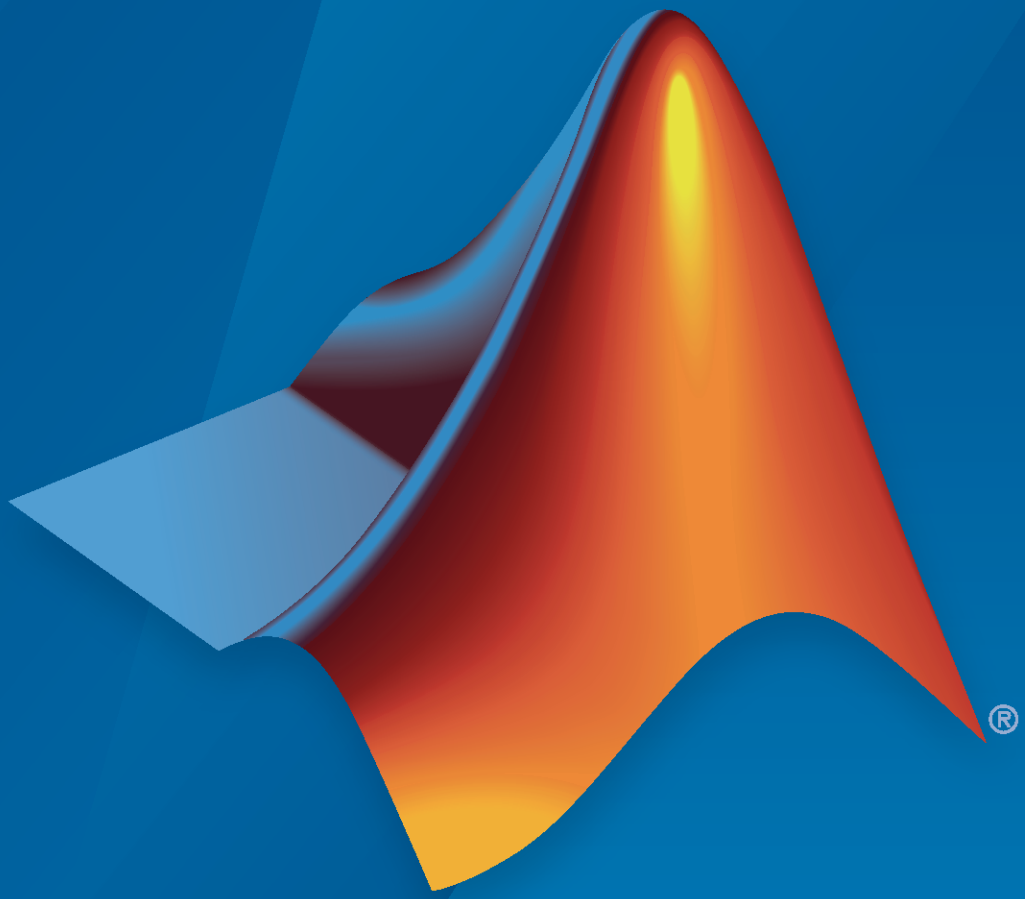
Deep Learning HDL Toolbox™

Reference



MATLAB®

R2023a

MathWorks®

How to Contact MathWorks

Latest news: `www.mathworks.com`

Sales and services: `www.mathworks.com/sales_and_services`

User community: `www.mathworks.com/matlabcentral`

Technical support: `www.mathworks.com/support/contact_us`

Phone: 508-647-7000

The MathWorks, Inc.
1 Apple Hill Drive
Natick, MA 01760-2098

Deep Learning HDL Toolbox™ Reference

Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See `www.mathworks.com/trademarks` for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Patents

MathWorks products are protected by one or more U.S. patents. Please see `www.mathworks.com/patents` for more information.

Revision History

September 2020	Online only	New for Version 1.0 (R2020b)
March 2021	Online only	Revised for Version 1.1 (R2021a)
September 2021	Online only	Revised for Version 1.2 (R2021b)
March 2022	Online only	Revised for Version 1.3 (Release R2022a)
September 2022	Online only	Revised for Version 1.4 (Release R2022b)
March 2023	Online only	Revised for Version 1.5 (Release R2023a)

Contents

Functions

dlhdl.Workflow class

Package: `dlhdl`

Configure deployment workflow for deep learning neural network

Description

Use the `dlhdl.Workflow` object to set options for compiling and deploying your deep learning network to a target FPGA. You create an object of the `dlhdl.Workflow` class for the specified deep learning network and FPGA bitstream. Use the object to:

- Compile the deep learning network.
- Estimate the speed and throughput of your network on the specified FPGA device.
- Compile and deploy the neural network onto the FPGA.
- Predict the class of input images.
- Profile the results for the specified network and the FPGA.

Creation

`hW = dlhdl.Workflow('Network',Network,'Bitstream',Bitstream)` creates a workflow configuration object with a network object, bitstream, and target object to deploy your custom pretrained deep learning network object.

`hW = dlhdl.Workflow('Network',Network,'Bitstream',Bitstream,Name,Value)` creates a workflow configuration object with a network object and bitstream to deploy your custom pretrained deep learning network object, with additional options specified by one or more name-value pair arguments.

Input Arguments

`Bitstream` — Name of the FPGA bitstream

'' (default) | character vector

Name of the FPGA bitstream, specified as a character vector. Make sure that the bitstream name matches the data type and the FPGA board that you are targeting. For a list of provided bitstream names, see “Use Deep Learning on FPGA Bitstreams”.

Example: `'Bitstream'`, `'arria10soc_single'` specifies that you want to deploy the trained network with `single` data types to an Arria10 SoC board.

`Network` — Network object

`SeriesNetwork` object | `DAGNetwork` object | `dlquantizer` object | `dlnetwork` object

Name of the deep learning network object.

Example: `'network'`, `net` creates a workflow object for the saved pretrained network `net`. To specify `net`, you can import any of the existing supported pretrained networks or use transfer learning to adapt the network to your issue. See “Supported Pretrained Networks”.

When the `network` is of type `dlnetwork` :

- The `dlnetwork` object must be initialized.
- The first layer of the `dlnetwork` object must be an image input layer.

```
net = resnet18;
hW = dlhdl.Workflow('Network',net,'Bitstream','zcu102_single');
```

Example: `'network', dlquantizeObj` creates a workflow object for the quantized network object `dlquantizeObj`. To specify `dlquantizeObj`, you can import any of the supported existing pretrained networks and create an object by using the `dlquantizer` class. For information on supported networks, see “Supported Pretrained Networks”.

```
net = resnet18;
dlquantObj = dlquantizer(net,'ExecutionEnvironment','FPGA');
dlquantObj.calibrate(imdsTrain);
hW = dlhdl.Workflow('Network',dlquantObj,'Bitstream','zcu102_int8');
```

Properties

`'Target'` — `dlhdl.Target` object to deploy network and bitstream to the target device

`hTarget`

Target object specified as `dlhdl.Target` object

Example: `'Target',hTarget`

```
hTarget = dlhdl.Target('Intel','Interface','JTAG')
hW = dlhdl.Workflow('network', snet,'Bitstream','arria10soc_single','Target',hTarget);
```

Methods

Public Methods

activations	Retrieve intermediate layer results for deployed deep learning network
compile	Compile workflow object
deploy	Deploy the specified neural network to the target FPGA board
getBuildInfo	Retrieve bitstream resource utilization
predict	Predict responses by using deployed network
predictAndUpdateState	Predict responses by using a trained and deployed recurrent neural network and update the deployed network state
resetState	Reset state parameters of deployed neural network

Examples

Create Workflow Object by using Property Name Value Pairs

```
snet = vgg19;
hW = dlhdl.Workflow('Network',snet,'Bitstream','arria10soc_single', 'Target', hTarget);
```

Create Workflow Object with Quantized Network Object

```
snet = getLogoNetwork;
dlquantObj = dlquantizer(snet,'ExecutionEnvironment','FPGA');
Image = imageDatastore('heineken.png','Labels','Heineken');
dlquantObj.calibrate(Image);
hTarget = dlhdl.Target('Xilinx','Interface','Ethernet');
hW = dlhdl.Workflow('Network',dlquantObj,'Bitstream','zcu102_int8','Target',hTarget);
```

Version History

Introduced in R2020b

See Also

Objects

`dlhdl.Target` | `dlquantizer` | `dlquantizationOptions`

Topics

“Prototype Deep Learning Networks on FPGA and SoC Devices”
“Quantization of Deep Neural Networks”

activations

Class: `dlhdl.Workflow`
Package: `dlhdl`

Retrieve intermediate layer results for deployed deep learning network

Syntax

```
act = activations(workflowObject,image,layername)
act = activations(workflowObject,image,layername, Name,Value)
```

Description

`act = activations(workflowObject,image,layername)` returns intermediate layer activation data results for the image data in `image`, and the name of the layer specified in `layername`. The result size depends on the output size of the layer. The layer output size can be retrieved by using `analyzeNetwork`.

`act = activations(workflowObject,image,layername, Name,Value)`returns intermediate layer activation data results for the image data in `image`, and the name of the layer specified in `layername`, with additional options specified by one or more `Name,Value` pair arguments. The result size depends on the output size of the layer. The layer output size can be retrieved by using `analyzeNetwork`.

Input Arguments

`workflowObject` — Workflow

`dlhdl.Workflow` object

Workflow, specified as a `dlhdl.Workflow` object.

`image` — Input image

m-by-*n*-by-*k* numeric array

Input image, specified as a *m*-by-*n*-by-*k* numeric array. *m*, *n*, and *k* must match the dimensions of the deep learning network input image layer. For example, for the LogoNet network, resize the input images to a 227-by-227-by-3 array.

Data Types: `single`

`layername` — Layer to extract activations from

'' (default) | character vector

Layer to extract activations from, specified as a character vector.

To compute the activations of a `SeriesNetwork` object, specify the layer as a character vector corresponding to the layer name.

To compute the activations of a `DAGNetwork` object, specify the layer as a character vector corresponding to the layer name. If the layer has multiple outputs, specify the layer and output as the

layer name, followed by the character "/", followed by the name of the layer output. That is, layername of the form 'layerName/outputName'.

These limitations apply when extracting activations from layers:

- A maxPooling2DLayer which has the HasUnpoolingIndices argument set to true has three outputs out, indices, and size. Activations are supported only for the output out.

Example: 'conv1'

Example: 'mpool/out'

Name-Value Pair Arguments

Specify optional pairs of arguments as Name1=Value1,...,NameN=ValueN, where Name is the argument name and Value is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose Name *in quotes.*

Profiler — Flag that returns profiling results
'off' (default) | 'on'

Flag to return profiling results for the deep learning network deployed to the target board.

Example: 'Profiler','on'

Output Arguments

act — Intermediate layer activation data
array of single

Intermediate layer activation data, returned as an array of singles. The array size depends on the layer output size. For example, for the ResNet-18 network pool1 layer, the size of the returned result array is 56-by-56-by-64.

Examples

Visualize Activations of a Deep Learning Network by Using LogoNet

This example shows how to feed an image to a convolutional neural network and display the activations of the different layers of the network. Examine the activations and discover which features the network learns by comparing areas of activation to the original image. Channels in earlier layers learn simple features like color and edges, while channels in the deeper layers learn complex features. Identifying features in this way can help you understand what the network has learned.

Logo Recognition Network

Logos assist in brand identification and recognition. Many companies incorporate their logos in advertising, documentation materials, and promotions. The logo recognition network (LogoNet) was developed in MATLAB® and can recognize 32 logos under various lighting conditions and camera motions. Because this network focuses only on recognition, you can use it in applications where localization is not required.

Prerequisites

- Intel® Arria10 SoC development kit
- Deep Learning HDL Toolbox™ Support Package for Intel FPGA and SoC
- Deep Learning Toolbox™
- Deep Learning HDL Toolbox™
- Computer Vision Toolbox™

Load Pretrained Series Network

To load the pretrained series network LogoNet, enter:

```
snet = getLogoNetwork;
```

Create Target Object

Create a target object that has a custom name for your target device and an interface to connect your target device to the host computer. Interface options are JTAG and Ethernet. To use JTAG, install Intel™ Quartus™ Prime Standard Edition 20.1. Set up the path to your installed Intel Quartus Prime executable if it is not already set up. For example, to set the toolpath, enter:

```
% hdlsetuptoolpath('ToolName', 'Altera Quartus II','ToolPath', 'C:\altera\20.1\quartus\bin64');
```

To create the target object, enter:

```
hTarget = dlhdl.Target('Intel','Interface','JTAG');
```

Create Workflow Object

Create an object of the `dlhdl.Workflow` class. When you create the object, specify the network and the bitstream name. Specify the saved pretrained LogoNet neural network, `snet`, as the network. Make sure that the bitstream name matches the data type and the FPGA board that you are targeting. In this example, the target FPGA board is the Intel Arria10 SOC board. The bitstream uses a single data type.

```
hW = dlhdl.Workflow('network', snet, 'Bitstream', 'arria10soc_single','Target',hTarget);
```

Read and show an image. Save its size for future use.

```
im = imread('ferrari.jpg');
imshow(im)
```

```
imgSize = size(im);
imgSize = imgSize(1:2);
```

View Network Architecture

Analyze the network to see which layers you can view. The convolutional layers perform convolutions by using learnable parameters. The network learns to identify useful features, often including one feature per channel. The first convolutional layer has 64 channels.

```
analyzeNetwork(snet)
```

The Image Input layer specifies the input size. Before passing the image through the network, you can resize it. The network can also process larger images.. If you feed the network larger images, the activations also become larger. Because the network is trained on images of size 227-by-227, it is not trained to recognize larger objects or features.

Show Activations of First Maxpool Layer

Investigate features by observing which areas in the maxpool layers activate on an image and comparing that image to the corresponding areas in the original images. Each layer of a convolutional neural network consists of many 2-D arrays called *channels*. Pass the image through the network and examine the output activations of the `maxpool_1` layer.

```
act1 = hW.activations(single(im),'maxpool_1','Profiler','on');
```

```
          offset_name          offset_address     allocated_space
    ___________________________    ______________    _________________

    "InputDataOffset"              "0x00000000"     "24.0 MB"
    "OutputResultOffset"           "0x01800000"     "136.0 MB"
    "SystemBufferOffset"           "0x0a000000"     "64.0 MB"
    "InstructionDataOffset"        "0x0e000000"     "8.0 MB"
    "ConvWeightDataOffset"         "0x0e800000"     "4.0 MB"
    "EndOffset"                    "0x0ec00000"     "Total: 236.0 MB"
```

```
### Programming FPGA Bitstream using JTAG...
### Programming the FPGA bitstream has been completed successfully.

### Finished writing input activations.
### Running single input activations.


              Deep Learning Processor Profiler Performance Results

                    LastLayerLatency(cycles)   LastLayerLatency(seconds)       FramesNum      Tota
                          -------------              -------------               ---------        --
Network                    10182024                    0.06788                     1           101
    conv_module            10182024                    0.06788
        conv_1              7088885                    0.04726
        maxpool_1           3093166                    0.02062
 * The clock frequency of the DL processor is: 150MHz
```

The activations are returned as a 3-D array, with the third dimension indexing the channel on the `maxpool_1` layer. To show these activations using the `imtile` function, reshape the array to 4-D. The third dimension in the input to `imtile` represents the image color. Set the third dimension to have size 1 because the activations do not have color. The fourth dimension indexes the channel.

```
sz = size(act1);
act1 = reshape(act1,[sz(1) sz(2) 1 sz(3)]);
```

Display the activations. Each activation can take any value, so normalize the output using the `mat2gray`. All activations are scaled so that the minimum activation is 0 and the maximum activation is 1. Display the 96 images on an 12-by-8 grid, one for each channel in the layer.

```
I = imtile(mat2gray(act1),'GridSize',[12 8]);
imshow(I)
```

Investigate Activations in Specific Channels

Each tile in the activations grid is the output of a channel in the `maxpool_1` layer. White pixels represent strong positive activations and black pixels represent strong negative activations. A channel that is mostly gray does not activate as strongly on the input image. The position of a pixel in the activation of a channel corresponds to the same position in the original image. A white pixel at a location in a channel indicates that the channel is strongly activated at that position.

Resize the activations in channel 33 to be the same size as the original image and display the activations.

```
act1ch33 = act1(:,:,:,22);
act1ch33 = mat2gray(act1ch33);
act1ch33 = imresize(act1ch33,imgSize);

I = imtile({im,act1ch33});
imshow(I)
```

Find Strongest Activation Channel

Find interesting channels by programmatically investigating channels with large activations. Find the channel that has the largest activation by using the `max` function, resize the channel output, and display the activations.

```
[maxValue,maxValueIndex] = max(max(max(act1)));
act1chMax = act1(:,:,:,maxValueIndex);
act1chMax = mat2gray(act1chMax);
act1chMax = imresize(act1chMax,imgSize);

I = imtile({im,act1chMax});
imshow(I)
```

Compare the strongest activation channel image to the original image. This channel activates on edges. It activates positively on light left/dark right edges and negatively on dark left/light right edges.

Version History

Introduced in R2020b

See Also

`compile` | `deploy` | `getBuildInfo` | `predict` | `predictAndUpdateState` | `resetState`

compile

Class: `dlhdl.Workflow`
Package: `dlhdl`

Compile workflow object

Syntax

```
compile(workflowObject)
compile(workflowObject,Name,Value)
```

Description

`compile(workflowObject)` compiles the `dlhdl.Workflow` object and generates the parameters for deploying the network on the target device.

`compile(workflowObject,Name,Value)` compiles the `dlhdl.Workflow` object and generates the parameters for deploying the network on the target device, with additional options specified by one or more `Name,Value` pair arguments.

The function returns two matrices. One matrix describes the layers of the network. The `Conv Controller (Scheduling)` and the `FC Controller (Scheduling)` modules in the deep learning processor IP use this matrix to schedule the convolution and fully connected layer operations. The second matrix contains the weights, biases, and inputs of the neural network. This information is loaded onto the DDR memory and used by the `Generic Convolution Processor` and the `Generic FC Processor` in the deep learning processor.

Input Arguments

`workflowObject` — Workflow

`dlhdl.Workflow` object

Workflow, specified as a `dlhdl.Workflow` object.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

`InputFrameNumberLimit` — Maximum input frame number limit

integer

Parameter to specify maximum input frame number limit to calculate DDR memory access allocation.

Example: `'InputFrameNumberLimit',30`

`HardwareNormalization` — Flag to enable hardware implementation of image input layer normalization function
'auto' (default) | 'on | 'off'

Flag to enable hardware implementation of image input layer normalization function , specified as a string or character vector.

Example: `HardwareNormalization = "auto"`

Examples

Compile the `dlhdl.Workflow` object

Compile the `dlhdl.Workflow` object, for deployment to the Intel® Arria® 10 SoC development kit that has `single` data types.

Create a `dlhdl.Workflow` object and then use the `compile` function to deploy the pretrained network to the target hardware.

```
snet = vgg19;
hT = dlhdl.Target('Intel');
hW = dlhdl.Workflow('network', snet, 'Bitstream', 'arria10soc_single','Target',hT);
hW.compile
```

Once the code is executed the result is:

```
  hW.compile
          offset_name          offset_address     allocated_space
    _____________________    ______________    _________________

    "InputDataOffset"          "0x00000000"     "24.0 MB"
    "OutputResultOffset"       "0x01800000"     "4.0 MB"
    "SystemBufferOffset"       "0x01c00000"     "52.0 MB"
    "InstructionDataOffset"    "0x05000000"     "20.0 MB"
    "ConvWeightDataOffset"     "0x06400000"     "276.0 MB"
    "FCWeightDataOffset"       "0x17800000"     "472.0 MB"
    "EndOffset"                "0x35000000"     "Total: 848.0 MB"


ans =

  struct with fields:

       Operators: [1×1 struct]
    LayerConfigs: [1×1 struct]
      NetConfigs: [1×1 struct]
```

Generate DDR Memory Offsets Based On Number of Input Frames

1. Create a `dlhdl.Workflow` object and then use the `compile` function with optional argument of `InputFrameNumberLimit` to deploy the pretrained network to the target hardware.

   ```
   net = resnet18;
   hT = dlhdl.Target('Xilinx');
   hW = dlhdl.Workflow('Network', net, 'Bitstream', 'zcu102_single','Target',hT);
   hW.compile('InputFrameNumberLimit',30);
   ```
2. The result of the code execution is:

```
### Compiling network for Deep Learning FPGA prototyping ...
### Targeting FPGA bitstream zcu102_single.
### The network includes the following layers:
     1   'data'                              Image Input                 224×224×3 images with 'zscore' normalization
     2   'conv1'                             Convolution                 64 7×7×3 convolutions with stride [2  2] and padding [3
     3   'bn_conv1'                          Batch Normalization         Batch normalization with 64 channels
     4   'conv1_relu'                        ReLU                        ReLU
     5   'pool1'                             Max Pooling                 3×3 max pooling with stride [2  2] and padding [1  1  1
     6   'res2a_branch2a'                    Convolution                 64 3×3×64 convolutions with stride [1  1] and padding [1
     7   'bn2a_branch2a'                     Batch Normalization         Batch normalization with 64 channels
     8   'res2a_branch2a_relu'               ReLU                        ReLU
     9   'res2a_branch2b'                    Convolution                 64 3×3×64 convolutions with stride [1  1] and padding [1
    10   'bn2a_branch2b'                     Batch Normalization         Batch normalization with 64 channels
    11   'res2a'                             Addition                    Element-wise addition of 2 inputs
    12   'res2a_relu'                        ReLU                        ReLU
    13   'res2b_branch2a'                    Convolution                 64 3×3×64 convolutions with stride [1  1] and padding [1
    14   'bn2b_branch2a'                     Batch Normalization         Batch normalization with 64 channels
    15   'res2b_branch2a_relu'               ReLU                        ReLU
    16   'res2b_branch2b'                    Convolution                 64 3×3×64 convolutions with stride [1  1] and padding [1
    17   'bn2b_branch2b'                     Batch Normalization         Batch normalization with 64 channels
    18   'res2b'                             Addition                    Element-wise addition of 2 inputs
    19   'res2b_relu'                        ReLU                        ReLU
    20   'res3a_branch2a'                    Convolution                 128 3×3×64 convolutions with stride [2  2] and padding [1
    21   'bn3a_branch2a'                     Batch Normalization         Batch normalization with 128 channels
    22   'res3a_branch2a_relu'               ReLU                        ReLU
    23   'res3a_branch2b'                    Convolution                 128 3×3×128 convolutions with stride [1  1] and padding
    24   'bn3a_branch2b'                     Batch Normalization         Batch normalization with 128 channels
    25   'res3a'                             Addition                    Element-wise addition of 2 inputs
    26   'res3a_relu'                        ReLU                        ReLU
    27   'res3a_branch1'                     Convolution                 128 1×1×64 convolutions with stride [2  2] and padding [0
    28   'bn3a_branch1'                      Batch Normalization         Batch normalization with 128 channels
    29   'res3b_branch2a'                    Convolution                 128 3×3×128 convolutions with stride [1  1] and padding
    30   'bn3b_branch2a'                     Batch Normalization         Batch normalization with 128 channels
    31   'res3b_branch2a_relu'               ReLU                        ReLU
    32   'res3b_branch2b'                    Convolution                 128 3×3×128 convolutions with stride [1  1] and padding
    33   'bn3b_branch2b'                     Batch Normalization         Batch normalization with 128 channels
    34   'res3b'                             Addition                    Element-wise addition of 2 inputs
    35   'res3b_relu'                        ReLU                        ReLU
    36   'res4a_branch2a'                    Convolution                 256 3×3×128 convolutions with stride [2  2] and padding
    37   'bn4a_branch2a'                     Batch Normalization         Batch normalization with 256 channels
    38   'res4a_branch2a_relu'               ReLU                        ReLU
    39   'res4a_branch2b'                    Convolution                 256 3×3×256 convolutions with stride [1  1] and padding
    40   'bn4a_branch2b'                     Batch Normalization         Batch normalization with 256 channels
    41   'res4a'                             Addition                    Element-wise addition of 2 inputs
    42   'res4a_relu'                        ReLU                        ReLU
    43   'res4a_branch1'                     Convolution                 256 1×1×128 convolutions with stride [2  2] and padding
    44   'bn4a_branch1'                      Batch Normalization         Batch normalization with 256 channels
    45   'res4b_branch2a'                    Convolution                 256 3×3×256 convolutions with stride [1  1] and padding
    46   'bn4b_branch2a'                     Batch Normalization         Batch normalization with 256 channels
    47   'res4b_branch2a_relu'               ReLU                        ReLU
    48   'res4b_branch2b'                    Convolution                 256 3×3×256 convolutions with stride [1  1] and padding
    49   'bn4b_branch2b'                     Batch Normalization         Batch normalization with 256 channels
    50   'res4b'                             Addition                    Element-wise addition of 2 inputs
    51   'res4b_relu'                        ReLU                        ReLU
    52   'res5a_branch2a'                    Convolution                 512 3×3×256 convolutions with stride [2  2] and padding
    53   'bn5a_branch2a'                     Batch Normalization         Batch normalization with 512 channels
    54   'res5a_branch2a_relu'               ReLU                        ReLU
    55   'res5a_branch2b'                    Convolution                 512 3×3×512 convolutions with stride [1  1] and padding
    56   'bn5a_branch2b'                     Batch Normalization         Batch normalization with 512 channels
    57   'res5a'                             Addition                    Element-wise addition of 2 inputs
    58   'res5a_relu'                        ReLU                        ReLU
    59   'res5a_branch1'                     Convolution                 512 1×1×256 convolutions with stride [2  2] and padding
    60   'bn5a_branch1'                      Batch Normalization         Batch normalization with 512 channels
    61   'res5b_branch2a'                    Convolution                 512 3×3×512 convolutions with stride [1  1] and padding
    62   'bn5b_branch2a'                     Batch Normalization         Batch normalization with 512 channels
    63   'res5b_branch2a_relu'               ReLU                        ReLU
    64   'res5b_branch2b'                    Convolution                 512 3×3×512 convolutions with stride [1  1] and padding
    65   'bn5b_branch2b'                     Batch Normalization         Batch normalization with 512 channels
    66   'res5b'                             Addition                    Element-wise addition of 2 inputs
    67   'res5b_relu'                        ReLU                        ReLU
    68   'pool5'                             2-D Global Average Pooling  2-D global average pooling
    69   'fc1000'                            Fully Connected             1000 fully connected layer
    70   'prob'                              Softmax                     softmax
    71   'ClassificationLayer_predictions'   Classification Output       crossentropyex with 'tench' and 999 other classes

### Optimizing network: Fused 'nnet.cnn.layer.BatchNormalizationLayer' into 'nnet.cnn.layer.Convolution2DLayer'
### Notice: The layer 'data' of type 'ImageInputLayer' is split into an image input layer 'data', an addition layer 'data_norm_add
### Notice: The layer 'prob' with type 'nnet.cnn.layer.SoftmaxLayer' is implemented in software.
### Notice: The layer 'ClassificationLayer_predictions' with type 'nnet.cnn.layer.ClassificationOutputLayer' is implemented in soft
### Compiling layer group: conv1>>pool1 ...
### Compiling layer group: conv1>>pool1 ... complete.
```

```
### Compiling layer group: res2a_branch2a>>res2a_branch2b ...
### Compiling layer group: res2a_branch2a>>res2a_branch2b ... complete.
### Compiling layer group: res2b_branch2a>>res2b_branch2b ...
### Compiling layer group: res2b_branch2a>>res2b_branch2b ... complete.
### Compiling layer group: res3a_branch1 ...
### Compiling layer group: res3a_branch1 ... complete.
### Compiling layer group: res3a_branch2a>>res3a_branch2b ...
### Compiling layer group: res3a_branch2a>>res3a_branch2b ... complete.
### Compiling layer group: res3b_branch2a>>res3b_branch2b ...
### Compiling layer group: res3b_branch2a>>res3b_branch2b ... complete.
### Compiling layer group: res4a_branch1 ...
### Compiling layer group: res4a_branch1 ... complete.
### Compiling layer group: res4a_branch2a>>res4a_branch2b ...
### Compiling layer group: res4a_branch2a>>res4a_branch2b ... complete.
### Compiling layer group: res4b_branch2a>>res4b_branch2b ...
### Compiling layer group: res4b_branch2a>>res4b_branch2b ... complete.
### Compiling layer group: res5a_branch1 ...
### Compiling layer group: res5a_branch1 ... complete.
### Compiling layer group: res5a_branch2a>>res5a_branch2b ...
### Compiling layer group: res5a_branch2a>>res5a_branch2b ... complete.
### Compiling layer group: res5b_branch2a>>res5b_branch2b ...
### Compiling layer group: res5b_branch2a>>res5b_branch2b ... complete.
### Compiling layer group: pool5 ...
### Compiling layer group: pool5 ... complete.
### Compiling layer group: fc1000 ...
### Compiling layer group: fc1000 ... complete.

### Allocating external memory buffers:

          offset_name          offset_address     allocated_space
    ___________________________    ______________    _______________

    "InputDataOffset"           "0x00000000"     "24.0 MB"
    "OutputResultOffset"        "0x01800000"     "4.0 MB"
    "SchedulerDataOffset"       "0x01c00000"     "8.0 MB"
    "SystemBufferOffset"        "0x02400000"     "28.0 MB"
    "InstructionDataOffset"     "0x04000000"     "4.0 MB"
    "ConvWeightDataOffset"      "0x04400000"     "52.0 MB"
    "FCWeightDataOffset"        "0x07800000"     "4.0 MB"
    "EndOffset"                 "0x07c00000"     "Total: 124.0 MB"

### Network compilation complete.
```

Compile dagnet network object

1. Create a `dlhdl.Workflow` object with `resnet18` as the network for deployment to a Xilinx® Zynq® UltraScale+™ MPSoC ZCU102 board which uses `single` data types.

```
net = resnet18;
hTarget = dlhdl.Target('Xilinx');
hW = dlhdl.Workflow('Network',snet,'Bitstream','zcu102_single','Target',hTarget);
```

2. Call the `compile` function on `hW`

```
hW.compile
```

Calling the `compile` function, returns:

```
### Compiling network for Deep Learning FPGA prototyping ...
### Targeting FPGA bitstream zcu102_single ...
### The network includes the following layers:

     1   'data'                          Image Input             224×224×3 images with 'zscore' normalization
     2   'conv1'                         Convolution             64 7×7×3 convolutions with stride [2  2] and p
     3   'bn_conv1'                      Batch Normalization     Batch normalization with 64 channels
     4   'conv1_relu'                    ReLU                    ReLU
     5   'pool1'                         Max Pooling             3×3 max pooling with stride [2  2] and padding
     6   'res2a_branch2a'                Convolution             64 3×3×64 convolutions with stride [1  1] and
     7   'bn2a_branch2a'                 Batch Normalization     Batch normalization with 64 channels
     8   'res2a_branch2a_relu'           ReLU                    ReLU
     9   'res2a_branch2b'                Convolution             64 3×3×64 convolutions with stride [1  1] and
    10   'bn2a_branch2b'                 Batch Normalization     Batch normalization with 64 channels
    11   'res2a'                         Addition                Element-wise addition of 2 inputs
```

```
    12   'res2a_relu'                        ReLU                    ReLU
    13   'res2b_branch2a'                    Convolution             64 3×3×64 convolutions with stride [1  1] and
    14   'bn2b_branch2a'                     Batch Normalization     Batch normalization with 64 channels
    15   'res2b_branch2a_relu'               ReLU                    ReLU
    16   'res2b_branch2b'                    Convolution             64 3×3×64 convolutions with stride [1  1] and
    17   'bn2b_branch2b'                     Batch Normalization     Batch normalization with 64 channels
    18   'res2b'                             Addition                Element-wise addition of 2 inputs
    19   'res2b_relu'                        ReLU                    ReLU
    20   'res3a_branch2a'                    Convolution             128 3×3×64 convolutions with stride [2  2] and
    21   'bn3a_branch2a'                     Batch Normalization     Batch normalization with 128 channels
    22   'res3a_branch2a_relu'               ReLU                    ReLU
    23   'res3a_branch2b'                    Convolution             128 3×3×128 convolutions with stride [1  1] an
    24   'bn3a_branch2b'                     Batch Normalization     Batch normalization with 128 channels
    25   'res3a'                             Addition                Element-wise addition of 2 inputs
    26   'res3a_relu'                        ReLU                    ReLU
    27   'res3a_branch1'                     Convolution             128 1×1×64 convolutions with stride [2  2] and
    28   'bn3a_branch1'                      Batch Normalization     Batch normalization with 128 channels
    29   'res3b_branch2a'                    Convolution             128 3×3×128 convolutions with stride [1  1] an
    30   'bn3b_branch2a'                     Batch Normalization     Batch normalization with 128 channels
    31   'res3b_branch2a_relu'               ReLU                    ReLU
    32   'res3b_branch2b'                    Convolution             128 3×3×128 convolutions with stride [1  1] an
    33   'bn3b_branch2b'                     Batch Normalization     Batch normalization with 128 channels
    34   'res3b'                             Addition                Element-wise addition of 2 inputs
    35   'res3b_relu'                        ReLU                    ReLU
    36   'res4a_branch2a'                    Convolution             256 3×3×128 convolutions with stride [2  2] an
    37   'bn4a_branch2a'                     Batch Normalization     Batch normalization with 256 channels
    38   'res4a_branch2a_relu'               ReLU                    ReLU
    39   'res4a_branch2b'                    Convolution             256 3×3×256 convolutions with stride [1  1] an
    40   'bn4a_branch2b'                     Batch Normalization     Batch normalization with 256 channels
    41   'res4a'                             Addition                Element-wise addition of 2 inputs
    42   'res4a_relu'                        ReLU                    ReLU
    43   'res4a_branch1'                     Convolution             256 1×1×128 convolutions with stride [2  2] an
    44   'bn4a_branch1'                      Batch Normalization     Batch normalization with 256 channels
    45   'res4b_branch2a'                    Convolution             256 3×3×256 convolutions with stride [1  1] an
    46   'bn4b_branch2a'                     Batch Normalization     Batch normalization with 256 channels
    47   'res4b_branch2a_relu'               ReLU                    ReLU
    48   'res4b_branch2b'                    Convolution             256 3×3×256 convolutions with stride [1  1] an
    49   'bn4b_branch2b'                     Batch Normalization     Batch normalization with 256 channels
    50   'res4b'                             Addition                Element-wise addition of 2 inputs
    51   'res4b_relu'                        ReLU                    ReLU
    52   'res5a_branch2a'                    Convolution             512 3×3×256 convolutions with stride [2  2] an
    53   'bn5a_branch2a'                     Batch Normalization     Batch normalization with 512 channels
    54   'res5a_branch2a_relu'               ReLU                    ReLU
    55   'res5a_branch2b'                    Convolution             512 3×3×512 convolutions with stride [1  1] an
    56   'bn5a_branch2b'                     Batch Normalization     Batch normalization with 512 channels
    57   'res5a'                             Addition                Element-wise addition of 2 inputs
    58   'res5a_relu'                        ReLU                    ReLU
    59   'res5a_branch1'                     Convolution             512 1×1×256 convolutions with stride [2  2] an
    60   'bn5a_branch1'                      Batch Normalization     Batch normalization with 512 channels
    61   'res5b_branch2a'                    Convolution             512 3×3×512 convolutions with stride [1  1] an
    62   'bn5b_branch2a'                     Batch Normalization     Batch normalization with 512 channels
    63   'res5b_branch2a_relu'               ReLU                    ReLU
    64   'res5b_branch2b'                    Convolution             512 3×3×512 convolutions with stride [1  1] an
    65   'bn5b_branch2b'                     Batch Normalization     Batch normalization with 512 channels
    66   'res5b'                             Addition                Element-wise addition of 2 inputs
    67   'res5b_relu'                        ReLU                    ReLU
    68   'pool5'                             Global Average Pooling  Global average pooling
    69   'fc1000'                            Fully Connected         1000 fully connected layer
    70   'prob'                              Softmax                 softmax
    71   'ClassificationLayer_predictions'   Classification Output   crossentropyex with 'tench' and 999 other clas

### Optimizing series network: Fused 'nnet.cnn.layer.BatchNormalizationLayer' into 'nnet.cnn.layer.Convolution2DLaye
5 Memory Regions created.

Skipping: data
Compiling leg: conv1>>pool1 ...
Compiling leg: conv1>>pool1 ... complete.
Compiling leg: res2a_branch2a>>res2a_branch2b ...
Compiling leg: res2a_branch2a>>res2a_branch2b ... complete.
Compiling leg: res2b_branch2a>>res2b_branch2b ...
Compiling leg: res2b_branch2a>>res2b_branch2b ... complete.
Compiling leg: res3a_branch2a>>res3a_branch2b ...
```

```
Compiling leg: res3a_branch2a>>res3a_branch2b ... complete.
Compiling leg: res3a_branch1 ...
Compiling leg: res3a_branch1 ... complete.
Compiling leg: res3b_branch2a>>res3b_branch2b ...
Compiling leg: res3b_branch2a>>res3b_branch2b ... complete.
Compiling leg: res4a_branch2a>>res4a_branch2b ...
Compiling leg: res4a_branch2a>>res4a_branch2b ... complete.
Compiling leg: res4a_branch1 ...
Compiling leg: res4a_branch1 ... complete.
Compiling leg: res4b_branch2a>>res4b_branch2b ...
Compiling leg: res4b_branch2a>>res4b_branch2b ... complete.
Compiling leg: res5a_branch2a>>res5a_branch2b ...
Compiling leg: res5a_branch2a>>res5a_branch2b ... complete.
Compiling leg: res5a_branch1 ...
Compiling leg: res5a_branch1 ... complete.
Compiling leg: res5b_branch2a>>res5b_branch2b ...
Compiling leg: res5b_branch2a>>res5b_branch2b ... complete.
Compiling leg: pool5 ...
Compiling leg: pool5 ... complete.
Compiling leg: fc1000 ...
Compiling leg: fc1000 ... complete.
Skipping: prob
Skipping: ClassificationLayer_predictions
Creating Schedule...
.........................
Creating Schedule...complete.
Creating Status Table...
.........................
Creating Status Table...complete.
Emitting Schedule...
.........................
Emitting Schedule...complete.
Emitting Status Table...
.........................
Emitting Status Table...complete.

### Allocating external memory buffers:

          offset_name          offset_address     allocated_space
    ___________________________    ______________    _________________

    "InputDataOffset"           "0x00000000"     "24.0 MB"
    "OutputResultOffset"        "0x01800000"     "4.0 MB"
    "SchedulerDataOffset"       "0x01c00000"     "4.0 MB"
    "SystemBufferOffset"        "0x02000000"     "28.0 MB"
    "InstructionDataOffset"     "0x03c00000"     "4.0 MB"
    "ConvWeightDataOffset"      "0x04000000"     "52.0 MB"
    "FCWeightDataOffset"        "0x07400000"     "4.0 MB"
    "EndOffset"                 "0x07800000"     "Total: 120.0 MB"

### Network compilation complete.


ans =

  struct with fields:

            weights: [1×1 struct]
       instructions: [1×1 struct]
          registers: [1×1 struct]
   syncInstructions: [1×1 struct]
```

Enable Hardware Implementation of Input Image Layer Normalization Function

1 Create a `dlhdl.Workflow` object with `resnet18` as the network for deployment to a Xilinx Zynq UltraScale+ MPSoC ZCU102 board which uses `single` data types.

```
net = resnet18;
hTarget = dlhdl.Target('Xilinx',Interface = 'Ethernet');
hW = dlhdl.Workflow(Network = net,Bitstream ='zcu102_single',Target = hTarget);
```

2 Call the `compile` function on `hW`. . Enable hardware implementation of the input image layer normalization function by setting the`HardwareNormalization` argument to `auto`.

```
hW.compile(HardwareNormalization = 'auto')
```

Calling the `compile` function, returns:

```
### Compiling network for Deep Learning FPGA prototyping ...
### Targeting FPGA bitstream zcu102_single.
### The network includes the following layers:
     1   'data'                              Image Input                 224×224×3 images with 'zscore' normalization
     2   'conv1'                             Convolution                 64 7×7×3 convolutions with stride [2  2] and padding [3
     3   'bn_conv1'                          Batch Normalization         Batch normalization with 64 channels
     4   'conv1_relu'                        ReLU                        ReLU
     5   'pool1'                             Max Pooling                 3×3 max pooling with stride [2  2] and padding [1  1  1
     6   'res2a_branch2a'                    Convolution                 64 3×3×64 convolutions with stride [1  1] and padding [1
     7   'bn2a_branch2a'                     Batch Normalization         Batch normalization with 64 channels
     8   'res2a_branch2a_relu'               ReLU                        ReLU
     9   'res2a_branch2b'                    Convolution                 64 3×3×64 convolutions with stride [1  1] and padding [1
    10   'bn2a_branch2b'                     Batch Normalization         Batch normalization with 64 channels
    11   'res2a'                             Addition                    Element-wise addition of 2 inputs
    12   'res2a_relu'                        ReLU                        ReLU
    13   'res2b_branch2a'                    Convolution                 64 3×3×64 convolutions with stride [1  1] and padding [1
    14   'bn2b_branch2a'                     Batch Normalization         Batch normalization with 64 channels
    15   'res2b_branch2a_relu'               ReLU                        ReLU
    16   'res2b_branch2b'                    Convolution                 64 3×3×64 convolutions with stride [1  1] and padding [1
    17   'bn2b_branch2b'                     Batch Normalization         Batch normalization with 64 channels
    18   'res2b'                             Addition                    Element-wise addition of 2 inputs
    19   'res2b_relu'                        ReLU                        ReLU
    20   'res3a_branch2a'                    Convolution                 128 3×3×64 convolutions with stride [2  2] and padding [1
    21   'bn3a_branch2a'                     Batch Normalization         Batch normalization with 128 channels
    22   'res3a_branch2a_relu'               ReLU                        ReLU
    23   'res3a_branch2b'                    Convolution                 128 3×3×128 convolutions with stride [1  1] and padding
    24   'bn3a_branch2b'                     Batch Normalization         Batch normalization with 128 channels
    25   'res3a'                             Addition                    Element-wise addition of 2 inputs
    26   'res3a_relu'                        ReLU                        ReLU
    27   'res3a_branch1'                     Convolution                 128 1×1×64 convolutions with stride [2  2] and padding [0
    28   'bn3a_branch1'                      Batch Normalization         Batch normalization with 128 channels
    29   'res3b_branch2a'                    Convolution                 128 3×3×128 convolutions with stride [1  1] and padding
    30   'bn3b_branch2a'                     Batch Normalization         Batch normalization with 128 channels
    31   'res3b_branch2a_relu'               ReLU                        ReLU
    32   'res3b_branch2b'                    Convolution                 128 3×3×128 convolutions with stride [1  1] and padding
    33   'bn3b_branch2b'                     Batch Normalization         Batch normalization with 128 channels
    34   'res3b'                             Addition                    Element-wise addition of 2 inputs
    35   'res3b_relu'                        ReLU                        ReLU
    36   'res4a_branch2a'                    Convolution                 256 3×3×128 convolutions with stride [2  2] and padding
    37   'bn4a_branch2a'                     Batch Normalization         Batch normalization with 256 channels
    38   'res4a_branch2a_relu'               ReLU                        ReLU
    39   'res4a_branch2b'                    Convolution                 256 3×3×256 convolutions with stride [1  1] and padding
    40   'bn4a_branch2b'                     Batch Normalization         Batch normalization with 256 channels
    41   'res4a'                             Addition                    Element-wise addition of 2 inputs
    42   'res4a_relu'                        ReLU                        ReLU
    43   'res4a_branch1'                     Convolution                 256 1×1×128 convolutions with stride [2  2] and padding
    44   'bn4a_branch1'                      Batch Normalization         Batch normalization with 256 channels
    45   'res4b_branch2a'                    Convolution                 256 3×3×256 convolutions with stride [1  1] and padding
    46   'bn4b_branch2a'                     Batch Normalization         Batch normalization with 256 channels
    47   'res4b_branch2a_relu'               ReLU                        ReLU
    48   'res4b_branch2b'                    Convolution                 256 3×3×256 convolutions with stride [1  1] and padding
    49   'bn4b_branch2b'                     Batch Normalization         Batch normalization with 256 channels
    50   'res4b'                             Addition                    Element-wise addition of 2 inputs
    51   'res4b_relu'                        ReLU                        ReLU
    52   'res5a_branch2a'                    Convolution                 512 3×3×256 convolutions with stride [2  2] and padding
    53   'bn5a_branch2a'                     Batch Normalization         Batch normalization with 512 channels
    54   'res5a_branch2a_relu'               ReLU                        ReLU
    55   'res5a_branch2b'                    Convolution                 512 3×3×512 convolutions with stride [1  1] and padding
    56   'bn5a_branch2b'                     Batch Normalization         Batch normalization with 512 channels
    57   'res5a'                             Addition                    Element-wise addition of 2 inputs
    58   'res5a_relu'                        ReLU                        ReLU
    59   'res5a_branch1'                     Convolution                 512 1×1×256 convolutions with stride [2  2] and padding
    60   'bn5a_branch1'                      Batch Normalization         Batch normalization with 512 channels
    61   'res5b_branch2a'                    Convolution                 512 3×3×512 convolutions with stride [1  1] and padding
    62   'bn5b_branch2a'                     Batch Normalization         Batch normalization with 512 channels
    63   'res5b_branch2a_relu'               ReLU                        ReLU
    64   'res5b_branch2b'                    Convolution                 512 3×3×512 convolutions with stride [1  1] and padding
    65   'bn5b_branch2b'                     Batch Normalization         Batch normalization with 512 channels
    66   'res5b'                             Addition                    Element-wise addition of 2 inputs
```

```
    67   'res5b_relu'                        ReLU                        ReLU
    68   'pool5'                             2-D Global Average Pooling  2-D global average pooling
    69   'fc1000'                            Fully Connected             1000 fully connected layer
    70   'prob'                              Softmax                     softmax
    71   'ClassificationLayer_predictions'   Classification Output       crossentropyex with 'tench' and 999 other classes

### Optimizing network: Fused 'nnet.cnn.layer.BatchNormalizationLayer' into 'nnet.cnn.layer.Convolution2DLayer'
### Notice: The layer 'data' of type 'ImageInputLayer' is split into an image input layer 'data', an addition layer 'data_norm_add
### Notice: The layer 'prob' with type 'nnet.cnn.layer.SoftmaxLayer' is implemented in software.
### Notice: The layer 'ClassificationLayer_predictions' with type 'nnet.cnn.layer.ClassificationOutputLayer' is implemented in soft
### Compiling layer group: conv1>>pool1 ...
### Compiling layer group: conv1>>pool1 ... complete.
### Compiling layer group: res2a_branch2a>>res2a_branch2b ...
### Compiling layer group: res2a_branch2a>>res2a_branch2b ... complete.
### Compiling layer group: res2b_branch2a>>res2b_branch2b ...
### Compiling layer group: res2b_branch2a>>res2b_branch2b ... complete.
### Compiling layer group: res3a_branch1 ...
### Compiling layer group: res3a_branch1 ... complete.
### Compiling layer group: res3a_branch2a>>res3a_branch2b ...
### Compiling layer group: res3a_branch2a>>res3a_branch2b ... complete.
### Compiling layer group: res3b_branch2a>>res3b_branch2b ...
### Compiling layer group: res3b_branch2a>>res3b_branch2b ... complete.
### Compiling layer group: res4a_branch1 ...
### Compiling layer group: res4a_branch1 ... complete.
### Compiling layer group: res4a_branch2a>>res4a_branch2b ...
### Compiling layer group: res4a_branch2a>>res4a_branch2b ... complete.
### Compiling layer group: res4b_branch2a>>res4b_branch2b ...
### Compiling layer group: res4b_branch2a>>res4b_branch2b ... complete.
### Compiling layer group: res5a_branch1 ...
### Compiling layer group: res5a_branch1 ... complete.
### Compiling layer group: res5a_branch2a>>res5a_branch2b ...
### Compiling layer group: res5a_branch2a>>res5a_branch2b ... complete.
### Compiling layer group: res5b_branch2a>>res5b_branch2b ...
### Compiling layer group: res5b_branch2a>>res5b_branch2b ... complete.
### Compiling layer group: pool5 ...
### Compiling layer group: pool5 ... complete.
### Compiling layer group: fc1000 ...
### Compiling layer group: fc1000 ... complete.

### Allocating external memory buffers:

          offset_name          offset_address     allocated_space
    _______________________    ______________    _________________

    "InputDataOffset"           "0x00000000"     "24.0 MB"
    "OutputResultOffset"        "0x01800000"     "4.0 MB"
    "SchedulerDataOffset"       "0x01c00000"     "8.0 MB"
    "SystemBufferOffset"        "0x02400000"     "28.0 MB"
    "InstructionDataOffset"     "0x04000000"     "4.0 MB"
    "ConvWeightDataOffset"      "0x04400000"     "52.0 MB"
    "FCWeightDataOffset"        "0x07800000"     "4.0 MB"
    "EndOffset"                 "0x07c00000"     "Total: 124.0 MB"

### Network compilation complete.


ans =

  struct with fields:

            weights: [1×1 struct]
       instructions: [1×1 struct]
          registers: [1×1 struct]
    syncInstructions: [1×1 struct]
        constantData: {{1×2 cell}  [0.0171 0.0175 0.0174 0 0.0171 0.0175 0.0174 0 0.0171 0.0175 0.0174 0 0.0171 0.0175 0.0174 0 …
```

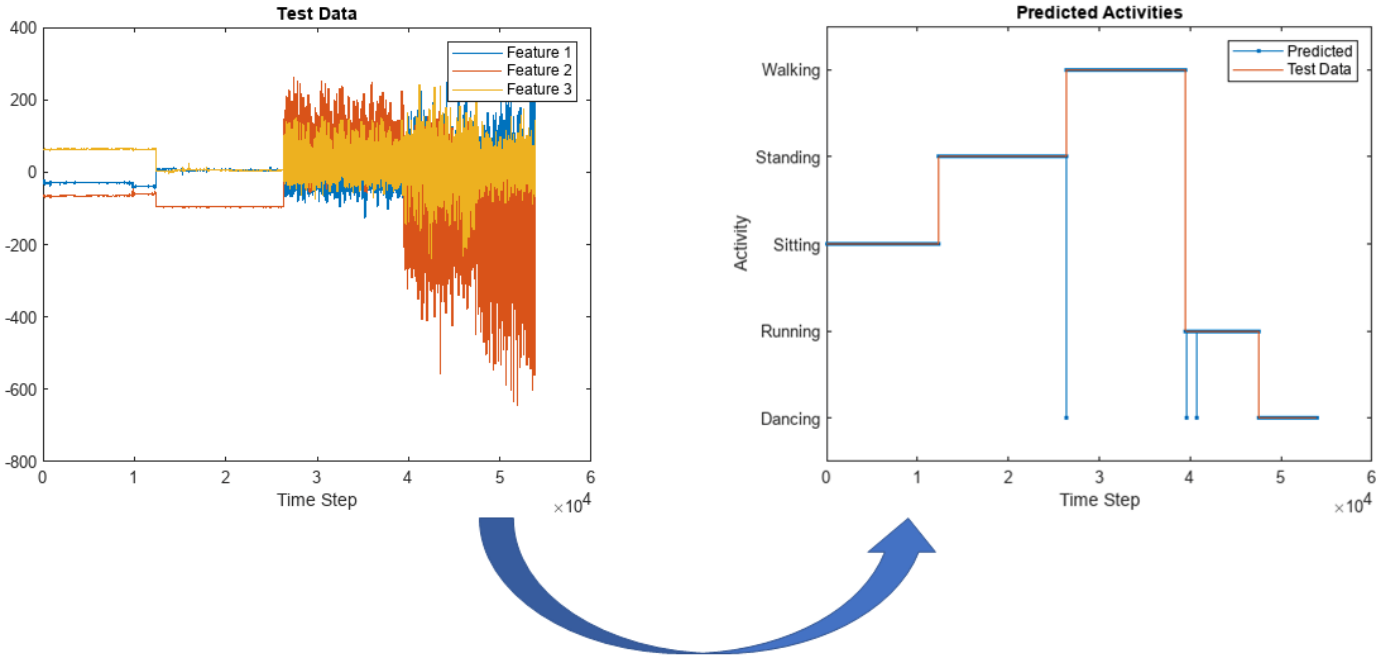
During compilation the compiler splits the image input layer into an image input layer, addition layer, and multiplication layer for hardware implementation.

Run Sequence-to-Sequence Classification on FPGAs by Using Deep Learning HDL Toolbox

This example shows how to create, compile, and deploy a long short-term memory (LSTM) network trained on accelerometer data from human movement by using the Deep Learning HDL Toolbox™

Support Package for Xilinx FPGA and SoC. Use the deployed network to classify human activity based on sequence input data. Use MATLAB® to retrieve the prediction results from the target device.

The network attached to this example was trained using the “Sequence-to-Sequence Classification Using Deep Learning”. This example uses sensor data obtained from a smartphone worn on the body. This example deploys an LSTM network trained to recognize the activity of the wearer given time series data that represents accelerometer readings in three different directions. The graphs below show the raw data for these accelerometer readings over time and the resulting classifications. The training data contains time series data for seven people. Each sequence has three features and varies in length. The data set contains six training observations and one test observation.



Prerequisites

- Xilinx® Zynq® Ultrascale+™ ZCU102 SoC development kit
- Deep Learning HDL Toolbox™ Support Package for Xilinx FPGA and SoC
- Deep Learning Toolbox™
- Deep Learning HDL Toolbox™

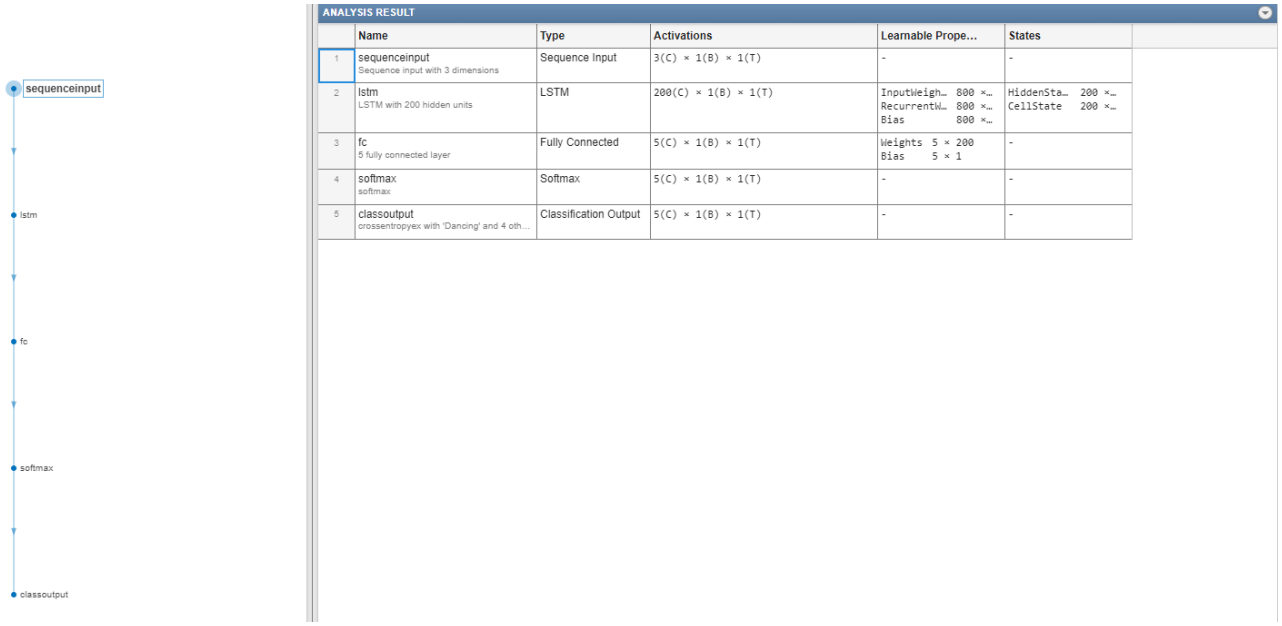
Load the Pretrained Network

To load the pretrained human body movement network, enter:

```
load SequenceToSequenceClassification
```

View the layers of the network by using the `analyzeNetwork` function. The function returns a graphical representation of the network and detailed parameter settings of the layers in the network.

```
analyzeNetwork(net)
```



Define FPGA Board Interface

Define the target FPGA board programming interface by using the `dlhdl.Target` object. Specify that the interface is for a Xilinx board with an Ethernet interface.

To create the target object, enter:

```
hTarget = dlhdl.Target('Xilinx','Interface','Ethernet');
```

To use the JTAG interface, install Xilinx™ Vivado™ Design Suite 2020.2. To set the Xilinx Vivado tool path, enter:

```
hdlsetuptoolpath('ToolName', 'Xilinx Vivado', 'ToolPath', 'C:\Xilinx\Vivado\2020.2\bin\vivado.ba
```

Prepare Network for Deployment

Prepare the network for deployment by creating a `dlhdl.Workflow` object. Specify the network and bitstream name. Ensure that the bitstream name matches the data type and FPGA board. In this example the target FPGA board is the Xilinx ZCU102 SOC board. The bitstream uses a single data type.

```
hW = dlhdl.Workflow('network', net, 'Bitstream', 'zcu102_lstm_single','Target',hTarget);
```

To run the example in a Xilinx ZC706 board, enter:

```
hW = dlhdl.Workflow('Network', snet, 'Bitstream', 'zc706_lstm_single','Target',hTarget);
```

Compile Network

Run the `compile` method of the `dlhdl.Workflow` object to compile the network and generate the instructions, weights, and biases for deployment. The total number of frames exceeds the default value of 30. Set the `InputFrameNumberLimit` name-value argument to `10000` to run predictions in chunks of 10,000 frames to prevent timeouts.

```
dn = compile(hW,'InputFrameNumberLimit',10000)
```

```
### Compiling network for Deep Learning FPGA prototyping ...
### Targeting FPGA bitstream zcu102_lstm_single.
### The network includes the following layers:
     1   'sequenceinput'   Sequence Input          Sequence input with 3 dimensions
     2   'lstm'            LSTM                    LSTM with 200 hidden units
     3   'fc'              Fully Connected         5 fully connected layer
     4   'softmax'         Softmax                 softmax
     5   'classoutput'     Classification Output   crossentropyex with 'Dancing' and 4 other clas

### Notice: The layer 'sequenceinput' with type 'nnet.cnn.layer.ImageInputLayer' is implemented i
### Notice: The layer 'softmax' with type 'nnet.cnn.layer.SoftmaxLayer' is implemented in softwar
### Notice: The layer 'classoutput' with type 'nnet.cnn.layer.ClassificationOutputLayer' is imple
### Compiling layer group: lstm.wi ...
### Compiling layer group: lstm.wi ... complete.
### Compiling layer group: lstm.wo ...
### Compiling layer group: lstm.wo ... complete.
### Compiling layer group: lstm.wg ...
### Compiling layer group: lstm.wg ... complete.
### Compiling layer group: lstm.wf ...
### Compiling layer group: lstm.wf ... complete.
### Compiling layer group: fc ...
### Compiling layer group: fc ... complete.

### Allocating external memory buffers:

          offset_name          offset_address    allocated_space
    ___________________________    ______________    _______________

    "InputDataOffset"             "0x00000000"     "4.0 MB"
    "OutputResultOffset"          "0x00400000"     "4.0 MB"
    "SchedulerDataOffset"         "0x00800000"     "4.0 MB"
    "SystemBufferOffset"          "0x00c00000"     "20.0 MB"
    "InstructionDataOffset"       "0x02000000"     "4.0 MB"
    "FCWeightDataOffset"          "0x02400000"     "4.0 MB"
    "EndOffset"                   "0x02800000"     "Total: 40.0 MB"

### Network compilation complete.

dn = struct with fields:
             weights: [1×1 struct]
        instructions: [1×1 struct]
           registers: [1×1 struct]
    syncInstructions: [1×1 struct]
        constantData: {}
```

Program Bitstream onto FPGA and Download Network Weights

To deploy the network on the Xilinx ZCU102 SoC hardware, run the `deploy` method of the `dlhdl.Workflow` object. This function uses the output of the `compile` function to program the FPGA board and download the network weights and biases. The `deploy` function starts programming the FPGA device and displays progress messages, and the required time to deploy the network.

```
deploy(hW)
```

```
### FPGA bitstream programming has been skipped as the same bitstream is already loaded on the ta
### Resetting network state.
```

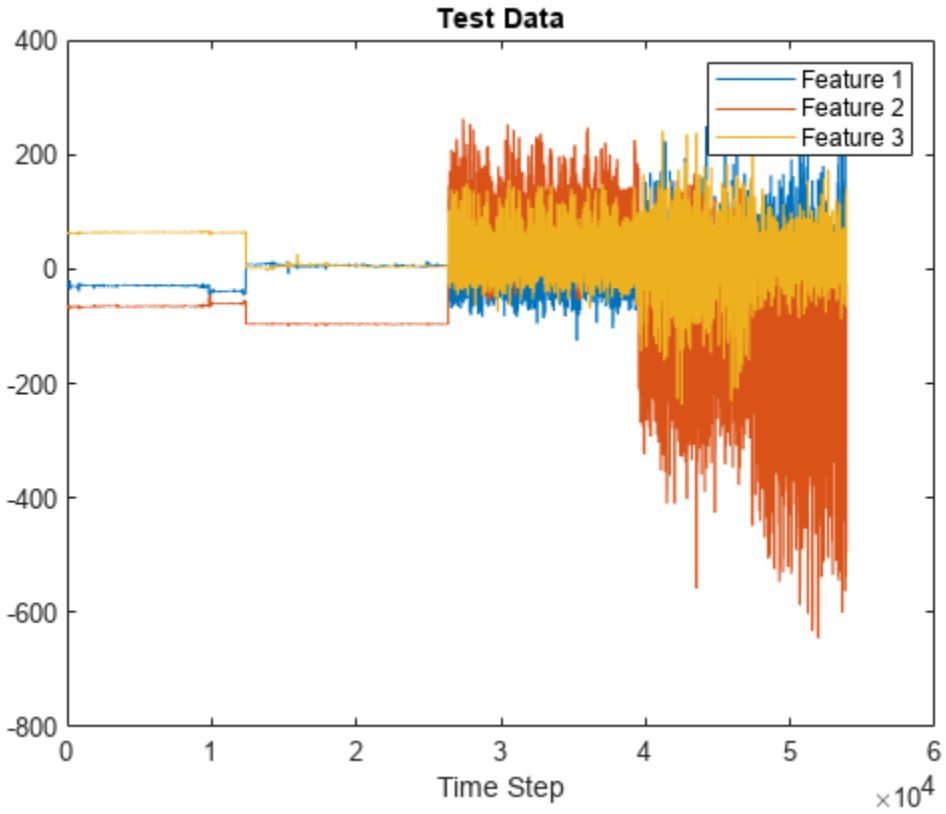
```
### Loading weights to FC Processor.
### FC Weights loaded. Current time is 30-Jun-2022 13:41:44
```

Load Human Activity Test Data

Load the test data and classify the activity at each time step. Each sequence has three features and varies in length. The three features correspond to the accelerometer readings in three different directions.

Load the human activity test data. `XTest` contains a single sequence of dimension 3. `YTest` contains a sequence of categorical labels that correspond to the activity at each time step.

```
load HumanActivityTest
numFeatures = 3;
figure
plot(XTest{1}')
xlabel("Time Step")
legend("Feature " + (1:numFeatures))
title("Test Data")
```



Run the Prediction

Classify the test data by using the `classify` function.

```
YPred = classify(hW.Network, XTest{1});
```

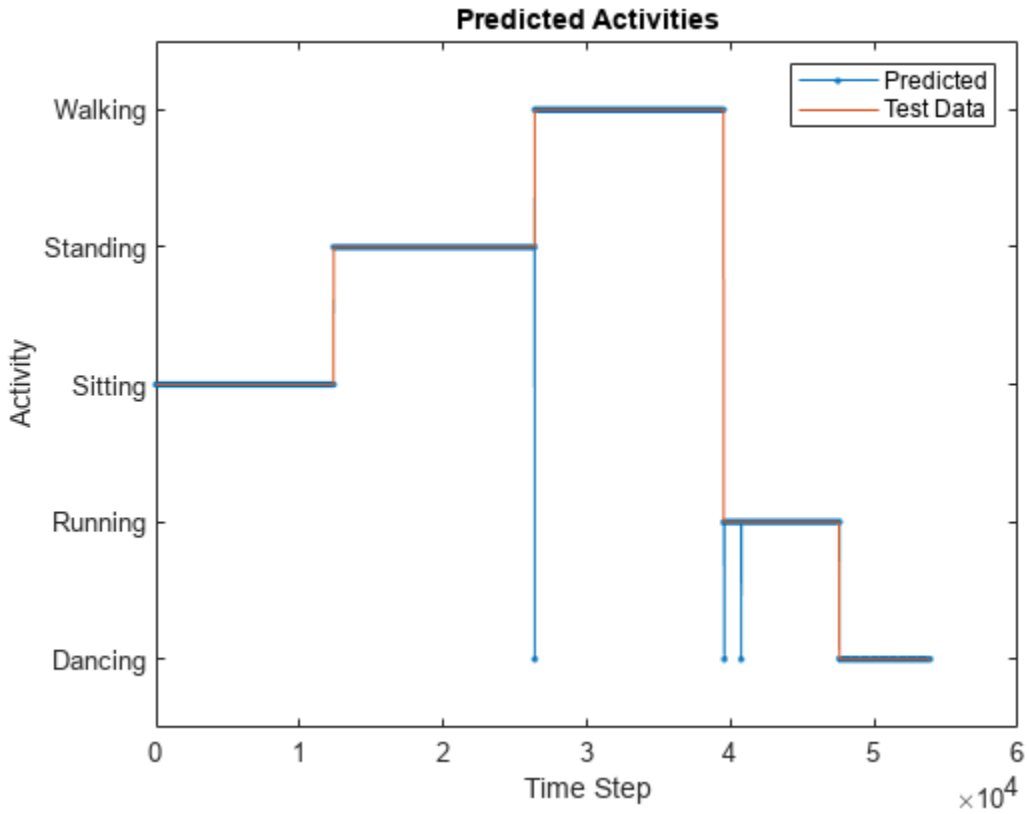
Calculate the accuracy of the prediction.

```
acc = sum(YPred == YTest{1})./numel(YTest{1})
```

```
acc = 0.9995
```

Compare the predictions with the test data by using a plot.

```
figure
plot(YPred,'.-')
hold on
plot(YTest{1})
hold off

xlabel("Time Step")
ylabel("Activity")
title("Predicted Activities")
legend(["Predicted" "Test Data"])
```



Compare this graph to the output of the `predict` method.

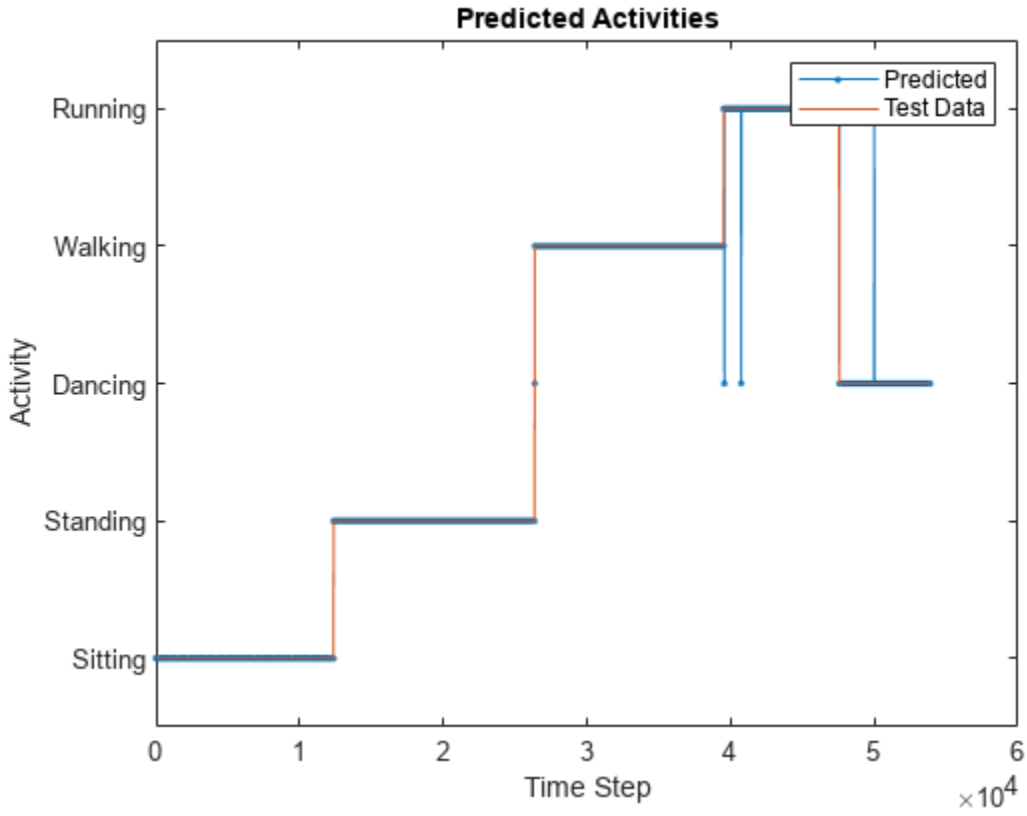
Run the `predict` method of the `dlhdl.Workflow` object, to retrieve the hardware prediction results.

```
predictions = hW.predict(XTest{1}(:,1:10000));
predictions = horzcat(predictions, hW.predict(XTest{1}(:,10001:20000)));
predictions = horzcat(predictions, hW.predict(XTest{1}(:,20001:30000)));
predictions = horzcat(predictions, hW.predict(XTest{1}(:,30001:40000)));
predictions = horzcat(predictions, hW.predict(XTest{1}(:,40001:50000)));
predictions = horzcat(predictions, hW.predict(XTest{1}(:,50001:end)));
```

```
save("hardwarepredictions.mat","predictions")
indices = [];
actions = [];
for x = 1:length(YPred)
    [r,i] = max(predictions(:,x));
    indices = [indices i];
    switch i
        case 1
            actions = [actions categorical("Dancing")];
        case 2
            actions = [actions categorical("Running")];
        case 5
            actions = [actions categorical("Walking")];
        case 4
            actions = [actions categorical("Standing")];
        case 3
            actions = [actions categorical("Sitting")];
    end
end
```

Plot the comparison between the FPGA board predictions and test data.

```
figure
plot(actions,'.-')
hold on
plot(YTest{1})
hold off

xlabel("Time Step")
ylabel("Activity")
title("Predicted Activities")
legend(["Predicted" "Test Data"])
```



The hardware-predicted activities are similar to the activities classified by the `classify` function.

Run Sequence Forecasting Using a GRU Layer on an FPGA

Reduce the time to train a sequence forecasting network by swapping out the LSTM later for a gated recurrent unit (GRU) layer. Use the deployed network to predict future values by using open-loop and closed-loop forecasting. Use MATLAB® to retrieve the prediction results from the target device.

Modified Waveform Data Network

The network attached to this example was trained using the “Time Series Forecasting Using Deep Learning”. In this example the LSTM layer was swapped out for a GRU layer. This example uses the `WaveformData.mat` data set, which contains 2000 synthetically generated waveforms of varying lengths with three channels. This example uses a trained network with a GRU layer to forecast future values of the waveforms given the values from the previous time steps using both closed loop and open loop forecasting.

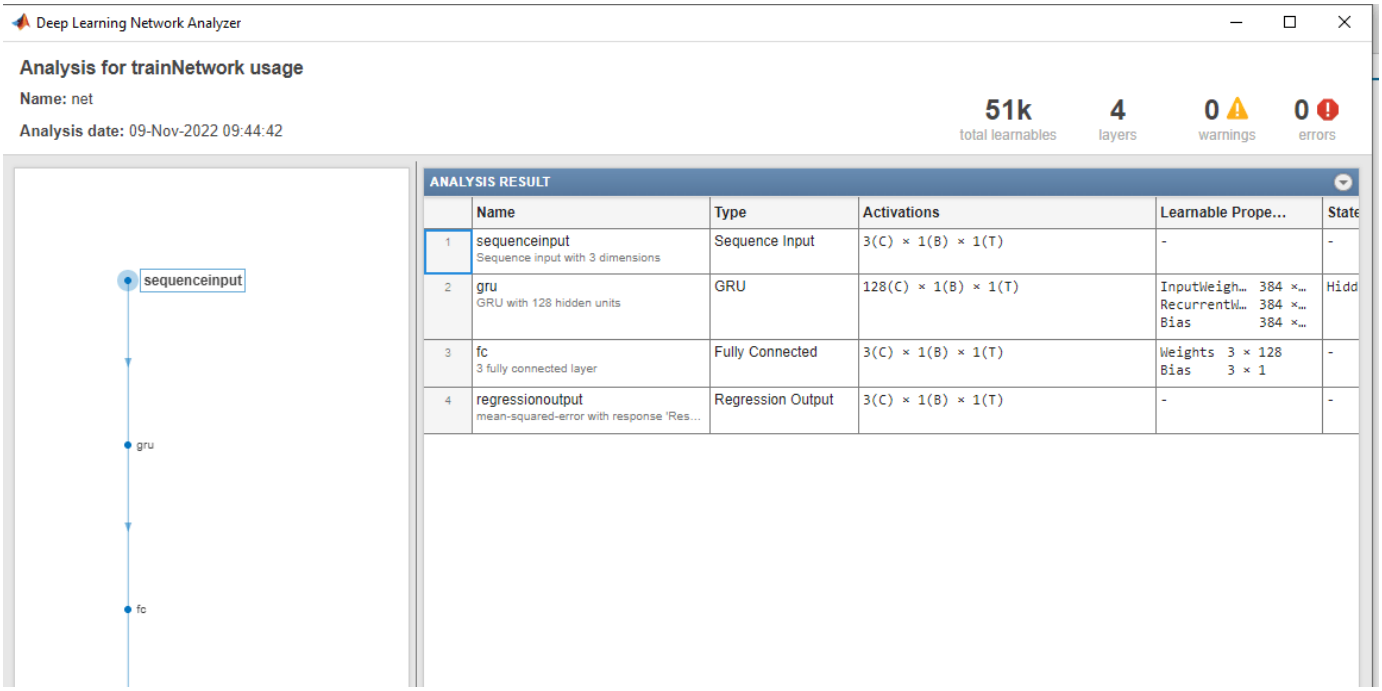
Load the Pretrained Network

To load the GRU layer network enter:

```
load grunet
```

Use the `analyzeNetwork` function to obtain information about the network layers. the function returns a graphical representation of the network that contains detailed parameter information for every layer in the network.

```
analyzeNetwork(net)
```



Define FPGA Board Interface

Define the target FPGA board programming interface by using the `dlhdl.Target` object. Specify that the interface is for a Xilinx board with an Ethernet interface.

To create the target object, enter:

```
hTarget_gru = dlhdl.Target('Xilinx',Interface='Ethernet');
```

To use the JTAG interface, install Xilinx™ Vivado™ Design Suite 2020.2. To set the Xilinx Vivado toolpath, enter:

```
hdlsetuptoolpath('ToolName', 'Xilinx Vivado', 'ToolPath', 'C:\Xilinx\Vivado\2020.2\bin\vivado.ba
hTarget = dlhdl.Target('Xilinx',Interface='JTAG');
```

Prepare Network for Deployment

Prepare the network for deployment by creating a `dlhdl.Workflow` object. Specify the network and the bitstream name. Ensure that the bitstream name matches the data type and the FPGA board. In this example the target FPGA board is the Xilinx ZCU102 SOC board. The bitstream uses a single data type.

```
hW_gru = dlhdl.Workflow(Network=net,Bitstream='zcu102_lstm_single',Target=hTarget_gru);
```

Tu run the example on the Xilinx ZC706 board, enter:

```
hW = dlhdl.Workflow(Network=net,Bitstream='zc706_lstm_single',Target=hTarget);
```

Compile the GRU Layer Network

Run the `compile` method of the `dlhdl.Workflow` object to compile the network and generate the instructions, weights, and biases for deployment. The total number of frames exceeds the default value of 30. Set the `InputFrameNumberLimit` name-value argument to `1000` to run predictions in chunks of 1000 frames to prevent timeouts.

```
dn = compile(hW_gru,'InputFrameNumberLimit',1000)
```

```
### Compiling network for Deep Learning FPGA prototyping ...
### Targeting FPGA bitstream zcu102_lstm_single.
### The network includes the following layers:
     1   'sequenceinput'      Sequence Input      Sequence input with 3 dimensions            (S
     2   'gru'                GRU                 GRU with 128 hidden units                   (H
     3   'fc'                 Fully Connected     3 fully connected layer                     (H
     4   'regressionoutput'   Regression Output   mean-squared-error with response 'Response'  (S

### Notice: The layer 'sequenceinput' with type 'nnet.cnn.layer.ImageInputLayer' is implemented i
### Notice: The layer 'regressionoutput' with type 'nnet.cnn.layer.RegressionOutputLayer' is impl
### Compiling layer group: gru.wh ...
### Compiling layer group: gru.wh ... complete.
### Compiling layer group: gru.rh ...
### Compiling layer group: gru.rh ... complete.
### Compiling layer group: gru.w1 ...
### Compiling layer group: gru.w1 ... complete.
### Compiling layer group: gru.w2 ...
### Compiling layer group: gru.w2 ... complete.
### Compiling layer group: fc ...
### Compiling layer group: fc ... complete.

### Allocating external memory buffers:

          offset_name          offset_address    allocated_space
    ___________________________    ______________    _________________

    "InputDataOffset"             "0x00000000"     "4.0 MB"
    "OutputResultOffset"          "0x00400000"     "4.0 MB"
    "SchedulerDataOffset"         "0x00800000"     "4.0 MB"
    "SystemBufferOffset"          "0x00c00000"     "20.0 MB"
    "InstructionDataOffset"       "0x02000000"     "4.0 MB"
    "FCWeightDataOffset"          "0x02400000"     "4.0 MB"
    "EndOffset"                   "0x02800000"     "Total: 40.0 MB"

### Network compilation complete.

dn = struct with fields:
             weights: [1×1 struct]
        instructions: [1×1 struct]
           registers: [1×1 struct]
    syncInstructions: [1×1 struct]
        constantData: {{1×2 cell}  [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
             ddrInfo: [1×1 struct]
```

Program Bitstream onto FPGA and Download Network Weights

To deploy the network on the Xilinx ZCU102 SoC hardware, run the `deploy` function of the `dlhdl.Workflow` object. This function uses the output of the `compile` function to program the

FPGA board by using the programming file. It also downloads the network weights and biases. The `deploy` function starts programming the FPGA device and displays progress messages, and the required time to deploy the network.

```
deploy(hW_gru)
```

```
### FPGA bitstream programming has been skipped as the same bitstream is already loaded on the ta
### Deep learning network programming has been skipped as the same network is already loaded on t
```

Test Network

Prepare the test data for prediction. Normalize the test data using the statistics calculated from the training data. Forecast the values using the GRU layer network. To forecast the values of future time steps of a sequence, specify the targets as the test sequences with values shifted by one time step. In other words, at each time step of the input sequence, the GRU layer network learns to predict the value of the next time step.

```
load Waveformdata
numChannels = size(data{1},1);
numObservations = numel(data);

idxTrain = 1:floor(0.9*numObservations);
idxTest = floor(0.9*numObservations)+1:numObservations;
dataTrain = data(idxTrain);
dataTest = data(idxTest);

for n = 1:numel(dataTrain)
    X = dataTrain{n};
    XTrain{n} = X(:,1:end-1);
    TTrain{n} = X(:,2:end);
end

muX = mean(cat(2,XTrain{:}),2);
sigmaX = std(cat(2,XTrain{:}),0,2);
muT = mean(cat(2,TTrain{:}),2);
sigmaT = std(cat(2,TTrain{:}),0,2);

for n = 1:size(dataTest,1)
    X = dataTest{n};
    XTest{n} = (X(:,1:end-1) - muX) ./ sigmaX;
    TTest{n} = (X(:,2:end) - muT) ./ sigmaT;
end
```

Make predictions using the test data.

```
YTest_gru = predict(hW_gru,XTest{1},Profile = 'on');
```

```
### Resetting network state.
### Finished writing input activations.
### Running a sequence of length 115.


              Deep Learning Processor Profiler Performance Results

                    LastFrameLatency(cycles)   LastFrameLatency(seconds)       FramesNum      Tota
                           -------------              -------------              ---------        --
Network                        32322                   0.00015                   115            37
    gru.wh                       548                   0.00000
```

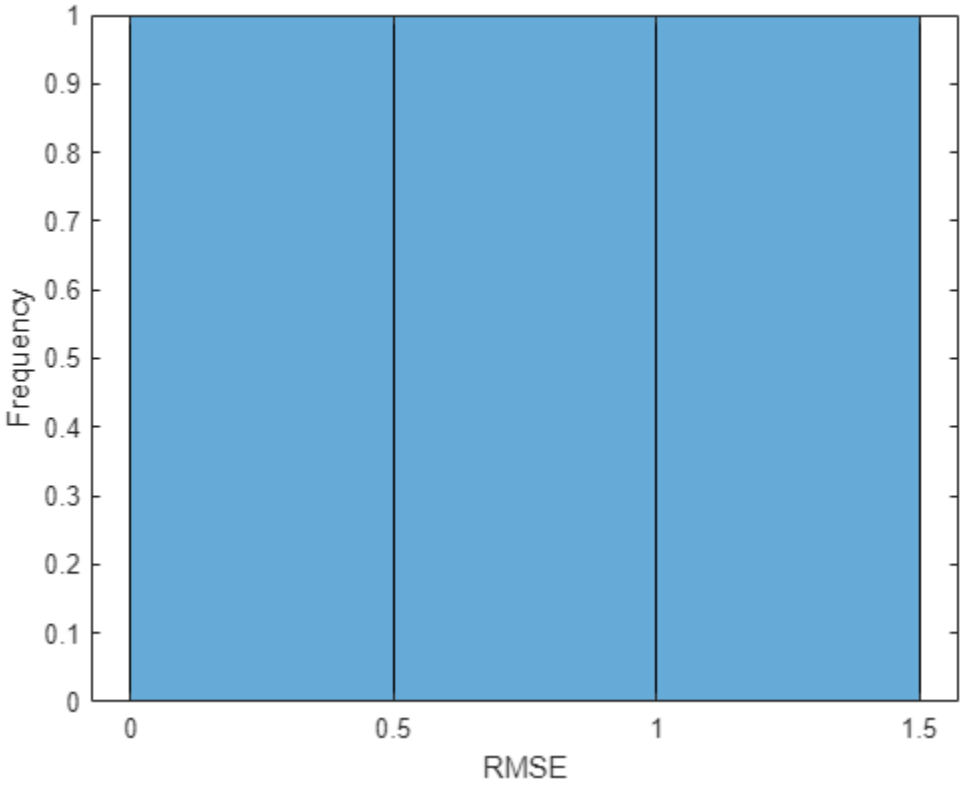
```
    gru.rh                    7538                  0.00003
    memSeparator_0              98                  0.00000
    gru.w1                    7469                  0.00003
    gru.w2                    7649                  0.00003
    gru.sigmoid_1              222                  0.00000
    gru.sigmoid_2              214                  0.00000
    gru.multiplication_2       288                  0.00000
    gru.multiplication_4       334                  0.00000
    gru.multiplication_1       344                  0.00000
    gru.addition_2             294                  0.00000
    gru.addition_1             294                  0.00000
    gru.tanh_1                 198                  0.00000
    gru.multiplication_3       288                  0.00000
    gru.addition_3             298                  0.00000
    fc                        6246                  0.00003
 * The clock frequency of the DL processor is: 220MHz
```

To evaluate the accuracy, calculate the root mean squared error (RMSE) between the predictions and the target for each test sequence.

```
for i = 1:size(YTest_gru,1)
    rmse(i) = sqrt(mean((YTest_gru(i) - TTest{1}(i)).^2,"all"));
end
```

Visualize the errors in a histogram. Lower values indicate greater accuracy.

```
figure
histogram(rmse)
xlabel("RMSE")
ylabel("Frequency")
```



Calculate the mean RMSE over all test observations.

```
mean(rmse)
```

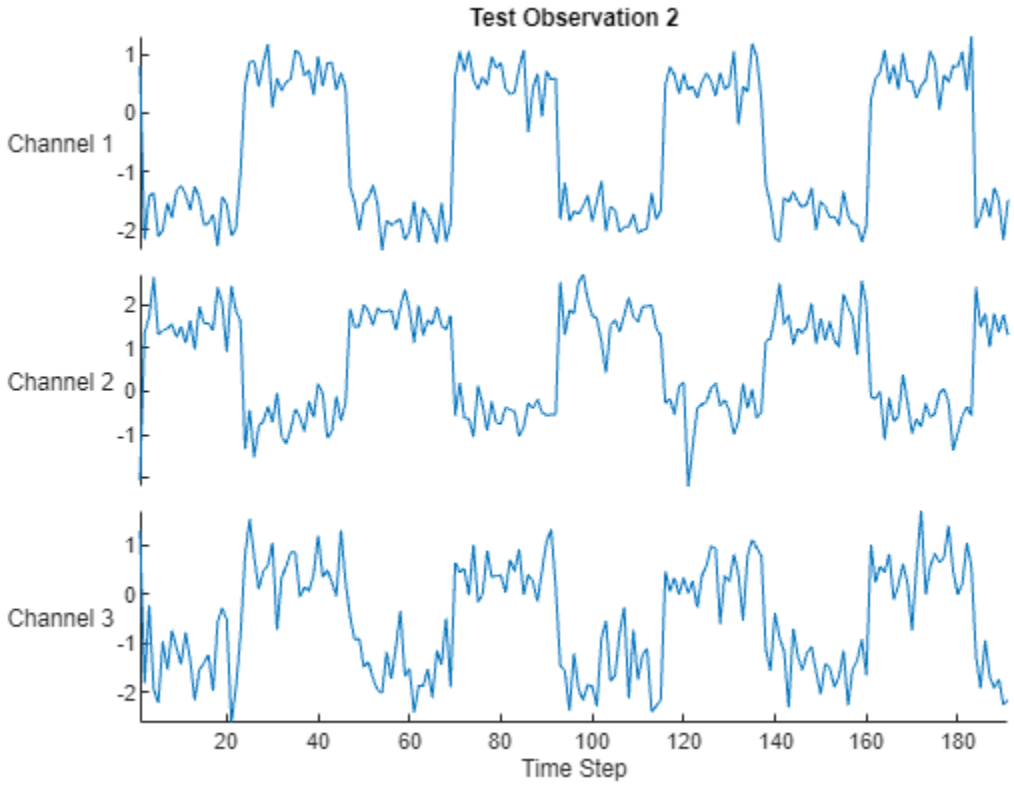
```
ans = single
    0.7688
```

Forecast Future Time Steps

To forecast the values of multiple future time steps, when given an input time series or sequence, use the `predictAndUpdateState` function. This function predicts time steps one at a time and updates the network state at each prediction. For each prediction, use the previous prediction as the input to the function.

Visualize one of the test sequences in a plot.

```
idx = 2;
X_gru = XTest{idx};
T_gru = TTest{idx};

figure
stackedplot(X_gru',DisplayLabels="Channel " + (1:numChannels))
xlabel("Time Step")
title("Test Observation " + idx)
```



Open-Loop Forecasting

Open-loop forecasting predicts the next time step in a sequence using only the input data. When making predictions for subsequent time steps, you collect the true values form your data source and use those as input. For example, suppose that you want to predict the value for time step t of a sequence by using data collected in time steps 1 through $t - 1$. To make predictions for time step $t + 1$, wait until you record the true value for time step t and use that value as input to make the next prediction. Use open-loop forecasting when you have true values to provide to the network before making the next prediction.

Initialize the network state by resetting the state using the `resetState` function, then make an initial prediction using the first few time steps of the input data. Update the network state by using the first 75 time steps of the input data.

```
resetState(hW_gru)
offset = 75;
[~,~] = predictAndUpdateState(hW_gru,X_gru(:,1:offset));
```

```
### Resetting network state.
### Finished writing input activations.
### Running a sequence of length 75.
```

To forecast further predictions, loop over time steps and update the network state by using the `predictAndUpdateState` function. Forecast values for the remaining time steps of the test observation by looping over the time steps of the input data and using them as input to the network. The first prediction is the value that corresponds to the time step `offset + 1`.

```
numTimeSteps = size(X_gru,2);
numPredictionTimeSteps = numTimeSteps - offset;
Y_gru = zeros(numChannels,numPredictionTimeSteps);

for t = 1:numPredictionTimeSteps
    Xt_gru = X_gru(:,offset+t);
    Y_gru(:,t) = predictAndUpdateState(hW_gru,Xt_gru);
end
```

```
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
```

```
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
```

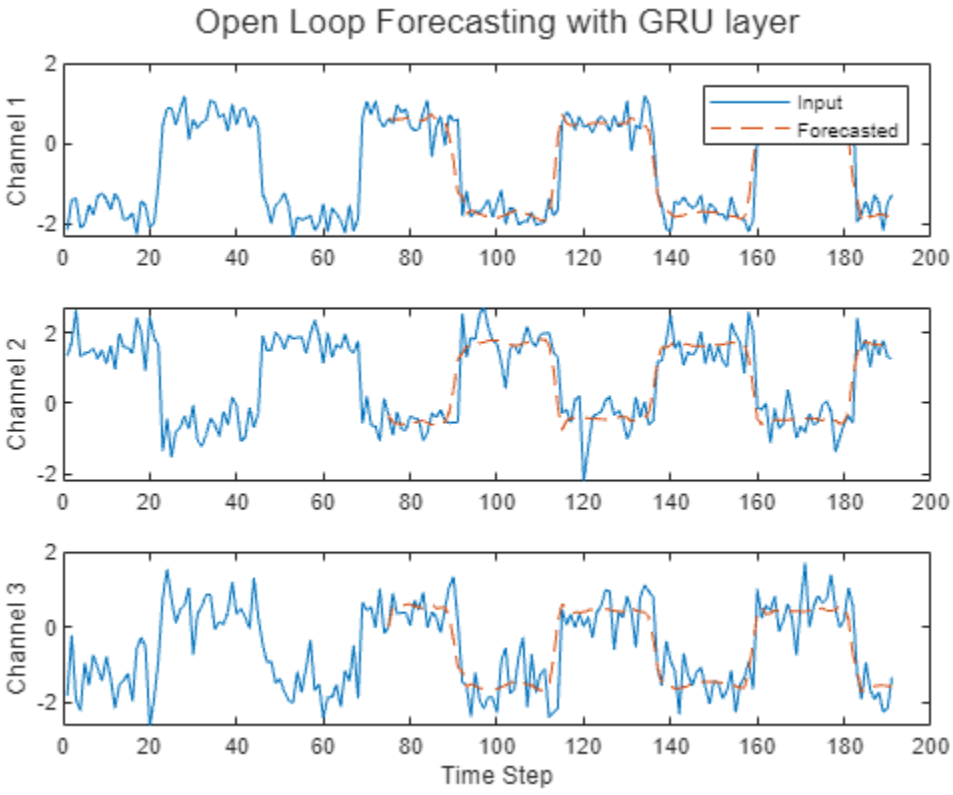
```
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
```

```
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
```

```
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
```

Compare the predictions with the target values.

```
figure
t = tiledlayout(numChannels,1);
title(t,"Open Loop Forecasting with GRU layer")

for i = 1:numChannels
    nexttile
    plot(T_gru(i,:))
    hold on
    plot(offset:numTimeSteps,[T_gru(i,offset) Y_gru(i,:)],'--')
    ylabel("Channel " + i)
end

xlabel("Time Step")
nexttile(1)
legend(["Input" "Forecasted"])
```



Closed-Loop Forecasting

Closed-loop forecasting predicts subsequent time steps in a sequence by using the previous predictions as input. In this case, the model does not require the true values to make the prediction. For example, suppose that you want to predict the value for time steps t through $t + k$ of the sequence by using data collected in time steps 1 through $t - 1$. To make predictions for time step i, use the predicted value for time step $i - 1$ as input. Use closed-loop forecasting to forecast multiple subsequent time steps or when you do not have true values to provide to the network before making the next prediction.

Initialize the network state by resetting the state using the `resetState` function, then make an initial prediction, `Z`, using the first few time steps of the input data. Update the network state by using the first 75 time steps of the input data.

```
resetState(hW_gru)
offset = size(X_gru,2);
[Z, ~] = predictAndUpdateState(hW_gru,X_gru);
```

```
### Resetting network state.
### Finished writing input activations.
### Running a sequence of length 191.
```

To forecast further predictions, loop over time steps and update the network state by using the `predictAndUpdateState` function. Forecast the next 200 time steps by iteratively passing the previously predicted value to the network. Because the network does not require the input data to make any further predictions, you can specify any number of time steps to forecast.

```
numPredictionTimeSteps = 200;
Xt_gru = Z(:,end);
Y_gru = zeros(numChannels,numPredictionTimeSteps);

for t = 1:numPredictionTimeSteps
    [Y_gru(:,t),~] =  predictAndUpdateState(hW_gru,Xt_gru);
    Xt_gru = Y_gru(:,t);
end
```

```
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
```

```
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
```

```
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
```

```
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
```

```
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
```

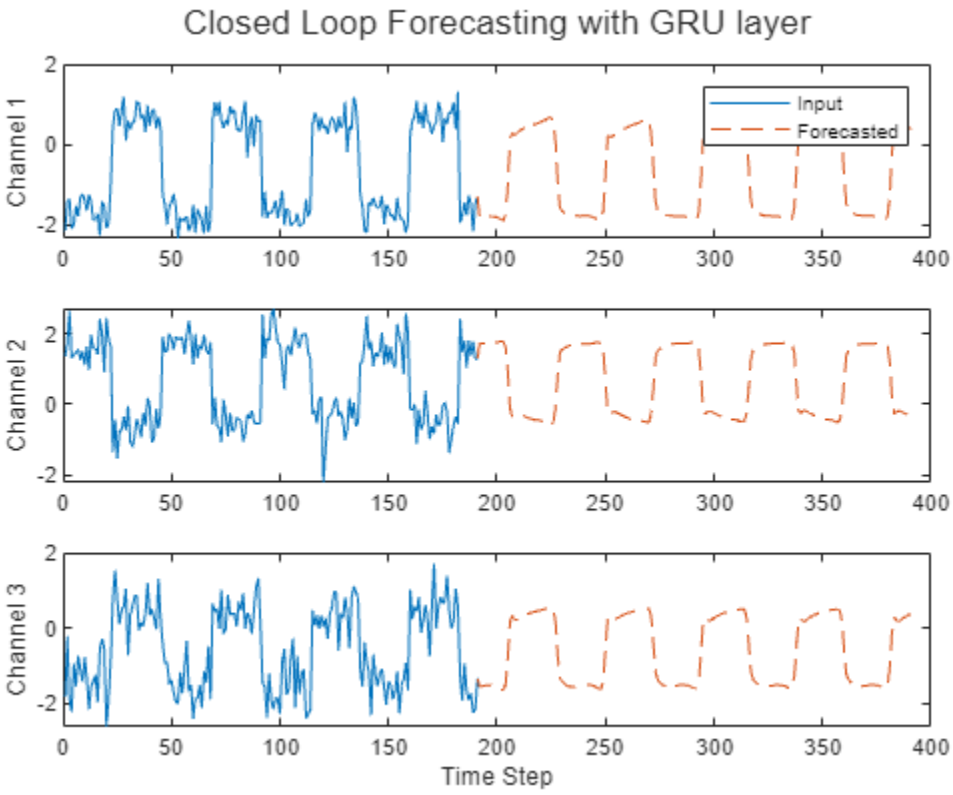
```
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
```

```
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
```

```
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
```

Visualize the forecasted values in a plot.

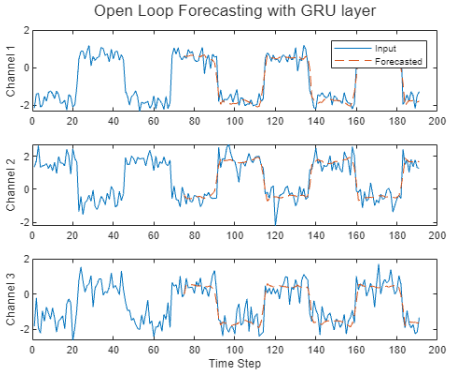
```
numTimeSteps = offset + numPredictionTimeSteps;

figure
t = tiledlayout(numChannels,1);
title(t,"Closed Loop Forecasting with GRU layer")

for i = 1:numChannels
    nexttile
    plot(T_gru(i,1:offset))
    hold on
    plot(offset:numTimeSteps,[T_gru(i,offset) Y_gru(i,:)],'--')
    ylabel("Channel " + i)
end

xlabel("Time Step")
nexttile(1)
legend(["Input" "Forecasted"])
```

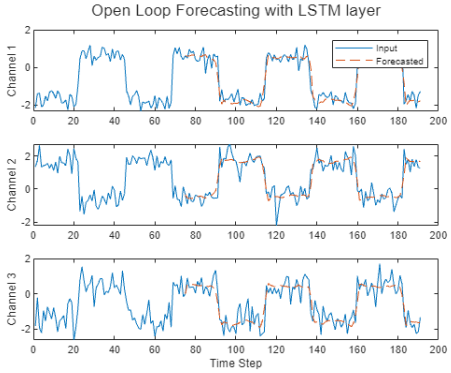


Closed-loop forecasting allows you to forecast an arbitrary number of time steps, but can be less accurate when compared to open-loop forecasting because the network does not have access to the true values during the forecasting process.

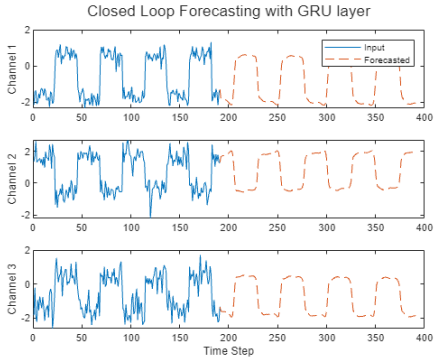
Compare Network Predictions

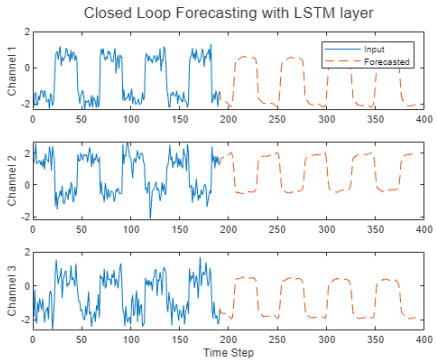
Compare the predictions of the LSTM layer network to the GRU layer network. This image shows the compariuson between the GRU layer network and LSTM layer network for open loop forecasting. The GRU layer network has a performance of 6734.9 frames per second and the LSTM layer network has a performance of 5632.3 frames per second. To learn how to deploy the LSTM layer network to an FPGA, see “Run Sequence Forecasting on FPGA by Using Deep Learning HDL Toolbox”.





This image shows the comparison between the GRU layer network and LSTM layer network for closed loop forecasting.





Version History

Introduced in R2020b

See Also

`deploy` | `getBuildInfo` | `predict` | `predictAndUpdateState` | `resetState`

Topics

“Use the Compiler Output for System Integration”
“Image Input Layer Normalization Hardware Implementation”

deploy

Class: `dlhdl.Workflow`
Package: `dlhdl`

Deploy the specified neural network to the target FPGA board

Syntax

```
deploy(workflowObject)
```

Description

`deploy(workflowObject)` programs the specified target board with the bitstream and deploys the deep learning network on it.

Input Arguments

`workflowObject` — Workflow
`dlhdl.Workflow` object

Workflow, specified as a `dlhdl.Workflow` object.

Examples

Get Started with Deep Learning FPGA Deployment on Intel Arria 10 SoC

This example shows how to create, compile, and deploy a `dlhdl.Workflow` object that has a handwritten character detection series network object by using the Deep Learning HDL Toolbox™ Support Package for Intel FPGA and SoC. Use MATLAB® to retrieve the prediction results from the target device.

Prerequisites

- Intel Arria™ 10 SoC development kit
- Deep Learning HDL Toolbox™ Support Package for Intel FPGA and SoC
- Deep Learning HDL Toolbox™
- Deep Learning Toolbox™

Load the Pretrained SeriesNetwork

To load the pretrained series network, that has been trained on the Modified National Institute Standards of Technology (MNIST) database[1], enter:

```
snet = getDigitsNetwork;
```

To view the layers of the pretrained series network, enter:

```
analyzeNetwork(snet)
```

Create Target Object

Create a target object that has a custom name for your target device and an interface to connect your target device to the host computer. Interface options are JTAG and Ethernet. To use JTAG, install Intel™ Quartus™ Prime Standard Edition 20.1. Set up the path to your installed Intel Quartus Prime executable if it is not already set up. For example, to set the toolpath, enter:

```
% hdlsetuptoolpath('ToolName', 'Altera Quartus II','ToolPath', 'C:\altera\20.1\quartus\bin64');
```

```
hTarget = dlhdl.Target('Intel')
```

```
hTarget =
  Target with properties:

       Vendor: 'Intel'
    Interface: JTAG
```

Create Workflow Object

Create an object of the `dlhdl.Workflow` class. When you create the object, specify the network and the bitstream name. Specify the saved pretrained MNIST neural network, snet, as the network. Make sure that the bitstream name matches the data type and the FPGA board that you are targeting. In this example, the target FPGA board is the Intel Arria 10 SOC board and the bitstream uses a single data type.

```
hW = dlhdl.Workflow('network', snet, 'Bitstream', 'arria10soc_single','Target',hTarget);
```

Compile the MNIST Series Network

To compile the MNIST series network, run the compile function of the `dlhdl.Workflow` object.

```
dn = hW.compile;
```

```
### Optimizing series network: Fused 'nnet.cnn.layer.BatchNormalizationLayer' into 'nnet.cnn.laye
          offset_name          offset_address    allocated_space
    _________________________    ______________    _______________

    "InputDataOffset"            "0x00000000"     "4.0 MB"
    "OutputResultOffset"         "0x00400000"     "4.0 MB"
    "SystemBufferOffset"         "0x00800000"     "28.0 MB"
    "InstructionDataOffset"      "0x02400000"     "4.0 MB"
    "ConvWeightDataOffset"       "0x02800000"     "4.0 MB"
    "FCWeightDataOffset"         "0x02c00000"     "4.0 MB"
    "EndOffset"                  "0x03000000"     "Total: 48.0 MB"
```

Program Bitstream onto FPGA and Download Network Weights

To deploy the network on the Intel Arria 10 SoC hardware, run the deploy function of the `dlhdl.Workflow` object. This function uses the output of the compile function to program the FPGA board by using the programming file. It also downloads the network weights and biases. The deploy function starts programming the FPGA device, displays progress messages, and the time it takes to deploy the network.

```
hW.deploy
```

```
### FPGA bitstream programming has been skipped as the same bitstream is already loaded on the ta
### Loading weights to FC Processor.
### FC Weights loaded. Current time is 28-Jun-2020 13:45:47
```

Run Prediction for Example Image

To load the example image, execute the predict function of the `dlhdl.Workflow` object, and then display the FPGA result, enter:

```
inputImg = imread('five_28x28.pgm');
```

Run prediction with the profile 'on' to see the latency and throughput results.

```
[prediction, speed] = hW.predict(single(inputImg),'Profile','on');
```

```
### Finished writing input activations.
### Running single input activations.


              Deep Learning Processor Profiler Performance Results

                    LastLayerLatency(cycles)   LastLayerLatency(seconds)       FramesNum      Tota
                          ------------             ------------               ---------        --
Network                       49243                 0.00033                     1
    conv_module               25983                 0.00017
        conv_1                 6813                 0.00005
        maxpool_1              4705                 0.00003
        conv_2                 5205                 0.00003
        maxpool_2              3839                 0.00003
        conv_3                 5481                 0.00004
    fc_module                 23260                 0.00016
        fc                    23260                 0.00016
 * The clock frequency of the DL processor is: 150MHz
```

```
[val, idx] = max(prediction);
fprintf('The prediction result is %d\n', idx-1);
```

```
The prediction result is 5
```

Bibliography

1 LeCun, Y., C. Cortes, and C. J. C. Burges. "The MNIST Database of Handwritten Digits." http://yann.lecun.com/exdb/mnist/.

Classify Images on FPGA Using Quantized Neural Network

This example shows how to use Deep Learning HDL Toolbox™ to deploy a quantized deep convolutional neural network (CNN) to an FPGA. In the example you use the pretrained ResNet-18 CNN to perform transfer learning and quantization. You then deploy the quantized network and use MATLAB ® to retrieve the prediction results.

ResNet-18 has been trained on over a million images and can classify images into 1000 object categories, such as keyboard, coffee mug, pencil, and many animals. The network has learned rich feature representations for a wide range of images. The network takes an image as input and outputs a label for the object in the image together with the probabilities for each of the object categories.

For this example, you need:

- Deep Learning Toolbox™

- Deep Learning HDL Toolbox™
- Deep Learning Toolbox Model for ResNet-18 Network
- Deep Learning HDL Toolbox™ Support Package for Xilinx® FPGA and SoC Devices
- Image Processing Toolbox™
- Deep Learning Toolbox Model Quantization Library
- MATLAB® Coder™ Interface for Deep Learning

To perform classification on a new set of images, you fine-tune a pretrained ResNet-18 CNN by transfer learning. In transfer learning, you can take a pretrained network and use it as a starting point to learn a new task. Fine-tuning a network with transfer learning is usually much faster and easier than training a network with randomly initialized weights. You can quickly transfer learned features to a new task using a smaller number of training images.

Load Pretrained Network

Load the pretrained ResNet-18 network.

```
net = resnet18;
```

View the layers of the pretrained network.

```
deepNetworkDesigner(net);
```

The first layer, the image input layer, requires input images of size 227-by-227-by-3, where three is the number of color channels.

```
inputSize = net.Layers(1).InputSize;
```

Load Data

This example uses the MathWorks MerchData data set. This is a small data set containing 75 images of MathWorks merchandise, belonging to five different classes (*cap*, *cube*, *playing cards*, *screwdriver*, and *torch*).

```
curDir = pwd;
unzip('MerchData.zip');
imds = imageDatastore('MerchData', ...
'IncludeSubfolders',true, ...
'LabelSource','foldernames');
```

Specify Training and Validation Sets

Divide the data into training and validation data sets, so that 30% percent of the images go to the training data set and 70% of the images to the validation data set. `splitEachLabel` splits the datastore `imds` into two new datastores, `imdsTrain` and `imdsValidation`.

```
[imdsTrain,imdsValidation] = splitEachLabel(imds,0.7,'randomized');
```

Replace Final layers

To retrain ResNet-18 to classify new images, replace the last fully connected layer and final classification layer of the network. In ResNet-18 , these layers have the names `'fc1000'` and `'ClassificationLayer_predictions'`, respectively. The fully connected layer and classification layer of the pretrained network `net` are configured for 1000 classes. These two layers `fc1000` and `ClassificationLayer_predictions` in ResNet-18, contain information on how to combine the

features that the network extracts into class probabilities and predicted labels. These two layers must be fine-tuned for the new classification problem. Extract all the layers, except the last two layers, from the pretrained network.

```
lgraph = layerGraph(net)
```

```
lgraph =
  LayerGraph with properties:

     InputNames: {'data'}
    OutputNames: {'ClassificationLayer_predictions'}
         Layers: [71×1 nnet.cnn.layer.Layer]
    Connections: [78×2 table]
```

```
numClasses = numel(categories(imdsTrain.Labels))
```

```
numClasses = 5
```

```
newLearnableLayer = fullyConnectedLayer(numClasses, ...
'Name','new_fc', ...
'WeightLearnRateFactor',10, ...
'BiasLearnRateFactor',10);
lgraph = replaceLayer(lgraph,'fc1000',newLearnableLayer);
newClassLayer = classificationLayer('Name','new_classoutput');
lgraph = replaceLayer(lgraph,'ClassificationLayer_predictions',newClassLayer);
```

Prepare Data for Training

The network requires input images of size 224-by-224-by-3, but the images in the image datastores have different sizes. Use an augmented image datastore to automatically resize the training images. Specify additional augmentation operations to perform on the training images, such as randomly flipping the training images along the vertical axis and randomly translating them up to 30 pixels horizontally and vertically. Data augmentation helps prevent the network from overfitting and memorizing the exact details of the training images.

```
pixelRange = [-30 30];
imageAugmenter = imageDataAugmenter( ...
'RandXReflection',true, ...
'RandXTranslation',pixelRange, ...
'RandYTranslation',pixelRange);
```

To automatically resize the validation images without performing further data augmentation, use an augmented image datastore without specifying any additional preprocessing operations.

```
augimdsTrain = augmentedImageDatastore(inputSize(1:2),imdsTrain, ...
'DataAugmentation',imageAugmenter);
augimdsValidation = augmentedImageDatastore(inputSize(1:2),imdsValidation);
```

Specify Training Options

Specify the training options. For transfer learning, keep the features from the early layers of the pretrained network (the transferred layer weights). To slow down learning in the transferred layers, set the initial learning rate to a small value. Specify the mini-batch size and validation data. The software validates the network every `ValidationFrequency` iterations during training.

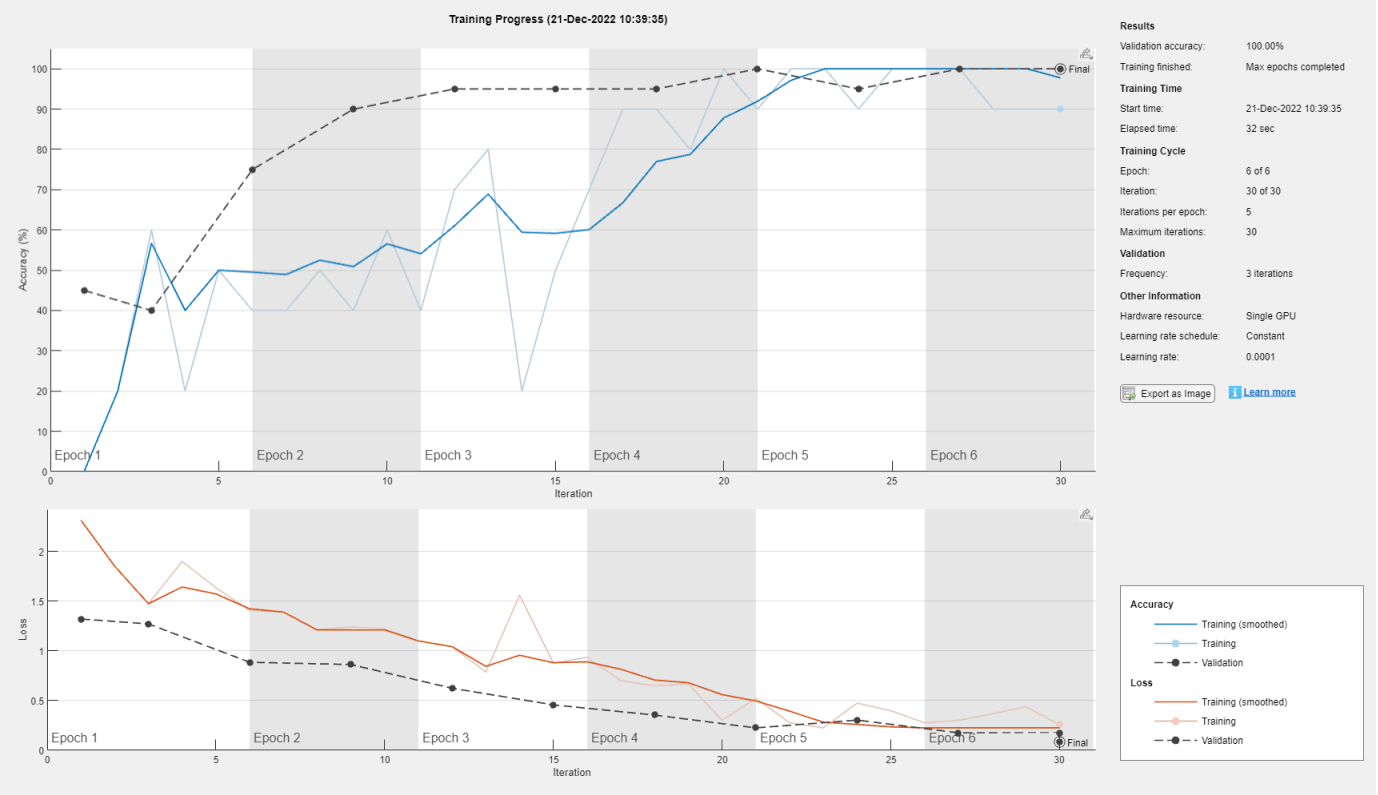
```
options = trainingOptions('sgdm', ...
'MiniBatchSize',10, ...
```

```
'MaxEpochs',6, ...
'InitialLearnRate',1e-4, ...
'Shuffle','every-epoch', ...
'ValidationData',augimdsValidation, ...
'ValidationFrequency',3, ...
'Verbose',false, ...
'Plots','training-progress');
```

Train Network

Train the network that consists of the transferred and new layers. By default, `trainNetwork` uses a GPU if one is available. Using this function on a GPU requires Parallel Computing Toolbox™ and a supported GPU device. For more information, see “GPU Computing Requirements” (Parallel Computing Toolbox). If a GPU is not available, the network uses a CPU (requires MATLAB Coder Interface for Deep learning). You can also specify the execution environment by using the `ExecutionEnvironment` name-value argument of `trainingOptions`.

```
netTransfer = trainNetwork(augimdsTrain,lgraph,options);
```



Quantize Network

Quantize the network using the `dlquantizer` object. Set the target execution environment to FPGA.

```
dlquantObj = dlquantizer(netTransfer,'ExecutionEnvironment','FPGA');
```

Calibrate Quantized Network

Use the `calibrate` function to exercise the network with sample inputs and collect the range information. The `calibrate` function collects the dynamic ranges of the weights and biases in the

convolution and fully connected layers of the network and the dynamic ranges of the activations in all layers of the network. The function returns the information as a table, in which each row contains range information for a learnable parameter of the quantized network.

```
calibrate(dlquantObj,augimdsTrain)
```

```
ans=95×5 table
       Optimized Layer Name       Network Layer Name    Learnables / Activations    MinValue    M
    ______________________________    _________________    ________________________    ________    -

    {'conv1_Weights'         }    {'conv1'         }          "Weights"              -0.79143
    {'conv1_Bias'            }    {'conv1'         }          "Bias"                 -0.66949    0
    {'res2a_branch2a_Weights'}    {'res2a_branch2a'}          "Weights"              -0.42074    0
    {'res2a_branch2a_Bias'   }    {'res2a_branch2a'}          "Bias"                  -0.8039
    {'res2a_branch2b_Weights'}    {'res2a_branch2b'}          "Weights"              -0.78524    0
    {'res2a_branch2b_Bias'   }    {'res2a_branch2b'}          "Bias"                  -1.3835
    {'res2b_branch2a_Weights'}    {'res2b_branch2a'}          "Weights"               -0.3174    0
    {'res2b_branch2a_Bias'   }    {'res2b_branch2a'}          "Bias"                  -1.1203
    {'res2b_branch2b_Weights'}    {'res2b_branch2b'}          "Weights"               -1.1915    0
    {'res2b_branch2b_Bias'   }    {'res2b_branch2b'}          "Bias"                 -0.81928
    {'res3a_branch2a_Weights'}    {'res3a_branch2a'}          "Weights"              -0.19735    0
    {'res3a_branch2a_Bias'   }    {'res3a_branch2a'}          "Bias"                 -0.53009    0
    {'res3a_branch2b_Weights'}    {'res3a_branch2b'}          "Weights"              -0.53557    0
    {'res3a_branch2b_Bias'   }    {'res3a_branch2b'}          "Bias"                 -0.67756
    {'res3a_branch1_Weights' }    {'res3a_branch1' }          "Weights"              -0.63395    0
    {'res3a_branch1_Bias'    }    {'res3a_branch1' }          "Bias"                 -0.95277    0
      ⋮
```

Define FPGA Board Interface

Define the target FPGA board programming interface by using the `dlhdl.Target` object. Create a programming interface with custom name for your target device and an Ethernet interface to connect the target device to the host computer.

```
hTarget = dlhdl.Target('Xilinx','Interface','Ethernet');
```

Prepare Network for Deployment

Prepare the network for deployment by creating a `dlhdl.Workflow` object. Specify the network and bitstream name. Ensure that the bitstream name matches the data type and the FPGA board that you are targeting. In this example, the target FPGA board is the Xilinx® Zynq® UltraScale+™ MPSoC ZCU102 board and the bitstream uses the `int8` data type.

```
hW = dlhdl.Workflow(Network=dlquantObj,Bitstream='zcu102_int8',Target=hTarget);
```

Compile Network

Run the `compile` method of the `dlhdl.Workflow` object to compile the network and generate the instructions, weights, and biases for deployment.

```
dn = compile(hW,'InputFrameNumberLimit',15)
```

```
### Compiling network for Deep Learning FPGA prototyping ...
### Targeting FPGA bitstream zcu102_int8.
### Optimizing network: Fused 'nnet.cnn.layer.BatchNormalizationLayer' into 'nnet.cnn.layer.Convo
### The network includes the following layers:
     1   'data'                  Image Input                  224×224×3 images with 'zscore' norm
```

```
     2   'conv1'                 2-D Convolution             64 7×7×3 convolutions with stride
     3   'conv1_relu'            ReLU                        ReLU
     4   'pool1'                 2-D Max Pooling             3×3 max pooling with stride [2  2]
     5   'res2a_branch2a'        2-D Convolution             64 3×3×64 convolutions with stride
     6   'res2a_branch2a_relu'   ReLU                        ReLU
     7   'res2a_branch2b'        2-D Convolution             64 3×3×64 convolutions with stride
     8   'res2a'                 Addition                    Element-wise addition of 2 inputs
     9   'res2a_relu'            ReLU                        ReLU
    10   'res2b_branch2a'        2-D Convolution             64 3×3×64 convolutions with stride
    11   'res2b_branch2a_relu'   ReLU                        ReLU
    12   'res2b_branch2b'        2-D Convolution             64 3×3×64 convolutions with stride
    13   'res2b'                 Addition                    Element-wise addition of 2 inputs
    14   'res2b_relu'            ReLU                        ReLU
    15   'res3a_branch2a'        2-D Convolution             128 3×3×64 convolutions with stride
    16   'res3a_branch2a_relu'   ReLU                        ReLU
    17   'res3a_branch2b'        2-D Convolution             128 3×3×128 convolutions with strid
    18   'res3a_branch1'         2-D Convolution             128 1×1×64 convolutions with stride
    19   'res3a'                 Addition                    Element-wise addition of 2 inputs
    20   'res3a_relu'            ReLU                        ReLU
    21   'res3b_branch2a'        2-D Convolution             128 3×3×128 convolutions with strid
    22   'res3b_branch2a_relu'   ReLU                        ReLU
    23   'res3b_branch2b'        2-D Convolution             128 3×3×128 convolutions with strid
    24   'res3b'                 Addition                    Element-wise addition of 2 inputs
    25   'res3b_relu'            ReLU                        ReLU
    26   'res4a_branch2a'        2-D Convolution             256 3×3×128 convolutions with strid
    27   'res4a_branch2a_relu'   ReLU                        ReLU
    28   'res4a_branch2b'        2-D Convolution             256 3×3×256 convolutions with strid
    29   'res4a_branch1'         2-D Convolution             256 1×1×128 convolutions with strid
    30   'res4a'                 Addition                    Element-wise addition of 2 inputs
    31   'res4a_relu'            ReLU                        ReLU
    32   'res4b_branch2a'        2-D Convolution             256 3×3×256 convolutions with strid
    33   'res4b_branch2a_relu'   ReLU                        ReLU
    34   'res4b_branch2b'        2-D Convolution             256 3×3×256 convolutions with strid
    35   'res4b'                 Addition                    Element-wise addition of 2 inputs
    36   'res4b_relu'            ReLU                        ReLU
    37   'res5a_branch2a'        2-D Convolution             512 3×3×256 convolutions with strid
    38   'res5a_branch2a_relu'   ReLU                        ReLU
    39   'res5a_branch2b'        2-D Convolution             512 3×3×512 convolutions with strid
    40   'res5a_branch1'         2-D Convolution             512 1×1×256 convolutions with strid
    41   'res5a'                 Addition                    Element-wise addition of 2 inputs
    42   'res5a_relu'            ReLU                        ReLU
    43   'res5b_branch2a'        2-D Convolution             512 3×3×512 convolutions with strid
    44   'res5b_branch2a_relu'   ReLU                        ReLU
    45   'res5b_branch2b'        2-D Convolution             512 3×3×512 convolutions with strid
    46   'res5b'                 Addition                    Element-wise addition of 2 inputs
    47   'res5b_relu'            ReLU                        ReLU
    48   'pool5'                 2-D Global Average Pooling  2-D global average pooling
    49   'new_fc'                Fully Connected             5 fully connected layer
    50   'prob'                  Softmax                     softmax
    51   'new_classoutput'       Classification Output       crossentropyex with 'MathWorks Cap

### Notice: The layer 'data' with type 'nnet.cnn.layer.ImageInputLayer' is implemented in softwar
### Notice: The layer 'prob' with type 'nnet.cnn.layer.SoftmaxLayer' is implemented in software.
### Notice: The layer 'new_classoutput' with type 'nnet.cnn.layer.ClassificationOutputLayer' is i
### Compiling layer group: conv1>>pool1 ...
### Compiling layer group: conv1>>pool1 ... complete.
### Compiling layer group: res2a_branch2a>>res2a_branch2b ...
### Compiling layer group: res2a_branch2a>>res2a_branch2b ... complete.
```

```
### Compiling layer group: res2b_branch2a>>res2b_branch2b ...
### Compiling layer group: res2b_branch2a>>res2b_branch2b ... complete.
### Compiling layer group: res3a_branch1 ...
### Compiling layer group: res3a_branch1 ... complete.
### Compiling layer group: res3a_branch2a>>res3a_branch2b ...
### Compiling layer group: res3a_branch2a>>res3a_branch2b ... complete.
### Compiling layer group: res3b_branch2a>>res3b_branch2b ...
### Compiling layer group: res3b_branch2a>>res3b_branch2b ... complete.
### Compiling layer group: res4a_branch1 ...
### Compiling layer group: res4a_branch1 ... complete.
### Compiling layer group: res4a_branch2a>>res4a_branch2b ...
### Compiling layer group: res4a_branch2a>>res4a_branch2b ... complete.
### Compiling layer group: res4b_branch2a>>res4b_branch2b ...
### Compiling layer group: res4b_branch2a>>res4b_branch2b ... complete.
### Compiling layer group: res5a_branch1 ...
### Compiling layer group: res5a_branch1 ... complete.
### Compiling layer group: res5a_branch2a>>res5a_branch2b ...
### Compiling layer group: res5a_branch2a>>res5a_branch2b ... complete.
### Compiling layer group: res5b_branch2a>>res5b_branch2b ...
### Compiling layer group: res5b_branch2a>>res5b_branch2b ... complete.
### Compiling layer group: pool5 ...
### Compiling layer group: pool5 ... complete.
### Compiling layer group: new_fc ...
### Compiling layer group: new_fc ... complete.

### Allocating external memory buffers:

          offset_name          offset_address    allocated_space
    ___________________________    ______________    _______________

    "InputDataOffset"             "0x00000000"     "8.0 MB"
    "OutputResultOffset"          "0x00800000"     "4.0 MB"
    "SchedulerDataOffset"         "0x00c00000"     "4.0 MB"
    "SystemBufferOffset"          "0x01000000"     "28.0 MB"
    "InstructionDataOffset"       "0x02c00000"     "4.0 MB"
    "ConvWeightDataOffset"        "0x03000000"     "16.0 MB"
    "FCWeightDataOffset"          "0x04000000"     "4.0 MB"
    "EndOffset"                   "0x04400000"     "Total: 68.0 MB"

### Network compilation complete.

dn = struct with fields:
             weights: [1×1 struct]
        instructions: [1×1 struct]
           registers: [1×1 struct]
    syncInstructions: [1×1 struct]
        constantData: {}
             ddrInfo: [1×1 struct]
```

Program Bitstream onto FPGA and Download Network Weights

To deploy the network on the Xilinx ZCU102 hardware, run the deploy function of the `dlhdl.Workflow` object. This function uses the output of the compile function to program the FPGA board by using the programming file. It also downloads the network weights and biases. The deploy function starts programming the FPGA device, displays progress messages, and the time it takes to deploy the network.

```
deploy(hW)
```

```
### Programming FPGA Bitstream using Ethernet...
### Attempting to connect to the hardware board at 192.168.1.101...
### Connection successful
### Programming FPGA device on Xilinx SoC hardware board at 192.168.1.101...
### Copying FPGA programming files to SD card...
### Setting FPGA bitstream and devicetree for boot...
# Copying Bitstream zcu102_int8.bit to /mnt/hdlcoder_rd
# Set Bitstream to hdlcoder_rd/zcu102_int8.bit
# Copying Devicetree devicetree_dlhdl.dtb to /mnt/hdlcoder_rd
# Set Devicetree to hdlcoder_rd/devicetree_dlhdl.dtb
# Set up boot for Reference Design: 'AXI-Stream DDR Memory Access : 3-AXIM'
### Rebooting Xilinx SoC at 192.168.1.101...
### Reboot may take several seconds...
### Attempting to connect to the hardware board at 192.168.1.101...
### Connection successful
### Programming the FPGA bitstream has been completed successfully.
### Loading weights to Conv Processor.
### Conv Weights loaded. Current time is 21-Dec-2022 10:45:19
### Loading weights to FC Processor.
### FC Weights loaded. Current time is 21-Dec-2022 10:45:19
```

Test Network

Load the example image.

```
imgFile = fullfile(pwd,'MerchData','MathWorks Cube','Mathworks cube_0.jpg');
inputImg = imresize(imread(imgFile),[224 224]);
imshow(inputImg)
```

Classify the image on the FPGA by using the `predict` method of the `dlhdl.Workflow` object and display the results.

```
[prediction,speed] = predict(hW,single(inputImg),'Profile','on');
```

```
### Finished writing input activations.
### Running single input activation.


              Deep Learning Processor Profiler Performance Results

                    LastFrameLatency(cycles)   LastFrameLatency(seconds)       FramesNum      Tota
                          -------------              -------------               ---------        --
Network                     7392114                   0.02957                      1            73
    conv1                   1115165                   0.00446
    pool1                    199164                   0.00080
    res2a_branch2a           270125                   0.00108
    res2a_branch2b           269946                   0.00108
    res2a                    102255                   0.00041
    res2b_branch2a           269792                   0.00108
    res2b_branch2b           269902                   0.00108
    res2b                    102695                   0.00041
    res3a_branch1            155120                   0.00062
    res3a_branch2a           156480                   0.00063
    res3a_branch2b           244913                   0.00098
    res3a                     51456                   0.00021
    res3b_branch2a           245366                   0.00098
    res3b_branch2b           245123                   0.00098
    res3b                     51286                   0.00021
    res4a_branch1            135535                   0.00054
    res4a_branch2a           136117                   0.00054
    res4a_branch2b           238454                   0.00095
    res4a                     25602                   0.00010
    res4b_branch2a           237909                   0.00095
    res4b_branch2b           238282                   0.00095
    res4b                     26742                   0.00011
    res5a_branch1            324642                   0.00130
    res5a_branch2a           325897                   0.00130
    res5a_branch2b           623521                   0.00249
    res5a                     13881                   0.00006
    res5b_branch2a           624028                   0.00250
    res5b_branch2b           624631                   0.00250
    res5b                     13051                   0.00005
    pool5                     37083                   0.00015
    new_fc                    17764                   0.00007
 * The clock frequency of the DL processor is: 250MHz
```

```
[val,idx] = max(prediction);
dlquantObj.NetworkObject.Layers(end).ClassNames{idx}
```

```
ans =
'MathWorks Cube'
```

Performance Comparison

Compare the performance of the quantized network to the performance of the single data type network.

```
optionsFPGA = dlquantizationOptions('Bitstream','zcu102_int8','Target',hTarget);
predictionFPGA = validate(dlquantObj,imdsValidation,optionsFPGA)
```

```
### Compiling network for Deep Learning FPGA prototyping ...
### Targeting FPGA bitstream zcu102_int8.
### Optimizing network: Fused 'nnet.cnn.layer.BatchNormalizationLayer' into 'nnet.cnn.layer.Convo
```

```
### The network includes the following layers:
     1   'data'                  Image Input                 224×224×3 images with 'zscore' norm
     2   'conv1'                 2-D Convolution             64 7×7×3 convolutions with stride
     3   'conv1_relu'            ReLU                        ReLU
     4   'pool1'                 2-D Max Pooling             3×3 max pooling with stride [2  2]
     5   'res2a_branch2a'        2-D Convolution             64 3×3×64 convolutions with stride
     6   'res2a_branch2a_relu'   ReLU                        ReLU
     7   'res2a_branch2b'        2-D Convolution             64 3×3×64 convolutions with stride
     8   'res2a'                 Addition                    Element-wise addition of 2 inputs
     9   'res2a_relu'            ReLU                        ReLU
    10   'res2b_branch2a'        2-D Convolution             64 3×3×64 convolutions with stride
    11   'res2b_branch2a_relu'   ReLU                        ReLU
    12   'res2b_branch2b'        2-D Convolution             64 3×3×64 convolutions with stride
    13   'res2b'                 Addition                    Element-wise addition of 2 inputs
    14   'res2b_relu'            ReLU                        ReLU
    15   'res3a_branch2a'        2-D Convolution             128 3×3×64 convolutions with stride
    16   'res3a_branch2a_relu'   ReLU                        ReLU
    17   'res3a_branch2b'        2-D Convolution             128 3×3×128 convolutions with strid
    18   'res3a_branch1'         2-D Convolution             128 1×1×64 convolutions with stride
    19   'res3a'                 Addition                    Element-wise addition of 2 inputs
    20   'res3a_relu'            ReLU                        ReLU
    21   'res3b_branch2a'        2-D Convolution             128 3×3×128 convolutions with strid
    22   'res3b_branch2a_relu'   ReLU                        ReLU
    23   'res3b_branch2b'        2-D Convolution             128 3×3×128 convolutions with strid
    24   'res3b'                 Addition                    Element-wise addition of 2 inputs
    25   'res3b_relu'            ReLU                        ReLU
    26   'res4a_branch2a'        2-D Convolution             256 3×3×128 convolutions with strid
    27   'res4a_branch2a_relu'   ReLU                        ReLU
    28   'res4a_branch2b'        2-D Convolution             256 3×3×256 convolutions with strid
    29   'res4a_branch1'         2-D Convolution             256 1×1×128 convolutions with strid
    30   'res4a'                 Addition                    Element-wise addition of 2 inputs
    31   'res4a_relu'            ReLU                        ReLU
    32   'res4b_branch2a'        2-D Convolution             256 3×3×256 convolutions with strid
    33   'res4b_branch2a_relu'   ReLU                        ReLU
    34   'res4b_branch2b'        2-D Convolution             256 3×3×256 convolutions with strid
    35   'res4b'                 Addition                    Element-wise addition of 2 inputs
    36   'res4b_relu'            ReLU                        ReLU
    37   'res5a_branch2a'        2-D Convolution             512 3×3×256 convolutions with strid
    38   'res5a_branch2a_relu'   ReLU                        ReLU
    39   'res5a_branch2b'        2-D Convolution             512 3×3×512 convolutions with strid
    40   'res5a_branch1'         2-D Convolution             512 1×1×256 convolutions with strid
    41   'res5a'                 Addition                    Element-wise addition of 2 inputs
    42   'res5a_relu'            ReLU                        ReLU
    43   'res5b_branch2a'        2-D Convolution             512 3×3×512 convolutions with strid
    44   'res5b_branch2a_relu'   ReLU                        ReLU
    45   'res5b_branch2b'        2-D Convolution             512 3×3×512 convolutions with strid
    46   'res5b'                 Addition                    Element-wise addition of 2 inputs
    47   'res5b_relu'            ReLU                        ReLU
    48   'pool5'                 2-D Global Average Pooling  2-D global average pooling
    49   'new_fc'                Fully Connected             5 fully connected layer
    50   'prob'                  Softmax                     softmax
    51   'new_classoutput'       Classification Output       crossentropyex with 'MathWorks Cap

### Notice: The layer 'data' with type 'nnet.cnn.layer.ImageInputLayer' is implemented in softwar
### Notice: The layer 'prob' with type 'nnet.cnn.layer.SoftmaxLayer' is implemented in software.
### Notice: The layer 'new_classoutput' with type 'nnet.cnn.layer.ClassificationOutputLayer' is i
### Compiling layer group: conv1>>pool1 ...
### Compiling layer group: conv1>>pool1 ... complete.
```

```
### Compiling layer group: res2a_branch2a>>res2a_branch2b ...
### Compiling layer group: res2a_branch2a>>res2a_branch2b ... complete.
### Compiling layer group: res2b_branch2a>>res2b_branch2b ...
### Compiling layer group: res2b_branch2a>>res2b_branch2b ... complete.
### Compiling layer group: res3a_branch1 ...
### Compiling layer group: res3a_branch1 ... complete.
### Compiling layer group: res3a_branch2a>>res3a_branch2b ...
### Compiling layer group: res3a_branch2a>>res3a_branch2b ... complete.
### Compiling layer group: res3b_branch2a>>res3b_branch2b ...
### Compiling layer group: res3b_branch2a>>res3b_branch2b ... complete.
### Compiling layer group: res4a_branch1 ...
### Compiling layer group: res4a_branch1 ... complete.
### Compiling layer group: res4a_branch2a>>res4a_branch2b ...
### Compiling layer group: res4a_branch2a>>res4a_branch2b ... complete.
### Compiling layer group: res4b_branch2a>>res4b_branch2b ...
### Compiling layer group: res4b_branch2a>>res4b_branch2b ... complete.
### Compiling layer group: res5a_branch1 ...
### Compiling layer group: res5a_branch1 ... complete.
### Compiling layer group: res5a_branch2a>>res5a_branch2b ...
### Compiling layer group: res5a_branch2a>>res5a_branch2b ... complete.
### Compiling layer group: res5b_branch2a>>res5b_branch2b ...
### Compiling layer group: res5b_branch2a>>res5b_branch2b ... complete.
### Compiling layer group: pool5 ...
### Compiling layer group: pool5 ... complete.
### Compiling layer group: new_fc ...
### Compiling layer group: new_fc ... complete.

### Allocating external memory buffers:

          offset_name          offset_address    allocated_space
    ___________________________    ______________    _______________

    "InputDataOffset"           "0x00000000"     "12.0 MB"
    "OutputResultOffset"        "0x00c00000"     "4.0 MB"
    "SchedulerDataOffset"       "0x01000000"     "4.0 MB"
    "SystemBufferOffset"        "0x01400000"     "28.0 MB"
    "InstructionDataOffset"     "0x03000000"     "4.0 MB"
    "ConvWeightDataOffset"      "0x03400000"     "16.0 MB"
    "FCWeightDataOffset"        "0x04400000"     "4.0 MB"
    "EndOffset"                 "0x04800000"     "Total: 72.0 MB"

### Network compilation complete.

### FPGA bitstream programming has been skipped as the same bitstream is already loaded on the ta
### Loading weights to Conv Processor.
### Conv Weights loaded. Current time is 21-Dec-2022 10:46:36
### Loading weights to FC Processor.
### FC Weights loaded. Current time is 21-Dec-2022 10:46:36
### Finished writing input activations.
### Running single input activation.
### Finished writing input activations.
### Running single input activation.
### Finished writing input activations.
### Running single input activation.
### Finished writing input activations.
### Running single input activation.
### Finished writing input activations.
### Running single input activation.
```

```
### Finished writing input activations.
### Running single input activation.
### Finished writing input activations.
### Running single input activation.
### Finished writing input activations.
### Running single input activation.
### Finished writing input activations.
### Running single input activation.
### Finished writing input activations.
### Running single input activation.
### Finished writing input activations.
### Running single input activation.
### Finished writing input activations.
### Running single input activation.
### Finished writing input activations.
### Running single input activation.
### Finished writing input activations.
### Running single input activation.
### Finished writing input activations.
### Running single input activation.
### Finished writing input activations.
### Running single input activation.
### Finished writing input activations.
### Running single input activation.


              Deep Learning Processor Bitstream Build Info

Resource                   Utilized           Total        Percentage
------------------        ----------      ------------    ------------
LUTs (CLB/ALM)*              249703            274080           91.11
DSPs                            391              2520           15.52
Block RAM                       583               912           63.93
* LUT count represents Configurable Logic Block(CLB) utilization in Xilinx devices and Adaptive L

### Optimizing network: Fused 'nnet.cnn.layer.BatchNormalizationLayer' into 'nnet.cnn.layer.Convo
### Notice: The layer 'data' of type 'ImageInputLayer' is split into an image input layer 'data'
### The network includes the following layers:
     1   'data'                  Image Input                 224×224×3 images with 'zscore' norm
     2   'conv1'                 2-D Convolution             64 7×7×3 convolutions with stride
     3   'conv1_relu'            ReLU                        ReLU
     4   'pool1'                 2-D Max Pooling             3×3 max pooling with stride [2  2]
     5   'res2a_branch2a'        2-D Convolution             64 3×3×64 convolutions with stride
     6   'res2a_branch2a_relu'   ReLU                        ReLU
     7   'res2a_branch2b'        2-D Convolution             64 3×3×64 convolutions with stride
     8   'res2a'                 Addition                    Element-wise addition of 2 inputs
     9   'res2a_relu'            ReLU                        ReLU
    10   'res2b_branch2a'        2-D Convolution             64 3×3×64 convolutions with stride
    11   'res2b_branch2a_relu'   ReLU                        ReLU
    12   'res2b_branch2b'        2-D Convolution             64 3×3×64 convolutions with stride
    13   'res2b'                 Addition                    Element-wise addition of 2 inputs
    14   'res2b_relu'            ReLU                        ReLU
```

```
    15   'res3a_branch2a'        2-D Convolution             128 3×3×64 convolutions with stride
    16   'res3a_branch2a_relu'   ReLU                        ReLU
    17   'res3a_branch2b'        2-D Convolution             128 3×3×128 convolutions with strid
    18   'res3a_branch1'         2-D Convolution             128 1×1×64 convolutions with stride
    19   'res3a'                 Addition                    Element-wise addition of 2 inputs
    20   'res3a_relu'            ReLU                        ReLU
    21   'res3b_branch2a'        2-D Convolution             128 3×3×128 convolutions with strid
    22   'res3b_branch2a_relu'   ReLU                        ReLU
    23   'res3b_branch2b'        2-D Convolution             128 3×3×128 convolutions with strid
    24   'res3b'                 Addition                    Element-wise addition of 2 inputs
    25   'res3b_relu'            ReLU                        ReLU
    26   'res4a_branch2a'        2-D Convolution             256 3×3×128 convolutions with strid
    27   'res4a_branch2a_relu'   ReLU                        ReLU
    28   'res4a_branch2b'        2-D Convolution             256 3×3×256 convolutions with strid
    29   'res4a_branch1'         2-D Convolution             256 1×1×128 convolutions with strid
    30   'res4a'                 Addition                    Element-wise addition of 2 inputs
    31   'res4a_relu'            ReLU                        ReLU
    32   'res4b_branch2a'        2-D Convolution             256 3×3×256 convolutions with strid
    33   'res4b_branch2a_relu'   ReLU                        ReLU
    34   'res4b_branch2b'        2-D Convolution             256 3×3×256 convolutions with strid
    35   'res4b'                 Addition                    Element-wise addition of 2 inputs
    36   'res4b_relu'            ReLU                        ReLU
    37   'res5a_branch2a'        2-D Convolution             512 3×3×256 convolutions with strid
    38   'res5a_branch2a_relu'   ReLU                        ReLU
    39   'res5a_branch2b'        2-D Convolution             512 3×3×512 convolutions with strid
    40   'res5a_branch1'         2-D Convolution             512 1×1×256 convolutions with strid
    41   'res5a'                 Addition                    Element-wise addition of 2 inputs
    42   'res5a_relu'            ReLU                        ReLU
    43   'res5b_branch2a'        2-D Convolution             512 3×3×512 convolutions with strid
    44   'res5b_branch2a_relu'   ReLU                        ReLU
    45   'res5b_branch2b'        2-D Convolution             512 3×3×512 convolutions with strid
    46   'res5b'                 Addition                    Element-wise addition of 2 inputs
    47   'res5b_relu'            ReLU                        ReLU
    48   'pool5'                 2-D Global Average Pooling  2-D global average pooling
    49   'new_fc'                Fully Connected             5 fully connected layer
    50   'prob'                  Softmax                     softmax
    51   'new_classoutput'       Classification Output       crossentropyex with 'MathWorks Cap

### Notice: The layer 'prob' with type 'nnet.cnn.layer.SoftmaxLayer' is implemented in software.
### Notice: The layer 'new_classoutput' with type 'nnet.cnn.layer.ClassificationOutputLayer' is i


              Deep Learning Processor Estimator Performance Results

                    LastFrameLatency(cycles)   LastFrameLatency(seconds)       FramesNum      Tota
                          -------------              -------------               ---------        --
Network                    23502752                 0.10683                       1           235
    data_norm_add            210750                 0.00096
    data_norm                210750                 0.00096
    conv1                   2164124                 0.00984
    pool1                    515064                 0.00234
    res2a_branch2a           966221                 0.00439
    res2a_branch2b           966221                 0.00439
    res2a                    210750                 0.00096
    res2b_branch2a           966221                 0.00439
    res2b_branch2b           966221                 0.00439
    res2b                    210750                 0.00096
    res3a_branch1            540861                 0.00246
```

```
    res3a_branch2a          540749                  0.00246
    res3a_branch2b          919117                  0.00418
    res3a                   105404                  0.00048
    res3b_branch2a          919117                  0.00418
    res3b_branch2b          919117                  0.00418
    res3b                   105404                  0.00048
    res4a_branch1           503405                  0.00229
    res4a_branch2a          509261                  0.00231
    res4a_branch2b          905421                  0.00412
    res4a                    52724                  0.00024
    res4b_branch2a          905421                  0.00412
    res4b_branch2b          905421                  0.00412
    res4b                    52724                  0.00024
    res5a_branch1          1039437                  0.00472
    res5a_branch2a         1046605                  0.00476
    res5a_branch2b         2005197                  0.00911
    res5a                    26368                  0.00012
    res5b_branch2a         2005197                  0.00911
    res5b_branch2b         2005197                  0.00911
    res5b                    26368                  0.00012
    pool5                    54594                  0.00025
    new_fc                   22571                  0.00010
 * The clock frequency of the DL processor is: 220MHz



              Deep Learning Processor Bitstream Build Info

Resource                   Utilized           Total        Percentage
------------------        ----------      ------------    ------------
LUTs (CLB/ALM)*              168099            274080           61.33
DSPs                            807              2520           32.02
Block RAM                       453               912           49.67
* LUT count represents Configurable Logic Block(CLB) utilization in Xilinx devices and Adaptive L

### Finished writing input activations.
### Running single input activation.

predictionFPGA = struct with fields:
       NumSamples: 20
    MetricResults: [1×1 struct]
       Statistics: [2×7 table]
```

View the frames per second performance for the quantized network and single-data-type network. The quantized network has a performance of 33.8 frames per second compared to 9.2 frames per second for the single-data-type network. You can use quantization to improve your frames per second performance, however yo could lose accuracy when you quantize your networks.

```
predictionFPGA.Statistics.FramesPerSecond
```

```
ans = 2×1

    9.3606
```

```
33.7719
```

Version History

Introduced in R2020b

See Also

`compile` | `getBuildInfo` | `predict` | `dlquantizer` | `dlquantizationOptions` | `calibrate` | `validate` | `predictAndUpdateState` | `resetState`

getBuildInfo

Class: `dlhdl.Workflow`
Package: `dlhdl`

Retrieve bitstream resource utilization

Syntax

```
area = getBuildInfo(workflowObject)
```

Description

`area = getBuildInfo(workflowObject)` returns a structure containing the bitstream resource utilization.

Input Arguments

`workflowObject` — Workflow

`dlhdl.Workflow` object

Workflow, specified as a `dlhdl.Workflow` object.

Output Arguments

`area` — Bitstream resource utilization

`structure`

Bitstream resource utilization returned as a structure.

- The `Block Memory Bits` utilization is available for Intel bitstreams only.
- The resource utilization results of Intel bitstreams show the `Block RAM` utilization as 100%. To analyze bitstream resource utilization, refer to the `Block Memory Bits` utilization instead.

Examples

Retrieve `arria10soc_single`Bitstream Resource Utilization

1. Create a file in your current working folder called `getLogoNetwork.m`. In the file, enter:

```
function net = getLogoNetwork
 if ~isfile('LogoNet.mat')
        url = 'https://www.mathworks.com/supportfiles/gpucoder/cnn_models/logo_detection/LogoNet.mat';
        websave('LogoNet.mat',url);
    end
    data = load('LogoNet.mat');
    net  = data.convnet;
end
```

2. Create a `dlhdl.Workflow` object that has `LogoNet` as the `Network` argument and `arria10soc_single` as the `Bitstream` argument.

```
snet = getLogoNetwork;
hW = dlhdl.Workflow('Network',snet,'Bitstream','arria10soc_single');
```

3 Call `getBuildInfo` argument to retrieve the `arria10soc_single` resource utilization. Store the resource utilization in `area`.

```
area = hW.getBuildInfo
```

```
              Deep Learning Processor Bitstream Build Info

Resource                   Utilized           Total        Percentage
------------------        ----------      ------------    ------------
LUTs (CLB/ALM)*               93578            251680           37.18
DSPs                            278              1687           16.48
Block RAM                      2131              2131          100.00
Block Memory Bits          23211920          43642880           53.19
* LUT count represents Configurable Logic Block(CLB) utilization in Xilinx devices and Adaptive Logic Module (ALM) u


area =

  struct with fields:

                LUT: [93578 251680]
    BlockMemoryBits: [23211920 43642880]
           BlockRAM: [2131 2131]
                DSP: [278 1687]
```

Version History

Introduced in R2021a

See Also

`compile` | `deploy` | `predict`

predict

Class: `dlhdl.Workflow`
Package: `dlhdl`

Predict responses by using deployed network

Syntax

```
Y = predict(workflowObject,images)
Y = predict(workflowObject,X1,...,XN)
[Y1,...,YM] = predict( ___ )
[Y,performance] = predict( ___ ,Name,Value)
```

Description

`Y = predict(workflowObject,images)` predicts responses for the image data, `images`, by using the deep learning network specified in the `dlhdl.Workflow` object, `workflowObject`.

`Y = predict(workflowObject,X1,...,XN)` predicts the responses for the data in the numeric or cell arrays `X1`, ..., `XN` for the multi-input network specified in the `Network` argument of the `workflowObject`. The input `XN` corresponds to the `workflowObject.Network.InputNames(N)`.

`[Y1,...,YM] = predict( ___ )` predicts responses for the `M` outputs of a multi-output network using any of the previous input arguments. The output `YM` corresponds to the output of the network specified in `workflowObject.Network.OutputNames(M)`.

`[Y,performance] = predict( ___ ,Name,Value)` predicts the responses with one or more arguments specified by optional name-value pair arguments.

Input Arguments

`workflowObject` — Workflow

`dlhdl.Workflow` object

Workflow, specified as a `dlhdl.Workflow` object.

`images` — Input image or input data

numeric array | formatted `dlarray` object

Input image, specified as a numeric array, cell array or formatted `dlarray` object. The numeric arrays can be 3-D or 4-D arrays. For 4-D arrays, the fourth dimension is the number of input images. If one of the members of the numeric array has four dimensions, then the other members of the numeric arrays must have four dimensions as well, with the value of the fourth dimension being the same for all members.

If the network specified in the `dlhdl.Workflow` object is a `dlnetwork` object, then the input image must be a formatted `dlarray` object. For more information about `dlarray` formats, see the `fmt` input argument of `dlarray`.

Data Types: `single` | `int8`

`X1,...,XN` — Numeric arrays for networks with multiple inputs

numeric array

Numeric or cell arrays for networks with multiple inputs, specified as a numeric array or cell array.

For multiple inputs to image prediction networks, the format of the predictors must match the formats described in the `images` argument descriptions.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example:

`Profile` — Flag that returns profiling results

"off" (default) | "on"

Flag to return profiling results for the deep learning network deployed to the target board, specified as `"off"` or `"on"`.

Example: `Profile = "on"`

Output Arguments

`Y` — Predicted responses

numeric array

Predicted responses, returned as a numeric array. The format of `Y` depends on the type of task.

Task	Format
2-D image regression	• *h*-by-*w*-by-*c*-by-*N* numeric array, where *h*, *w*, and *c* are the height, width, and number of channels of the images, respectively, and *N* is the number of images
3-D image regression	• *h*-by-*w*-by-*d*-by-*c*-by-*N* numeric array, where *h*, *w*, *d*, and *c* are the height, width, depth, and number of channels of the images, respectively, and *N* is the number of images
Sequence-to-one regression	*N*-by-*R* matrix, where *N* is the number of sequences and *R* is the number of responses
Sequence-to-sequence regression	*N*-by-*R* matrix, where *N* is the number of sequences and *R* is the number of responses
Feature regression	*N*-by-*R* matrix, where *N* is the number of observations and *R* is the number of responses

For sequence-to-sequence regression problems with one observation, `images` can be a matrix. In this case, `Y` is a matrix of responses.

If the output layer of the network is a classification layer, then `Y` is the predicted classification scores. This table describes the format of the scores for classification tasks.

Task	Format
Image classification	*N*-by-*K* matrix, where *N* is the number of observations and *K* is the number of classes
Sequence-to-label classification	
Feature classification	

`Y1,...,YM` — Predicted scores or responses of networks with multiple outputs
numeric array

Predicted scores or responses of networks with multiple outputs, returned as numeric arrays.

Each output `Yj` corresponds to the network output `net.OutputNames(j)` and has format as described in the `Y` output argument.

`performance` — Deployed network performance data
table

Deployed network performance data, returned as an *N*-by-5 table, where *N* is the number of layers in the network. This method returns performance only when the `Profile` name-value argument is set to `'on'`. To learn about the data in the performance table, see “Profile Inference Run”.

Examples

Bicyclist and Pedestrian Classification by Using FPGA

This example shows how to deploy a custom trained series network to detect pedestrians and bicyclists based on their micro-Doppler signatures. This network is taken from the Pedestrian and Bicyclist Classification Using Deep Learning example from the Phased Array Toolbox. For more details on network training and input data, see “Pedestrian and Bicyclist Classification Using Deep Learning” (Radar Toolbox).

Prerequisites

- Xilinx™ Vivado™ Design Suite 2020.2
- Zynq® UltraScale+™ MPSoC ZCU102 Evaluation Kit
- HDL Verifier™ Support Package for XIlinx FPGA Boards
- MATLAB™ Coder ™ Interface for Deep Learning Libraries
- Deep Learning Toolbox™
- Deep Learning HDL Toolbox™

The data files used in this example are:

- The MAT File `trainedNetBicPed.mat` contains a model trained on training data set `trainDataNoCar` and its label set `trainLabelNoCar`.
- The MAT File `testDataBicPed.mat` contains the test data set `testDataNoCar` and its label set `testLabelNoCar`.

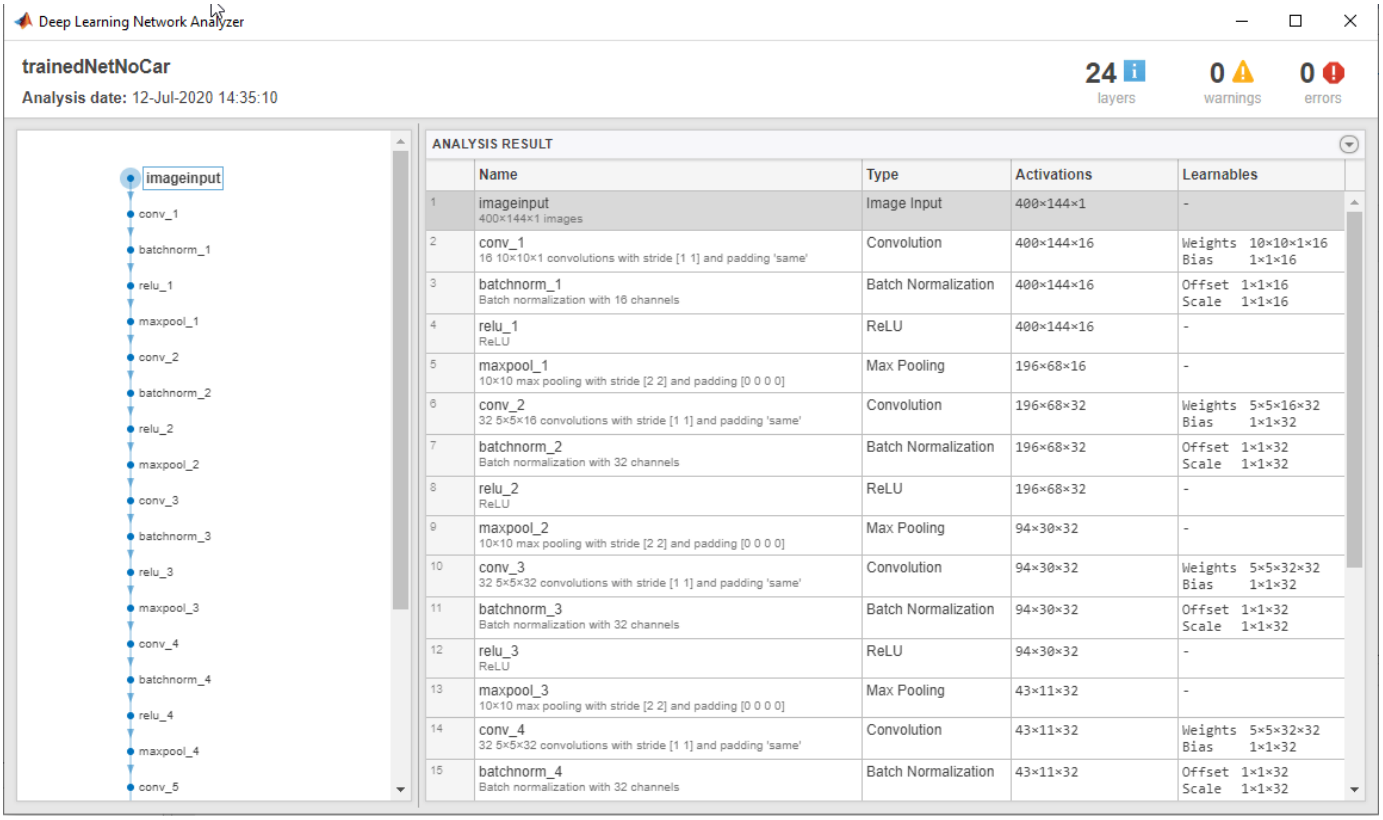
Load Data and Network

Load a pretrained network. Load test data and its labels.

```
load('trainedNetBicPed.mat','trainedNetNoCar')
load('testDataBicPed.mat')
```

View the layers of the pre-trained series network

```
analyzeNetwork(trainedNetNoCar);
```



Set up HDL Toolpath

Set up the path to your installed Xilinx™ Vivado™ Design Suite 2020.2 executable if it is not already set up. For example, to set the toolpath, enter:

```
% hdlsetuptoolpath('ToolName', 'Xilinx Vivado','ToolPath', 'C:\Vivado\2020.2\bin');
```

Create Target Object

Create a target object for your target device with a vendor name and an interface to connect your target device to the host computer. Interface options are JTAG (default) and Ethernet. Vendor options are Intel or Xilinx. Use the installed Xilinx Vivado Design Suite over an Ethernet connection to program the device.

```
hT = dlhdl.Target('Xilinx', 'Interface', 'Ethernet');
```

Create Workflow Object

Create an object of the `dlhdl.Workflow` class. When you create the object, specify the network and the bitstream name. Specify the saved pre-trained series network, `trainedNetNoCar`, as the network. Make sure the bitstream name matches the data type and the FPGA board that you are targeting. In this example, the target FPGA board is the Zynq UltraScale+ MPSoC ZCU102 board. The bitstream uses a single data type. .

```
hW = dlhdl.Workflow('Network', trainedNetNoCar, 'Bitstream', 'zcu102_single', 'Target', hT);
```

Compile `trainedNetNoCar` Series Network

To compile the `trainedNetNoCar` series network, run the compile function of the `dlhdl.Workflow` object .

```
dn = hW.compile;
```

```
### Optimizing series network: Fused 'nnet.cnn.layer.BatchNormalizationLayer' into 'nnet.cnn.laye
          offset_name          offset_address    allocated_space
    _________________________    ______________    _______________

    "InputDataOffset"            "0x00000000"      "28.0 MB"
    "OutputResultOffset"         "0x01c00000"      "4.0 MB"
    "SystemBufferOffset"         "0x02000000"      "28.0 MB"
    "InstructionDataOffset"      "0x03c00000"      "4.0 MB"
    "ConvWeightDataOffset"       "0x04000000"      "4.0 MB"
    "FCWeightDataOffset"         "0x04400000"      "4.0 MB"
    "EndOffset"                  "0x04800000"      "Total: 72.0 MB"
```

Program the Bitstream onto FPGA and Download Network Weights

To deploy the network on the Zynq® UltraScale+™ MPSoC ZCU102 hardware, run the deploy function of the `dlhdl.Workflow` object . This function uses the output of the compile function to program the FPGA board by using the programming file.The function also downloads the network weights and biases. The deploy function checks for the Xilinx Vivado tool and the supported tool version. It then starts programming the FPGA device by using the bitstream, displays progress messages and the time it takes to deploy the network.

```
hW.deploy;
```

```
### FPGA bitstream programming has been skipped as the same bitstream is already loaded on the ta
### Deep learning network programming has been skipped as the same network is already loaded on t
```

Run Predictions on Micro-Doppler Signatures

Classify one input from the sample test data set by using the predict function of the `dlhdl.Workflow` object and display the label. The inputs to the network correspond to the sonograms of the micro-Doppler signatures for a pedestrian or a bicyclist or a combination of both.

```
testImg = single(testDataNoCar(:, :, :, 1));
testLabel = testLabelNoCar(1);
classnames = trainedNetNoCar.Layers(end).Classes;

% Get predictions from network on single test input
score = hW.predict(testImg, 'Profile', 'On')
```

```
### Finished writing input activations.
### Running single input activations.
```

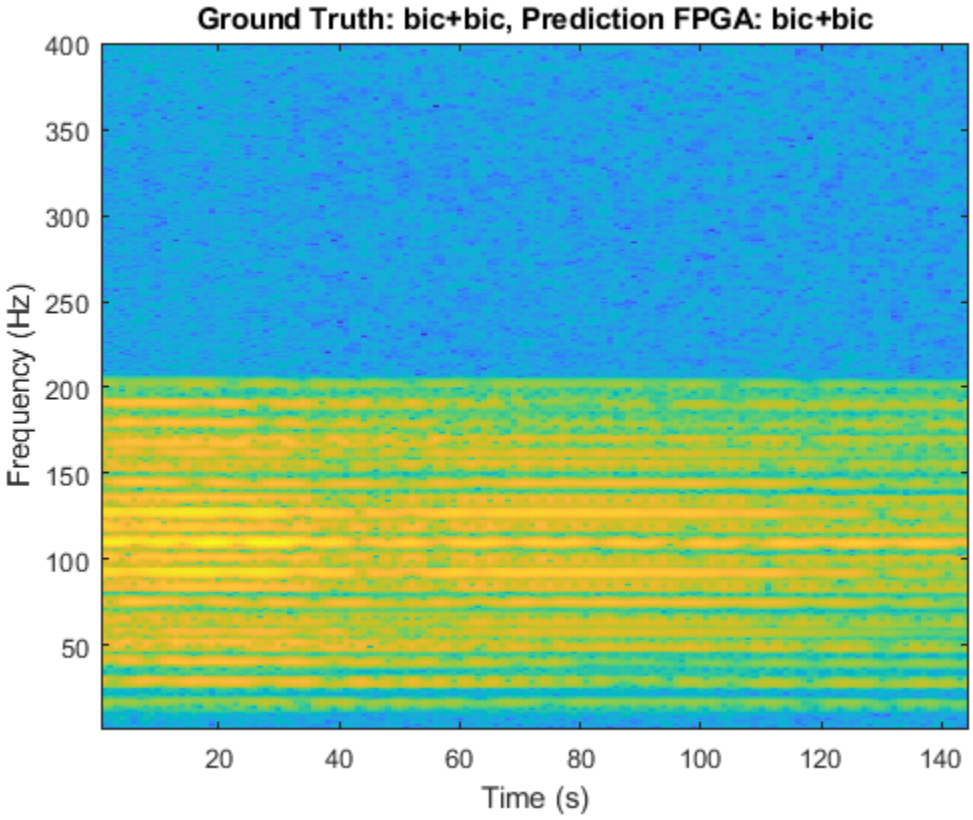
```
              Deep Learning Processor Profiler Performance Results

                     LastLayerLatency(cycles)   LastLayerLatency(seconds)       FramesNum      Tota
                           -------------              -------------               ---------        --
Network                      9430692                  0.04287                       1            94
    conv_module              9411355                  0.04278
        conv_1               4178753                  0.01899
        maxpool_1            1394883                  0.00634
        conv_2               1975197                  0.00898
        maxpool_2             706156                  0.00321
        conv_3                813598                  0.00370
        maxpool_3             121790                  0.00055
        conv_4                148165                  0.00067
        maxpool_4              22255                  0.00010
        conv_5                 41999                  0.00019
        avgpool2d               8674                  0.00004
    fc_module                  19337                  0.00009
        fc                     19337                  0.00009
 * The clock frequency of the DL processor is: 220MHz

score = 1×5 single row vector

    0.9956    0.0000    0.0000    0.0044    0.0000


[~, idx1] = max(score);
predTestLabel = classnames(idx1)

predTestLabel = categorical
     ped
```

Load five random images from the sample test data set and execute the predict function of the `dlhdl.Workflow` object to display the labels alongside the signatures. The predictions will happen at once since the input is concatenated along the fourth dimension.

```
numTestFrames = size(testDataNoCar, 4);
numView = 5;
listIndex = randperm(numTestFrames, numView);
testImgBatch = single(testDataNoCar(:, :, :, listIndex));
testLabelBatch = testLabelNoCar(listIndex);

% Get predictions from network using DL HDL Toolbox on FPGA
[scores, speed] = hW.predict(testImgBatch, 'Profile', 'On');

### Finished writing input activations.
### Running single input activations.


              Deep Learning Processor Profiler Performance Results

                     LastLayerLatency(cycles)   LastLayerLatency(seconds)       FramesNum      Tota
                           -------------              -------------               ---------        --
Network                      9446929                  0.04294                       5           471
    conv_module              9427488                  0.04285
        conv_1               4195175                  0.01907
        maxpool_1            1394705                  0.00634
```

```
        conv_2             1975204                  0.00898
        maxpool_2           706332                  0.00321
        conv_3              813499                  0.00370
        maxpool_3           121869                  0.00055
        conv_4              148063                  0.00067
        maxpool_4            22019                  0.00010
        conv_5               42053                  0.00019
        avgpool2d             8684                  0.00004
    fc_module                19441                  0.00009
        fc                   19441                  0.00009
 * The clock frequency of the DL processor is: 220MHz
```

```
[~, idx2] = max(scores, [], 2);
predTestLabelBatch = classnames(idx2);

% Display the micro-doppler signatures along with the ground truth and
% predictions.
for k = 1:numView
    index = listIndex(k);
    imagesc(testDataNoCar(:, :, :, index));
    axis xy
    xlabel('Time (s)')
    ylabel('Frequency (Hz)')
    title('Ground Truth: '+string(testLabelNoCar(index))+', Prediction FPGA: '+string(predTestLa
    drawnow;
    pause(3);
end
```



The image shows the micro-Doppler signatures of two bicyclists (bic+bic) which is the ground truth. The ground truth is the classification of the image against which the network prediction is compared. The network prediction retrieved from the FPGA correctly predicts that the image has two bicyclists.

Classify Images on FPGA Using Quantized Neural Network

This example shows how to use Deep Learning HDL Toolbox™ to deploy a quantized deep convolutional neural network (CNN) to an FPGA. In the example you use the pretrained ResNet-18 CNN to perform transfer learning and quantization. You then deploy the quantized network and use MATLAB ® to retrieve the prediction results.

ResNet-18 has been trained on over a million images and can classify images into 1000 object categories, such as keyboard, coffee mug, pencil, and many animals. The network has learned rich feature representations for a wide range of images. The network takes an image as input and outputs a label for the object in the image together with the probabilities for each of the object categories.

For this example, you need:

- Deep Learning Toolbox™
- Deep Learning HDL Toolbox™
- Deep Learning Toolbox Model for ResNet-18 Network
- Deep Learning HDL Toolbox™ Support Package for Xilinx® FPGA and SoC Devices
- Image Processing Toolbox™
- Deep Learning Toolbox Model Quantization Library
- MATLAB® Coder™ Interface for Deep Learning

To perform classification on a new set of images, you fine-tune a pretrained ResNet-18 CNN by transfer learning. In transfer learning, you can take a pretrained network and use it as a starting point to learn a new task. Fine-tuning a network with transfer learning is usually much faster and easier than training a network with randomly initialized weights. You can quickly transfer learned features to a new task using a smaller number of training images.

Load Pretrained Network

Load the pretrained ResNet-18 network.

```
net = resnet18;
```

View the layers of the pretrained network.

```
deepNetworkDesigner(net);
```

The first layer, the image input layer, requires input images of size 227-by-227-by-3, where three is the number of color channels.

```
inputSize = net.Layers(1).InputSize;
```

Load Data

This example uses the MathWorks MerchData data set. This is a small data set containing 75 images of MathWorks merchandise, belonging to five different classes (*cap*, *cube*, *playing cards*, *screwdriver*, and *torch*).

```
curDir = pwd;
unzip('MerchData.zip');
imds = imageDatastore('MerchData', ...
'IncludeSubfolders',true, ...
'LabelSource','foldernames');
```

Specify Training and Validation Sets

Divide the data into training and validation data sets, so that 30% percent of the images go to the training data set and 70% of the images to the validation data set. `splitEachLabel` splits the datastore `imds` into two new datastores, `imdsTrain` and `imdsValidation`.

```
[imdsTrain,imdsValidation] = splitEachLabel(imds,0.7,'randomized');
```

Replace Final layers

To retrain ResNet-18 to classify new images, replace the last fully connected layer and final classification layer of the network. In ResNet-18 , these layers have the names `'fc1000'` and `'ClassificationLayer_predictions'`, respectively. The fully connected layer and classification layer of the pretrained network `net` are configured for 1000 classes. These two layers `fc1000` and `ClassificationLayer_predictions` in ResNet-18, contain information on how to combine the features that the network extracts into class probabilities and predicted labels. These two layers must be fine-tuned for the new classification problem. Extract all the layers, except the last two layers, from the pretrained network.

```
lgraph = layerGraph(net)
```

```
lgraph =
  LayerGraph with properties:

     InputNames: {'data'}
    OutputNames: {'ClassificationLayer_predictions'}
         Layers: [71×1 nnet.cnn.layer.Layer]
    Connections: [78×2 table]
```

```
numClasses = numel(categories(imdsTrain.Labels))
```

```
numClasses = 5
```

```
newLearnableLayer = fullyConnectedLayer(numClasses, ...
'Name','new_fc', ...
'WeightLearnRateFactor',10, ...
'BiasLearnRateFactor',10);
lgraph = replaceLayer(lgraph,'fc1000',newLearnableLayer);
newClassLayer = classificationLayer('Name','new_classoutput');
lgraph = replaceLayer(lgraph,'ClassificationLayer_predictions',newClassLayer);
```

Prepare Data for Training

The network requires input images of size 224-by-224-by-3, but the images in the image datastores have different sizes. Use an augmented image datastore to automatically resize the training images. Specify additional augmentation operations to perform on the training images, such as randomly flipping the training images along the vertical axis and randomly translating them up to 30 pixels horizontally and vertically. Data augmentation helps prevent the network from overfitting and memorizing the exact details of the training images.

```
pixelRange = [-30 30];
imageAugmenter = imageDataAugmenter( ...
```

```
'RandXReflection',true, ...
'RandXTranslation',pixelRange, ...
'RandYTranslation',pixelRange);
```

To automatically resize the validation images without performing further data augmentation, use an augmented image datastore without specifying any additional preprocessing operations.

```
augimdsTrain = augmentedImageDatastore(inputSize(1:2),imdsTrain, ...
'DataAugmentation',imageAugmenter);
augimdsValidation = augmentedImageDatastore(inputSize(1:2),imdsValidation);
```

Specify Training Options

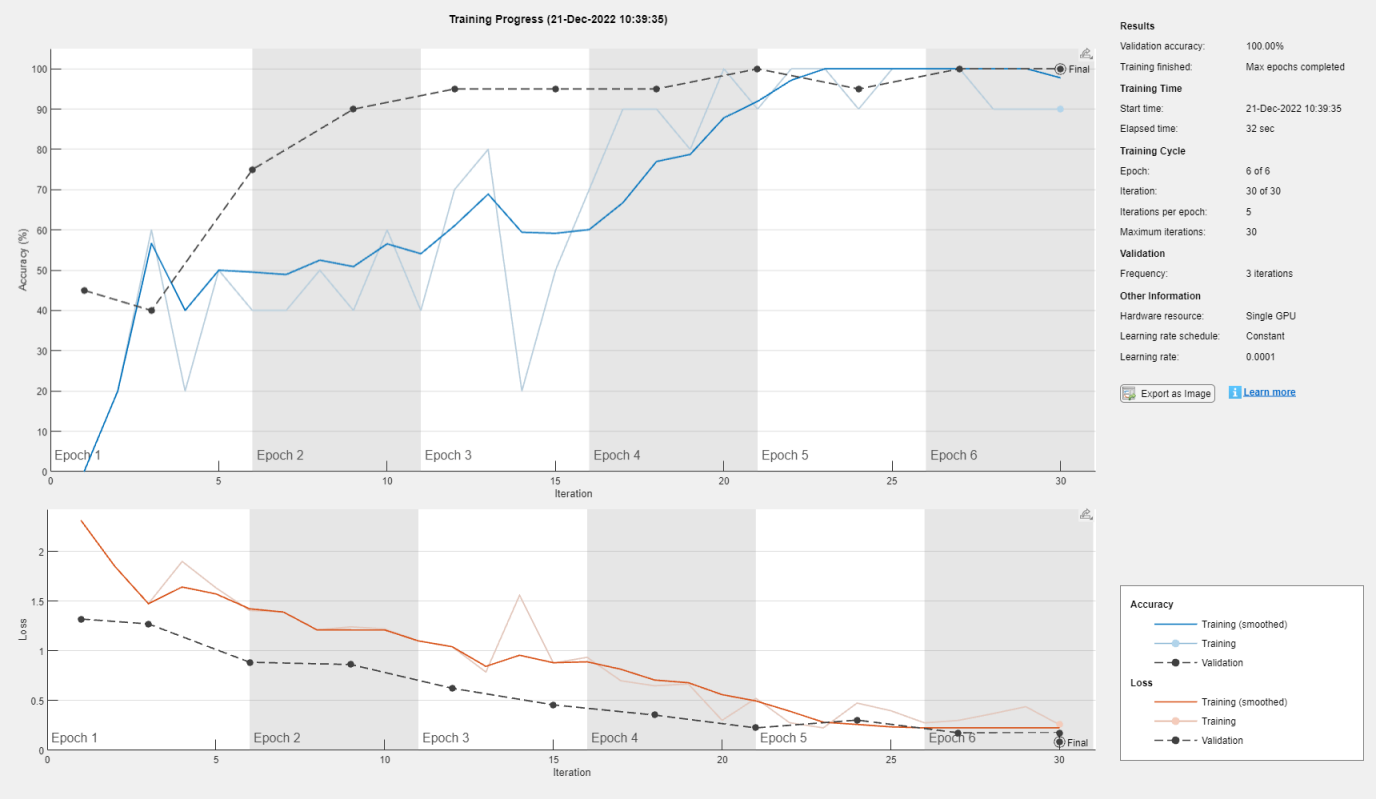
Specify the training options. For transfer learning, keep the features from the early layers of the pretrained network (the transferred layer weights). To slow down learning in the transferred layers, set the initial learning rate to a small value. Specify the mini-batch size and validation data. The software validates the network every `ValidationFrequency` iterations during training.

```
options = trainingOptions('sgdm', ...
'MiniBatchSize',10, ...
'MaxEpochs',6, ...
'InitialLearnRate',1e-4, ...
'Shuffle','every-epoch', ...
'ValidationData',augimdsValidation, ...
'ValidationFrequency',3, ...
'Verbose',false, ...
'Plots','training-progress');
```

Train Network

Train the network that consists of the transferred and new layers. By default, `trainNetwork` uses a GPU if one is available. Using this function on a GPU requires Parallel Computing Toolbox™ and a supported GPU device. For more information, see “GPU Computing Requirements” (Parallel Computing Toolbox). If a GPU is not available, the network uses a CPU (requires MATLAB Coder Interface for Deep learning). You can also specify the execution environment by using the `ExecutionEnvironment` name-value argument of `trainingOptions`.

```
netTransfer = trainNetwork(augimdsTrain,lgraph,options);
```



Quantize Network

Quantize the network using the `dlquantizer` object. Set the target execution environment to FPGA.

```
dlquantObj = dlquantizer(netTransfer,'ExecutionEnvironment','FPGA');
```

Calibrate Quantized Network

Use the `calibrate` function to exercise the network with sample inputs and collect the range information. The `calibrate` function collects the dynamic ranges of the weights and biases in the convolution and fully connected layers of the network and the dynamic ranges of the activations in all layers of the network. The function returns the information as a table, in which each row contains range information for a learnable parameter of the quantized network.

```
calibrate(dlquantObj,augimdsTrain)
```

```
ans=95×5 table
       Optimized Layer Name       Network Layer Name    Learnables / Activations    MinValue    M
    ______________________________    ________________    ________________________    ________    _

    {'conv1_Weights'         }    {'conv1'         }          "Weights"             -0.79143
    {'conv1_Bias'            }    {'conv1'         }          "Bias"                -0.66949    0
    {'res2a_branch2a_Weights'}    {'res2a_branch2a'}          "Weights"             -0.42074    0
    {'res2a_branch2a_Bias'   }    {'res2a_branch2a'}          "Bias"                 -0.8039
    {'res2a_branch2b_Weights'}    {'res2a_branch2b'}          "Weights"             -0.78524    0
    {'res2a_branch2b_Bias'   }    {'res2a_branch2b'}          "Bias"                 -1.3835
    {'res2b_branch2a_Weights'}    {'res2b_branch2a'}          "Weights"              -0.3174    0
    {'res2b_branch2a_Bias'   }    {'res2b_branch2a'}          "Bias"                 -1.1203
    {'res2b_branch2b_Weights'}    {'res2b_branch2b'}          "Weights"              -1.1915    0
```

```
    {'res2b_branch2b_Bias'   }    {'res2b_branch2b'}          "Bias"              -0.81928
    {'res3a_branch2a_Weights'}    {'res3a_branch2a'}          "Weights"           -0.19735    0
    {'res3a_branch2a_Bias'   }    {'res3a_branch2a'}          "Bias"              -0.53009    0
    {'res3a_branch2b_Weights'}    {'res3a_branch2b'}          "Weights"           -0.53557    0
    {'res3a_branch2b_Bias'   }    {'res3a_branch2b'}          "Bias"              -0.67756
    {'res3a_branch1_Weights' }    {'res3a_branch1' }          "Weights"           -0.63395    0
    {'res3a_branch1_Bias'    }    {'res3a_branch1' }          "Bias"              -0.95277    0
      ⋮
```

Define FPGA Board Interface

Define the target FPGA board programming interface by using the `dlhdl.Target` object. Create a programming interface with custom name for your target device and an Ethernet interface to connect the target device to the host computer.

```
hTarget = dlhdl.Target('Xilinx','Interface','Ethernet');
```

Prepare Network for Deployment

Prepare the network for deployment by creating a `dlhdl.Workflow` object. Specify the network and bitstream name. Ensure that the bitstream name matches the data type and the FPGA board that you are targeting. In this example, the target FPGA board is the Xilinx® Zynq® UltraScale+™ MPSoC ZCU102 board and the bitstream uses the `int8` data type.

```
hW = dlhdl.Workflow(Network=dlquantObj,Bitstream='zcu102_int8',Target=hTarget);
```

Compile Network

Run the `compile` method of the `dlhdl.Workflow` object to compile the network and generate the instructions, weights, and biases for deployment.

```
dn = compile(hW,'InputFrameNumberLimit',15)
```

```
### Compiling network for Deep Learning FPGA prototyping ...
### Targeting FPGA bitstream zcu102_int8.
### Optimizing network: Fused 'nnet.cnn.layer.BatchNormalizationLayer' into 'nnet.cnn.layer.Conv
### The network includes the following layers:
     1   'data'                  Image Input                 224×224×3 images with 'zscore' norm
     2   'conv1'                 2-D Convolution             64 7×7×3 convolutions with stride
     3   'conv1_relu'            ReLU                        ReLU
     4   'pool1'                 2-D Max Pooling             3×3 max pooling with stride [2  2]
     5   'res2a_branch2a'        2-D Convolution             64 3×3×64 convolutions with stride
     6   'res2a_branch2a_relu'   ReLU                        ReLU
     7   'res2a_branch2b'        2-D Convolution             64 3×3×64 convolutions with stride
     8   'res2a'                 Addition                    Element-wise addition of 2 inputs
     9   'res2a_relu'            ReLU                        ReLU
    10   'res2b_branch2a'        2-D Convolution             64 3×3×64 convolutions with stride
    11   'res2b_branch2a_relu'   ReLU                        ReLU
    12   'res2b_branch2b'        2-D Convolution             64 3×3×64 convolutions with stride
    13   'res2b'                 Addition                    Element-wise addition of 2 inputs
    14   'res2b_relu'            ReLU                        ReLU
    15   'res3a_branch2a'        2-D Convolution             128 3×3×64 convolutions with stride
    16   'res3a_branch2a_relu'   ReLU                        ReLU
    17   'res3a_branch2b'        2-D Convolution             128 3×3×128 convolutions with stri
    18   'res3a_branch1'         2-D Convolution             128 1×1×64 convolutions with stride
    19   'res3a'                 Addition                    Element-wise addition of 2 inputs
    20   'res3a_relu'            ReLU                        ReLU
```

```
    21   'res3b_branch2a'        2-D Convolution             128 3×3×128 convolutions with stric
    22   'res3b_branch2a_relu'   ReLU                        ReLU
    23   'res3b_branch2b'        2-D Convolution             128 3×3×128 convolutions with stric
    24   'res3b'                 Addition                    Element-wise addition of 2 inputs
    25   'res3b_relu'            ReLU                        ReLU
    26   'res4a_branch2a'        2-D Convolution             256 3×3×128 convolutions with stric
    27   'res4a_branch2a_relu'   ReLU                        ReLU
    28   'res4a_branch2b'        2-D Convolution             256 3×3×256 convolutions with stric
    29   'res4a_branch1'         2-D Convolution             256 1×1×128 convolutions with stric
    30   'res4a'                 Addition                    Element-wise addition of 2 inputs
    31   'res4a_relu'            ReLU                        ReLU
    32   'res4b_branch2a'        2-D Convolution             256 3×3×256 convolutions with stric
    33   'res4b_branch2a_relu'   ReLU                        ReLU
    34   'res4b_branch2b'        2-D Convolution             256 3×3×256 convolutions with stric
    35   'res4b'                 Addition                    Element-wise addition of 2 inputs
    36   'res4b_relu'            ReLU                        ReLU
    37   'res5a_branch2a'        2-D Convolution             512 3×3×256 convolutions with stric
    38   'res5a_branch2a_relu'   ReLU                        ReLU
    39   'res5a_branch2b'        2-D Convolution             512 3×3×512 convolutions with stric
    40   'res5a_branch1'         2-D Convolution             512 1×1×256 convolutions with stric
    41   'res5a'                 Addition                    Element-wise addition of 2 inputs
    42   'res5a_relu'            ReLU                        ReLU
    43   'res5b_branch2a'        2-D Convolution             512 3×3×512 convolutions with stric
    44   'res5b_branch2a_relu'   ReLU                        ReLU
    45   'res5b_branch2b'        2-D Convolution             512 3×3×512 convolutions with stric
    46   'res5b'                 Addition                    Element-wise addition of 2 inputs
    47   'res5b_relu'            ReLU                        ReLU
    48   'pool5'                 2-D Global Average Pooling  2-D global average pooling
    49   'new_fc'                Fully Connected             5 fully connected layer
    50   'prob'                  Softmax                     softmax
    51   'new_classoutput'       Classification Output       crossentropyex with 'MathWorks Cap

### Notice: The layer 'data' with type 'nnet.cnn.layer.ImageInputLayer' is implemented in softwa
### Notice: The layer 'prob' with type 'nnet.cnn.layer.SoftmaxLayer' is implemented in software.
### Notice: The layer 'new_classoutput' with type 'nnet.cnn.layer.ClassificationOutputLayer' is i
### Compiling layer group: conv1>>pool1 ...
### Compiling layer group: conv1>>pool1 ... complete.
### Compiling layer group: res2a_branch2a>>res2a_branch2b ...
### Compiling layer group: res2a_branch2a>>res2a_branch2b ... complete.
### Compiling layer group: res2b_branch2a>>res2b_branch2b ...
### Compiling layer group: res2b_branch2a>>res2b_branch2b ... complete.
### Compiling layer group: res3a_branch1 ...
### Compiling layer group: res3a_branch1 ... complete.
### Compiling layer group: res3a_branch2a>>res3a_branch2b ...
### Compiling layer group: res3a_branch2a>>res3a_branch2b ... complete.
### Compiling layer group: res3b_branch2a>>res3b_branch2b ...
### Compiling layer group: res3b_branch2a>>res3b_branch2b ... complete.
### Compiling layer group: res4a_branch1 ...
### Compiling layer group: res4a_branch1 ... complete.
### Compiling layer group: res4a_branch2a>>res4a_branch2b ...
### Compiling layer group: res4a_branch2a>>res4a_branch2b ... complete.
### Compiling layer group: res4b_branch2a>>res4b_branch2b ...
### Compiling layer group: res4b_branch2a>>res4b_branch2b ... complete.
### Compiling layer group: res5a_branch1 ...
### Compiling layer group: res5a_branch1 ... complete.
### Compiling layer group: res5a_branch2a>>res5a_branch2b ...
### Compiling layer group: res5a_branch2a>>res5a_branch2b ... complete.
### Compiling layer group: res5b_branch2a>>res5b_branch2b ...
```

```
### Compiling layer group: res5b_branch2a>>res5b_branch2b ... complete.
### Compiling layer group: pool5 ...
### Compiling layer group: pool5 ... complete.
### Compiling layer group: new_fc ...
### Compiling layer group: new_fc ... complete.

### Allocating external memory buffers:

          offset_name          offset_address    allocated_space
    _______________________    ______________    _______________

    "InputDataOffset"           "0x00000000"     "8.0 MB"
    "OutputResultOffset"        "0x00800000"     "4.0 MB"
    "SchedulerDataOffset"       "0x00c00000"     "4.0 MB"
    "SystemBufferOffset"        "0x01000000"     "28.0 MB"
    "InstructionDataOffset"     "0x02c00000"     "4.0 MB"
    "ConvWeightDataOffset"      "0x03000000"     "16.0 MB"
    "FCWeightDataOffset"        "0x04000000"     "4.0 MB"
    "EndOffset"                 "0x04400000"     "Total: 68.0 MB"

### Network compilation complete.

dn = struct with fields:
             weights: [1×1 struct]
        instructions: [1×1 struct]
           registers: [1×1 struct]
    syncInstructions: [1×1 struct]
        constantData: {}
             ddrInfo: [1×1 struct]
```

Program Bitstream onto FPGA and Download Network Weights

To deploy the network on the Xilinx ZCU102 hardware, run the deploy function of the `dlhdl.Workflow` object. This function uses the output of the compile function to program the FPGA board by using the programming file. It also downloads the network weights and biases. The deploy function starts programming the FPGA device, displays progress messages, and the time it takes to deploy the network.

```
deploy(hW)
```

```
### Programming FPGA Bitstream using Ethernet...
### Attempting to connect to the hardware board at 192.168.1.101...
### Connection successful
### Programming FPGA device on Xilinx SoC hardware board at 192.168.1.101...
### Copying FPGA programming files to SD card...
### Setting FPGA bitstream and devicetree for boot...
# Copying Bitstream zcu102_int8.bit to /mnt/hdlcoder_rd
# Set Bitstream to hdlcoder_rd/zcu102_int8.bit
# Copying Devicetree devicetree_dlhdl.dtb to /mnt/hdlcoder_rd
# Set Devicetree to hdlcoder_rd/devicetree_dlhdl.dtb
# Set up boot for Reference Design: 'AXI-Stream DDR Memory Access : 3-AXIM'
### Rebooting Xilinx SoC at 192.168.1.101...
### Reboot may take several seconds...
### Attempting to connect to the hardware board at 192.168.1.101...
### Connection successful
### Programming the FPGA bitstream has been completed successfully.
### Loading weights to Conv Processor.
```

```
### Conv Weights loaded. Current time is 21-Dec-2022 10:45:19
### Loading weights to FC Processor.
### FC Weights loaded. Current time is 21-Dec-2022 10:45:19
```

Test Network

Load the example image.

```
imgFile = fullfile(pwd,'MerchData','MathWorks Cube','Mathworks cube_0.jpg');
inputImg = imresize(imread(imgFile),[224 224]);
imshow(inputImg)
```

Classify the image on the FPGA by using the `predict` method of the `dlhdl.Workflow` object and display the results.

```
[prediction,speed] = predict(hW,single(inputImg),'Profile','on');
```

```
### Finished writing input activations.
### Running single input activation.


              Deep Learning Processor Profiler Performance Results

                    LastFrameLatency(cycles)   LastFrameLatency(seconds)       FramesNum      Tota
                          -------------              -------------               ---------        --
Network                     7392114                 0.02957                       1            73
    conv1                   1115165                 0.00446
    pool1                    199164                 0.00080
    res2a_branch2a           270125                 0.00108
    res2a_branch2b           269946                 0.00108
    res2a                    102255                 0.00041
    res2b_branch2a           269792                 0.00108
    res2b_branch2b           269902                 0.00108
    res2b                    102695                 0.00041
    res3a_branch1            155120                 0.00062
    res3a_branch2a           156480                 0.00063
```

```
    res3a_branch2b          244913                  0.00098
    res3a                    51456                  0.00021
    res3b_branch2a          245366                  0.00098
    res3b_branch2b          245123                  0.00098
    res3b                    51286                  0.00021
    res4a_branch1           135535                  0.00054
    res4a_branch2a          136117                  0.00054
    res4a_branch2b          238454                  0.00095
    res4a                    25602                  0.00010
    res4b_branch2a          237909                  0.00095
    res4b_branch2b          238282                  0.00095
    res4b                    26742                  0.00011
    res5a_branch1           324642                  0.00130
    res5a_branch2a          325897                  0.00130
    res5a_branch2b          623521                  0.00249
    res5a                    13881                  0.00006
    res5b_branch2a          624028                  0.00250
    res5b_branch2b          624631                  0.00250
    res5b                    13051                  0.00005
    pool5                    37083                  0.00015
    new_fc                   17764                  0.00007
 * The clock frequency of the DL processor is: 250MHz
```

```
[val,idx] = max(prediction);
dlquantObj.NetworkObject.Layers(end).ClassNames{idx}
```

```
ans =
'MathWorks Cube'
```

Performance Comparison

Compare the performance of the quantized network to the performance of the single data type network.

```
optionsFPGA = dlquantizationOptions('Bitstream','zcu102_int8','Target',hTarget);
predictionFPGA = validate(dlquantObj,imdsValidation,optionsFPGA)
```

```
### Compiling network for Deep Learning FPGA prototyping ...
### Targeting FPGA bitstream zcu102_int8.
### Optimizing network: Fused 'nnet.cnn.layer.BatchNormalizationLayer' into 'nnet.cnn.layer.Convo
### The network includes the following layers:
     1   'data'                  Image Input                 224×224×3 images with 'zscore' norm
     2   'conv1'                 2-D Convolution             64 7×7×3 convolutions with stride
     3   'conv1_relu'            ReLU                        ReLU
     4   'pool1'                 2-D Max Pooling             3×3 max pooling with stride [2  2]
     5   'res2a_branch2a'        2-D Convolution             64 3×3×64 convolutions with stride
     6   'res2a_branch2a_relu'   ReLU                        ReLU
     7   'res2a_branch2b'        2-D Convolution             64 3×3×64 convolutions with stride
     8   'res2a'                 Addition                    Element-wise addition of 2 inputs
     9   'res2a_relu'            ReLU                        ReLU
    10   'res2b_branch2a'        2-D Convolution             64 3×3×64 convolutions with stride
    11   'res2b_branch2a_relu'   ReLU                        ReLU
    12   'res2b_branch2b'        2-D Convolution             64 3×3×64 convolutions with stride
    13   'res2b'                 Addition                    Element-wise addition of 2 inputs
    14   'res2b_relu'            ReLU                        ReLU
    15   'res3a_branch2a'        2-D Convolution             128 3×3×64 convolutions with stride
    16   'res3a_branch2a_relu'   ReLU                        ReLU
    17   'res3a_branch2b'        2-D Convolution             128 3×3×128 convolutions with strid
    18   'res3a_branch1'         2-D Convolution             128 1×1×64 convolutions with stride
```

```
    19   'res3a'                 Addition                    Element-wise addition of 2 inputs
    20   'res3a_relu'            ReLU                        ReLU
    21   'res3b_branch2a'        2-D Convolution             128 3×3×128 convolutions with stric
    22   'res3b_branch2a_relu'   ReLU                        ReLU
    23   'res3b_branch2b'        2-D Convolution             128 3×3×128 convolutions with stric
    24   'res3b'                 Addition                    Element-wise addition of 2 inputs
    25   'res3b_relu'            ReLU                        ReLU
    26   'res4a_branch2a'        2-D Convolution             256 3×3×128 convolutions with stric
    27   'res4a_branch2a_relu'   ReLU                        ReLU
    28   'res4a_branch2b'        2-D Convolution             256 3×3×256 convolutions with stric
    29   'res4a_branch1'         2-D Convolution             256 1×1×128 convolutions with stric
    30   'res4a'                 Addition                    Element-wise addition of 2 inputs
    31   'res4a_relu'            ReLU                        ReLU
    32   'res4b_branch2a'        2-D Convolution             256 3×3×256 convolutions with stric
    33   'res4b_branch2a_relu'   ReLU                        ReLU
    34   'res4b_branch2b'        2-D Convolution             256 3×3×256 convolutions with stric
    35   'res4b'                 Addition                    Element-wise addition of 2 inputs
    36   'res4b_relu'            ReLU                        ReLU
    37   'res5a_branch2a'        2-D Convolution             512 3×3×256 convolutions with stric
    38   'res5a_branch2a_relu'   ReLU                        ReLU
    39   'res5a_branch2b'        2-D Convolution             512 3×3×512 convolutions with stric
    40   'res5a_branch1'         2-D Convolution             512 1×1×256 convolutions with stric
    41   'res5a'                 Addition                    Element-wise addition of 2 inputs
    42   'res5a_relu'            ReLU                        ReLU
    43   'res5b_branch2a'        2-D Convolution             512 3×3×512 convolutions with stric
    44   'res5b_branch2a_relu'   ReLU                        ReLU
    45   'res5b_branch2b'        2-D Convolution             512 3×3×512 convolutions with stric
    46   'res5b'                 Addition                    Element-wise addition of 2 inputs
    47   'res5b_relu'            ReLU                        ReLU
    48   'pool5'                 2-D Global Average Pooling  2-D global average pooling
    49   'new_fc'                Fully Connected             5 fully connected layer
    50   'prob'                  Softmax                     softmax
    51   'new_classoutput'       Classification Output       crossentropyex with 'MathWorks Cap

### Notice: The layer 'data' with type 'nnet.cnn.layer.ImageInputLayer' is implemented in softwa
### Notice: The layer 'prob' with type 'nnet.cnn.layer.SoftmaxLayer' is implemented in software.
### Notice: The layer 'new_classoutput' with type 'nnet.cnn.layer.ClassificationOutputLayer' is i
### Compiling layer group: conv1>>pool1 ...
### Compiling layer group: conv1>>pool1 ... complete.
### Compiling layer group: res2a_branch2a>>res2a_branch2b ...
### Compiling layer group: res2a_branch2a>>res2a_branch2b ... complete.
### Compiling layer group: res2b_branch2a>>res2b_branch2b ...
### Compiling layer group: res2b_branch2a>>res2b_branch2b ... complete.
### Compiling layer group: res3a_branch1 ...
### Compiling layer group: res3a_branch1 ... complete.
### Compiling layer group: res3a_branch2a>>res3a_branch2b ...
### Compiling layer group: res3a_branch2a>>res3a_branch2b ... complete.
### Compiling layer group: res3b_branch2a>>res3b_branch2b ...
### Compiling layer group: res3b_branch2a>>res3b_branch2b ... complete.
### Compiling layer group: res4a_branch1 ...
### Compiling layer group: res4a_branch1 ... complete.
### Compiling layer group: res4a_branch2a>>res4a_branch2b ...
### Compiling layer group: res4a_branch2a>>res4a_branch2b ... complete.
### Compiling layer group: res4b_branch2a>>res4b_branch2b ...
### Compiling layer group: res4b_branch2a>>res4b_branch2b ... complete.
### Compiling layer group: res5a_branch1 ...
### Compiling layer group: res5a_branch1 ... complete.
### Compiling layer group: res5a_branch2a>>res5a_branch2b ...
```

```
### Compiling layer group: res5a_branch2a>>res5a_branch2b ... complete.
### Compiling layer group: res5b_branch2a>>res5b_branch2b ...
### Compiling layer group: res5b_branch2a>>res5b_branch2b ... complete.
### Compiling layer group: pool5 ...
### Compiling layer group: pool5 ... complete.
### Compiling layer group: new_fc ...
### Compiling layer group: new_fc ... complete.

### Allocating external memory buffers:

          offset_name          offset_address    allocated_space
    ___________________________    _____________    _________________

    "InputDataOffset"           "0x00000000"     "12.0 MB"
    "OutputResultOffset"        "0x00c00000"     "4.0 MB"
    "SchedulerDataOffset"       "0x01000000"     "4.0 MB"
    "SystemBufferOffset"        "0x01400000"     "28.0 MB"
    "InstructionDataOffset"     "0x03000000"     "4.0 MB"
    "ConvWeightDataOffset"      "0x03400000"     "16.0 MB"
    "FCWeightDataOffset"        "0x04400000"     "4.0 MB"
    "EndOffset"                 "0x04800000"     "Total: 72.0 MB"

### Network compilation complete.

### FPGA bitstream programming has been skipped as the same bitstream is already loaded on the ta
### Loading weights to Conv Processor.
### Conv Weights loaded. Current time is 21-Dec-2022 10:46:36
### Loading weights to FC Processor.
### FC Weights loaded. Current time is 21-Dec-2022 10:46:36
### Finished writing input activations.
### Running single input activation.
### Finished writing input activations.
### Running single input activation.
### Finished writing input activations.
### Running single input activation.
### Finished writing input activations.
### Running single input activation.
### Finished writing input activations.
### Running single input activation.
### Finished writing input activations.
### Running single input activation.
### Finished writing input activations.
### Running single input activation.
### Finished writing input activations.
### Running single input activation.
### Finished writing input activations.
### Running single input activation.
### Finished writing input activations.
### Running single input activation.
### Finished writing input activations.
### Running single input activation.
### Finished writing input activations.
### Running single input activation.
### Finished writing input activations.
```

```
### Running single input activation.
### Finished writing input activations.
### Running single input activation.
### Finished writing input activations.
### Running single input activation.
### Finished writing input activations.
### Running single input activation.
### Finished writing input activations.
### Running single input activation.
### Finished writing input activations.
### Running single input activation.


              Deep Learning Processor Bitstream Build Info

Resource                   Utilized           Total        Percentage
------------------        ----------      ------------    ------------
LUTs (CLB/ALM)*              249703            274080           91.11
DSPs                            391              2520           15.52
Block RAM                       583               912           63.93
* LUT count represents Configurable Logic Block(CLB) utilization in Xilinx devices and Adaptive L

### Optimizing network: Fused 'nnet.cnn.layer.BatchNormalizationLayer' into 'nnet.cnn.layer.Convo
### Notice: The layer 'data' of type 'ImageInputLayer' is split into an image input layer 'data'
### The network includes the following layers:
     1   'data'                  Image Input                224×224×3 images with 'zscore' norm
     2   'conv1'                 2-D Convolution            64 7×7×3 convolutions with stride
     3   'conv1_relu'            ReLU                       ReLU
     4   'pool1'                 2-D Max Pooling            3×3 max pooling with stride [2  2]
     5   'res2a_branch2a'        2-D Convolution            64 3×3×64 convolutions with stride
     6   'res2a_branch2a_relu'   ReLU                       ReLU
     7   'res2a_branch2b'        2-D Convolution            64 3×3×64 convolutions with stride
     8   'res2a'                 Addition                   Element-wise addition of 2 inputs
     9   'res2a_relu'            ReLU                       ReLU
    10   'res2b_branch2a'        2-D Convolution            64 3×3×64 convolutions with stride
    11   'res2b_branch2a_relu'   ReLU                       ReLU
    12   'res2b_branch2b'        2-D Convolution            64 3×3×64 convolutions with stride
    13   'res2b'                 Addition                   Element-wise addition of 2 inputs
    14   'res2b_relu'            ReLU                       ReLU
    15   'res3a_branch2a'        2-D Convolution            128 3×3×64 convolutions with stride
    16   'res3a_branch2a_relu'   ReLU                       ReLU
    17   'res3a_branch2b'        2-D Convolution            128 3×3×128 convolutions with strid
    18   'res3a_branch1'         2-D Convolution            128 1×1×64 convolutions with stride
    19   'res3a'                 Addition                   Element-wise addition of 2 inputs
    20   'res3a_relu'            ReLU                       ReLU
    21   'res3b_branch2a'        2-D Convolution            128 3×3×128 convolutions with strid
    22   'res3b_branch2a_relu'   ReLU                       ReLU
    23   'res3b_branch2b'        2-D Convolution            128 3×3×128 convolutions with strid
    24   'res3b'                 Addition                   Element-wise addition of 2 inputs
    25   'res3b_relu'            ReLU                       ReLU
    26   'res4a_branch2a'        2-D Convolution            256 3×3×128 convolutions with strid
    27   'res4a_branch2a_relu'   ReLU                       ReLU
    28   'res4a_branch2b'        2-D Convolution            256 3×3×256 convolutions with strid
    29   'res4a_branch1'         2-D Convolution            256 1×1×128 convolutions with strid
    30   'res4a'                 Addition                   Element-wise addition of 2 inputs
    31   'res4a_relu'            ReLU                       ReLU
    32   'res4b_branch2a'        2-D Convolution            256 3×3×256 convolutions with strid
    33   'res4b_branch2a_relu'   ReLU                       ReLU
```

```
    34   'res4b_branch2b'        2-D Convolution             256 3×3×256 convolutions with stric
    35   'res4b'                 Addition                    Element-wise addition of 2 inputs
    36   'res4b_relu'            ReLU                        ReLU
    37   'res5a_branch2a'        2-D Convolution             512 3×3×256 convolutions with stric
    38   'res5a_branch2a_relu'   ReLU                        ReLU
    39   'res5a_branch2b'        2-D Convolution             512 3×3×512 convolutions with stric
    40   'res5a_branch1'         2-D Convolution             512 1×1×256 convolutions with stric
    41   'res5a'                 Addition                    Element-wise addition of 2 inputs
    42   'res5a_relu'            ReLU                        ReLU
    43   'res5b_branch2a'        2-D Convolution             512 3×3×512 convolutions with stric
    44   'res5b_branch2a_relu'   ReLU                        ReLU
    45   'res5b_branch2b'        2-D Convolution             512 3×3×512 convolutions with stric
    46   'res5b'                 Addition                    Element-wise addition of 2 inputs
    47   'res5b_relu'            ReLU                        ReLU
    48   'pool5'                 2-D Global Average Pooling   2-D global average pooling
    49   'new_fc'                Fully Connected             5 fully connected layer
    50   'prob'                  Softmax                     softmax
    51   'new_classoutput'       Classification Output       crossentropyex with 'MathWorks Cap

### Notice: The layer 'prob' with type 'nnet.cnn.layer.SoftmaxLayer' is implemented in software.
### Notice: The layer 'new_classoutput' with type 'nnet.cnn.layer.ClassificationOutputLayer' is i


             Deep Learning Processor Estimator Performance Results

                   LastFrameLatency(cycles)   LastFrameLatency(seconds)       FramesNum      Tota
                         -------------              -------------               ---------        --
Network                   23502752                    0.10683                     1           235
    data_norm_add           210750                    0.00096
    data_norm               210750                    0.00096
    conv1                  2164124                    0.00984
    pool1                   515064                    0.00234
    res2a_branch2a          966221                    0.00439
    res2a_branch2b          966221                    0.00439
    res2a                   210750                    0.00096
    res2b_branch2a          966221                    0.00439
    res2b_branch2b          966221                    0.00439
    res2b                   210750                    0.00096
    res3a_branch1           540861                    0.00246
    res3a_branch2a          540749                    0.00246
    res3a_branch2b          919117                    0.00418
    res3a                   105404                    0.00048
    res3b_branch2a          919117                    0.00418
    res3b_branch2b          919117                    0.00418
    res3b                   105404                    0.00048
    res4a_branch1           503405                    0.00229
    res4a_branch2a          509261                    0.00231
    res4a_branch2b          905421                    0.00412
    res4a                    52724                    0.00024
    res4b_branch2a          905421                    0.00412
    res4b_branch2b          905421                    0.00412
    res4b                    52724                    0.00024
    res5a_branch1          1039437                    0.00472
    res5a_branch2a         1046605                    0.00476
    res5a_branch2b         2005197                    0.00911
    res5a                    26368                    0.00012
    res5b_branch2a         2005197                    0.00911
    res5b_branch2b         2005197                    0.00911
```

```
    res5b                    26368                  0.00012
    pool5                    54594                  0.00025
    new_fc                   22571                  0.00010
 * The clock frequency of the DL processor is: 220MHz



              Deep Learning Processor Bitstream Build Info

Resource                   Utilized           Total        Percentage
------------------        ----------      ------------    ------------
LUTs (CLB/ALM)*              168099            274080           61.33
DSPs                            807              2520           32.02
Block RAM                       453               912           49.67
* LUT count represents Configurable Logic Block(CLB) utilization in Xilinx devices and Adaptive L

### Finished writing input activations.
### Running single input activation.

predictionFPGA = struct with fields:
       NumSamples: 20
    MetricResults: [1×1 struct]
       Statistics: [2×7 table]
```

View the frames per second performance for the quantized network and single-data-type network. The quantized network has a performance of 33.8 frames per second compared to 9.2 frames per second for the single-data-type network. You can use quantization to improve your frames per second performance, however yo could lose accuracy when you quantize your networks.

```
predictionFPGA.Statistics.FramesPerSecond
```

```
ans = 2×1

    9.3606
   33.7719
```

Detect Objects Using YOLO v3 Network Deployed to FPGA

This example shows how to deploy a trained you only look once (YOLO) v3 object detector to a target FPGA board. You then use MATLAB to retrieve the object classification from the FPGA board.

Compared to YOLO v2 networks, YOLO v3 networks have additional detection heads that help detect smaller objects.

Create YOLO v3 Detector Object

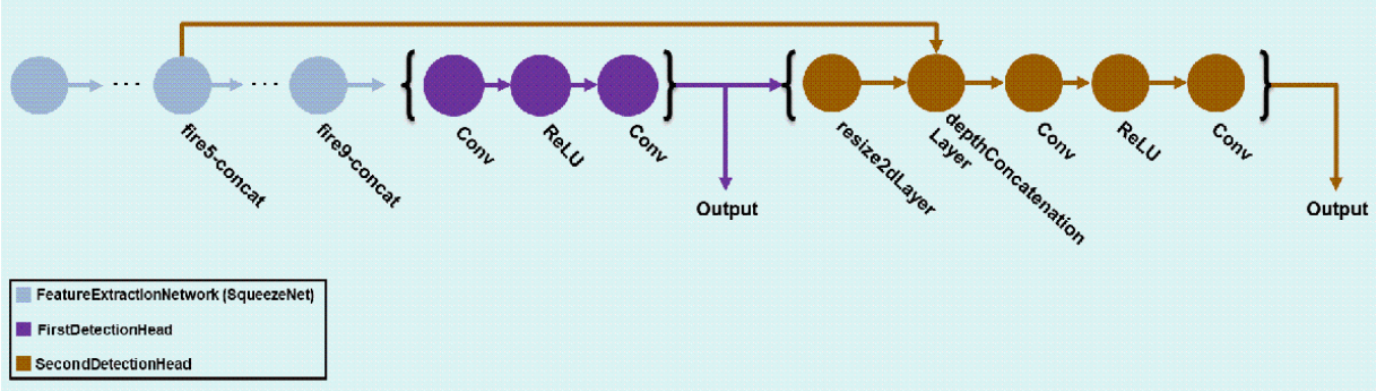
In this example, you use a pretrained YOLO v3 object detector. To construct and train a custom YOLO v3 detector, see “Object Detection Using YOLO v3 Deep Learning” (Computer Vision Toolbox).

Use the `downloadPretrainedYOLOv3Detector` function to generate a `dlnetwork` object. To get the code for this function, see the downloadPretrainedYOLOv3Detector Function on page 1-94 section.

```
preTrainedDetector = downloadPretrainedYOLOv3Detector;
```

```
Downloaded pretrained detector
```

The generated network uses training data to estimate the anchor boxes, which help the detector learn to predict the boxes. For more information about anchor boxes, see “Anchor Boxes for Object Detection” (Computer Vision Toolbox). The `downloadPretrainedYOLOv3Detector` function creates this YOLO v3 network:



Load the Pretrained network

Extract the network from the pretrained YOLO v3 detector object.

```
yolov3Detector = preTrainedDetector;
net = yolov3Detector.Network;
```

Extract the attributes of the network as variables.

```
anchorBoxes = yolov3Detector.AnchorBoxes;
outputNames = yolov3Detector.Network.OutputNames;
inputSize = yolov3Detector.InputSize;
classNames = yolov3Detector.ClassNames;
```

Use the `analyzeNetwork` function to obtain information about the network layers. the function returns a graphical representation of the network that contains detailed parameter information for every layer in the network.

```
analyzeNetwork(net);
```

Define FPGA Board Interface

Define the target FPGA board programming interface by using the `dlhdl.Target` object. Create a programming interface with custom name for your target device and an Ethernet interface to connect the target device to the host computer.

```
hTarget = dlhdl.Target('Xilinx','Interface','Ethernet');
```

Prepare Network for Deployment

Prepare the network for deployment by creating a `dlhdl.Workflow` object. Specify the network and bitstream name. Ensure that the bitstream name matches the data type and the FPGA board that you are targeting. In this example, the target FPGA board is the Xilinx® Zynq® UltraScale+™ MPSoC ZCU102 board and the bitstream uses the single data type.

```
hW = dlhdl.Workflow('Network',net,'Bitstream','zcu102_single','Target',hTarget);
```

Compile Network

Run the `compile` method of the `dlhdl.Workflow` object to compile the network and generate the instructions, weights, and biases for deployment.

```
dn = compile(hW);
```

```
### Compiling network for Deep Learning FPGA prototyping ...
### Targeting FPGA bitstream zcu102_single.
### An output layer called 'Output1_customOutputConv1' of type 'nnet.cnn.layer.RegressionOutputLa
### An output layer called 'Output2_customOutputConv2' of type 'nnet.cnn.layer.RegressionOutputLa
### Optimizing network: Fused 'nnet.cnn.layer.BatchNormalizationLayer' into 'nnet.cnn.layer.Convo
### The network includes the following layers:
     1   'data'                       Image Input                   227×227×3 images
     2   'conv1'                      2-D Convolution               64 3×3×3 convolutions with
     3   'relu_conv1'                 ReLU                          ReLU
     4   'pool1'                      2-D Max Pooling               3×3 max pooling with stride
     5   'fire2-squeeze1x1'           2-D Convolution               16 1×1×64 convolutions with
     6   'fire2-relu_squeeze1x1'      ReLU                          ReLU
     7   'fire2-expand1x1'            2-D Convolution               64 1×1×16 convolutions with
     8   'fire2-relu_expand1x1'       ReLU                          ReLU
     9   'fire2-expand3x3'            2-D Convolution               64 3×3×16 convolutions with
    10   'fire2-relu_expand3x3'       ReLU                          ReLU
    11   'fire2-concat'               Depth concatenation           Depth concatenation of 2 in
    12   'fire3-squeeze1x1'           2-D Convolution               16 1×1×128 convolutions wit
    13   'fire3-relu_squeeze1x1'      ReLU                          ReLU
    14   'fire3-expand1x1'            2-D Convolution               64 1×1×16 convolutions with
    15   'fire3-relu_expand1x1'       ReLU                          ReLU
    16   'fire3-expand3x3'            2-D Convolution               64 3×3×16 convolutions with
    17   'fire3-relu_expand3x3'       ReLU                          ReLU
    18   'fire3-concat'               Depth concatenation           Depth concatenation of 2 in
    19   'pool3'                      2-D Max Pooling               3×3 max pooling with stride
    20   'fire4-squeeze1x1'           2-D Convolution               32 1×1×128 convolutions wit
    21   'fire4-relu_squeeze1x1'      ReLU                          ReLU
    22   'fire4-expand1x1'            2-D Convolution               128 1×1×32 convolutions wit
    23   'fire4-relu_expand1x1'       ReLU                          ReLU
    24   'fire4-expand3x3'            2-D Convolution               128 3×3×32 convolutions wit
    25   'fire4-relu_expand3x3'       ReLU                          ReLU
    26   'fire4-concat'               Depth concatenation           Depth concatenation of 2 in
    27   'fire5-squeeze1x1'           2-D Convolution               32 1×1×256 convolutions wit
    28   'fire5-relu_squeeze1x1'      ReLU                          ReLU
    29   'fire5-expand1x1'            2-D Convolution               128 1×1×32 convolutions wit
    30   'fire5-relu_expand1x1'       ReLU                          ReLU
    31   'fire5-expand3x3'            2-D Convolution               128 3×3×32 convolutions wit
    32   'fire5-relu_expand3x3'       ReLU                          ReLU
    33   'fire5-concat'               Depth concatenation           Depth concatenation of 2 in
    34   'pool5'                      2-D Max Pooling               3×3 max pooling with stride
    35   'fire6-squeeze1x1'           2-D Convolution               48 1×1×256 convolutions wit
    36   'fire6-relu_squeeze1x1'      ReLU                          ReLU
    37   'fire6-expand1x1'            2-D Convolution               192 1×1×48 convolutions wit
    38   'fire6-relu_expand1x1'       ReLU                          ReLU
    39   'fire6-expand3x3'            2-D Convolution               192 3×3×48 convolutions wit
    40   'fire6-relu_expand3x3'       ReLU                          ReLU
    41   'fire6-concat'               Depth concatenation           Depth concatenation of 2 in
    42   'fire7-squeeze1x1'           2-D Convolution               48 1×1×384 convolutions wit
    43   'fire7-relu_squeeze1x1'      ReLU                          ReLU
```

```
    44   'fire7-expand1x1'             2-D Convolution               192 1×1×48 convolutions wit
    45   'fire7-relu_expand1x1'        ReLU                          ReLU
    46   'fire7-expand3x3'             2-D Convolution               192 3×3×48 convolutions wit
    47   'fire7-relu_expand3x3'        ReLU                          ReLU
    48   'fire7-concat'                Depth concatenation           Depth concatenation of 2 in
    49   'fire8-squeeze1x1'            2-D Convolution               64 1×1×384 convolutions wit
    50   'fire8-relu_squeeze1x1'       ReLU                          ReLU
    51   'fire8-expand1x1'             2-D Convolution               256 1×1×64 convolutions wit
    52   'fire8-relu_expand1x1'        ReLU                          ReLU
    53   'fire8-expand3x3'             2-D Convolution               256 3×3×64 convolutions wit
    54   'fire8-relu_expand3x3'        ReLU                          ReLU
    55   'fire8-concat'                Depth concatenation           Depth concatenation of 2 in
    56   'fire9-squeeze1x1'            2-D Convolution               64 1×1×512 convolutions wit
    57   'fire9-relu_squeeze1x1'       ReLU                          ReLU
    58   'fire9-expand1x1'             2-D Convolution               256 1×1×64 convolutions wit
    59   'fire9-relu_expand1x1'        ReLU                          ReLU
    60   'fire9-expand3x3'             2-D Convolution               256 3×3×64 convolutions wit
    61   'fire9-relu_expand3x3'        ReLU                          ReLU
    62   'fire9-concat'                Depth concatenation           Depth concatenation of 2 in
    63   'customConv1'                 2-D Convolution               1024 3×3×512 convolutions w
    64   'customRelu1'                 ReLU                          ReLU
    65   'customOutputConv1'           2-D Convolution               18 1×1×1024 convolutions wi
    66   'featureConv2'                2-D Convolution               128 1×1×512 convolutions wi
    67   'featureRelu2'                ReLU                          ReLU
    68   'Output1_customOutputConv1'   Regression Output             mean-squared-error
    69   'featureResize2'              dnnfpga.custom.Resize2DLayer  dnnfpga.custom.Resize2DLaye
    70   'depthConcat2'                Depth concatenation           Depth concatenation of 2 in
    71   'customConv2'                 2-D Convolution               256 3×3×384 convolutions wi
    72   'customRelu2'                 ReLU                          ReLU
    73   'customOutputConv2'           2-D Convolution               18 1×1×256 convolutions wit
    74   'Output2_customOutputConv2'   Regression Output             mean-squared-error

### Notice: The layer 'data' with type 'nnet.cnn.layer.ImageInputLayer' is implemented in softwar
### Notice: The layer 'Output1_customOutputConv1' with type 'nnet.cnn.layer.RegressionOutputLayer
### Notice: The layer 'Output2_customOutputConv2' with type 'nnet.cnn.layer.RegressionOutputLayer
### Compiling layer group: conv1>>fire2-relu_squeeze1x1 ...
### Compiling layer group: conv1>>fire2-relu_squeeze1x1 ... complete.
### Compiling layer group: fire2-expand1x1>>fire2-relu_expand1x1 ...
### Compiling layer group: fire2-expand1x1>>fire2-relu_expand1x1 ... complete.
### Compiling layer group: fire2-expand3x3>>fire2-relu_expand3x3 ...
### Compiling layer group: fire2-expand3x3>>fire2-relu_expand3x3 ... complete.
### Compiling layer group: fire3-squeeze1x1>>fire3-relu_squeeze1x1 ...
### Compiling layer group: fire3-squeeze1x1>>fire3-relu_squeeze1x1 ... complete.
### Compiling layer group: fire3-expand1x1>>fire3-relu_expand1x1 ...
### Compiling layer group: fire3-expand1x1>>fire3-relu_expand1x1 ... complete.
### Compiling layer group: fire3-expand3x3>>fire3-relu_expand3x3 ...
### Compiling layer group: fire3-expand3x3>>fire3-relu_expand3x3 ... complete.
### Compiling layer group: pool3>>fire4-relu_squeeze1x1 ...
### Compiling layer group: pool3>>fire4-relu_squeeze1x1 ... complete.
### Compiling layer group: fire4-expand1x1>>fire4-relu_expand1x1 ...
### Compiling layer group: fire4-expand1x1>>fire4-relu_expand1x1 ... complete.
### Compiling layer group: fire4-expand3x3>>fire4-relu_expand3x3 ...
### Compiling layer group: fire4-expand3x3>>fire4-relu_expand3x3 ... complete.
### Compiling layer group: fire5-squeeze1x1>>fire5-relu_squeeze1x1 ...
### Compiling layer group: fire5-squeeze1x1>>fire5-relu_squeeze1x1 ... complete.
### Compiling layer group: fire5-expand1x1>>fire5-relu_expand1x1 ...
### Compiling layer group: fire5-expand1x1>>fire5-relu_expand1x1 ... complete.
### Compiling layer group: fire5-expand3x3>>fire5-relu_expand3x3 ...
```

```
### Compiling layer group: fire5-expand3x3>>fire5-relu_expand3x3 ... complete.
### Compiling layer group: pool5>>fire6-relu_squeeze1x1 ...
### Compiling layer group: pool5>>fire6-relu_squeeze1x1 ... complete.
### Compiling layer group: fire6-expand1x1>>fire6-relu_expand1x1 ...
### Compiling layer group: fire6-expand1x1>>fire6-relu_expand1x1 ... complete.
### Compiling layer group: fire6-expand3x3>>fire6-relu_expand3x3 ...
### Compiling layer group: fire6-expand3x3>>fire6-relu_expand3x3 ... complete.
### Compiling layer group: fire7-squeeze1x1>>fire7-relu_squeeze1x1 ...
### Compiling layer group: fire7-squeeze1x1>>fire7-relu_squeeze1x1 ... complete.
### Compiling layer group: fire7-expand1x1>>fire7-relu_expand1x1 ...
### Compiling layer group: fire7-expand1x1>>fire7-relu_expand1x1 ... complete.
### Compiling layer group: fire7-expand3x3>>fire7-relu_expand3x3 ...
### Compiling layer group: fire7-expand3x3>>fire7-relu_expand3x3 ... complete.
### Compiling layer group: fire8-squeeze1x1>>fire8-relu_squeeze1x1 ...
### Compiling layer group: fire8-squeeze1x1>>fire8-relu_squeeze1x1 ... complete.
### Compiling layer group: fire8-expand1x1>>fire8-relu_expand1x1 ...
### Compiling layer group: fire8-expand1x1>>fire8-relu_expand1x1 ... complete.
### Compiling layer group: fire8-expand3x3>>fire8-relu_expand3x3 ...
### Compiling layer group: fire8-expand3x3>>fire8-relu_expand3x3 ... complete.
### Compiling layer group: fire9-squeeze1x1>>fire9-relu_squeeze1x1 ...
### Compiling layer group: fire9-squeeze1x1>>fire9-relu_squeeze1x1 ... complete.
### Compiling layer group: fire9-expand1x1>>fire9-relu_expand1x1 ...
### Compiling layer group: fire9-expand1x1>>fire9-relu_expand1x1 ... complete.
### Compiling layer group: fire9-expand3x3>>fire9-relu_expand3x3 ...
### Compiling layer group: fire9-expand3x3>>fire9-relu_expand3x3 ... complete.
### Compiling layer group: customConv1>>customOutputConv1 ...
### Compiling layer group: customConv1>>customOutputConv1 ... complete.
### Compiling layer group: featureConv2>>featureRelu2 ...
### Compiling layer group: featureConv2>>featureRelu2 ... complete.
### Compiling layer group: customConv2>>customOutputConv2 ...
### Compiling layer group: customConv2>>customOutputConv2 ... complete.

### Allocating external memory buffers:

          offset_name          offset_address     allocated_space
    ___________________________    ______________    _________________

    "InputDataOffset"             "0x00000000"     "24.0 MB"
    "OutputResultOffset"          "0x01800000"     "4.0 MB"
    "SchedulerDataOffset"         "0x01c00000"     "4.0 MB"
    "SystemBufferOffset"          "0x02000000"     "28.0 MB"
    "InstructionDataOffset"       "0x03c00000"     "8.0 MB"
    "ConvWeightDataOffset"        "0x04400000"     "104.0 MB"
    "EndOffset"                   "0x0ac00000"     "Total: 172.0 MB"

### Network compilation complete.
```

Program Bitstream onto FPGA and Download Network Weights

To deploy the network on the Xilinx® Zynq® UltraScale+ MPSoC ZCU102 hardware, run the `deploy` method of the `dlhdl.Workflow` object. This method programs the FPGA board using the output of the compile method and the programming file, downloads the network weights and biases, displays progress messages, and the time it takes to deploy the network.

```
deploy(hW);
```

```
### Programming FPGA Bitstream using Ethernet...
### Attempting to connect to the hardware board at 192.168.1.101...
```

```
### Connection successful
### Programming FPGA device on Xilinx SoC hardware board at 192.168.1.101...
### Copying FPGA programming files to SD card...
### Setting FPGA bitstream and devicetree for boot...
# Copying Bitstream zcu102_single.bit to /mnt/hdlcoder_rd
# Set Bitstream to hdlcoder_rd/zcu102_single.bit
# Copying Devicetree devicetree_dlhdl.dtb to /mnt/hdlcoder_rd
# Set Devicetree to hdlcoder_rd/devicetree_dlhdl.dtb
# Set up boot for Reference Design: 'AXI-Stream DDR Memory Access : 3-AXIM'
### Rebooting Xilinx SoC at 192.168.1.101...
### Reboot may take several seconds...
### Attempting to connect to the hardware board at 192.168.1.101...
### Connection successful
### Programming the FPGA bitstream has been completed successfully.
### Loading weights to Conv Processor.
### Conv Weights loaded. Current time is 27-Oct-2022 13:44:50
```

Test Network

Load the example image and convert the image into a `dlarray`. Then classify the image on the FPGA by using the `predict` method of the `dlhdl.Workflow` object and display the results.

```
img = imread('vehicle_image.jpg');
I = single(rescale(img));
I = imresize(I, yolov3Detector.InputSize(1:2));
dlX = dlarray(I,'SSC');
```

Store the output of each detection head of the network in the `features` variable. Pass `features` to the post-processing function `processYOLOv3Ouputs` to combine the multiple outputs and compute the final results. To get the code for this function, see the processYOLOv3Output Function on page 1-95 section.

```
features = cell(size(net.OutputNames'));
[features{:}] = hW.predict(dlX, 'Profiler', 'on');
```

```
### Finished writing input activations.
### Running single input activation.


              Deep Learning Processor Profiler Performance Results

                    LastFrameLatency(cycles)   LastFrameLatency(seconds)       FramesNum      Tota
                          -------------              -------------               ---------        --
Network                    34469645                 0.15668                       1          344
    conv1                    673148                 0.00306
    pool1                    509022                 0.00231
    fire2-squeeze1x1         308280                 0.00140
    fire2-expand1x1          305546                 0.00139
    fire2-expand3x3          305227                 0.00139
    fire3-squeeze1x1         628018                 0.00285
    fire3-expand1x1          305219                 0.00139
    fire3-expand3x3          305220                 0.00139
    pool3                    286781                 0.00130
    fire4-squeeze1x1         264346                 0.00120
    fire4-expand1x1          264777                 0.00120
    fire4-expand3x3          264750                 0.00120
    fire5-squeeze1x1         749166                 0.00341
    fire5-expand1x1          264800                 0.00120
```

```
    fire5-expand3x3         264880                  0.00120
    pool5                   219686                  0.00100
    fire6-squeeze1x1        195193                  0.00089
    fire6-expand1x1         145091                  0.00066
    fire6-expand3x3         145075                  0.00066
    fire7-squeeze1x1        290001                  0.00132
    fire7-expand1x1         144830                  0.00066
    fire7-expand3x3         145390                  0.00066
    fire8-squeeze1x1        369605                  0.00168
    fire8-expand1x1         245085                  0.00111
    fire8-expand3x3         245208                  0.00111
    fire9-squeeze1x1        490784                  0.00223
    fire9-expand1x1         244864                  0.00111
    fire9-expand3x3         245458                  0.00112
    customConv1           17592876                  0.07997
    customOutputConv1       952889                  0.00433
    featureConv2            913457                  0.00415
    featureResize2           57819                  0.00026
    customConv2             5600648                 0.02546
    customOutputConv2       526143                  0.00239
 * The clock frequency of the DL processor is: 220MHz
```

```
[bboxes, scores, labels] = processYOLOv3Output(anchorBoxes, inputSize, classNames, features, I);
resultImage = insertObjectAnnotation(I,'rectangle',bboxes,scores);
imshow(resultImage)
```

The FPGA returns a score prediction of `0.89605` with a bounding box drawn around the object in the image. The FPGA also returns a prediction of vehicle to the `labels` variable.

`downloadPretrainedYOLOv3Detector` Function

The `downloadPretrainedYOLOv3Detector` function to download the pretrained YOLO v3 detector network

```
function detector = downloadPretrainedYOLOv3Detector
if ~exist('yolov3SqueezeNetVehicleExample_21aSPKG.mat', 'file')
```

```
    if ~exist('yolov3SqueezeNetVehicleExample_21aSPKG.zip', 'file')
        zipFile = matlab.internal.examples.downloadSupportFile('vision/data', 'yolov3SqueezeNetVe
        copyfile(zipFile);
    end
    unzip('yolov3SqueezeNetVehicleExample_21aSPKG.zip');
end
pretrained = load("yolov3SqueezeNetVehicleExample_21aSPKG.mat");
detector = pretrained.detector;
disp('Downloaded pretrained detector');
end
```

`processYOLOv3Output` Function

The `processYOLOv3Output` function is attached as a helper file in this example's directory. This function converts the feature maps from multiple detection heads to bounding boxes, scores and labels. A code snippet of the function is shown below.

```
function [bboxes, scores, labels] = processYOLOv3Output(anchorBoxes, inputSize, classNames, featu
% This function converts the feature maps from multiple detection heads to bounding boxes, scores
% processYOLOv3Output is C code generatable

% Breaks down the raw output from predict function into Confidence score, X, Y, Width,
% Height and Class probabilities for each output from detection head
predictions = iYolov3Transform(features, anchorBoxes);

% Initialize parameters for post-processing
inputSize2d = inputSize(1:2);
info.PreprocessedImageSize = inputSize2d(1:2);
info.ScaleX = size(img,1)/inputSize2d(1);
info.ScaleY = size(img,2)/inputSize2d(1);
params.MinSize = [1 1];
params.MaxSize = size(img(:,:,1));
params.Threshold = 0.5;
params.FractionDownsampling = 1;
params.DetectionInputWasBatchOfImages = false;
params.NetworkInputSize = inputSize;
params.DetectionPreprocessing = "none";
params.SelectStrongest = 1;
bboxes = [];
scores = [];
labels = [];

% Post-process the predictions to get bounding boxes, scores and labels
[bboxes, scores, labels] = iPostprocessMultipleDetection(anchorBoxes, inputSize, classNames, prec
end

function [bboxes, scores, labels] = iPostprocessMultipleDetection (anchorBoxes, inputSize, classN
% Post-process the predictions to get bounding boxes, scores and labels

% YpredData is a (x,8) cell array, where x = number of detection heads
% Information in each column is:
% column 1 -> confidence scores
% column 2 to column 5 -> X offset, Y offset, Width, Height of anchor boxes
% column 6 -> class probabilities
% column 7-8 -> copy of width and height of anchor boxes

% Initialize parameters for post-processing
classes = classNames;
```

```
predictions = YPredData;
extractPredictions = cell(size(predictions));
% Extract dlarray data
for i = 1:size(extractPredictions,1)
    for j = 1:size(extractPredictions,2)
        extractPredictions{i,j} = extractdata(predictions{i,j});
    end
end

% Storing the values of columns 2 to 5 of extractPredictions
% Columns 2 to 5 represent information about X-coordinate, Y-coordinate, Width and Height of pred
extractedCoordinates = cell(size(predictions,1),4);
for i = 1:size(predictions,1)
    for j = 2:5
        extractedCoordinates{i,j-1} = extractPredictions{i,j};
    end
end

% Convert predictions from grid cell coordinates to box coordinates.
boxCoordinates = anchorBoxGenerator(anchorBoxes, inputSize, classNames, extractedCoordinates, par
% Replace grid cell coordinates in extractPredictions with box coordinates
for i = 1:size(YPredData,1)
    for j = 2:5
        extractPredictions{i,j} = single(boxCoordinates{i,j-1});
    end
end

% 1. Convert bboxes from spatial to pixel dimension
% 2. Combine the prediction from different heads.
% 3. Filter detections based on threshold.

% Reshaping the matrices corresponding to confidence scores and  bounding boxes
detections = cell(size(YPredData,1),6);
for i = 1:size(detections,1)
    for j = 1:5
        detections{i,j} = reshapePredictions(extractPredictions{i,j});
    end
end
% Reshaping the matrices corresponding to class probablities
numClasses = repmat({numel(classes)},[size(detections,1),1]);
for i = 1:size(detections,1)
    detections{i,6} = reshapeClasses(extractPredictions{i,6},numClasses{i,1});
end

% cell2mat converts the cell of matrices into one matrix, this combines the
% predictions of all detection heads
detections = cell2mat(detections);

% Getting the most probable class and corresponding index
[classProbs, classIdx] = max(detections(:,6:end),[],2);
detections(:,1) = detections(:,1).*classProbs;
detections(:,6) = classIdx;

% Keep detections whose confidence score is greater than threshold.
detections = detections(detections(:,1) >= params.Threshold,:);

[bboxes, scores, labels] = iPostProcessDetections(detections, classes, info, params);
end
```

```
function [bboxes, scores, labels] = iPostProcessDetections(detections, classes, info, params)
% Resizes the anchor boxes, filters anchor boxes based on size and apply
% NMS to eliminate overlapping anchor boxes
if ~isempty(detections)

    % Obtain bounding boxes and class data for pre-processed image
    scorePred = detections(:,1);
    bboxesTmp = detections(:,2:5);
    classPred = detections(:,6);
    inputImageSize = ones(1,2);
    inputImageSize(2) = info.ScaleX.*info.PreprocessedImageSize(2);
    inputImageSize(1) = info.ScaleY.*info.PreprocessedImageSize(1);
    % Resize boxes to actual image size.
    scale = [inputImageSize(2) inputImageSize(1) inputImageSize(2) inputImageSize(1)];
    bboxPred = bboxesTmp.*scale;
    % Convert x and y position of detections from centre to top-left.
    bboxPred = iConvertCenterToTopLeft(bboxPred);

    % Filter boxes based on MinSize, MaxSize.
    [bboxPred, scorePred, classPred] = filterBBoxes(params.MinSize, params.MaxSize, bboxPred, sc

    % Apply NMS to eliminate boxes having significant overlap
    if params.SelectStrongest
        [bboxes, scores, classNames] = selectStrongestBboxMulticlass(bboxPred, scorePred, classP
            'RatioType', 'Union', 'OverlapThreshold', 0.4);
    else
        bboxes = bboxPred;
        scores = scorePred;
        classNames = classPred;
    end

    % Limit width detections
    detectionsWd = min((bboxes(:,1) + bboxes(:,3)),inputImageSize(1,2));
    bboxes(:,3) = detectionsWd(:,1) - bboxes(:,1);

    % Limit height detections
    detectionsHt = min((bboxes(:,2) + bboxes(:,4)),inputImageSize(1,1));
    bboxes(:,4) = detectionsHt(:,1) - bboxes(:,2);
    bboxes(bboxes<1) = 1;

    % Convert classId to classNames.
    labels = categorical(classes,cellstr(classes));
    labels = labels(classNames);

else
    % If detections are empty then bounding boxes, scores and labels should
    % be empty
    bboxes = zeros(0,4,'single');
    scores = zeros(0,1,'single');
    labels = categorical(classes);
end
end

function x = reshapePredictions(pred)
% Reshapes the matrices corresponding to scores, X, Y, Width and Height to
% make them compatible for combining the outputs of different detection
% heads
```

```
[h,w,c,n] = size(pred);
x = reshape(pred,h*w*c,1,n);
end

function x = reshapeClasses(pred,numClasses)
% Reshapes the matrices corresponding to the class probabilities, to make it
% compatible for combining the outputs of different detection heads
[h,w,c,n] = size(pred);
numAnchors = c/numClasses;
x = reshape(pred,h*w,numClasses,numAnchors,n);
x = permute(x,[1,3,2,4]);
[h,w,c,n] = size(x);
x = reshape(x,h*w,c,n);
end

function bboxes = iConvertCenterToTopLeft(bboxes)
% Convert x and y position of detections from centre to top-left.
bboxes(:,1) = bboxes(:,1) - bboxes(:,3)/2 + 0.5;
bboxes(:,2) = bboxes(:,2) - bboxes(:,4)/2 + 0.5;
bboxes = floor(bboxes);
bboxes(bboxes<1) = 1;
end

function tiledAnchors = anchorBoxGenerator(anchorBoxes, inputSize, classNames,YPredCell,inputImag
% Convert grid cell coordinates to box coordinates.
% Generate tiled anchor offset.
tiledAnchors = cell(size(YPredCell));
for i = 1:size(YPredCell,1)
    anchors = anchorBoxes{i,:};
    [h,w,~,n] = size(YPredCell{i,1});
    [tiledAnchors{i,2},tiledAnchors{i,1}] = ndgrid(0:h-1,0:w-1,1:size(anchors,1),1:n);
    [~,~,tiledAnchors{i,3}] = ndgrid(0:h-1,0:w-1,anchors(:,2),1:n);
    [~,~,tiledAnchors{i,4}] = ndgrid(0:h-1,0:w-1,anchors(:,1),1:n);
end

for i = 1:size(YPredCell,1)
    [h,w,~,~] = size(YPredCell{i,1});
    tiledAnchors{i,1} = double((tiledAnchors{i,1} + YPredCell{i,1})./w);
    tiledAnchors{i,2} = double((tiledAnchors{i,2} + YPredCell{i,2})./h);
    tiledAnchors{i,3} = double((tiledAnchors{i,3}.*YPredCell{i,3})./inputImageSize(2));
    tiledAnchors{i,4} = double((tiledAnchors{i,4}.*YPredCell{i,4})./inputImageSize(1));
end
end

function predictions = iYolov3Transform(YPredictions, anchorBoxes)
% This function breaks down the raw output from predict function into Confidence score, X, Y, Wi
% Height and Class probabilities for each output from detection head

predictions = cell(size(YPredictions,1),size(YPredictions,2) + 2);

for idx = 1:size(YPredictions,1)
    % Get the required info on feature size.
    numChannelsPred = size(YPredictions{idx},3);  %number of channels in a feature map
    numAnchors = size(anchorBoxes{idx},1);    %number of anchor boxes per grid
    numPredElemsPerAnchors = numChannelsPred/numAnchors;
    channelsPredIdx = 1:numChannelsPred;
    predictionIdx = ones([1,numAnchors.*5]);
```

```
        % X positions.
        startIdx = 1;
        endIdx = numChannelsPred;
        stride = numPredElemsPerAnchors;
        predictions{idx,2} = YPredictions{idx}(:,:,startIdx:stride:endIdx,:);
        predictionIdx = [predictionIdx startIdx:stride:endIdx];

        % Y positions.
        startIdx = 2;
        endIdx = numChannelsPred;
        stride = numPredElemsPerAnchors;
        predictions{idx,3} = YPredictions{idx}(:,:,startIdx:stride:endIdx,:);
        predictionIdx = [predictionIdx startIdx:stride:endIdx];

        % Width.
        startIdx = 3;
        endIdx = numChannelsPred;
        stride = numPredElemsPerAnchors;
        predictions{idx,4} = YPredictions{idx}(:,:,startIdx:stride:endIdx,:);
        predictionIdx = [predictionIdx startIdx:stride:endIdx];

        % Height.
        startIdx = 4;
        endIdx = numChannelsPred;
        stride = numPredElemsPerAnchors;
        predictions{idx,5} = YPredictions{idx}(:,:,startIdx:stride:endIdx,:);
        predictionIdx = [predictionIdx startIdx:stride:endIdx];

        % Confidence scores.
        startIdx = 5;
        endIdx = numChannelsPred;
        stride = numPredElemsPerAnchors;
        predictions{idx,1} = YPredictions{idx}(:,:,startIdx:stride:endIdx,:);
        predictionIdx = [predictionIdx startIdx:stride:endIdx];

        % Class probabilities.
        classIdx = setdiff(channelsPredIdx,predictionIdx);
        predictions{idx,6} = YPredictions{idx}(:,:,classIdx,:);
    end

    for i = 1:size(predictions,1)
        predictions{i,7} = predictions{i,4};
        predictions{i,8} = predictions{i,5};
    end

    % Apply activation to the predicted cell array
    % Apply sigmoid activation to columns 1-3 (Confidence score, X, Y)
    for i = 1:size(predictions,1)
        for j = 1:3
            predictions{i,j} = sigmoid(predictions{i,j});
        end
    end
    % Apply exponentiation to columns 4-5 (Width, Height)
    for i = 1:size(predictions,1)
        for j = 4:5
            predictions{i,j} = exp(predictions{i,j});
        end
    end
```

```
% Apply sigmoid activation to column 6 (Class probabilities)
for i = 1:size(predictions,1)
    for j = 6
        predictions{i,j} = sigmoid(predictions{i,j});
    end
end
end

function [bboxPred, scorePred, classPred] = filterBBoxes(minSize, maxSize, bboxPred, scorePred, c
% Filter boxes based on MinSize, MaxSize
[bboxPred, scorePred, classPred] = filterSmallBBoxes(minSize, bboxPred, scorePred, classPred);
[bboxPred, scorePred, classPred] = filterLargeBBoxes(maxSize, bboxPred, scorePred, classPred);
end

function varargout = filterSmallBBoxes(minSize, varargin)
% Filter boxes based on MinSize
bboxes = varargin{1};
tooSmall = any((bboxes(:,[4 3]) < minSize),2);
for ii = 1:numel(varargin)
    varargout{ii} = varargin{ii}(~tooSmall,:);
end
end

function varargout = filterLargeBBoxes(maxSize, varargin)
% Filter boxes based on MaxSize
bboxes = varargin{1};
tooBig = any((bboxes(:,[4 3]) > maxSize),2);
for ii = 1:numel(varargin)
    varargout{ii} = varargin{ii}(~tooBig,:);
end
end

function m = cell2mat(c)
% Converts the cell of matrices into one matrix by concatenating
% the output corresponding to each feature map

elements = numel(c);
% If number of elements is 0 return an empty array
if elements == 0
    m = [];
    return
end
% If number of elements is 1, return same element as matrix
if elements == 1
    if isnumeric(c{1}) || ischar(c{1}) || islogical(c{1}) || isstruct(c{1})
        m = c{1};
        return
    end
end
% Error out for unsupported cell content
ciscell = iscell(c{1});
cisobj = isobject(c{1});
if cisobj || ciscell
    disp('CELL2MAT does not support cell arrays containing cell arrays or objects.');
end
% If input input is struct, extract field names of structure into a cell
if isstruct(c{1})
    cfields = cell(elements,1);
```

```
    for n = 1:elements
        cfields{n} = fieldnames(c{n});
    end
    if ~isequal(cfields{:})
        disp('The field names of each cell array element must be consistent and in consistent orc
    end
end
% If number of dimensions is 2
if ndims(c) == 2
    rows = size(c,1);
    cols = size(c,2);
    if (rows < cols)
        % If rows is less than columns first concatenate each column into 1
        % row then concatenate all the rows
        m = cell(rows,1);
        for n = 1:rows
            m{n} = cat(2,c{n,:});
        end
        m = cat(1,m{:});
    else
        % If columns is less than rows, first concatenate each corresponding
        % row into columns, then combine all columns into 1
        m = cell(1,cols);
        for n = 1:cols
            m{n} = cat(1,c{:,n});
        end
        m = cat(2,m{:});
    end
    return
end
end
```

References

[1] Redmon, Joseph, and Ali Farhadi. “YOLOv3: An Incremental Improvement.” Preprint, submitted April 8, 2018. https://arxiv.org/abs/1804.02767.

Version History

Introduced in R2020b

See Also

`compile` | `deploy` | `getBuildInfo` | `dlquantizer` | `dlquantizationOptions` | `calibrate` | `validate` | `predictAndUpdateState` | `resetState`

Topics

“Profile Inference Run”
“Profile Network to Determine Performance Bottlenecks”

predictAndUpdateState

Class: `dlhdl.Workflow`
Package: `dlhdl`

Predict responses by using a trained and deployed recurrent neural network and update the deployed network state

Syntax

```
Y = predictAndUpdateState(workflowObject,sequences)
[Y,performance] = predictAndUpdateState(workflowObject,sequences,Name,Value)
```

Description

`Y = predictAndUpdateState(workflowObject,sequences)` predicts responses for data in `sequences` using the deployed network and updates the network state. The method does not initialize the network state before running. This method supports recurrent neural networks only. The specified network must have at least one recurrent layer, such as an LSTM layer or a custom layer with state parameters.

`[Y,performance] = predictAndUpdateState(workflowObject,sequences,Name,Value)` predicts responses and updates the network state with one or more arguments specified by optional name-value pair arguments.

Input Arguments

`workflowObject` — Workflow

`dlhdl.Workflow` object

Workflow, specified as a `dlhdl.Workflow` object.

`sequences` — Sequence or time series data

numeric array

For numeric array input, the dimensions of the numeric arrays containing the sequences depend on the type of data.

Input	Description
Vector sequences	*c*-by-*s* matrices, where *c* is the number of features of the sequences and *s* is the sequence length.
1-D image sequences	*h*-by-*c*-by-*s* arrays, where *h* and *c* correspond to the height and number of channels of the images, respectively, and *s* is the sequence length.

Input	Description
2-D image sequences	*h*-by-*w*-by-*c*-by-*s* arrays, where *h*, *w*, and *c* correspond to the height, width, and number of channels of the images, respectively, and *s* is the sequence length.
3-D image sequences	*h*-by-*w*-by-*d*-by-*c*-by-*s*, where *h*, *w*, *d*, and *c* correspond to the height, width, depth, and number of channels of the 3-D images, respectively, and *s* is the sequence length.

The dimensions of the sequence data must correspond to the table.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example:

`Profile` — Flag that returns profiling results
"off" (default) | "on"

Flag to return profiling results, for the deep learning network deployed to the target board, specified as `"off"` or `"on"`.

Example: `"Profile", "On"`

Output Arguments

Y — Predicted responses
numeric array

Predicted responses, returned as a numeric array. The format of `Y` depends on the type of task.

This table describes the format for regression problems.

Task	Format
2-D image regression	• *h*-by-*w*-by-*c*-by-*N* numeric array, where *h*, *w*, and *c* are the height, width, and number of channels of the images, respectively, and *N* is the number of images
3-D image regression	• *h*-by-*w*-by-*d*-by-*c*-by-*N* numeric array, where *h*, *w*, *d*, and *c* are the height, width, depth, and number of channels of the images, respectively, and *N* is the number of images
Sequence-to-one regression	*N*-by-*R* matrix, where *N* is the number of sequences and *R* is the number of responses

Task	Format
Sequence-to-sequence regression	*N*-by-*R* matrix, where *N* is the number of sequences and *R* is the number of responses
Feature regression	*N*-by-*R* matrix, where *N* is the number of observations and *R* is the number of responses

For sequence-to-sequence regression problems with one observation, `sequences` can be a matrix. In this case, `Y` is a matrix of responses.

If the output layer of the network is a classification layer, then `Y` is the predicted classification scores. This table describes the format of the scores for classification tasks.

Task	Format
Image classification	*N*-by-*K* matrix, where *N* is the number of observations and *K* is the number of classes
Sequence-to-label classification	
Feature classification	

`performance` — Deployed network performance data
table

Deployed network performance data, returned as an *N*-by-5 table, where *N* is the number of layers in the network. This method returns performance only when the `Profile` name-value argument is set to `'on'`. To learn about the data in the performance table, see “Profile Inference Run”.

Examples

Run Sequence Forecasting on FPGA by Using Deep Learning HDL Toolbox

This example shows how to create, compile, and deploy a long short-term memory (LSTM) network trained on waveform data by using the Deep Learning HDL Toolbox™ Support Package for Xilinx FPGA and SoC. Use the deployed network to predict future values by using open-loop and closed-loop forecasting. Use MATLAB® to retrieve the prediction results from the target device.

Waveform Data Network

The network attached to this example was trained using the “Time Series Forecasting Using Deep Learning”. This example uses the `WaveformData.mat` data set, which contains 2000 synthetically generated waveforms of varying lengths with three channels. This example uses a trained LSTM network to forecast future values of the waveforms given the values from the previous time steps using both closed loop and open loop forecasting.

Prerequisites

- Xilinx® Zynq® Ultrascale+™ ZCU102 SoC development kit
- Deep Learning HDL Toolbox™ Support Package for Xilinx FPGA and SoC
- Deep Learning Toolbox™
- Deep Learning HDL Toolbox™

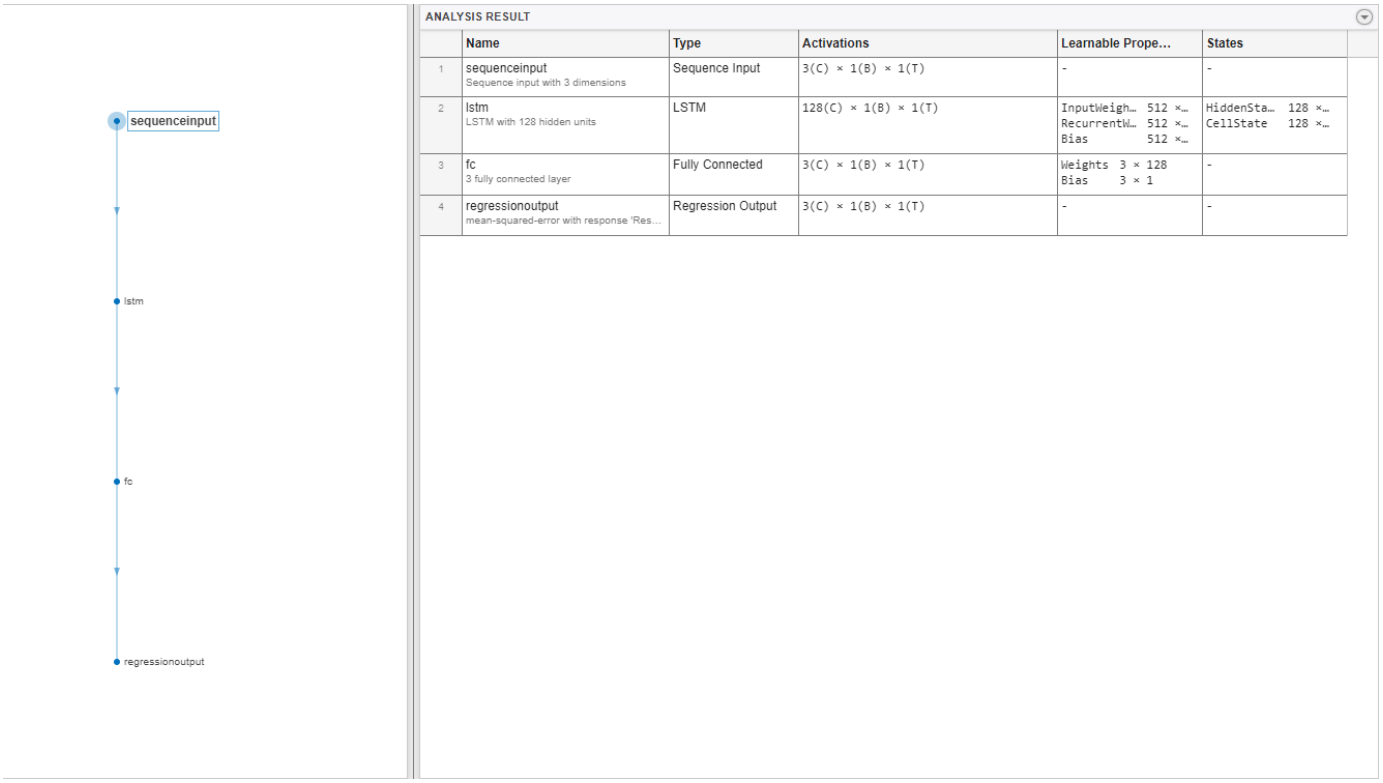
Load the Pretrained Network

To load the LSTM network enter:

```
load WaveformForcastingNet
```

Use the `analyzeNetwork` function to obtain information about the network layers. the function returns a graphical representation of the network that contains detailed parameter information for every layer in the network.

```
analyzeNetwork(net)
```



Define FPGA Board Interface

Define the target FPGA board programming interface by using the `dlhdl.Target` object. Specify that the interface is for a Xilinx board with an Ethernet interface.

To create the target object, enter:

```
hTarget = dlhdl.Target('Xilinx','Interface','Ethernet');
```

To use the JTAG interface, install Xilinx™ Vivado™ Design Suite 2020.2. To set the Xilinx Vivado toolpath, enter:

```
hdlsetuptoolpath('ToolName', 'Xilinx Vivado', 'ToolPath', 'C:\Xilinx\Vivado\2020.2\bin\vivado.ba
hTarget = dlhdl.Target('Xilinx','Interface','JTAG');
```

Prepare Network for Deployment

Prepare the network for deployment by creating a `dlhdl.Workflow` object. Specify the network and the bitstream name. Ensure that the bitstream name matches the data type and the FPGA board. In

this example the target FPGA board is the Xilinx ZCU102 SOC board. The bitstream uses a single data type.

```
hW = dlhdl.Workflow('network', net, 'Bitstream', 'zcu102_lstm_single','Target',hTarget);
```

Tu run the example on the Xilinx ZC706 board, enter:

```
hW = dlhdl.Workflow('Network', snet, 'Bitstream', 'zc706_lstm_single','Target',hTarget);
```

Compile the LSTM Network

Run the `compile` method of the `dlhdl.Workflow` object to compile the network and generate the instructions, weights, and biases for deployment. The total number of frames exceeds the default value of 30. Set the `InputFrameNumberLimit` name-value argument to `1000` to run predictions in chunks of 1000 frames to prevent timeouts.

```
dn = compile(hW,'InputFrameNumberLimit',1000)
```

```
### Compiling network for Deep Learning FPGA prototyping ...
### Targeting FPGA bitstream zcu102_lstm_single.
### The network includes the following layers:
     1   'sequenceinput'      Sequence Input      Sequence input with 3 dimensions            (S
     2   'lstm'               LSTM                LSTM with 128 hidden units                  (H
     3   'fc'                 Fully Connected     3 fully connected layer                     (H
     4   'regressionoutput'   Regression Output   mean-squared-error with response 'Response'  (S

### Notice: The layer 'sequenceinput' with type 'nnet.cnn.layer.ImageInputLayer' is implemented i
### Notice: The layer 'regressionoutput' with type 'nnet.cnn.layer.RegressionOutputLayer' is imp
### Compiling layer group: lstm.wi ...
### Compiling layer group: lstm.wi ... complete.
### Compiling layer group: lstm.wo ...
### Compiling layer group: lstm.wo ... complete.
### Compiling layer group: lstm.wg ...
### Compiling layer group: lstm.wg ... complete.
### Compiling layer group: lstm.wf ...
### Compiling layer group: lstm.wf ... complete.
### Compiling layer group: fc ...
### Compiling layer group: fc ... complete.

### Allocating external memory buffers:

          offset_name          offset_address    allocated_space
    _______________________    ______________    _______________

    "InputDataOffset"           "0x00000000"     "4.0 MB"
    "OutputResultOffset"        "0x00400000"     "4.0 MB"
    "SchedulerDataOffset"       "0x00800000"     "4.0 MB"
    "SystemBufferOffset"        "0x00c00000"     "20.0 MB"
    "InstructionDataOffset"     "0x02000000"     "4.0 MB"
    "FCWeightDataOffset"        "0x02400000"     "4.0 MB"
    "EndOffset"                 "0x02800000"     "Total: 40.0 MB"

### Network compilation complete.

dn = struct with fields:
             weights: [1×1 struct]
        instructions: [1×1 struct]
           registers: [1×1 struct]
```

```
    syncInstructions: [1×1 struct]
        constantData: {}
```

Program Bitstream onto FPGA and Download Network Weights

To deploy the network on the Xilinx ZCU102 SoC hardware, run the `deploy` function of the `dlhdl.Workflow` object. This function uses the output of the `compile` function to program the FPGA board by using the programming file. It also downloads the network weights and biases. The `deploy` function starts programming the FPGA device and displays progress messages, and the required time to deploy the network.

```
deploy(hW)
```

```
### FPGA bitstream programming has been skipped as the same bitstream is already loaded on the ta
### Resetting network state.
### Loading weights to FC Processor.
### FC Weights loaded. Current time is 09-Nov-2022 09:35:06
```

Test Network

Prepare the test data for prediction. Normalize the test data using the statistics calculated from the training data. To forecast the values of future time steps of a sequence, specify the targets as the test sequences with values shifted by one time step. In other words, at each time step of the input sequence, the LSTM network learns to predict the value of the next time step. The predictors as the test sequences without the final time step.

```
load Waveformdata
numChannels = size(data{1},1);
numObservations = numel(data);

idxTrain = 1:floor(0.9*numObservations);
idxTest = floor(0.9*numObservations)+1:numObservations;
dataTrain = data(idxTrain);
dataTest = data(idxTest);

for n = 1:numel(dataTrain)
    X = dataTrain{n};
    XTrain{n} = X(:,1:end-1);
    TTrain{n} = X(:,2:end);
end

muX = mean(cat(2,XTrain{:}),2);
sigmaX = std(cat(2,XTrain{:}),0,2);
muT = mean(cat(2,TTrain{:}),2);
sigmaT = std(cat(2,TTrain{:}),0,2);

for n = 1:size(dataTest,1)
    X = dataTest{n};
    XTest{n} = (X(:,1:end-1) - muX) ./ sigmaX;
    TTest{n} = (X(:,2:end) - muT) ./ sigmaT;
end
```

Make predictions using the test data.

```
YTest = hW.predict(XTest{1},Profile ='on');
```

```
### Resetting network state.
### Finished writing input activations.
```

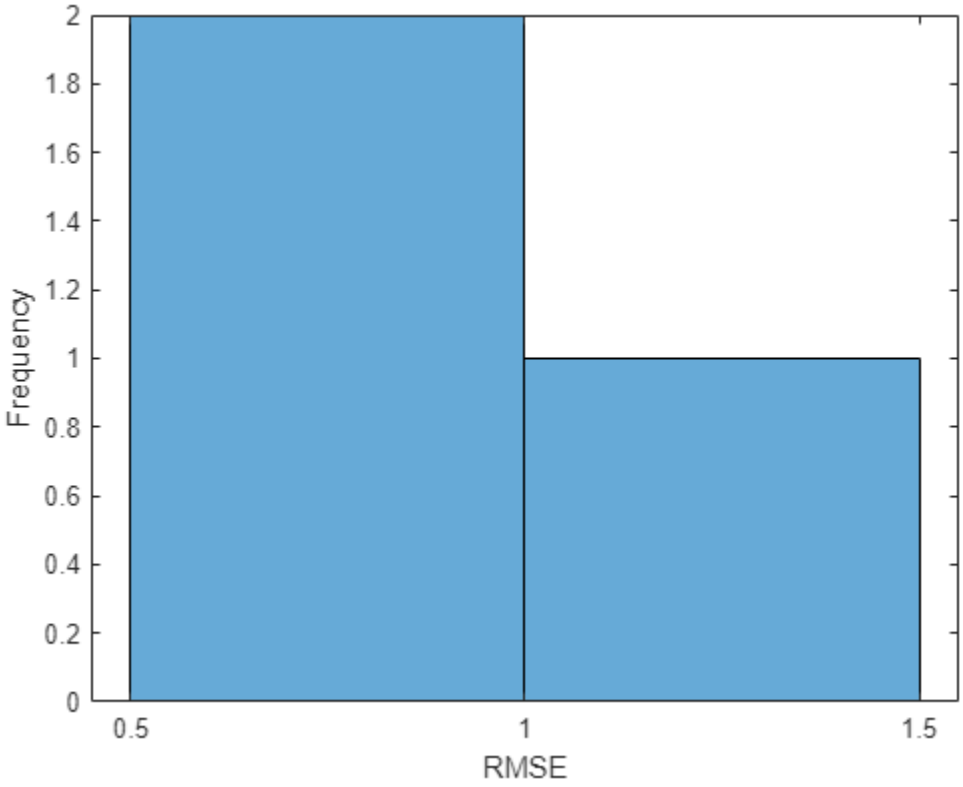
```
### Running a sequence of length 115.


              Deep Learning Processor Profiler Performance Results

                     LastFrameLatency(cycles)   LastFrameLatency(seconds)       FramesNum      Tota
                           -------------              -------------               ---------        --
Network                         38755                 0.00018                     115            44
    memSeparator_0                88                  0.00000
    lstm.wi                     7478                  0.00003
    lstm.wo                     7549                  0.00003
    lstm.wg                     7619                  0.00003
    lstm.wf                     7519                  0.00003
    lstm.sigmoid_1               222                  0.00000
    lstm.sigmoid_3               224                  0.00000
    lstm.tanh_1                  204                  0.00000
    lstm.sigmoid_2               224                  0.00000
    lstm.multiplication_2         294                  0.00000
    lstm.multiplication_1         314                  0.00000
    lstm.c_add                   308                  0.00000
    lstm.tanh_2                  229                  0.00000
    lstm.multiplication_3         287                  0.00000
    fc                          6196                  0.00003
 * The clock frequency of the DL processor is: 220MHz
```

To evaluate the accuracy, calculate the root mean squared error (RMSE) between the predictions and the target for each test sequence.

```
for i = 1:size(YTest,1)
    rmse(i) = sqrt(mean((YTest(i) - TTest{1}(i)).^2,"all"));
end
```

Visualize the errors in a histogram. Lower values indicate greater accuracy.

```
figure
histogram(rmse)
xlabel("RMSE")
ylabel("Frequency")
```



Calculate the mean RMSE over all test observations.

```
mean(rmse)
```

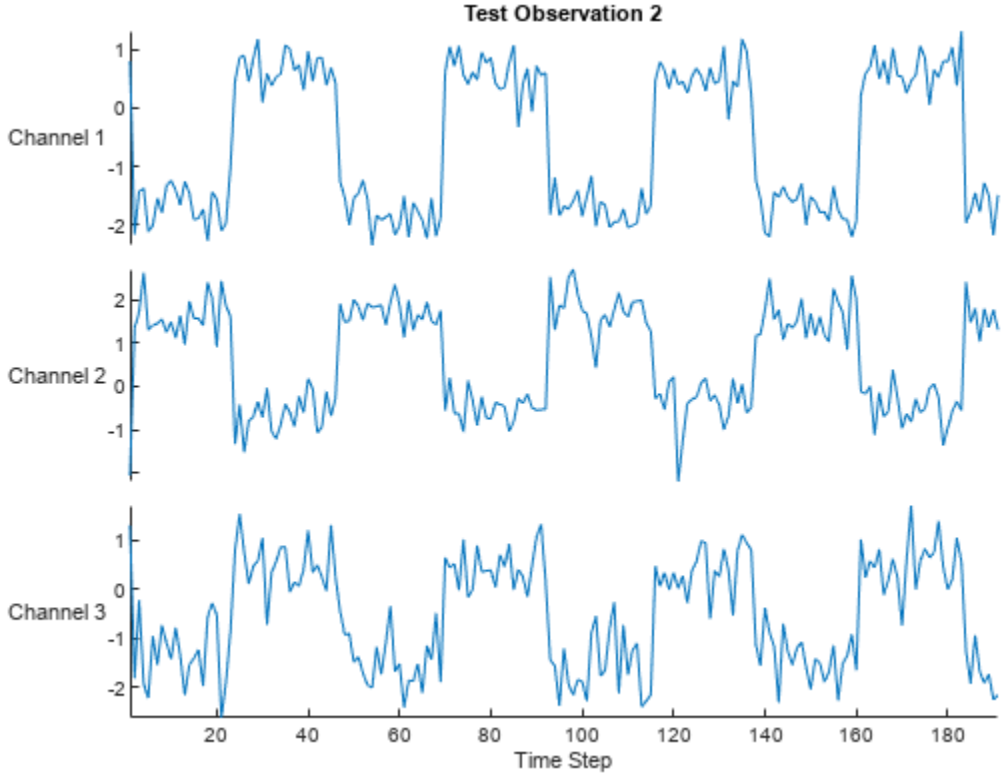
```
ans = single
    0.8385
```

Forecast Future Time Steps

To forecast the values of multiple future time steps, when given an input time series or sequence, use the `predictAndUpdateState` function. This function predicts time steps one at a time and updates the network state at each prediction. For each prediction, use the previous prediction as the input to the function.

Visualize one of the test sequences in a plot.

```
idx = 2;
X = XTest{idx};
T = TTest{idx};

figure
stackedplot(X',DisplayLabels="Channel " + (1:numChannels))
xlabel("Time Step")
title("Test Observation " + idx)
```



Open-Loop Forecasting

Open-loop forecasting predicts the next time step in a sequence using only the input data. When making predictions for subsequent time steps, you collect the true values form your data source and use those as input. For example, suppose that you want to predict the value for time step t of a sequence by using data collected in time steps 1 through $t - 1$. To make predictions for time step $t + 1$, wait until you record the true value for time step t and use that value as input to make the next prediction. Use open-loop forecasting when you have true values to provide to the network before making the next prediction.

Initialize the network state by resetting the state using the `resetState` function, then make an initial prediction using the first few time steps of the input data. Update the network state by using the first 75 time steps of the input data.

```
resetState(hW)
offset = 75;
[~,~] = hW.predictAndUpdateState(X(:,1:offset));
```

```
### Resetting network state.
### Finished writing input activations.
### Running a sequence of length 75.
```

To forecast further predictions, loop over time steps and update the network state by using the `predictAndUpdateState` function. Forecast values for the remaining time steps of the test observation by looping over the time steps of the input data and using them as input to the network. The first prediction is the value that corresponds to the time step `offset + 1`.

```
numTimeSteps = size(X,2);
numPredictionTimeSteps = numTimeSteps - offset;
Y = zeros(numChannels,numPredictionTimeSteps);

for t = 1:numPredictionTimeSteps
    Xt = X(:,offset+t);
    Y(:,t) = predictAndUpdateState(hW,Xt);
end
```

```
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
```

```
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
```

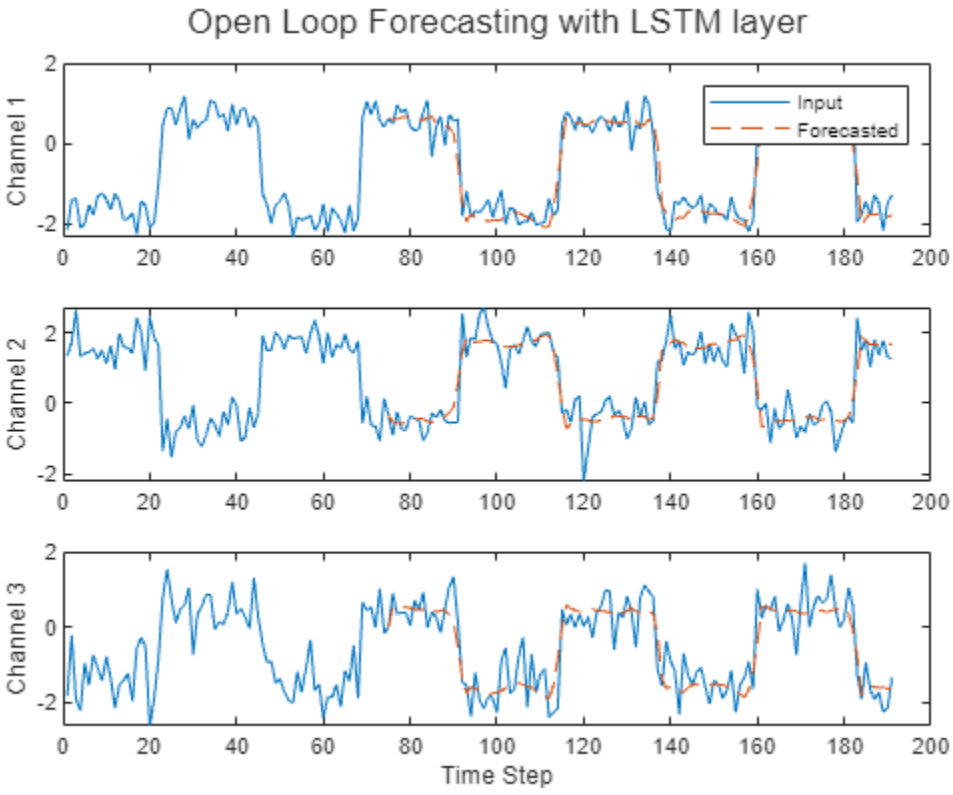
```
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
```

```
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
```

```
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
```

Compare the predictions with the target values.

```
figure
t = tiledlayout(numChannels,1);
title(t,"Open Loop Forecasting with LSTM layer")

for i = 1:numChannels
    nexttile
    plot(T(i,:))
    hold on
    plot(offset:numTimeSteps,[T(i,offset) Y(i,:)],'--')
    ylabel("Channel " + i)
end

xlabel("Time Step")
nexttile(1)
legend(["Input" "Forecasted"])
```



Closed-Loop Forecasting

Closed-loop forecasting predicts subsequent time steps in a sequence by using the previous predictions as input. In this case, the model does not require the true values to make the prediction. For example, suppose that you want to predict the value for time steps t through $t + k$ of the sequence by using data collected in time steps 1 through $t - 1$. To make predictions for time step i, use the predicted value for time step $i - 1$ as input. Use closed-loop forecasting to forecast multiple subsequent time steps or when you do not have true values to provide to the network before making the next prediction.

Initialize the network state by resetting the state using the `resetState` function, then make an initial prediction, `Z`, using the first few time steps of the input data. Update the network state by using the first 75 time steps of the input data.

```
resetState(hW)
offset = size(X,2);
[Z, ~] = predictAndUpdateState(hW,X);
```

```
### Resetting network state.
### Finished writing input activations.
### Running a sequence of length 191.
```

To forecast further predictions, loop over time steps and update the network state by using the `predictAndUpdateState` function. Forecast the next 200 time steps by iteratively passing the previously predicted value to the network. Because the network does not require the input data to make any further predictions, you can specify any number of time steps to forecast.

```
numPredictionTimeSteps = 200;
Xt = Z(:,end);
Y = zeros(numChannels,numPredictionTimeSteps);

for t = 1:numPredictionTimeSteps
    [Y(:,t),~] =  predictAndUpdateState(hW,Xt);
    Xt = Y(:,t);
end
```

```
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
```

```
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
```

```
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
```

```
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
```

```
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
```

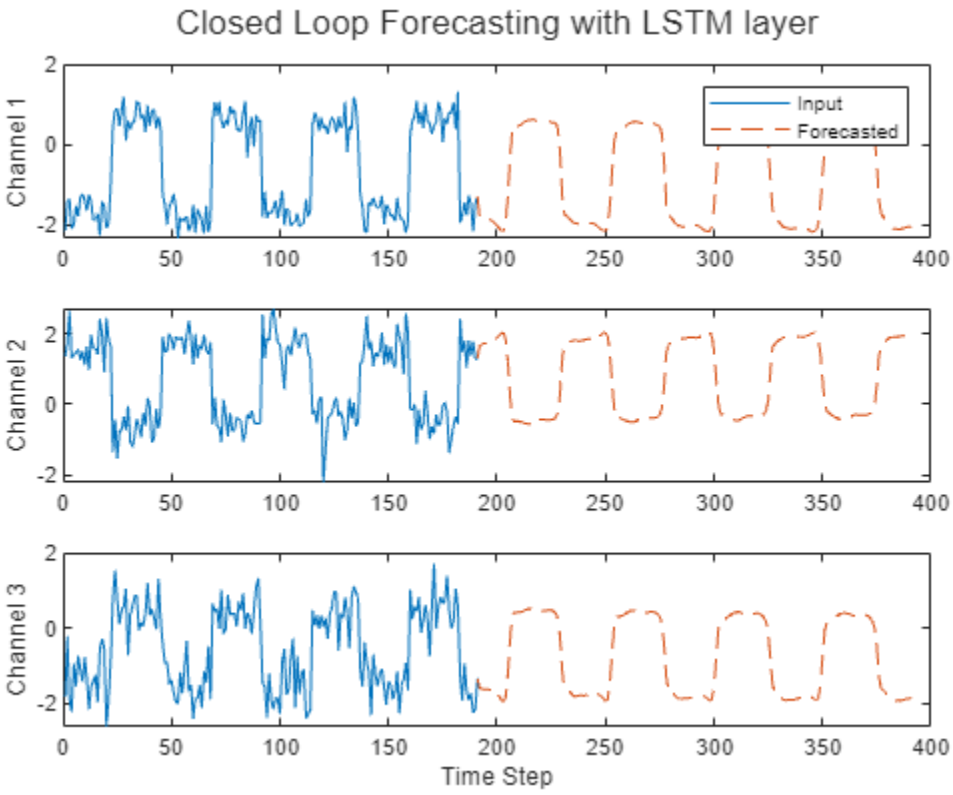
```
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
```

```
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
```

```
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
```

Visualize the forecasted values in a plot.

```
numTimeSteps = offset + numPredictionTimeSteps;

figure
t = tiledlayout(numChannels,1);
title(t,"Closed Loop Forecasting with LSTM layer")

for i = 1:numChannels
    nexttile
    plot(T(i,1:offset))
    hold on
    plot(offset:numTimeSteps,[T(i,offset) Y(i,:)],'--')
    ylabel("Channel " + i)
end

xlabel("Time Step")
nexttile(1)
legend(["Input" "Forecasted"])
```



Closed-loop forecasting allows you to forecast an arbitrary number of time steps, but can be less accurate when compared to open-loop forecasting because the network does not have access to the true values during the forecasting process.

Version History

Introduced in R2022b

See Also

`activations` | `compile` | `deploy` | `predict` | `resetState`

resetState

Class: `dlhdl.Workflow`
Package: `dlhdl`

Reset state parameters of deployed neural network

Syntax

```
resetState(workflowObject)
```

Description

`resetState(workflowObject)` resets the state parameters of the deployed neural network. Use this method to reset the state of a recurrent neural network, such as an LSTM network.

Input Arguments

`workflowObject` — Workflow
`dlhdl.Workflow` object

Workflow, specified as a `dlhdl.Workflow` object.

Examples

Run Sequence Forecasting on FPGA by Using Deep Learning HDL Toolbox

This example shows how to create, compile, and deploy a long short-term memory (LSTM) network trained on waveform data by using the Deep Learning HDL Toolbox™ Support Package for Xilinx FPGA and SoC. Use the deployed network to predict future values by using open-loop and closed-loop forecasting. Use MATLAB® to retrieve the prediction results from the target device.

Waveform Data Network

The network attached to this example was trained using the “Time Series Forecasting Using Deep Learning”. This example uses the `WaveformData.mat` data set, which contains 2000 synthetically generated waveforms of varying lengths with three channels. This example uses a trained LSTM network to forecast future values of the waveforms given the values from the previous time steps using both closed loop and open loop forecasting.

Prerequisites

- Xilinx® Zynq® Ultrascale+™ ZCU102 SoC development kit
- Deep Learning HDL Toolbox™ Support Package for Xilinx FPGA and SoC
- Deep Learning Toolbox™
- Deep Learning HDL Toolbox™

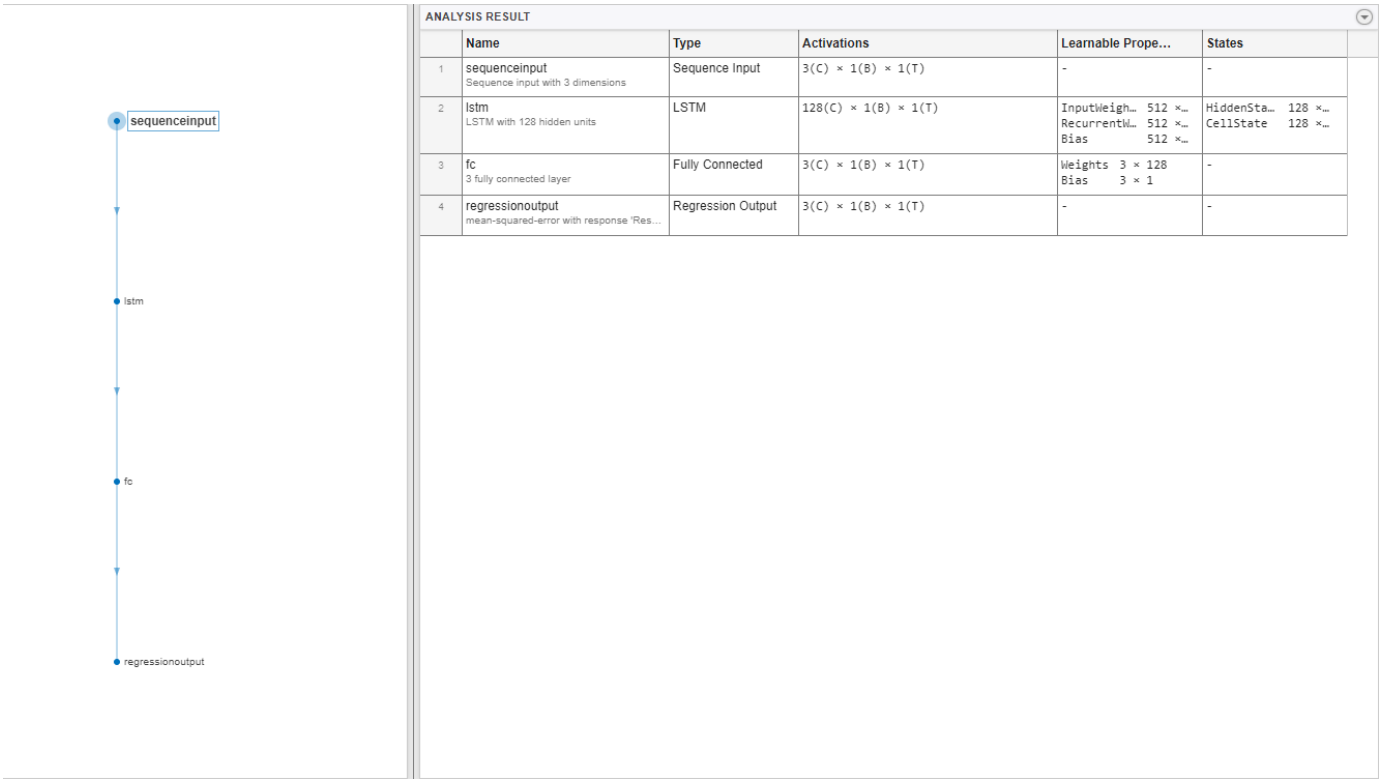
Load the Pretrained Network

To load the LSTM network enter:

```
load WaveformForcastingNet
```

Use the `analyzeNetwork` function to obtain information about the network layers. the function returns a graphical representation of the network that contains detailed parameter information for every layer in the network.

```
analyzeNetwork(net)
```



Define FPGA Board Interface

Define the target FPGA board programming interface by using the `dlhdl.Target` object. Specify that the interface is for a Xilinx board with an Ethernet interface.

To create the target object, enter:

```
hTarget = dlhdl.Target('Xilinx','Interface','Ethernet');
```

To use the JTAG interface, install Xilinx™ Vivado™ Design Suite 2020.2. To set the Xilinx Vivado toolpath, enter:

```
hdlsetuptoolpath('ToolName', 'Xilinx Vivado', 'ToolPath', 'C:\Xilinx\Vivado\2020.2\bin\vivado.ba
hTarget = dlhdl.Target('Xilinx','Interface','JTAG');
```

Prepare Network for Deployment

Prepare the network for deployment by creating a `dlhdl.Workflow` object. Specify the network and the bitstream name. Ensure that the bitstream name matches the data type and the FPGA board. In

this example the target FPGA board is the Xilinx ZCU102 SOC board. The bitstream uses a single data type.

```
hW = dlhdl.Workflow('network', net, 'Bitstream', 'zcu102_lstm_single','Target',hTarget);
```

Tu run the example on the Xilinx ZC706 board, enter:

```
hW = dlhdl.Workflow('Network', snet, 'Bitstream', 'zc706_lstm_single','Target',hTarget);
```

Compile the LSTM Network

Run the `compile` method of the `dlhdl.Workflow` object to compile the network and generate the instructions, weights, and biases for deployment. The total number of frames exceeds the default value of 30. Set the `InputFrameNumberLimit` name-value argument to `1000` to run predictions in chunks of 1000 frames to prevent timeouts.

```
dn = compile(hW,'InputFrameNumberLimit',1000)
```

```
### Compiling network for Deep Learning FPGA prototyping ...
### Targeting FPGA bitstream zcu102_lstm_single.
### The network includes the following layers:
     1   'sequenceinput'      Sequence Input      Sequence input with 3 dimensions            (S
     2   'lstm'               LSTM                LSTM with 128 hidden units                  (H
     3   'fc'                 Fully Connected     3 fully connected layer                     (H
     4   'regressionoutput'   Regression Output   mean-squared-error with response 'Response'  (S

### Notice: The layer 'sequenceinput' with type 'nnet.cnn.layer.ImageInputLayer' is implemented i
### Notice: The layer 'regressionoutput' with type 'nnet.cnn.layer.RegressionOutputLayer' is imp
### Compiling layer group: lstm.wi ...
### Compiling layer group: lstm.wi ... complete.
### Compiling layer group: lstm.wo ...
### Compiling layer group: lstm.wo ... complete.
### Compiling layer group: lstm.wg ...
### Compiling layer group: lstm.wg ... complete.
### Compiling layer group: lstm.wf ...
### Compiling layer group: lstm.wf ... complete.
### Compiling layer group: fc ...
### Compiling layer group: fc ... complete.

### Allocating external memory buffers:

          offset_name          offset_address    allocated_space
    _______________________    ______________    _______________

    "InputDataOffset"           "0x00000000"     "4.0 MB"
    "OutputResultOffset"        "0x00400000"     "4.0 MB"
    "SchedulerDataOffset"       "0x00800000"     "4.0 MB"
    "SystemBufferOffset"        "0x00c00000"     "20.0 MB"
    "InstructionDataOffset"     "0x02000000"     "4.0 MB"
    "FCWeightDataOffset"        "0x02400000"     "4.0 MB"
    "EndOffset"                 "0x02800000"     "Total: 40.0 MB"

### Network compilation complete.

dn = struct with fields:
             weights: [1×1 struct]
        instructions: [1×1 struct]
           registers: [1×1 struct]
```

```
    syncInstructions: [1×1 struct]
        constantData: {}
```

Program Bitstream onto FPGA and Download Network Weights

To deploy the network on the Xilinx ZCU102 SoC hardware, run the `deploy` function of the `dlhdl.Workflow` object. This function uses the output of the `compile` function to program the FPGA board by using the programming file. It also downloads the network weights and biases. The `deploy` function starts programming the FPGA device and displays progress messages, and the required time to deploy the network.

```
deploy(hW)
```

```
### FPGA bitstream programming has been skipped as the same bitstream is already loaded on the ta
### Resetting network state.
### Loading weights to FC Processor.
### FC Weights loaded. Current time is 09-Nov-2022 09:35:06
```

Test Network

Prepare the test data for prediction. Normalize the test data using the statistics calculated from the training data. To forecast the values of future time steps of a sequence, specify the targets as the test sequences with values shifted by one time step. In other words, at each time step of the input sequence, the LSTM network learns to predict the value of the next time step. The predictors as the test sequences without the final time step.

```
load Waveformdata
numChannels = size(data{1},1);
numObservations = numel(data);

idxTrain = 1:floor(0.9*numObservations);
idxTest = floor(0.9*numObservations)+1:numObservations;
dataTrain = data(idxTrain);
dataTest = data(idxTest);

for n = 1:numel(dataTrain)
    X = dataTrain{n};
    XTrain{n} = X(:,1:end-1);
    TTrain{n} = X(:,2:end);
end

muX = mean(cat(2,XTrain{:}),2);
sigmaX = std(cat(2,XTrain{:}),0,2);
muT = mean(cat(2,TTrain{:}),2);
sigmaT = std(cat(2,TTrain{:}),0,2);

for n = 1:size(dataTest,1)
    X = dataTest{n};
    XTest{n} = (X(:,1:end-1) - muX) ./ sigmaX;
    TTest{n} = (X(:,2:end) - muT) ./ sigmaT;
end
```

Make predictions using the test data.

```
YTest = hW.predict(XTest{1},Profile ='on');
```

```
### Resetting network state.
### Finished writing input activations.
```

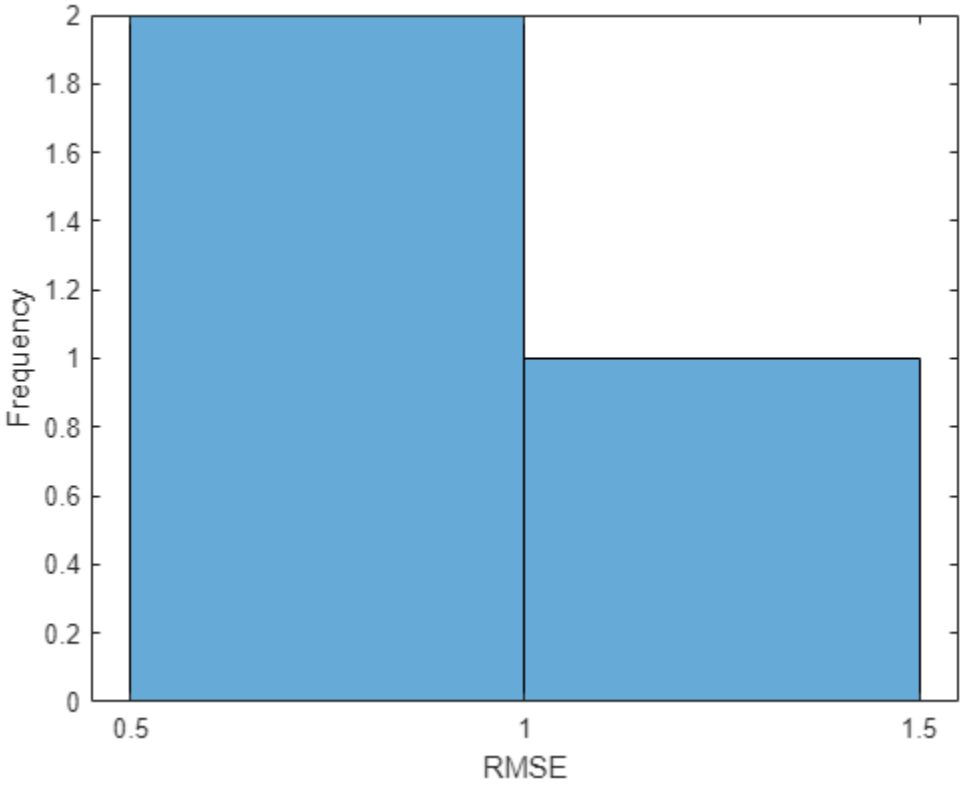
```
### Running a sequence of length 115.


              Deep Learning Processor Profiler Performance Results

                     LastFrameLatency(cycles)   LastFrameLatency(seconds)       FramesNum      Tota
                           -------------              -------------               ---------        --
Network                         38755                 0.00018                     115            44
    memSeparator_0                88                  0.00000
    lstm.wi                     7478                  0.00003
    lstm.wo                     7549                  0.00003
    lstm.wg                     7619                  0.00003
    lstm.wf                     7519                  0.00003
    lstm.sigmoid_1               222                  0.00000
    lstm.sigmoid_3               224                  0.00000
    lstm.tanh_1                  204                  0.00000
    lstm.sigmoid_2               224                  0.00000
    lstm.multiplication_2        294                   0.00000
    lstm.multiplication_1        314                   0.00000
    lstm.c_add                   308                  0.00000
    lstm.tanh_2                  229                  0.00000
    lstm.multiplication_3        287                   0.00000
    fc                          6196                  0.00003
 * The clock frequency of the DL processor is: 220MHz
```

To evaluate the accuracy, calculate the root mean squared error (RMSE) between the predictions and the target for each test sequence.

```
for i = 1:size(YTest,1)
    rmse(i) = sqrt(mean((YTest(i) - TTest{1}(i)).^2,"all"));
end
```

Visualize the errors in a histogram. Lower values indicate greater accuracy.

```
figure
histogram(rmse)
xlabel("RMSE")
ylabel("Frequency")
```



Calculate the mean RMSE over all test observations.

```
mean(rmse)
```

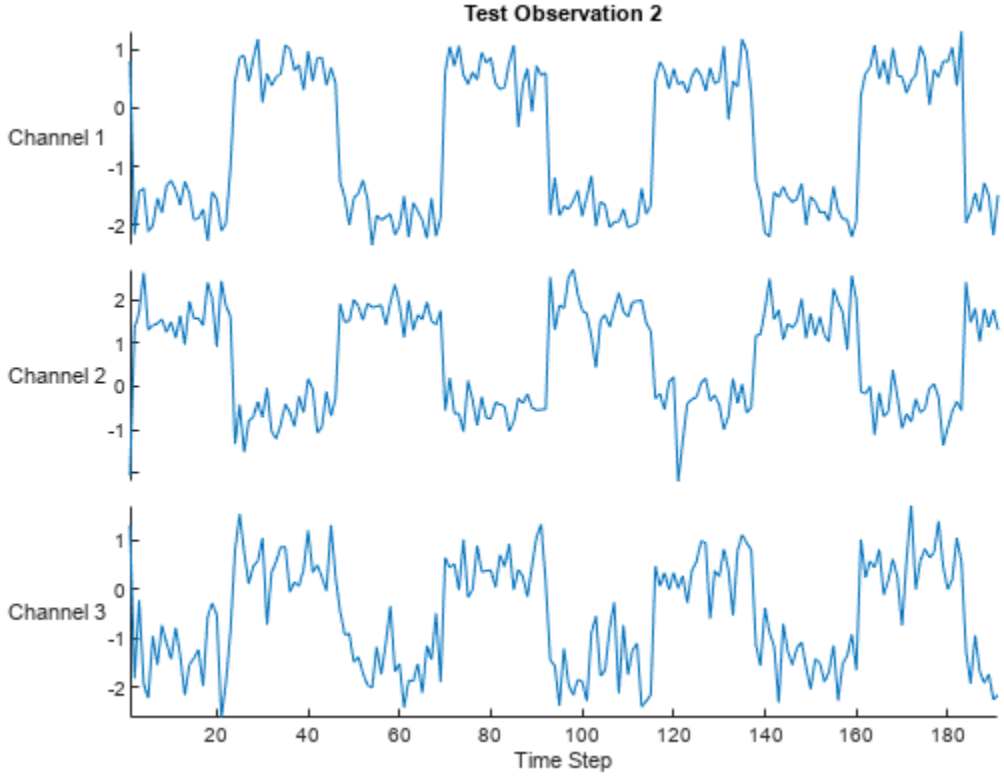
```
ans = single
    0.8385
```

Forecast Future Time Steps

To forecast the values of multiple future time steps, when given an input time series or sequence, use the `predictAndUpdateState` function. This function predicts time steps one at a time and updates the network state at each prediction. For each prediction, use the previous prediction as the input to the function.

Visualize one of the test sequences in a plot.

```
idx = 2;
X = XTest{idx};
T = TTest{idx};

figure
stackedplot(X',DisplayLabels="Channel " + (1:numChannels))
xlabel("Time Step")
title("Test Observation " + idx)
```



Open-Loop Forecasting

Open-loop forecasting predicts the next time step in a sequence using only the input data. When making predictions for subsequent time steps, you collect the true values form your data source and use those as input. For example, suppose that you want to predict the value for time step t of a sequence by using data collected in time steps 1 through $t - 1$. To make predictions for time step $t + 1$, wait until you record the true value for time step t and use that value as input to make the next prediction. Use open-loop forecasting when you have true values to provide to the network before making the next prediction.

Initialize the network state by resetting the state using the `resetState` function, then make an initial prediction using the first few time steps of the input data. Update the network state by using the first 75 time steps of the input data.

```
resetState(hW)
offset = 75;
[~,~] = hW.predictAndUpdateState(X(:,1:offset));
```

```
### Resetting network state.
### Finished writing input activations.
### Running a sequence of length 75.
```

To forecast further predictions, loop over time steps and update the network state by using the `predictAndUpdateState` function. Forecast values for the remaining time steps of the test observation by looping over the time steps of the input data and using them as input to the network. The first prediction is the value that corresponds to the time step `offset + 1`.

```
numTimeSteps = size(X,2);
numPredictionTimeSteps = numTimeSteps - offset;
Y = zeros(numChannels,numPredictionTimeSteps);

for t = 1:numPredictionTimeSteps
    Xt = X(:,offset+t);
    Y(:,t) = predictAndUpdateState(hW,Xt);
end
```

```
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
```

```
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
```

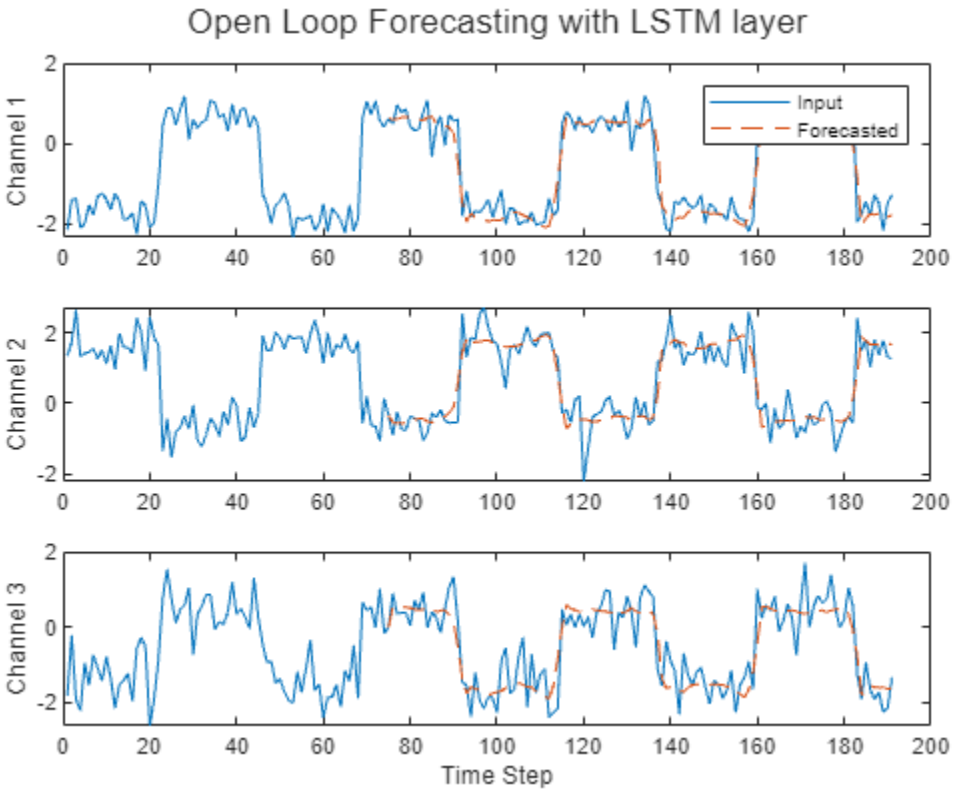
```
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
```

```
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
```

```
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
```

Compare the predictions with the target values.

```
figure
t = tiledlayout(numChannels,1);
title(t,"Open Loop Forecasting with LSTM layer")

for i = 1:numChannels
    nexttile
    plot(T(i,:))
    hold on
    plot(offset:numTimeSteps,[T(i,offset) Y(i,:)],'--')
    ylabel("Channel " + i)
end

xlabel("Time Step")
nexttile(1)
legend(["Input" "Forecasted"])
```



Closed-Loop Forecasting

Closed-loop forecasting predicts subsequent time steps in a sequence by using the previous predictions as input. In this case, the model does not require the true values to make the prediction. For example, suppose that you want to predict the value for time steps t through $t + k$ of the sequence by using data collected in time steps 1 through $t - 1$. To make predictions for time step i, use the predicted value for time step $i - 1$ as input. Use closed-loop forecasting to forecast multiple subsequent time steps or when you do not have true values to provide to the network before making the next prediction.

Initialize the network state by resetting the state using the `resetState` function, then make an initial prediction, `Z`, using the first few time steps of the input data. Update the network state by using the first 75 time steps of the input data.

```
resetState(hW)
offset = size(X,2);
[Z, ~] = predictAndUpdateState(hW,X);
```

```
### Resetting network state.
### Finished writing input activations.
### Running a sequence of length 191.
```

To forecast further predictions, loop over time steps and update the network state by using the `predictAndUpdateState` function. Forecast the next 200 time steps by iteratively passing the previously predicted value to the network. Because the network does not require the input data to make any further predictions, you can specify any number of time steps to forecast.

```
numPredictionTimeSteps = 200;
Xt = Z(:,end);
Y = zeros(numChannels,numPredictionTimeSteps);

for t = 1:numPredictionTimeSteps
    [Y(:,t),~] =  predictAndUpdateState(hW,Xt);
    Xt = Y(:,t);
end
```

```
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
```

```
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
```

```
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
```

```
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
```

```
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
```

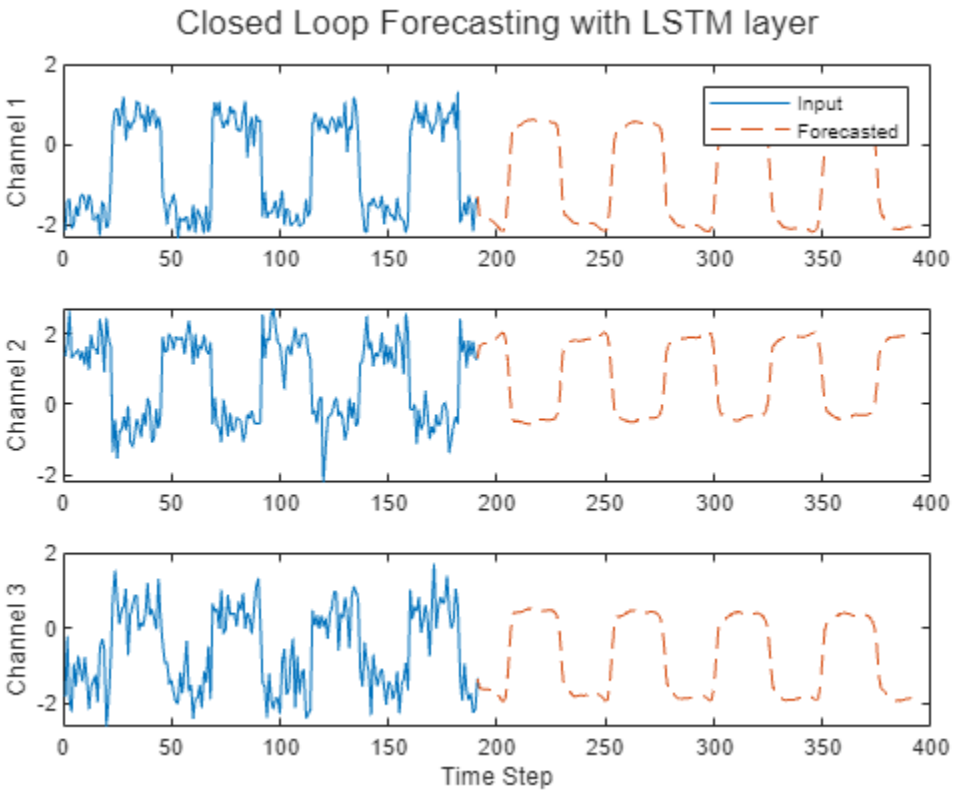
```
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
```

```
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
```

```
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
### Finished writing input activations.
### Running a sequence of length 1.
```

Visualize the forecasted values in a plot.

```
numTimeSteps = offset + numPredictionTimeSteps;

figure
t = tiledlayout(numChannels,1);
title(t,"Closed Loop Forecasting with LSTM layer")

for i = 1:numChannels
    nexttile
    plot(T(i,1:offset))
    hold on
    plot(offset:numTimeSteps,[T(i,offset) Y(i,:)],'--')
    ylabel("Channel " + i)
end

xlabel("Time Step")
nexttile(1)
legend(["Input" "Forecasted"])
```



Closed-loop forecasting allows you to forecast an arbitrary number of time steps, but can be less accurate when compared to open-loop forecasting because the network does not have access to the true values during the forecasting process.

Version History

Introduced in R2022b

See Also

`activations` | `compile` | `deploy` | `predict` | `predictAndUpdateState`

dlhdl.Target class

Package: `dlhdl`

Configure interface to target board for workflow deployment

Description

Use the `dlhdl.Target` object to create the interface to deploy the `dlhdl.Workflow` object to your target hardware.

Creation

`hTarget = dlhdl.Target(Vendor)` creates a target object that you pass on to `dlhdl.Workflow` to deploy your deep learning network to your target device.

`hTarget = dlhdl.Target(Vendor,Name,Value)` creates a target object that you pass on to `dlhdl.Workflow`, with additional properties specified by one or more `Name,Value` pair arguments.

Input Arguments

`Vendor` — Target board vendor name

`'Xilinx'` (default) | `'Intel'` | `"Xilinx"` | `"Intel"`

Target device vendor name, specified as a character vector or string.

Example: `'Xilinx'` or `"Xilinx"`

Properties

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

`Interface` — Interface to connect to the target board

`'JTAG'` (default) | `'Ethernet'` | `'File'`

Name of the interface specified as a character vector.

Example: `'Interface','JTAG'` creates a target configuration object with `'JTAG'` as the interface to the target device.

`IPAddress` — IP address for the target device with Ethernet interface

`''` (default)

IP address for the target device with the Ethernet interface specified as a character vector.

Example: `'IPAddress','192.168.1.101'` creates a target configuration object with `'192.168.1.101'` as the target device IP address.

`Username` — SSH user name

`'root'` (default)

SSH user name specified as a character vector.

Example: `'Username','root'` creates a target configuration object with `'root'` as the SSH user name.

`Password` — SSH password
`'root'` | `'cyclonevsoc'`

Password of the root user specified as a character vector. Use `'root'` on the Xilinx SoC boards and `'cyclonevsoc'` on the Intel SoC boards.

Example: `'Password','root'` creates a target configuration object with `'root'` as the SSH password for Xilinx SoC boards.

Example: `'Password','cyclonevsoc'` creates a target configuration object with `'cyclonevsoc'` as the SSH password for Intel SoC boards.

`Port` — SSH connection port number
22 (default)

SSH port number specified as an integer.

Example: `'Port',22` creates a target configuration object with 22 as the SSH port number.

`Filename` — Name of the deployment file
`' '` (default)

Name of the deployment file specified as a string or character vector. When you do not specify the file name, the name of the bitstream is the file name.

Example: `'Filename','zcu102socinit"` creates a `'zcu102soc_init.dln'` file.

Methods

Public Methods

release	Release the connection to the target device
validateConnection	Validate SSH connection and deployed bitstream

Examples

Create Target Object That Has a JTAG interface

```
hTarget = dlhdl.Target('Xilinx','Interface','JTAG')
hTarget =

  Target with properties:

       Vendor: 'Xilinx'
    Interface: JTAG
```

Create Target Object That Has an Ethernet Interface and Set IP Address

```
hTarget = dlhdl.Target('Xilinx','Interface','Ethernet','IPAddress','192.168.1.101')
hTarget =

  Target with properties:

       Vendor: 'Xilinx'
```

```
    Interface: Ethernet
    IPAddress: '192.168.1.101'
     Username: 'root'
         Port: 22
```

Create Target Object That has a File Interface

```
hTarget = dlhdl.Target('Xilinx',Interface = 'File', Filename = 'zcu102socinit')
```

```
hTarget =

  TargetFile with properties:

      Interface: File
           Path: './'
     FileHandle: -1
             CR: '←↵'
       Filename: 'zcu102socinit'
    IsConnected: 0
          hConn: []
         Vendor: 'Xilinx'
```

Version History

Introduced in R2020b

See Also

Functions

`release` | `validateConnection`

Objects

`dlhdl.Workflow`

release

Class: `dlhdl.Target`
Package: `dlhdl`

Release the connection to the target device

Syntax

```
release(targetObject)
```

Description

`release(targetObject)` releases the connection to the target board.

Input Arguments

`targetObject` — Instance of target object

`dlhdl.Target` object

Instance of the target object, specified as an `dlhdl.Target` object.

Examples

Release Connection to Target Device

1. Create a `dlhdl.Target` object that has an Ethernet interface and SSH connection.

```
hTarget = dlhdl.Target('Xilinx','Interface','Ethernet','IPAddress','192.168.1.101');
```

2. Create a `dlhdl.Workflow` object and deploy the object to the target board.

```
net = resnet18;
hW = dlhdl.Workflow('Network',net,'Bitstream','zcu102_single', 'Target', hTarget);
hW.deploy;
```

3. Obtain a prediction.

   Use this image to run the code:

```
% Load input images and resize them according to the network specifications
image = imread('zebra.jpeg');
inputImg = imresize(image, [224, 224]);
imshow(inputImg);
% Predict the outcome and optionally profile the results to measure performance.
[prediction, speed] = hW.predict(single(inputImg), 'Profile', 'on' );
```

4 Release the connection.

```
hTarget.release;
```

Version History

Introduced in R2020b

See Also

`validateConnection`

validateConnection

Class: `dlhdl.Target`
Package: `dlhdl`

Validate SSH connection and deployed bitstream

Syntax

```
validateConnection(targetObject)
```

Description

`validateConnection(targetObject)`:

1. First validates the SSH connection for an Ethernet interface. This step is skipped for a JTAG interface.
2. Validates the connection for a deployed bitstream.

Input Arguments

`targetObject` — Instance of target object

`dlhdl.Target` object

Instance of the target object, specified as an `dlhdl.Target` object.

Examples

Validate `dlhdl.Target` Object that has a JTAG Interface

Validate deployed bitstream and SSH connection to the target device.

1. Create a `dlhdl.Target` object with a JTAG interface.

```
hTarget = dlhdl.Target('Intel','Interface','JTAG');
```

2. Create a `dlhdl.Workflow` object and deploy the object to the target board.

```
snet = vgg19;
hW = dlhdl.Workflow('Network',snet,'Bitstream','arria10soc_single', 'Target', hTarget);
hW.deploy;
```

3. Validate the connection and bitstream.

```
hTarget.validateConnection
### Validating connection to bitstream over JTAG interface
### Bitstream connection over JTAG interface successful
```

Validate `dlhdl.Target` Object that has an Ethernet Interface

Validate deployed bitstream and SSH connection to the target device.

1. Create a `dlhdl.Target` object that has an Ethernet interface.

```
hTarget = dlhdl.Target('Xilinx','Interface','Ethernet','IPAddress','10.10.10.14');
```

2. Create a `dlhdl.Workflow` object and deploy the object to the target board.

```
net = resnet18;
hW = dlhdl.Workflow('Network',net,'Bitstream','zcu102_single', 'Target', hTarget);
hW.deploy;
```

3. Validate the connection and bitstream.

```
hTarget.validateConnection
### Validating connection to target over SSH
### SSH connection successful
### Validating connection to bitstream over Ethernet interface
### Bitstream connection over Ethernet interface successful
```

Version History

Introduced in R2020b

See Also

`release`

dlhdl.ProcessorConfig class

Package: `dlhdl`

Configure custom deep learning processor

Description

Use the `dlhdl.ProcessorConfig` class to configure a custom processor, which is then passed on to the `dlhdl.buildProcessor` class to generate a custom deep learning processor.

Creation

The `dlhdl.ProcessorConfig` class creates a custom processor configuration object that you can use to specify the processor parameters. The processor parameters are then used by the `dlhdl.buildProcessor` class to build and generate code for your custom deep learning processor.

`dlhdl.ProcessorConfig(Name,Value)` creates a custom processor configuration object, with additional options specified by one or more name-value arguments.

Properties

System Level Properties

`ReferenceDesign` — Reference design name
'AXI-Stream DDR Memory Access : 3-AXIM' (default) | 'Deep Learning with Preprocessing Interface' | character vector

Reference design name, specified as a character vector

Example: `'AXI-Stream DDR memory Access : 3-AXIM'`

`SynthesisTool` — Synthesis tool name
'Xilinx Vivado' (default) | 'Altera Quartus II' | 'Xilinx ISE' | character vector

Synthesis tool name, specified as a character vector.

Example: `Xilinx Vivado`

`SynthesisToolChipFamily` — Synthesis tool chip family name
'Zynq Ultrascale+ (default) | 'Artix7' | 'Kintex7' | 'Kintex Ultrascale+' | 'Spartan7' | 'Virtex7' | 'Virtex Ultrascale+' | 'Zynq' | 'Arria 10' | 'Arria V GZ' | 'Cyclone 10 GX' | 'Stratix 10' | 'Stratix V' | character vector

Specify the target device chip family name as a character vector

Example: `'Zynq'`

`TargetFrequency` — Target frequency in MHz
200 (default) | integer

Specify the target board frequency in MHz.

Example: `220`

`TargetPlatform` — Name of the target board
'Xilinx Zynq Ultrascale+ MPSoC ZCU 102 Evaluation Kit' (default) | 'Generic Deep Learning Processor' | 'Intel Arria 10 SoC development kit' | 'Xilinx Zynq ZC706 evaluation kit' | 'Xilinx Kintex-Ultrascale KCU105 evaluation board' | character vector

Specify the name of the target board as a character vector.

Example: `'Xilinx Zynq ZC706 evaluation kit'`

`Bitstream` — Name of the bitstream
'arria10soc_single' | 'arria10soc_int8' | 'zc706_single' | 'zc706_int8' | 'zcu102_single' | 'zcu102_int8' | 'zcu102_lstm_single' | 'zc706_lstm_single'

Specify the name of the bitstream whose processor configuration must be retrieved as a character vector.

Example: 'Bitstream','zcu102_single'

Top Level Properties

`RunTimeControl` — Deep learning processor IP core mode setting
'register' (default) | 'port' | character vector

Specify whether the runtime input signals to the deep learning processor IP core are implemented as registers or ports.

Example: `'register`

`RunTimeStatus` — Feedback signals from deep learning processor IP core control
'register' (default) | 'port'

Specify whether the runtime output signals from the deep learning processor IP core are implemented as registers or ports.

Example: 'register'

`InputStreamControl` — Deep learning processor IP core input interface control
'register' (default) | 'port' | character vector

Specify the deep learning processor IP core input interface option as a character vector

Example: `'register'`

`OutputStreamControl` — Deep learning processor IP core output interface control
'register' (default) | 'port' | character vector

Specify the deep learning processor IP core output interface option as a character vector

Example: `'port'`

`SetupControl` — Deep learning processor IP core user interface control
'register' (default) | 'port' | character vector

Specify the deep learning processor IP core user interface option as a character vector

Example: `'port'`

`ProcessorDataType` — Deep learning processor IP core module data type
'single (default) | 'int8' | character vector

Specify the deep learning processor IP core module data type as a character vector.

Example: `'single'`

Processing Module conv Properties

`ModuleGeneration` — Enable or disable convolution module generation as a part of the deep learning processor configuration
'on' (default) | 'off' | character vector

Use this parameter to control generation of the convolution module as a part of the deep learning processor configuration.

`LRNBlockGeneration` — Enable or disable local response normalization (LRN) block generation as a part of the convolution module of the deep learning processor configuration
'off' (default) | 'on' | character vector

Use this parameter to control generation of the LRN block as a part of the convolution module of the deep learning processor configuration.

`SegmentationBlockGeneration` — Enable or disable segmentation block generation as a part of the convolution module of the deep learning processor configuration
'on' (default) | 'off' | character vector

Use this parameter to control generation of the segmentation block as a part of the convolution module of the deep learning processor configuration.

`ConvThreadNumber` — Number of parallel convolution processor kernel threads
16 (default) | 4 | 9 | 16 | 25 | 36 | 64 | 256 | unsigned integer

This parameter is the number of parallel 3-by-3 convolution kernel threads that are a part of the `conv` module within the `dlhdl.ProcessorConfig` object.

`InputMemorySize` — Cache block RAM (BRAM) sizes
[227 227 3] (default) | 3D positive integer array

This parameter is a 3D matrix representing input image size limited by the `conv` module BRAM size within the `dlhdl.ProcessorConfig` object.

`OutputMemorySize` — Cache block RAM (BRAM) sizes
[227 227 3] (default) | 3D positive integer array

This parameter is a 3D matrix representing output image size limited by the `conv` module BRAM size within the `dlhdl.ProcessorConfig` object.

`FeatureSizeLimit` — Maximum input and output feature size
2048 (default) | positive integer

This parameter is a positive integer representing the maximum input and output feature size as a part of the `conv` module within the `dlhdl.ProcessorConfig` object.

Processing Module fc Properties

`ModuleGeneration` — Enable or disable fully connected module generation as a part of the deep learning processor configuration
'on' (default) | 'off' | character vector

Use this parameter to control generation of the fully connected module as a part of the deep learning processor configuration.

`SoftmaxBlockGeneration` — Enable or disable Softmax block generation as a part of the fully connected module of the deep learning processor configuration
'off' (default) | 'on' | character vector

Use this parameter to control generation of the Softmax block as a part of the fully connected module of the deep learning processor configuration. When you set this property to `off`, the Softmax layer is still implemented in software.

`SigmoidBlockGeneration` — Enable or disable sigmoid block generation as a part of the fully connected module of the deep learning processor configuration
'off' (default) | 'on' | character vector

Use this parameter to control generation of the sigmoid block as a part of the fully connected module of the deep learning processor configuration.

`FCThreadNumber` — Number of parallel fully connected (fc) MAC threads
4 (default) | 4 | 8 | 16 | 32 | 64 | unsigned integer

This parameter is the number of parallel fc MAC threads that are a part of the `fc` module within the `dlhdl.ProcessorConfig` object.

`InputMemorySize` — Cache block RAM (BRAM) sizes
25088 (default) | unsigned integer

This parameter is an unsigned integer representing cache BRAM size limited by the `fc` module BRAM size within the `dlhdl.ProcessorConfig` object.

`OutputMemorySize` — Cache block RAM (BRAM) sizes
4096 (default) | unsigned integer

This parameter is an unsigned integer representing cache BRAM size limited by the `fc` module BRAM size within the `dlhdl.ProcessorConfig` object.

Processing Module custom Properties

`ModuleGeneration` — Enable or disable custom module generation as a part of the deep learning processor configuration
'on' (default) | 'off' | character vector

Use this parameter to control generation of the adder module as a part of the deep learning processor configuration.

`Addition` — Enable or disable addition layer generation as a part of the custom module of the deep learning processor configuration
'on' (default) | 'off' | character vector

Use this parameter to control generation of the addition layer as a part of the custom module of the deep learning processor configuration.

`Multiplication` — Enable or disable multiplication layer generation as a part of the custom module of the deep learning processor configuration
'on' (default) | 'off' | character vector

Use this parameter to control generation of the multiplication layer as a part of the custom module of the deep learning processor configuration.

`Resize2D` — Enable or disable `resize2d` layer generation as a part of the custom module of the deep learning processor configuration
'off' (default) | 'on' | character vector

Use this parameter to control generation of the `resize2d` layer as a part of the custom module of the deep learning processor configuration.

`Sigmoid` — Enable or disable sigmoid layer generation as a part of the custom module of the deep learning processor configuration
'off' (default) | 'on' | character vector

Use this parameter to control generation of the sigmoid layer as a part of the custom module of the deep learning processor configuration.

`TanhLayer` — Enable or disable `tanh` layer generation as a part of the custom module of the deep learning processor configuration
'off' (default) | 'on' | character vector

Use this parameter to control generation of the `tanh` layer as a part of the custom module of the deep learning processor configuration.

`InputMemorySize` — Cache block RAM (BRAM) sizes
40 (default) | unsigned integer

This parameter is an unsigned integer representing cache BRAM size limited by the `adder` module BRAM size within the `dlhdl.ProcessorConfig` object.

`OutputMemorySize` — Cache block RAM (BRAM) sizes
120 (default) | unsigned integer

This parameter is an unsigned integer representing cache BRAM size limited by the `adder` module BRAM size within the `dlhdl.ProcessorConfig` object.

Methods

Public Methods

buildCalibrationBitstream	Generate calibration bitstream and path to generated bitstream files
deployCalibrationBitstream	Deploy calibration bitstream and generate calibration data file
getModuleProperty	Use the getModuleProperty method to get values of module properties within the dlhdl.ProcessorConfig object
setModuleProperty	Use the setModuleProperty method to set properties of modules within the dlhdl.ProcessorConfig object

estimatePerformance	Retrieve layer-level latencies and performance by using estimatePerformance method
estimateResources	Return estimated resources used by custom bitstream configuration
optimizeConfigurationForNetwork	Update network-specific deep learning processor configuration with optimized deep learning processor configuration
openCustomLayerModel	Open a generated custom layer verification model to verify your custom layers
registerCustomLayer	Register the custom layer definition and Simulink model representation of the custom layer
verifyCustomLayerModel	Verify the functionality and accuracy of the custom layer by using the generated custom layer verification model

Examples

Create a `ProcessorConfig` Object

Create a custom processor configuration. Save the `ProcessorConfig` object to `hPC`.

```
hPC = dlhdl.ProcessorConfig
```

The result is:

```
hPC =

                    Processing Module "conv"
                            ModuleGeneration: 'on'
                           LRNBlockGeneration: 'off'
                      SegmentationBlockGeneration: 'on'
                            ConvThreadNumber: 16
                             InputMemorySize: [227 227 3]
                            OutputMemorySize: [227 227 3]
                            FeatureSizeLimit: 2048

                       Processing Module "fc"
                            ModuleGeneration: 'on'
                          SoftmaxBlockGeneration: 'off'
                          SigmoidBlockGeneration: 'off'
                                  FCThreadNumber: 4
                                 InputMemorySize: 25088
                                OutputMemorySize: 4096

                   Processing Module "custom"
                            ModuleGeneration: 'on'
                                    Addition: 'on'
                              Multiplication: 'on'
                                    Resize2D: 'off'
                                     Sigmoid: 'off'
                                   TanhLayer: 'off'
                             InputMemorySize: 40
                            OutputMemorySize: 120

               Processor Top Level Properties
                              RunTimeControl: 'register'
                               RunTimeStatus: 'register'
                          InputStreamControl: 'register'
                         OutputStreamControl: 'register'
                                SetupControl: 'register'
                           ProcessorDataType: 'single'

                     System Level Properties
                              TargetPlatform: 'Xilinx Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit'
                             TargetFrequency: 200
                               SynthesisTool: 'Xilinx Vivado'
                             ReferenceDesign: 'AXI-Stream DDR Memory Access : 3-AXIM'
                      SynthesisToolChipFamily: 'Zynq UltraScale+'
                      SynthesisToolDeviceName: 'xczu9eg-ffvb1156-2-e'
                     SynthesisToolPackageName: ''
                      SynthesisToolSpeedValue: ''
```

Modify Properties of `ProcessorConfig` Object

Modify the `TargetPlatform`, `SynthesisTool`, and `TargetFrequency` properties of `hPC`.

```
hPC.TargetPlatform = 'Xilinx Zynq ZC706 evaluation kit';
>> hPC.SynthesisTool = 'Xilinx Vivado';
>> hPC.TargetFrequency = 180;
hPC
```

The result is:

```
hPC =

                   Processing Module "conv"
                           ModuleGeneration: 'on'
                         LRNBlockGeneration: 'off'
                  SegmentationBlockGeneration: 'on'
                           ConvThreadNumber: 16
                            InputMemorySize: [227 227 3]
                           OutputMemorySize: [227 227 3]
                           FeatureSizeLimit: 2048

                     Processing Module "fc"
                           ModuleGeneration: 'on'
                        SoftmaxBlockGeneration: 'off'
                        SigmoidBlockGeneration: 'off'
                                FCThreadNumber: 4
                               InputMemorySize: 25088
                              OutputMemorySize: 4096

                  Processing Module "custom"
                            ModuleGeneration: 'on'
                                    Addition: 'on'
                              Multiplication: 'on'
                                    Resize2D: 'off'
                                     Sigmoid: 'off'
                                   TanhLayer: 'off'
                             InputMemorySize: 40
                            OutputMemorySize: 120

            Processor Top Level Properties
                           RunTimeControl: 'register'
                            RunTimeStatus: 'register'
                       InputStreamControl: 'register'
                      OutputStreamControl: 'register'
                             SetupControl: 'register'
                        ProcessorDataType: 'single'

                  System Level Properties
                           TargetPlatform: 'Xilinx Zynq ZC706 evaluation kit'
                          TargetFrequency: 180
                            SynthesisTool: 'Xilinx Vivado'
                          ReferenceDesign: 'AXI-Stream DDR Memory Access : 3-AXIM'
                   SynthesisToolChipFamily: 'Zynq'
                   SynthesisToolDeviceName: 'xc7z045'
                  SynthesisToolPackageName: 'ffg900'
                   SynthesisToolSpeedValue: '-2'
```

Retrieve `ProcessorConfig` object for `zcu102_single` bitstream

Retrieve the `ProcessorConfig` object for the `zcu102_single` bitstream and store the object in `hPC`.

```
hPC = dlhdl.ProcessorConfig('Bitstream','zcu102_single')
```

The result is:

```
hPC =
```

```
                Processing Module "conv"
                        ModuleGeneration: 'on'
                      LRNBlockGeneration: 'off'
           SegmentationBlockGeneration: 'on'
                      ConvThreadNumber: 16
                       InputMemorySize: [227 227 3]
                      OutputMemorySize: [227 227 3]
                      FeatureSizeLimit: 2048

                Processing Module "fc"
                        ModuleGeneration: 'on'
                    SoftmaxBlockGeneration: 'off'
                    SigmoidBlockGeneration: 'off'
                         FCThreadNumber: 4
                       InputMemorySize: 25088
                      OutputMemorySize: 4096

             Processing Module "custom"
                      ModuleGeneration: 'on'
                              Addition: 'on'
                        Multiplication: 'on'
                              Resize2D: 'on'
                               Sigmoid: 'off'
                             TanhLayer: 'off'
                       InputMemorySize: 40
                      OutputMemorySize: 120

        Processor Top Level Properties
                        RunTimeControl: 'register'
                         RunTimeStatus: 'register'
                    InputStreamControl: 'register'
                   OutputStreamControl: 'register'
                          SetupControl: 'register'
                     ProcessorDataType: 'single'

                System Level Properties
                         TargetPlatform: 'Xilinx Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit'
                        TargetFrequency: 220
                          SynthesisTool: 'Xilinx Vivado'
                        ReferenceDesign: 'AXI-Stream DDR Memory Access : 3-AXIM'
                    SynthesisToolChipFamily: 'Zynq UltraScale+'
                    SynthesisToolDeviceName: 'xczu9eg-ffvb1156-2-e'
                  SynthesisToolPackageName: ''
                   SynthesisToolSpeedValue: ''
```

Set the `ProcessorConfig` object module data type to `int8`

Create a custom processor configuration. Save the `ProcessorConfig` object to `hPC`.

```
hPC = dlhdl.ProcessorConfig
```

Modify the `ProcessorDataType` to `int8`.

```
hPC.ProcessorDataType = 'int8'
```

The result is:

```
hPC =

                Processing Module "conv"
                        ModuleGeneration: 'on'
                      LRNBlockGeneration: 'off'
           SegmentationBlockGeneration: 'on'
                      ConvThreadNumber: 16
                       InputMemorySize: [227 227 3]
                      OutputMemorySize: [227 227 3]
                      FeatureSizeLimit: 2048

                Processing Module "fc"
                        ModuleGeneration: 'on'
                    SoftmaxBlockGeneration: 'off'
                    SigmoidBlockGeneration: 'off'
                         FCThreadNumber: 4
                       InputMemorySize: 25088
                      OutputMemorySize: 4096

             Processing Module "custom"
                      ModuleGeneration: 'on'
```

```
                    Addition: 'on'
              Multiplication: 'on'
                    Resize2D: 'off'
                     Sigmoid: 'off'
                   TanhLayer: 'off'
             InputMemorySize: 40
            OutputMemorySize: 120

Processor Top Level Properties
              RunTimeControl: 'register'
               RunTimeStatus: 'register'
          InputStreamControl: 'register'
         OutputStreamControl: 'register'
                SetupControl: 'register'
           ProcessorDataType: 'int8'

       System Level Properties
                TargetPlatform: 'Xilinx Zynq ZC706 evaluation kit'
               TargetFrequency: 180
                 SynthesisTool: 'Xilinx Vivado'
               ReferenceDesign: 'AXI-Stream DDR Memory Access : 3-AXIM'
        SynthesisToolChipFamily: 'Zynq'
        SynthesisToolDeviceName: 'xc7z045'
      SynthesisToolPackageName: 'ffg900'
        SynthesisToolSpeedValue: '-2'
```

Version History

Introduced in R2020b

See Also

Functions

`dlhdl.buildProcessor`

Classes

`dlhdl.Workflow`

Topics

“Custom Processor Configuration Workflow”
“Deep Learning Processor IP Core”
“Estimate Performance of Deep Learning Network”
“Estimate Resource Utilization for Custom Processor Configuration”

buildCalibrationBitstream

Class: `dlhdl.ProcessorConfig`
Package: `dlhdl`

Generate calibration bitstream and path to generated bitstream files

Syntax

```
bitstreamPath = buildCalibrationBitstream(processorConfigObject)
```

Description

`bitstreamPath = buildCalibrationBitstream(processorConfigObject)` generates the board-specific calibration bitstream files for `processorConfigObject` and returns the path to the generated bitstream files.

Input Arguments

`processorConfigObject` — Processor configuration
`dlhdl.ProcessorConfig` object

Processor configuration, specified as a `dlhdl.ProcessorConfig` object.

Output Arguments

`bitstreamPath` — Path to folder containing generated calibration bitstream files
`character vector`

Path to folder containing generated calibration bitstream files, returned as a `character vector`

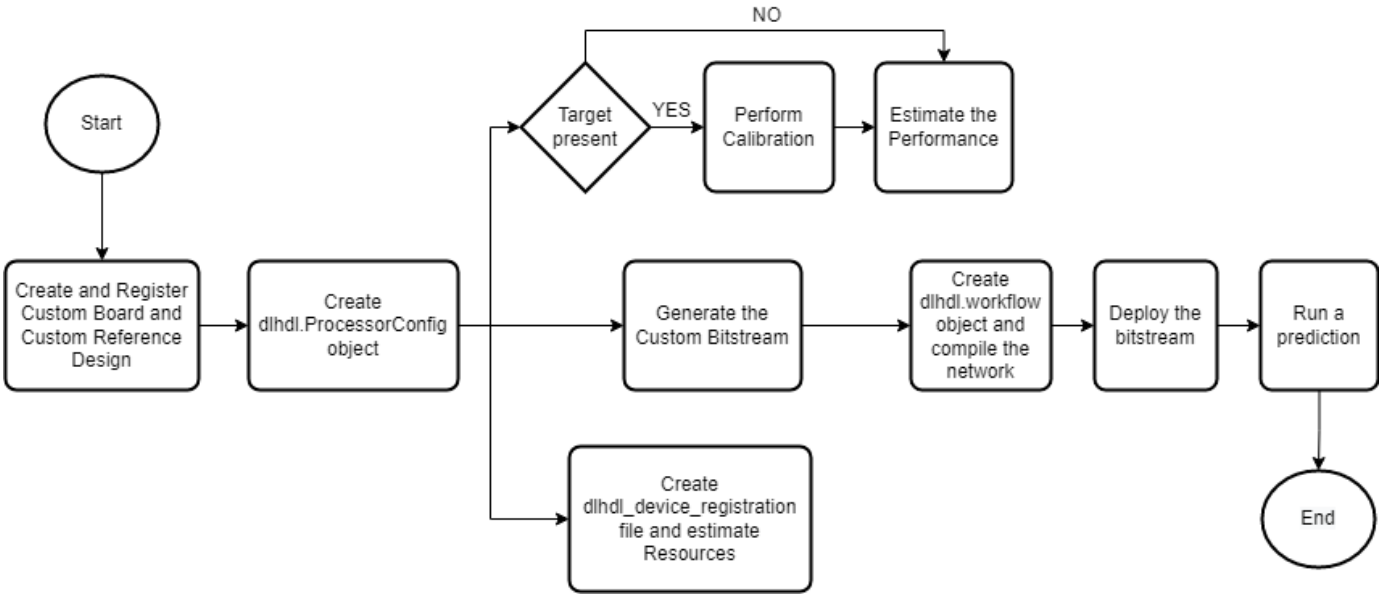
Examples

Deep Learning Processor IP Core Generation for Custom Board

This example shows how to create custom board and generate a deep learning processor IP core for the custom board. In this example you:

- Create a custom board and reference design
- Estimate the network performance and board resource utilization
- Generate a custom processor and bitstream
- Deploy the network by using the custom bitstream

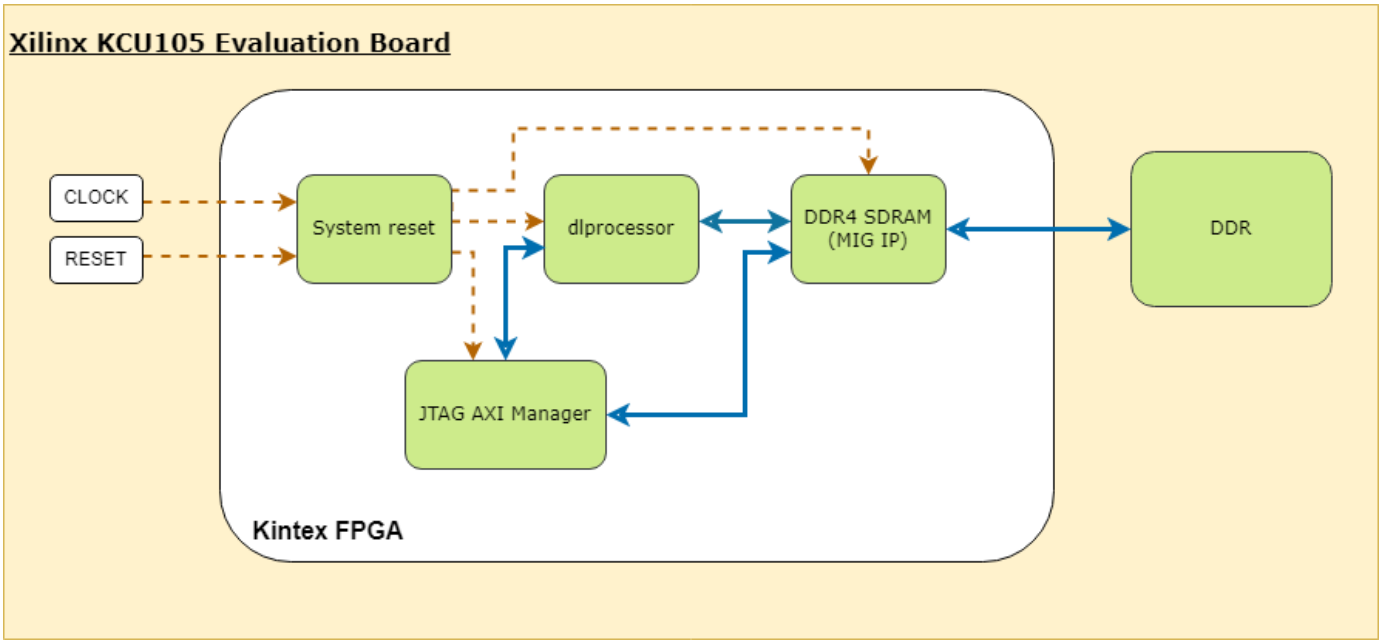
The image shows the process of deploying a network to a custom board and retrieving a prediction from the deployed network.



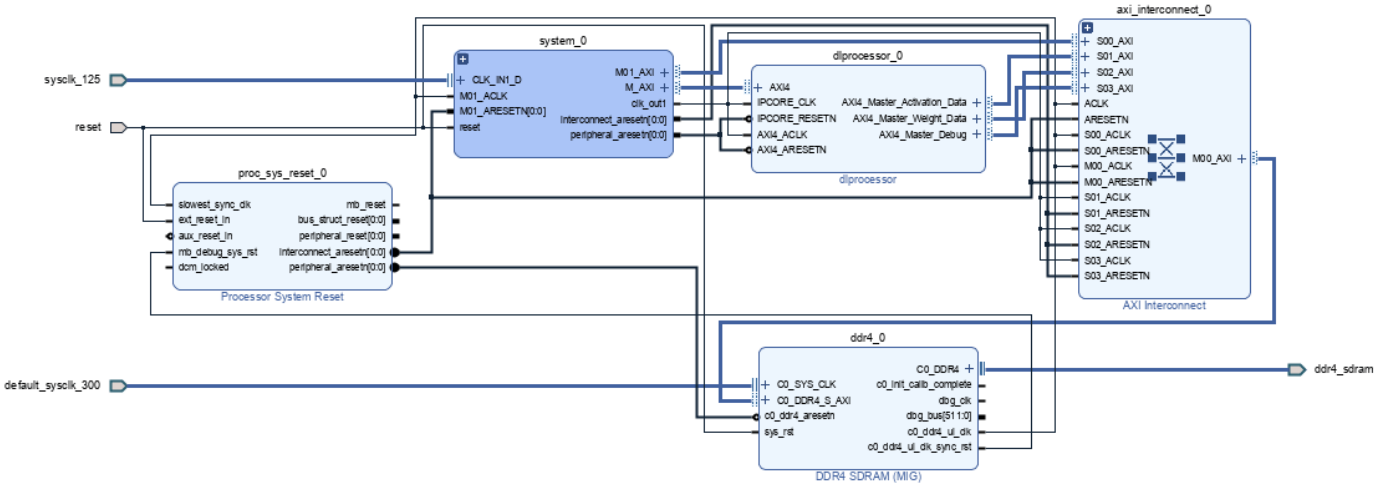
This example uses the Xilinx® Kintex® UltraScale™ KCU105 board. The board contains these blocks:

- System reset block — Used to feed the clock and reset signals to the design.
- Memory Interface Generator (MIG) IP block — Used to generate memory controllers and interfaces for Xilinx FPGAs.
- MATLAB JTAG AXI Manager block — Used by MATLAB to access onboard memory location. For more details, see “Set Up AXI Manager” (HDL Verifier).

Integrate the generated deep learning processor IP core into your reference design. For more details, see “Board and Reference Design Registration System” (HDL Coder).



This image shows the generated deep learning processor IP core `dl_processor0` integrated into the reference design.



Register Custom Board

Define the interface and attributes of a custom SoC board. To register the Xilinx® Kintex® UltraScale™ KCU105 board:

1. Create a board registration file with the name `hdlcoder_board_customization.m` and add it to the MATLAB path. The `hdlcoder_board_customization.m` function must return a second output. For more information, see “Register a Custom Board” (HDL Coder).

Set the target workflow to `DeepLearningProcessor`. For information on other target workflows supported by HDL Coder™ , see “Workflows in HDL Workflow Advisor” (HDL Coder).

```
function [boardList, workflow] = hdlcoder_board_customization
  % Board plugin registration file
  % 1. Any registration file with this name on MATLAB path will be picked up
  % 2. Registration file returns a cell array pointing to the location of
  %    the board plugins
  % 3. Board plugin must be a package folder accessible from MATLAB path,
  %    and contains a board definition file
  %
  % Copyright 2022 The MathWorks, Inc.

boardList = { ...
     'DLKCU105.plugin_board', ...
      };
workflow = hdlcoder.Workflow.DeepLearningProcessor;
end
```

2. Create the board definition file. To generate a deep learning processor, you must define the `ExternalMemorySize`. This property defines the memory size of the DDR on the target board.

```
% Copyright 2022 The MathWorks, Inc.

% Board definition of KCU105
function hB = plugin_board()
```

```
% Construct board object
hB = hdlcoder.Board;

hB.BoardName    = 'Xilinx Kintex-Ultrascale KCU105 evaluation board';

% FPGA device information
hB.FPGAVendor   = 'Xilinx';
hB.FPGAFamily   = 'KintexU';
hB.FPGADevice   = 'xcku040-ffva1156-2-e';
hB.FPGAPackage  = '';
hB.FPGASpeed    = '';

% Tool information
hB.SupportedTool = {'Xilinx Vivado'};

% FPGA JTAG chain position
hB.JTAGChainPosition = 1;

% Size of external DDR memory in bytes
hB.ExternalMemorySize = 0x80000000; % 2 GB

% Add interfaces
% Standard "External Port" interface
hB.addExternalPortInterface( ...
    'IOPadConstraint', {'IOSTANDARD = LVCMOS18'});

% Custom board external I/O interface
hB.addExternalIOInterface( ...
     'InterfaceID',    'LEDs General Purpose', ...
     'InterfaceType',  'OUT', ...
     'PortName',       'GPLEDs', ...
     'PortWidth',      8, ...
     'FPGAPin',        {'AP8', 'H23', 'P20', 'P21', 'N22', 'M22', 'R23','P23'}, ...
     'IOPadConstraint', {'IOSTANDARD = LVCMOS18'});

% Custom board external I/O interface
hB.addExternalIOInterface( ...
    'InterfaceID',    'User Push Buttons', ...
    'InterfaceType',  'IN', ...
    'PortName',       'PB', ...
    'PortWidth',      1, ...
    'FPGAPin',        {'AE10'}, ...
    'IOPadConstraint', {'IOSTANDARD = LVCMOS18'});
```

Register Custom Reference Design

Define the interface and attributes of a custom SoC reference design. To create a custom reference design:

1. Create a reference design registration file named `hdlcoder_ref_design_customization.m` that contains the list of reference design plugins associated with the board. For more information, see “Register a Custom Reference Design” (HDL Coder).

```
function [rd, boardName] = hdlcoder_ref_design_customization
% Reference design plugin registration file
% 1. The registration file with this name inside of a board plugin folder
%    will be picked up
% 2. Any registration file with this name on MATLAB path will also be picked up
% 3. The registration file returns a cell array pointing to the location of
```

```
%    the reference design plugins
% 4. The registration file also returns its associated board name
% 5. Reference design plugin must be a package folder accessible from
%    MATLAB path, and contains a reference design definition file
%
%   Copyright 2022 The MathWorks, Inc.

rd = {...
        'DLKCU105.matlab_3axi4_master_2020_1.plugin_rd', ...
  };

boardName = 'Xilinx Kintex-Ultrascale KCU105 evaluation board';

end
```

2. Create the reference design definition file. To generate a deep learning processor IP core, you must define these three AXI4 Master Interfaces:

- AXI4 Master Activation Data
- AXI4 Master Weight Data
- AXI4 Master Debug

```
function hRD = plugin_rd()
% Reference design definition

% Copyright 2022 The MathWorks, Inc.

% Construct reference design object
hRD = hdlcoder.ReferenceDesign('SynthesisTool', 'Xilinx Vivado');

hRD.ReferenceDesignName = 'AXI-Stream DDR Memory Access : 3-AXIM';
hRD.BoardName = 'Xilinx Kintex-Ultrascale KCU105 evaluation board';

% Tool information
hRD.SupportedToolVersion = {'2020.1','2020.2'};

% Add custom design files

% add custom Vivado design
hRD.addCustomVivadoDesign( ...
  'CustomBlockDesignTcl', 'system_top.tcl',...
  'VivadoBoardPart',      'xilinx.com:kcu105:part0:1.0');

% Add HDL Verifier JTAG as AXI Master IP from support package
hRD.addIPRepository( ...
  'IPListFunction','hdlverifier.fpga.vivado.iplist', ...
  'NotExistMessage', 'IP Repository not found.');

% Add interfaces

% add clock interface
hRD.addClockInterface( ...
     'ClockConnection',      'system_0/clk_out1', ...
     'ResetConnection',      'system_0/peripheral_aresetn',...
     'DefaultFrequencyMHz',  125,...
     'MinFrequencyMHz',      10,...
     'MaxFrequencyMHz',      250,...
     'ClockNumber',          1,...
     'ClockModuleInstance',  'system_0/clk_wiz_0');
```

```
% add AXI4 and AXI4-Lite slave interfaces
% This slave interface is used for intracting between DDR4 and Deep Learning IP
hRD.addAXI4SlaveInterface( ...
    'InterfaceConnection', 'system_0/M_AXI', ...
    'BaseAddress',         '0x44A00000',...
    'MasterAddressSpace',  'system_0/hdlverifier_axi_manager_0/axi4m',...
    'InterfaceType',       'AXI4');

% AXI4 Master Interface for the layer activation data with max data bit-width of 512
hRD.addAXI4MasterInterface(...
        'InterfaceID',         'AXI4 Master Activation Data', ...
        'ReadSupport',          true, ...
        'WriteSupport',         true, ...
        'MaxDataWidth',         512, ...
        'AddrWidth',            32, ...
        'InterfaceConnection', 'axi_interconnect_0/S01_AXI',...
        'TargetAddressSegments', {{'ddr4_0/C0_DDR4_MEMORY_MAP/C0_DDR4_ADDRESS_BLOCK',hex2dec('80

% AXI4 Master Interface for the layer weight data with max data bit-width of 512
hRD.addAXI4MasterInterface(...
        'InterfaceID',         'AXI4 Master Weight Data', ...
        'ReadSupport',          true, ...
        'WriteSupport',         true, ...
        'MaxDataWidth',         512, ...
        'AddrWidth',            32, ...
        'InterfaceConnection', 'axi_interconnect_0/S02_AXI',...
        'TargetAddressSegments', {{'ddr4_0/C0_DDR4_MEMORY_MAP/C0_DDR4_ADDRESS_BLOCK',hex2dec('80

% AXI4 Master Interface for the debugger with max data bit-width of 512
hRD.addAXI4MasterInterface(...
        'InterfaceID',         'AXI4 Master Debug', ...
        'ReadSupport',          true, ...
        'WriteSupport',         true, ...
        'MaxDataWidth',         512, ...
        'AddrWidth',            32, ...
        'InterfaceConnection', 'axi_interconnect_0/S03_AXI',...
        'TargetAddressSegments', {{'ddr4_0/C0_DDR4_MEMORY_MAP/C0_DDR4_ADDRESS_BLOCK',hex2dec('80
```

3. The reference design plugin file must contain information about the target interface and the deep learning processor IP core, the memory address space for the deep learning processor IP core, and a command to validate the reference design. The file also requires information on the resources consumed by the reference design. This information is used during resource estimation. Add the deep learning processor information to the reference design file:

```
% Deep learning specific properties
hRD.registerDeepLearningTargetInterface("JTAG");
hRD.registerDeepLearningMemoryAddressSpace(0x80000000, 0x80000000); % 2GB

% Resource utilization information
hRD.ResourcesUsed.LogicElements = 30500;
hRD.ResourcesUsed.DSP = 3;
hRD.ResourcesUsed.RAM = 26.5;
```

Performance Estimation

Reduce the time required to design and deploy a custom deep learning network that meets performance requirements by analyzing the layer-level latencies before deploying the network.

Estimate the performance of network for your custom board by collecting calibration data from the custom board, by:

1. Generating a calibration bitstream
2. Deploying the calibration bitstream to the target custom board
3. Retrieving the external to internal memory transaction latencies

Create a Processor Configuration object.

```
hPC = dlhdl.ProcessorConfig;
```

Specify the `TargetPlatform`. This automatically sets the `SynthesisToolChipFamily`, `SynthesisToolDeviceName`, and `ReferenceDesign` properties.

```
hPC.TargetPlatform = 'Xilinx Kintex-Ultrascale KCU105 evaluation board';
```

Set the target frequency.

```
hPC.TargetFrequency = 100;
```

This example uses a ResNet-18 pretrained network. For more details, see `resnet18`. Set the deep learning network:

```
net = resnet18;
```

To fit this design onto the target, reduce the number of parallel convolution processor kernel threads for the `conv` module to 9.

```
setModuleProperty(hPC, 'conv', 'ConvThreadNumber', 9);
```

Set the Xilinx Vivado toolpath to your design tool using the `hdlsetuptoolpath` function, then build the calibration bitstream.

```
hdlsetuptoolpath('ToolName','Xilinx Vivado','ToolPath','C:\Xilinx\Vivado\2020.2\bin\vivado.bat');
bitstreamPath = buildCalibrationBitstream(hPC);
```

Deploy the bitstream to the hardware and obtain the external- to-internal memory transaction latencies. You can use these values to get better estimates for the layer-level latencies.

```
deployCalibrationBitstream(hPC, bitstreamPath);
```

The `deployCalibrationBitstream` saves the calibration data from the hardware as a structure in the `CalibrationData` property of the `dlhdl.ProcessorConfig` object. The function also saves the calibration data as a MAT-file with the name `calibrationData.mat`. You can load this data into a new `dlhdl.ProcessorConfig` object by entering:

```
load('calibrationData.mat');
hPC.CalibrationData = calData;
```

Estimate the performance of the network for the custom processor configuration.

```
estimatePerformance(hPC, net);
```

```
            Deep Learning Processor Estimator Performance Results

                  LastFrameLatency(cycles)   LastFrameLatency(seconds)      FramesNum      Total Latency      Frames/s
                       -------------              -------------              ---------       ---------       ---------
Network                   34817713                    0.34818                   1           34817713             2.9
    ____data_norm_add       117907                    0.00118
    ____data_norm           117907                    0.00118
    ____conv1              3356175                    0.03356
    ____pool1               635032                    0.00635
    ____res2a_branch2a     1850730                    0.01851
    ____res2a_branch2b     1850730                    0.01851
    ____res2a               162115                    0.00162
    ____res2b_branch2a     1850730                    0.01851
    ____res2b_branch2b     1850730                    0.01851
    ____res2b               162115                    0.00162
    ____res3a_branch1      1012642                    0.01013
    ____res3a_branch2a     1011307                    0.01011
    ____res3a_branch2b     1599289                    0.01599
    ____res3a                79207                    0.00079
    ____res3b_branch2a     1599289                    0.01599
    ____res3b_branch2b     1599289                    0.01599
    ____res3b                79207                    0.00079
    ____res4a_branch1        823109                    0.00823
    ____res4a_branch2a       825037                    0.00825
    ____res4a_branch2b      1552695                    0.01553
    ____res4a                 39605                    0.00040
    ____res4b_branch2a      1552695                    0.01553
    ____res4b_branch2b      1552695                    0.01553
    ____res4b                 39605                    0.00040
    ____res5a_branch1       1182129                    0.01182
    ____res5a_branch2a      1186237                    0.01186
    ____res5a_branch2b      2284751                    0.02285
    ____res5a                 19703                    0.00020
    ____res5b_branch2a      2284751                    0.02285
    ____res5b_branch2b      2284751                    0.02285
    ____res5b                 19703                    0.00020
    ____pool5                 47376                    0.00047
    ____fc1000               188470                    0.00188
 * The clock frequency of the DL processor is: 100MHz
```

Resource Estimation

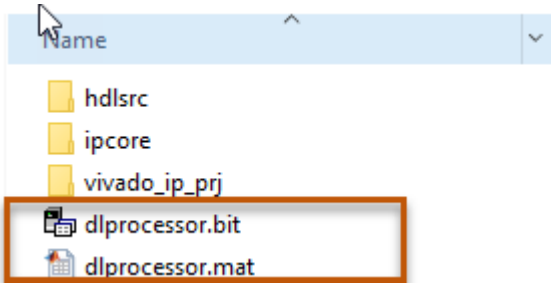
Verify that the generated bistream and network fit on your target custom board, by using `estimateResources` to estimate the resource utilization. To learn how to estimate the resource utilization for your custom boards, see “Estimate Resource Utilization for Custom Board and Reference Design”.

Generate Custom Bitstream for Custom Processor Configuration

Generate a bitstream for the custom processor configuration `hPC`.

```
dlhdl.buildProcessor(hPC);
```

Locate the bitstream file and associated MAT file at `cwd\dlhdl_prj\`, where `cwd` is your current working folder. The name of the bitstream file is `dlprocessor.bit`. The name of the MAT file is `dlprocessor.mat`. To use the generated bitstream for the supported Xilinx boards, copy the `dlprocessor.bit` and `dlprocessor.mat` files to the current working folder.



Deploy the Custom Bitstream and Run Predictions on the Network

After you generate the bitstream, deploy the network and run the predictions on the network. For more information, refer to the “Prototype Deep Learning Networks on FPGA and SoC Devices” page. For an example on prototyping, see “Bicyclist and Pedestrian Classification by Using FPGA”.

Create Target Object

Create a target object with the vendor name of the target device. Specify the interface to connect the target device to the host using the `Interface` name-value pair. This example connects to the target using the JTAG interface.

```
hT = dlhdl.Target('Xilinx', 'Interface', 'JTAG')
```

Create Workflow Object for ResNet-18 Network

Create an object of the `dlhdl.Workflow` class. Specify the network, the bitstream name, and the target object.

```
hW = dlhdl.Workflow('Network', net, 'Bitstream', 'dlprocessor.bit', 'Target', hT);
```

Compile the Network

Run the `compile` function of the `dlhdl.Workflow` object.

```
compile(hW)
```

Deploy the Bitstream to the FPGA

To deploy the network on the Xilinx KCU105 Kintex hardware, run the `deploy` function of the `dlhdl.Workflow` object.

```
deploy(hW)
```

```
### Programming FPGA Bitstream using JTAG...
### Programming the FPGA bitstream has been completed successfully.
### Loading weights to Conv Processor.
### Conv Weights loaded. Current time is 07-Jun-2022 17:44:19
### Loading weights to FC Processor.
### FC Weights loaded. Current time is 07-Jun-2022 17:44:27
```

Run Prediction for the Network

Load the sample image.

```
img = imread('sampleImage1.png');
imshow(img);
```

Run a prediction on the image. The `result` output argument contains the output of the layer preceding the `ClassificationOutputLayer` and `speed` contains the profiler table.

```
[result, speed] = predict(hW, img, 'Profile', 'on');
```

```
              Deep Learning Processor Profiler Performance Results

                    LastFrameLatency(cycles)    LastFrameLatency(seconds)       FramesNum       Total Latency     Frames/s
                          -------------               -------------             ---------         ---------      ---------
Network                     34197670                    0.34198                     1             34200121            2.9
    data_norm_add             157975                    0.00158
    data_norm                 157887                    0.00158
    conv1                    2984413                    0.02984
    pool1                     516698                    0.00517
    res2a_branch2a           1734527                    0.01735
    res2a_branch2b           1734368                    0.01734
    res2a                     224302                    0.00224
    res2b_branch2a           1734604                    0.01735
    res2b_branch2b           1734348                    0.01734
    res2b                     224312                    0.00224
    res3a_branch1             979411                    0.00979
    res3a_branch2a            974770                    0.00975
    res3a_branch2b           1561164                    0.01561
    res3a                     109624                    0.00110
    res3b_branch2a           1561119                    0.01561
    res3b_branch2b           1561136                    0.01561
    res3b                     109624                    0.00110
    res4a_branch1             817239                    0.00817
    res4a_branch2a            819776                    0.00820
    res4a_branch2b           1555224                    0.01555
    res4a                      54841                    0.00055
    res4b_branch2a           1555469                    0.01555
    res4b_branch2b           1555284                    0.01555
    res4b                      54831                    0.00055
    res5a_branch1            1197847                    0.01198
    res5a_branch2a           1199724                    0.01200
    res5a_branch2b           2331422                    0.02331
    res5a                      27324                    0.00027
    res5b_branch2a           2331507                    0.02332
    res5b_branch2b           2331802                    0.02332
    res5b                      27274                    0.00027
    pool5                      81101                    0.00081
    fc1000                    196536                    0.00197
 * The clock frequency of the DL processor is: 100MHz
```

Get the output class from the prediction.

```
[value,idx] = max(result);
classNames = net.Layers(end).Classes;
classNames(idx)
```

```
ans =

  categorical

     monitor
```

Version History

Introduced in R2022b

See Also

deployCalibrationBitstream | estimatePerformance | estimateResources | getModuleProperty | optimizeConfigurationForNetwork | setModuleProperty

deployCalibrationBitstream

Class: `dlhdl.ProcessorConfig`
Package: `dlhdl`

Deploy calibration bitstream and generate calibration data file

Syntax

```
deployCalibrationBitstream(processorConfigObject,bitstreamPath)
```

Description

`deployCalibrationBitstream(processorConfigObject,bitstreamPath)` deploys the bitstream generated by `buildCalibrationBitstream` and generates a `calibrationData.mat` file that contains the read and write latency values for the target hardware board that correspond to different data burst lengths.

Input Arguments

`processorConfigObject` — Processor configuration

`dlhdl.ProcessorConfig` object

Processor configuration, specified as a `dlhdl.ProcessorConfig` object.

`bitstreamPath` — Path to generated calibration bitstream files

' ' (default) | char

Path to the generated bitstream files returned by the `buildCalibrationBitstream` method, specified as a character vector.

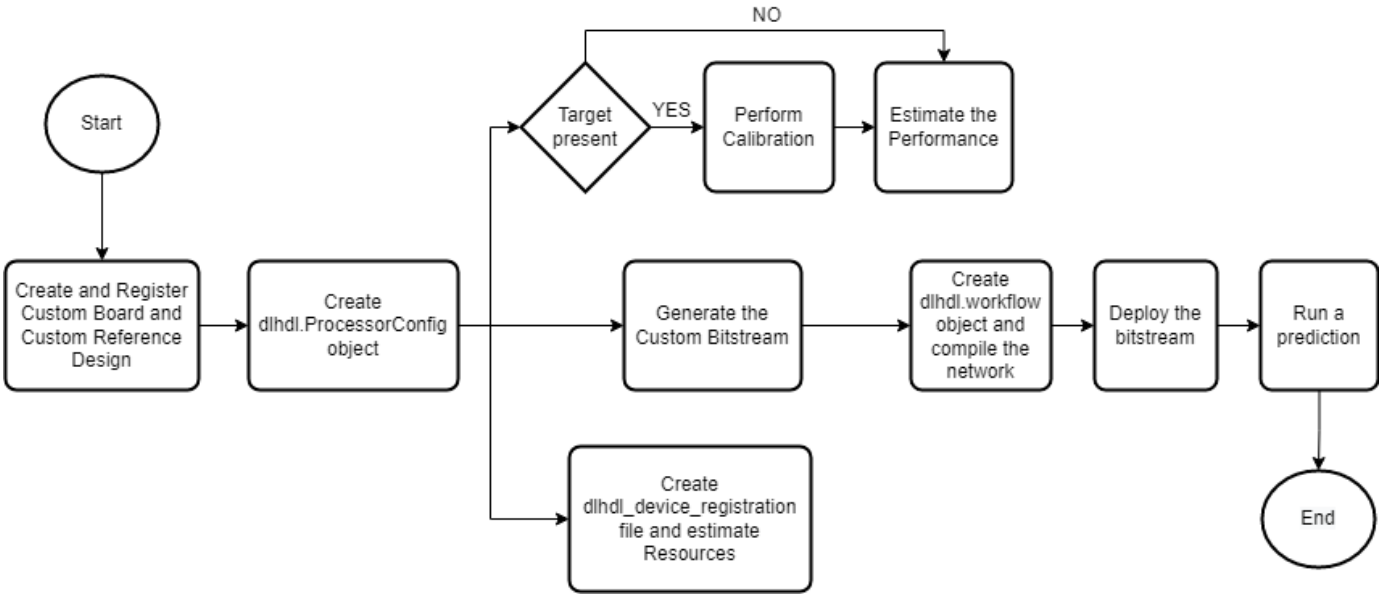
Examples

Deep Learning Processor IP Core Generation for Custom Board

This example shows how to create custom board and generate a deep learning processor IP core for the custom board. In this example you:

- Create a custom board and reference design
- Estimate the network performance and board resource utilization
- Generate a custom processor and bitstream
- Deploy the network by using the custom bitstream

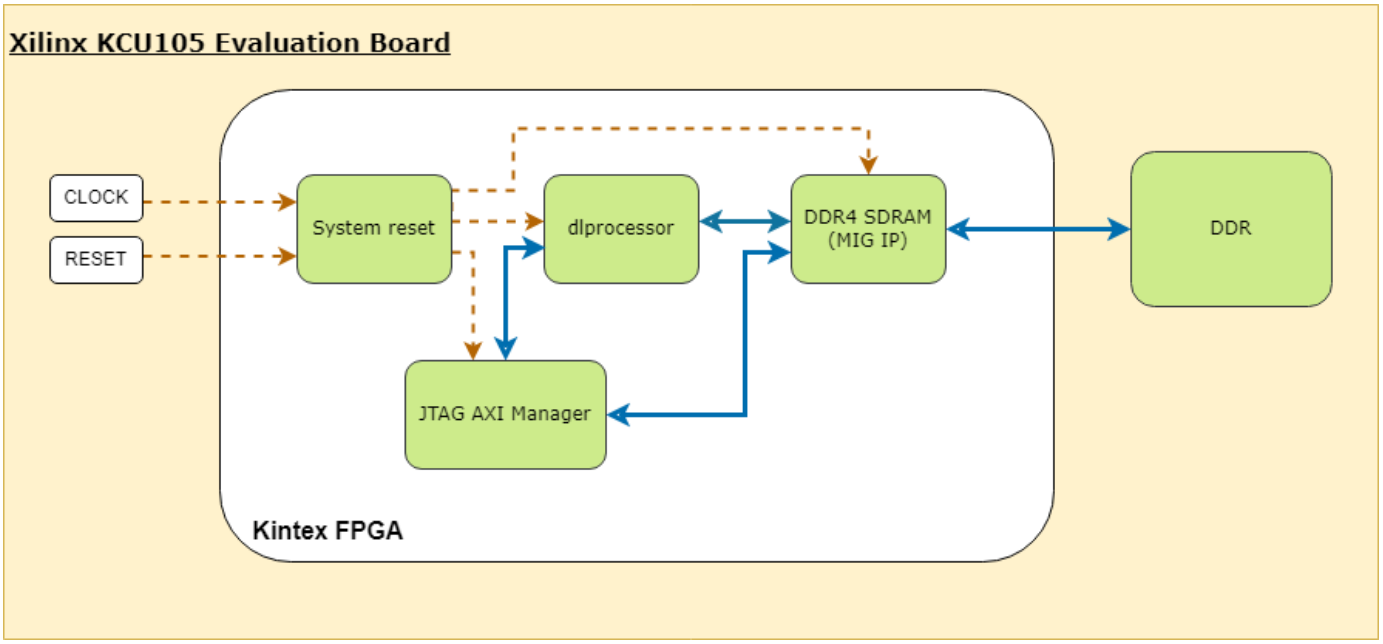
The image shows the process of deploying a network to a custom board and retrieving a prediction from the deployed network.



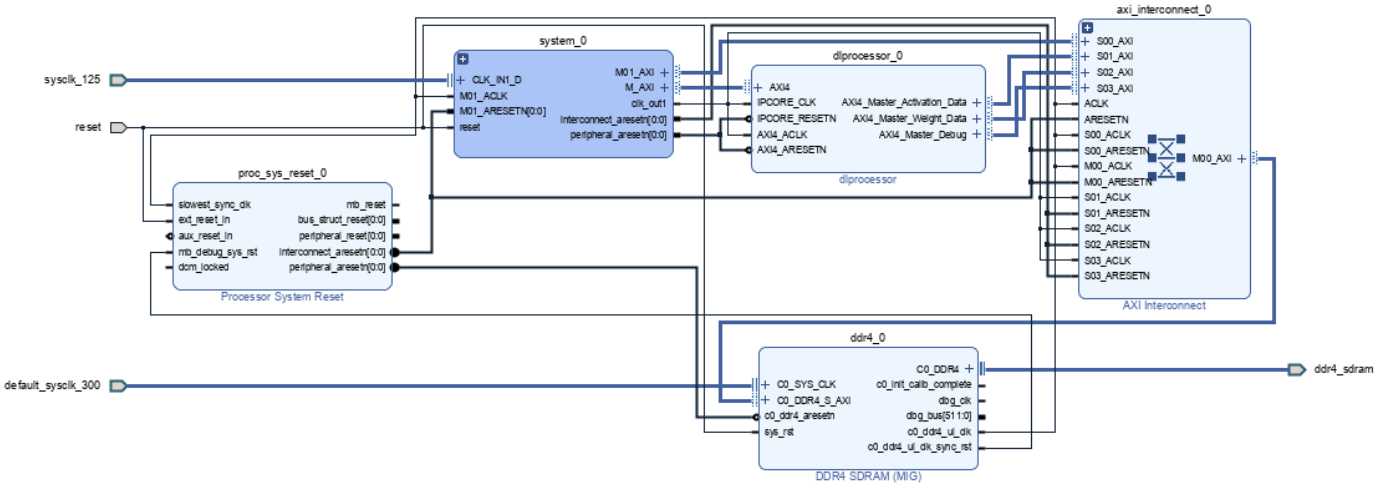
This example uses the Xilinx® Kintex® UltraScale™ KCU105 board. The board contains these blocks:

- System reset block — Used to feed the clock and reset signals to the design.
- Memory Interface Generator (MIG) IP block — Used to generate memory controllers and interfaces for Xilinx FPGAs.
- MATLAB JTAG AXI Manager block — Used by MATLAB to access onboard memory location. For more details, see “Set Up AXI Manager” (HDL Verifier).

Integrate the generated deep learning processor IP core into your reference design. For more details, see “Board and Reference Design Registration System” (HDL Coder).



This image shows the generated deep learning processor IP core `dl_processor0` integrated into the reference design.



Register Custom Board

Define the interface and attributes of a custom SoC board. To register the Xilinx® Kintex® UltraScale™ KCU105 board:

1. Create a board registration file with the name `hdlcoder_board_customization.m` and add it to the MATLAB path. The `hdlcoder_board_customization.m` function must return a second output. For more information, see “Register a Custom Board” (HDL Coder).

Set the target workflow to `DeepLearningProcessor`. For information on other target workflows supported by HDL Coder™ , see “Workflows in HDL Workflow Advisor” (HDL Coder).

```
function [boardList, workflow] = hdlcoder_board_customization
  % Board plugin registration file
  % 1. Any registration file with this name on MATLAB path will be picked up
  % 2. Registration file returns a cell array pointing to the location of
  %    the board plugins
  % 3. Board plugin must be a package folder accessible from MATLAB path,
  %    and contains a board definition file
  %
  % Copyright 2022 The MathWorks, Inc.

boardList = { ...
     'DLKCU105.plugin_board', ...
      };
workflow = hdlcoder.Workflow.DeepLearningProcessor;
end
```

2. Create the board definition file. To generate a deep learning processor, you must define the `ExternalMemorySize`. This property defines the memory size of the DDR on the target board.

```
% Copyright 2022 The MathWorks, Inc.

% Board definition of KCU105
function hB = plugin_board()
```

```
% Construct board object
hB = hdlcoder.Board;

hB.BoardName    = 'Xilinx Kintex-Ultrascale KCU105 evaluation board';

% FPGA device information
hB.FPGAVendor   = 'Xilinx';
hB.FPGAFamily   = 'KintexU';
hB.FPGADevice   = 'xcku040-ffva1156-2-e';
hB.FPGAPackage  = '';
hB.FPGASpeed    = '';

% Tool information
hB.SupportedTool = {'Xilinx Vivado'};

% FPGA JTAG chain position
hB.JTAGChainPosition = 1;

% Size of external DDR memory in bytes
hB.ExternalMemorySize = 0x80000000; % 2 GB

% Add interfaces
% Standard "External Port" interface
hB.addExternalPortInterface( ...
    'IOPadConstraint', {'IOSTANDARD = LVCMOS18'});

% Custom board external I/O interface
hB.addExternalIOInterface( ...
     'InterfaceID',    'LEDs General Purpose', ...
     'InterfaceType',  'OUT', ...
     'PortName',       'GPLEDs', ...
     'PortWidth',      8, ...
     'FPGAPin',        {'AP8', 'H23', 'P20', 'P21', 'N22', 'M22', 'R23','P23'}, ...
     'IOPadConstraint', {'IOSTANDARD = LVCMOS18'});

% Custom board external I/O interface
hB.addExternalIOInterface( ...
    'InterfaceID',    'User Push Buttons', ...
    'InterfaceType',  'IN', ...
    'PortName',       'PB', ...
    'PortWidth',      1, ...
    'FPGAPin',        {'AE10'}, ...
    'IOPadConstraint', {'IOSTANDARD = LVCMOS18'});
```

Register Custom Reference Design

Define the interface and attributes of a custom SoC reference design. To create a custom reference design:

1. Create a reference design registration file named `hdlcoder_ref_design_customization.m` that contains the list of reference design plugins associated with the board. For more information, see “Register a Custom Reference Design” (HDL Coder).

```
function [rd, boardName] = hdlcoder_ref_design_customization
% Reference design plugin registration file
% 1. The registration file with this name inside of a board plugin folder
%    will be picked up
% 2. Any registration file with this name on MATLAB path will also be picked up
% 3. The registration file returns a cell array pointing to the location of
```

```
%    the reference design plugins
% 4. The registration file also returns its associated board name
% 5. Reference design plugin must be a package folder accessible from
%    MATLAB path, and contains a reference design definition file
%
%   Copyright 2022 The MathWorks, Inc.

rd = {...
        'DLKCU105.matlab_3axi4_master_2020_1.plugin_rd', ...
  };

boardName = 'Xilinx Kintex-Ultrascale KCU105 evaluation board';

end
```

2. Create the reference design definition file. To generate a deep learning processor IP core, you must define these three AXI4 Master Interfaces:

- AXI4 Master Activation Data
- AXI4 Master Weight Data
- AXI4 Master Debug

```
function hRD = plugin_rd()
% Reference design definition

% Copyright 2022 The MathWorks, Inc.

% Construct reference design object
hRD = hdlcoder.ReferenceDesign('SynthesisTool', 'Xilinx Vivado');

hRD.ReferenceDesignName = 'AXI-Stream DDR Memory Access : 3-AXIM';
hRD.BoardName = 'Xilinx Kintex-Ultrascale KCU105 evaluation board';

% Tool information
hRD.SupportedToolVersion = {'2020.1','2020.2'};

% Add custom design files

% add custom Vivado design
hRD.addCustomVivadoDesign( ...
  'CustomBlockDesignTcl', 'system_top.tcl',...
  'VivadoBoardPart',      'xilinx.com:kcu105:part0:1.0');

% Add HDL Verifier JTAG as AXI Master IP from support package
hRD.addIPRepository( ...
  'IPListFunction','hdlverifier.fpga.vivado.iplist', ...
  'NotExistMessage', 'IP Repository not found.');

% Add interfaces

% add clock interface
hRD.addClockInterface( ...
     'ClockConnection',      'system_0/clk_out1', ...
     'ResetConnection',      'system_0/peripheral_aresetn',...
     'DefaultFrequencyMHz',  125,...
     'MinFrequencyMHz',      10,...
     'MaxFrequencyMHz',      250,...
     'ClockNumber',          1,...
     'ClockModuleInstance',  'system_0/clk_wiz_0');
```

```
% add AXI4 and AXI4-Lite slave interfaces
% This slave interface is used for intracting between DDR4 and Deep Learning IP
hRD.addAXI4SlaveInterface( ...
    'InterfaceConnection', 'system_0/M_AXI', ...
    'BaseAddress',         '0x44A00000',...
    'MasterAddressSpace',  'system_0/hdlverifier_axi_manager_0/axi4m',...
    'InterfaceType',       'AXI4');

% AXI4 Master Interface for the layer activation data with max data bit-width of 512
hRD.addAXI4MasterInterface(...
        'InterfaceID',         'AXI4 Master Activation Data', ...
        'ReadSupport',          true, ...
        'WriteSupport',         true, ...
        'MaxDataWidth',         512, ...
        'AddrWidth',            32, ...
        'InterfaceConnection', 'axi_interconnect_0/S01_AXI',...
        'TargetAddressSegments', {{'ddr4_0/C0_DDR4_MEMORY_MAP/C0_DDR4_ADDRESS_BLOCK',hex2dec('80

% AXI4 Master Interface for the layer weight data with max data bit-width of 512
hRD.addAXI4MasterInterface(...
        'InterfaceID',         'AXI4 Master Weight Data', ...
        'ReadSupport',          true, ...
        'WriteSupport',         true, ...
        'MaxDataWidth',         512, ...
        'AddrWidth',            32, ...
        'InterfaceConnection', 'axi_interconnect_0/S02_AXI',...
        'TargetAddressSegments', {{'ddr4_0/C0_DDR4_MEMORY_MAP/C0_DDR4_ADDRESS_BLOCK',hex2dec('80

% AXI4 Master Interface for the debugger with max data bit-width of 512
hRD.addAXI4MasterInterface(...
        'InterfaceID',         'AXI4 Master Debug', ...
        'ReadSupport',          true, ...
        'WriteSupport',         true, ...
        'MaxDataWidth',         512, ...
        'AddrWidth',            32, ...
        'InterfaceConnection', 'axi_interconnect_0/S03_AXI',...
        'TargetAddressSegments', {{'ddr4_0/C0_DDR4_MEMORY_MAP/C0_DDR4_ADDRESS_BLOCK',hex2dec('80
```

3. The reference design plugin file must contain information about the target interface and the deep learning processor IP core, the memory address space for the deep learning processor IP core, and a command to validate the reference design. The file also requires information on the resources consumed by the reference design. This information is used during resource estimation. Add the deep learning processor information to the reference design file:

```
% Deep learning specific properties
hRD.registerDeepLearningTargetInterface("JTAG");
hRD.registerDeepLearningMemoryAddressSpace(0x80000000, 0x80000000); % 2GB

% Resource utilization information
hRD.ResourcesUsed.LogicElements = 30500;
hRD.ResourcesUsed.DSP = 3;
hRD.ResourcesUsed.RAM = 26.5;
```

Performance Estimation

Reduce the time required to design and deploy a custom deep learning network that meets performance requirements by analyzing the layer-level latencies before deploying the network.

Estimate the performance of network for your custom board by collecting calibration data from the custom board, by:

1. Generating a calibration bitstream
2. Deploying the calibration bitstream to the target custom board
3. Retrieving the external to internal memory transaction latencies

Create a Processor Configuration object.

```
hPC = dlhdl.ProcessorConfig;
```

Specify the `TargetPlatform`. This automatically sets the `SynthesisToolChipFamily`, `SynthesisToolDeviceName`, and `ReferenceDesign` properties.

```
hPC.TargetPlatform = 'Xilinx Kintex-Ultrascale KCU105 evaluation board';
```

Set the target frequency.

```
hPC.TargetFrequency = 100;
```

This example uses a ResNet-18 pretrained network. For more details, see `resnet18`. Set the deep learning network:

```
net = resnet18;
```

To fit this design onto the target, reduce the number of parallel convolution processor kernel threads for the `conv` module to 9.

```
setModuleProperty(hPC, 'conv', 'ConvThreadNumber', 9);
```

Set the Xilinx Vivado toolpath to your design tool using the `hdlsetuptoolpath` function, then build the calibration bitstream.

```
hdlsetuptoolpath('ToolName','Xilinx Vivado','ToolPath','C:\Xilinx\Vivado\2020.2\bin\vivado.bat');
bitstreamPath = buildCalibrationBitstream(hPC);
```

Deploy the bitstream to the hardware and obtain the external- to-internal memory transaction latencies. You can use these values to get better estimates for the layer-level latencies.

```
deployCalibrationBitstream(hPC, bitstreamPath);
```

The `deployCalibrationBitstream` saves the calibration data from the hardware as a structure in the `CalibrationData` property of the `dlhdl.ProcessorConfig` object. The function also saves the calibration data as a MAT-file with the name `calibrationData.mat`. You can load this data into a new `dlhdl.ProcessorConfig` object by entering:

```
load('calibrationData.mat');
hPC.CalibrationData = calData;
```

Estimate the performance of the network for the custom processor configuration.

```
estimatePerformance(hPC, net);
```

```
            Deep Learning Processor Estimator Performance Results

                  LastFrameLatency(cycles)   LastFrameLatency(seconds)      FramesNum      Total Latency     Frames/s
                       -------------              -------------              ---------       ---------       ---------
Network                   34817713                    0.34818                   1           34817713             2.9
    ____data_norm_add       117907                    0.00118
    ____data_norm           117907                    0.00118
    ____conv1              3356175                    0.03356
    ____pool1               635032                    0.00635
    ____res2a_branch2a     1850730                    0.01851
    ____res2a_branch2b     1850730                    0.01851
    ____res2a               162115                    0.00162
    ____res2b_branch2a     1850730                    0.01851
    ____res2b_branch2b     1850730                    0.01851
    ____res2b               162115                    0.00162
    ____res3a_branch1      1012642                    0.01013
    ____res3a_branch2a     1011307                    0.01011
    ____res3a_branch2b     1599289                    0.01599
    ____res3a                79207                    0.00079
    ____res3b_branch2a     1599289                    0.01599
    ____res3b_branch2b     1599289                    0.01599
    ____res3b                79207                    0.00079
    ____res4a_branch1        823109                   0.00823
    ____res4a_branch2a       825037                   0.00825
    ____res4a_branch2b      1552695                   0.01553
    ____res4a                 39605                    0.00040
    ____res4b_branch2a      1552695                   0.01553
    ____res4b_branch2b      1552695                   0.01553
    ____res4b                 39605                    0.00040
    ____res5a_branch1       1182129                   0.01182
    ____res5a_branch2a      1186237                   0.01186
    ____res5a_branch2b      2284751                   0.02285
    ____res5a                 19703                    0.00020
    ____res5b_branch2a      2284751                   0.02285
    ____res5b_branch2b      2284751                   0.02285
    ____res5b                 19703                    0.00020
    ____pool5                 47376                    0.00047
    ____fc1000               188470                    0.00188
 * The clock frequency of the DL processor is: 100MHz
```

Resource Estimation

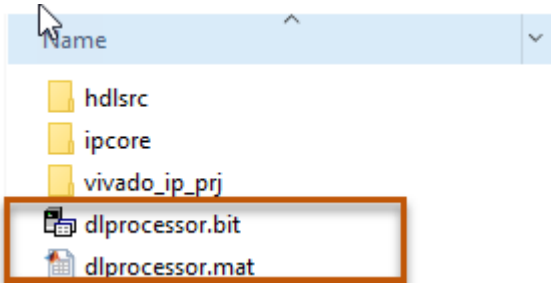
Verify that the generated bistream and network fit on your target custom board, by using `estimateResources` to estimate the resource utilization. To learn how to estimate the resource utilization for your custom boards, see “Estimate Resource Utilization for Custom Board and Reference Design”.

Generate Custom Bitstream for Custom Processor Configuration

Generate a bitstream for the custom processor configuration `hPC`.

```
dlhdl.buildProcessor(hPC);
```

Locate the bitstream file and associated MAT file at `cwd\dlhdl_prj\`, where `cwd` is your current working folder. The name of the bitstream file is `dlprocessor.bit`. The name of the MAT file is `dlprocessor.mat`. To use the generated bitstream for the supported Xilinx boards, copy the `dlprocessor.bit` and `dlprocessor.mat` files to the current working folder.



Deploy the Custom Bitstream and Run Predictions on the Network

After you generate the bitstream, deploy the network and run the predictions on the network. For more information, refer to the “Prototype Deep Learning Networks on FPGA and SoC Devices” page. For an example on prototyping, see “Bicyclist and Pedestrian Classification by Using FPGA”.

Create Target Object

Create a target object with the vendor name of the target device. Specify the interface to connect the target device to the host using the `Interface` name-value pair. This example connects to the target using the JTAG interface.

```
hT = dlhdl.Target('Xilinx', 'Interface', 'JTAG')
```

Create Workflow Object for ResNet-18 Network

Create an object of the `dlhdl.Workflow` class. Specify the network, the bitstream name, and the target object.

```
hW = dlhdl.Workflow('Network', net, 'Bitstream', 'dlprocessor.bit', 'Target', hT);
```

Compile the Network

Run the `compile` function of the `dlhdl.Workflow` object.

```
compile(hW)
```

Deploy the Bitstream to the FPGA

To deploy the network on the Xilinx KCU105 Kintex hardware, run the `deploy` function of the `dlhdl.Workflow` object.

```
deploy(hW)
```

```
### Programming FPGA Bitstream using JTAG...
### Programming the FPGA bitstream has been completed successfully.
### Loading weights to Conv Processor.
### Conv Weights loaded. Current time is 07-Jun-2022 17:44:19
### Loading weights to FC Processor.
### FC Weights loaded. Current time is 07-Jun-2022 17:44:27
```

Run Prediction for the Network

Load the sample image.

```
img = imread('sampleImage1.png');
imshow(img);
```

Run a prediction on the image. The `result` output argument contains the output of the layer preceding the `ClassificationOutputLayer` and `speed` contains the profiler table.

```
[result, speed] = predict(hW, img, 'Profile', 'on');
```

```
              Deep Learning Processor Profiler Performance Results

                    LastFrameLatency(cycles)    LastFrameLatency(seconds)       FramesNum       Total Latency     Frames/s
                          -------------               -------------             ---------         ---------      ---------
Network                     34197670                    0.34198                     1             34200121            2.9
    data_norm_add             157975                    0.00158
    data_norm                 157887                    0.00158
    conv1                    2984413                    0.02984
    pool1                     516698                    0.00517
    res2a_branch2a           1734527                    0.01735
    res2a_branch2b           1734368                    0.01734
    res2a                     224302                    0.00224
    res2b_branch2a           1734604                    0.01735
    res2b_branch2b           1734348                    0.01734
    res2b                     224312                    0.00224
    res3a_branch1             979411                    0.00979
    res3a_branch2a            974770                    0.00975
    res3a_branch2b           1561164                    0.01561
    res3a                     109624                    0.00110
    res3b_branch2a           1561119                    0.01561
    res3b_branch2b           1561136                    0.01561
    res3b                     109624                    0.00110
    res4a_branch1             817239                    0.00817
    res4a_branch2a            819776                    0.00820
    res4a_branch2b           1555224                    0.01555
    res4a                      54841                    0.00055
    res4b_branch2a           1555469                    0.01555
    res4b_branch2b           1555284                    0.01555
    res4b                      54831                    0.00055
    res5a_branch1            1197847                    0.01198
    res5a_branch2a           1199724                    0.01200
    res5a_branch2b           2331422                    0.02331
    res5a                      27324                    0.00027
    res5b_branch2a           2331507                    0.02332
    res5b_branch2b           2331802                    0.02332
    res5b                      27274                    0.00027
    pool5                      81101                    0.00081
    fc1000                    196536                    0.00197
 * The clock frequency of the DL processor is: 100MHz
```

Get the output class from the prediction.

```
[value,idx] = max(result);
classNames = net.Layers(end).Classes;
classNames(idx)
```

```
ans =

  categorical

     monitor
```

Version History

Introduced in R2022b

See Also

buildCalibrationBitstream | estimatePerformance | estimateResources | getModuleProperty | optimizeConfigurationForNetwork | setModuleProperty

estimatePerformance

Class: `dlhdl.ProcessorConfig`
Package: `dlhdl`

Retrieve layer-level latencies and performance by using `estimatePerformance` method

Syntax

```
estimatePerformance(processorConfigObject, network)
performance = estimatePerformance(processorConfigObject,network)
performance = estimatePerformance(processorConfigObject,network,Name,Value)
```

Description

`estimatePerformance(processorConfigObject, network)` returns the layer-level latencies and network performance for the object specified by the `network` argument.

`performance = estimatePerformance(processorConfigObject,network)` returns a table containing the `network` object layer-level latencies and performance.

`performance = estimatePerformance(processorConfigObject,network,Name,Value)` returns a table containing the `network` object layer-level latencies and performance, with one or more arguments specified by optional name-value pair arguments.

Input Arguments

`processorConfigObject` — Processor configuration

`dlhdl.ProcessorConfig` object

Processor configuration, specified as a `dlhdl.ProcessorConfig` object.

`network` — Network object

`SeriesNetwork` object | `DAGNetwork` object | `dlnetwork` object | `dlquantizer` object

Name of network object for performance estimate.

Example: `estimatePerformance(snet)`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

`FrameCount` — Number of frames to consider for the calculation of performance estimation

1 (default) | integer

Number of frames to consider for the calculation of performance estimation, specified as a positive number integer.

Example: `'FrameCount',10`

Output Arguments

`performance` — Network object performance

`table`

Network object performance for the `ProcessorConfig` object, returned as a table.

Examples

Estimate Performance of LogoNet Network

1. Create a file in your current working folder called `getLogoNetwork.m`. In the file, enter:

```
function net = getLogoNetwork
 if ~isfile('LogoNet.mat')
        url = 'https://www.mathworks.com/supportfiles/gpucoder/cnn_models/logo_detection/LogoNet.mat';
        websave('LogoNet.mat',url);
    end
    data = load('LogoNet.mat');
    net  = data.convnet;
end
```

2. Create a `dlhdl.ProcessorConfig` object.

```
snet = getLogoNetwork;
hPC = dlhdl.ProcessorConfig;
```

3. To retrieve the layer-level latencies and performance for the LogoNet network, call the `estimatePerformance` method.

```
hPC.estimatePerformance(snet)
```

```
### Notice: The layer 'imageinput' of type 'ImageInputLayer' is split into an image input layer 'imageinput' and an addition layer
### Notice: The layer 'softmax' with type 'nnet.cnn.layer.SoftmaxLayer' is implemented in software.
### Notice: The layer 'classoutput' with type 'nnet.cnn.layer.ClassificationOutputLayer' is implemented in software.


             Deep Learning Processor Estimator Performance Results

                  LastFrameLatency(cycles)   LastFrameLatency(seconds)       FramesNum      Total Latency     Frames/s
                  -------------              -------------                   ---------      ---------         ---------
Network                  38810151                 0.19405                       1           38810151              5.2
    ____imageinput_norm     216472                 0.00108
    ____conv_1             6829224                 0.03415
    ____maxpool_1          3705912                 0.01853
    ____conv_2            10454501                 0.05227
    ____maxpool_2          1173810                 0.00587
    ____conv_3             9364533                 0.04682
    ____maxpool_3          1229970                 0.00615
    ____conv_4             1384564                 0.00692
    ____maxpool_4            24450                 0.00012
    ____fc_1               2644886                 0.01322
    ____fc_2               1692534                 0.00846
    ____fc_3                 89295                 0.00045
 * The clock frequency of the DL processor is: 200MHz
```

Estimate Performance of ResNet-18 for Multiple Frame Inputs

Estimate the performance of the ResNet-18 network for multiple frames by using the `dlhdl.ProcessorConfig` object.

Load the ResNet-18 network and save it to *net*

```
net = resnet18;
```

Create a `dlhdl.ProcessorConfig` object and save to *hPC*

```
hPC = dlhdl.ProcessorConfig;
```

Retrieve layer level latencies and performance in frames per second (FPS) for multiple frames by using the `estimatePerformance` method with `FrameNumber` as an optional input argument.

```
hPC.estimatePerformance(net,'FrameCount',10);
```

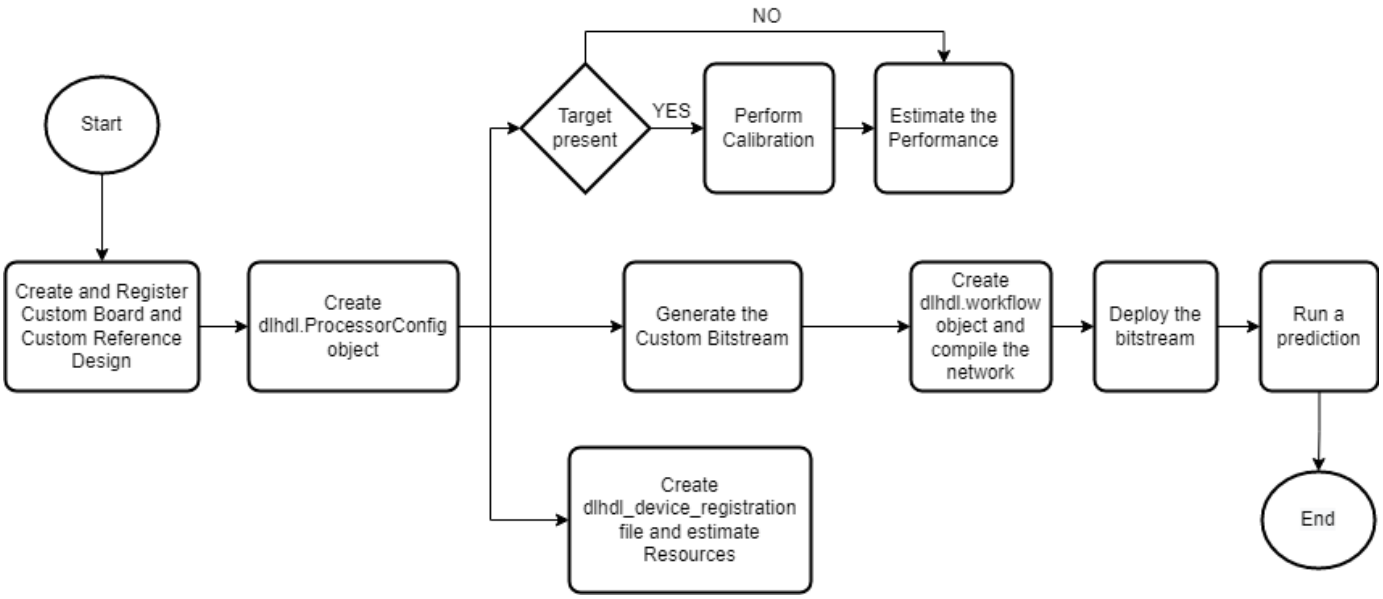
```
### Optimizing network: Fused 'nnet.cnn.layer.BatchNormalizationLayer' into 'nnet.cnn.layer.Convolution2DLayer'
### Notice: The layer 'data' of type 'ImageInputLayer' is split into an image input layer 'data', an addition layer 'data_norm_add', and
### Notice: The layer 'prob' with type 'nnet.cnn.layer.SoftmaxLayer' is implemented in software.
### Notice: The layer 'ClassificationLayer_predictions' with type 'nnet.cnn.layer.ClassificationOutputLayer' is implemented in software.


             Deep Learning Processor Estimator Performance Results

                  LastFrameLatency(cycles)   LastFrameLatency(seconds)       FramesNum      Total Latency     Frames/s
                       -------------              -------------               ---------        ---------       ---------
Network                 21328236                 0.10664                     10          211208850             9.5
    ____data_norm_add       210750                 0.00105
    ____data_norm           210750                 0.00105
    ____conv1              2164124                 0.01082
    ____pool1               515064                 0.00258
    ____res2a_branch2a      966221                 0.00483
    ____res2a_branch2b      966221                 0.00483
    ____res2a               210750                 0.00105
    ____res2b_branch2a      966221                 0.00483
    ____res2b_branch2b      966221                 0.00483
    ____res2b               210750                 0.00105
    ____res3a_branch1       540861                 0.00270
    ____res3a_branch2a      540749                 0.00270
    ____res3a_branch2b      919117                 0.00460
    ____res3a               105404                 0.00053
    ____res3b_branch2a      919117                 0.00460
    ____res3b_branch2b      919117                 0.00460
    ____res3b               105404                 0.00053
    ____res4a_branch1       503405                 0.00252
    ____res4a_branch2a      509261                 0.00255
    ____res4a_branch2b      905421                 0.00453
    ____res4a                52724                 0.00026
    ____res4b_branch2a      905421                 0.00453
    ____res4b_branch2b      905421                 0.00453
    ____res4b                52724                 0.00026
    ____res5a_branch1       744525                 0.00372
    ____res5a_branch2a      751693                 0.00376
    ____res5a_branch2b     1415373                 0.00708
    ____res5a                26368                 0.00013
    ____res5b_branch2a     1415373                 0.00708
    ____res5b_branch2b     1415373                 0.00708
    ____res5b                26368                 0.00013
    ____pool5                54594                 0.00027
    ____fc1000              207351                 0.00104
 * The clock frequency of the DL processor is: 200MHz
```

Deep Learning Processor IP Core Generation for Custom Board

This example shows how to create custom board and generate a deep learning processor IP core for the custom board. In this example you:

- Create a custom board and reference design
- Estimate the network performance and board resource utilization
- Generate a custom processor and bitstream
- Deploy the network by using the custom bitstream

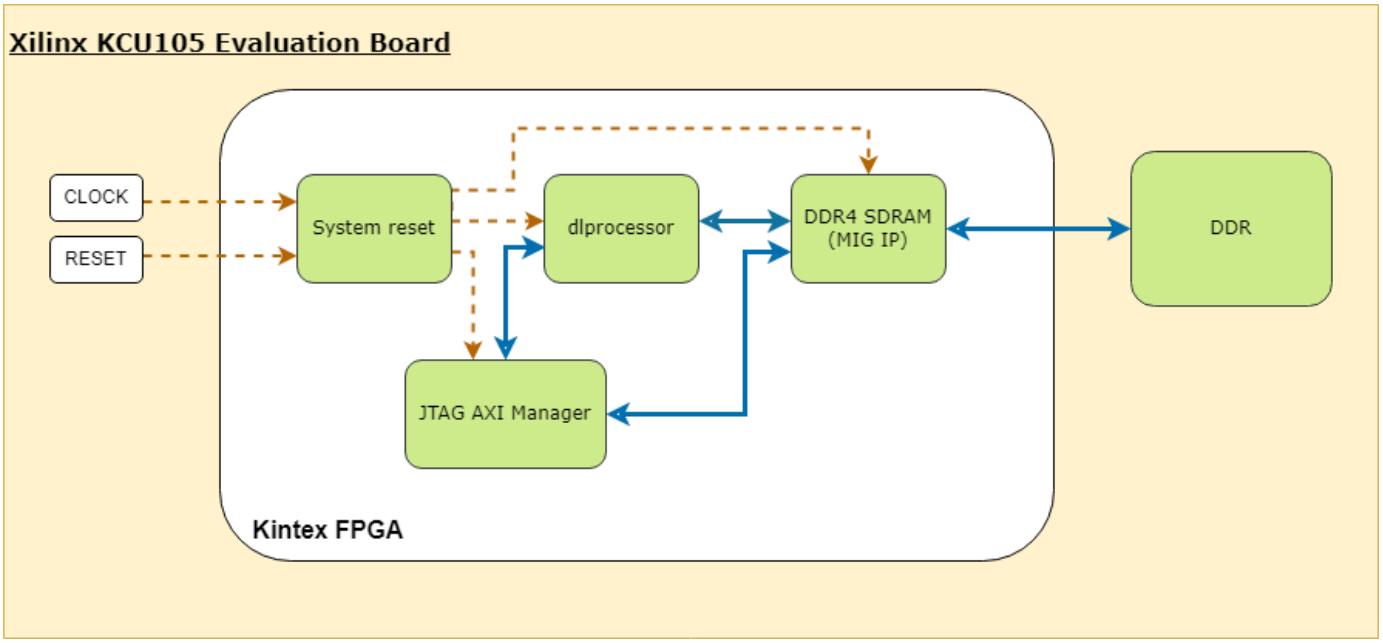
The image shows the process of deploying a network to a custom board and retrieving a prediction from the deployed network.



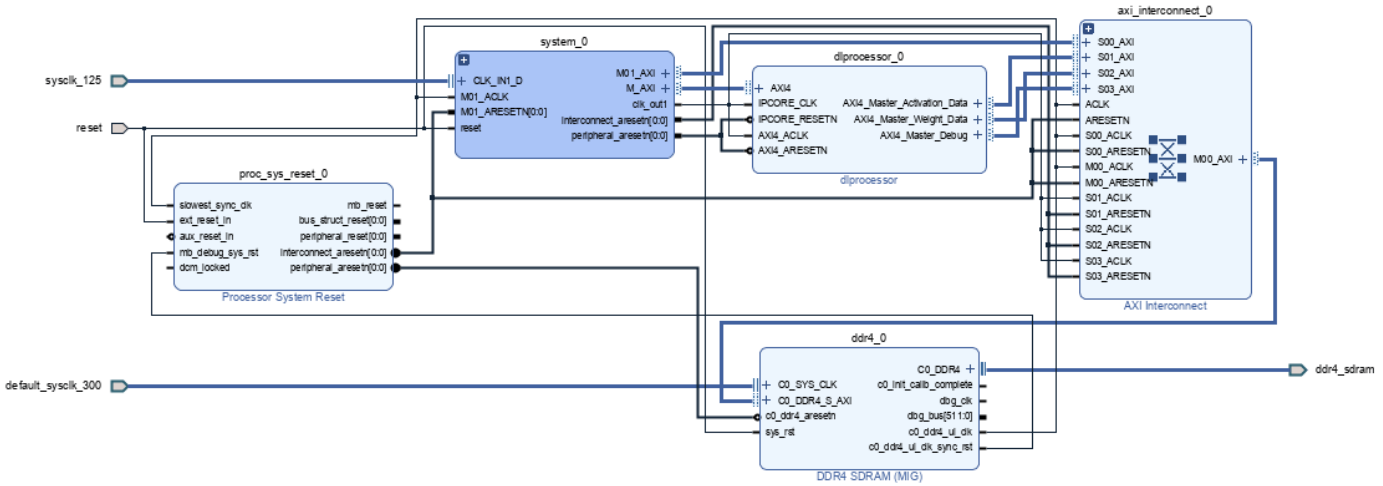
This example uses the Xilinx® Kintex® UltraScale™ KCU105 board. The board contains these blocks:

- System reset block — Used to feed the clock and reset signals to the design.
- Memory Interface Generator (MIG) IP block — Used to generate memory controllers and interfaces for Xilinx FPGAs.
- MATLAB JTAG AXI Manager block — Used by MATLAB to access onboard memory location. For more details, see “Set Up AXI Manager” (HDL Verifier).

Integrate the generated deep learning processor IP core into your reference design. For more details, see “Board and Reference Design Registration System” (HDL Coder).



This image shows the generated deep learning processor IP core `dl_processor0` integrated into the reference design.



Register Custom Board

Define the interface and attributes of a custom SoC board. To register the Xilinx® Kintex® UltraScale™ KCU105 board:

1. Create a board registration file with the name `hdlcoder_board_customization.m` and add it to the MATLAB path. The `hdlcoder_board_customization.m` function must return a second output. For more information, see “Register a Custom Board” (HDL Coder).

Set the target workflow to `DeepLearningProcessor`. For information on other target workflows supported by HDL Coder™ , see “Workflows in HDL Workflow Advisor” (HDL Coder).

```
function [boardList, workflow] = hdlcoder_board_customization
  % Board plugin registration file
  % 1. Any registration file with this name on MATLAB path will be picked up
  % 2. Registration file returns a cell array pointing to the location of
  %    the board plugins
  % 3. Board plugin must be a package folder accessible from MATLAB path,
  %    and contains a board definition file
  %
  % Copyright 2022 The MathWorks, Inc.

boardList = { ...
     'DLKCU105.plugin_board', ...
      };
workflow = hdlcoder.Workflow.DeepLearningProcessor;
end
```

2. Create the board definition file. To generate a deep learning processor, you must define the `ExternalMemorySize`. This property defines the memory size of the DDR on the target board.

```
% Copyright 2022 The MathWorks, Inc.

% Board definition of KCU105
function hB = plugin_board()

% Construct board object
hB = hdlcoder.Board;

hB.BoardName    = 'Xilinx Kintex-Ultrascale KCU105 evaluation board';

% FPGA device information
hB.FPGAVendor   = 'Xilinx';
hB.FPGAFamily   = 'KintexU';
hB.FPGADevice   = 'xcku040-ffva1156-2-e';
hB.FPGAPackage  = '';
hB.FPGASpeed    = '';

% Tool information
hB.SupportedTool = {'Xilinx Vivado'};

% FPGA JTAG chain position
hB.JTAGChainPosition = 1;

% Size of external DDR memory in bytes
hB.ExternalMemorySize = 0x80000000; % 2 GB

% Add interfaces
% Standard "External Port" interface
hB.addExternalPortInterface( ...
    'IOPadConstraint', {'IOSTANDARD = LVCMOS18'});

% Custom board external I/O interface
hB.addExternalIOInterface( ...
     'InterfaceID',    'LEDs General Purpose', ...
     'InterfaceType',  'OUT', ...
     'PortName',       'GPLEDs', ...
     'PortWidth',      8, ...
     'FPGAPin',        {'AP8', 'H23', 'P20', 'P21', 'N22', 'M22', 'R23','P23'}, ...
     'IOPadConstraint', {'IOSTANDARD = LVCMOS18'});

% Custom board external I/O interface
hB.addExternalIOInterface( ...
```

```
    'InterfaceID',    'User Push Buttons', ...
    'InterfaceType',  'IN', ...
    'PortName',       'PB', ...
    'PortWidth',      1, ...
    'FPGAPin',        {'AE10'}, ...
    'IOPadConstraint', {'IOSTANDARD = LVCMOS18'});
```

Register Custom Reference Design

Define the interface and attributes of a custom SoC reference design. To create a custom reference design:

1. Create a reference design registration file named `hdlcoder_ref_design_customization.m` that contains the list of reference design plugins associated with the board. For more information, see “Register a Custom Reference Design” (HDL Coder).

```
function [rd, boardName] = hdlcoder_ref_design_customization
% Reference design plugin registration file
% 1. The registration file with this name inside of a board plugin folder
%    will be picked up
% 2. Any registration file with this name on MATLAB path will also be picked up
% 3. The registration file returns a cell array pointing to the location of
%    the reference design plugins
% 4. The registration file also returns its associated board name
% 5. Reference design plugin must be a package folder accessible from
%    MATLAB path, and contains a reference design definition file
%
%   Copyright 2022 The MathWorks, Inc.

rd = {...
        'DLKCU105.matlab_3axi4_master_2020_1.plugin_rd', ...
  };

boardName = 'Xilinx Kintex-Ultrascale KCU105 evaluation board';

end
```

2. Create the reference design definition file. To generate a deep learning processor IP core, you must define these three AXI4 Master Interfaces:

- AXI4 Master Activation Data
- AXI4 Master Weight Data
- AXI4 Master Debug

```
function hRD = plugin_rd()
% Reference design definition

% Copyright 2022 The MathWorks, Inc.

% Construct reference design object
hRD = hdlcoder.ReferenceDesign('SynthesisTool', 'Xilinx Vivado');

hRD.ReferenceDesignName = 'AXI-Stream DDR Memory Access : 3-AXIM';
hRD.BoardName = 'Xilinx Kintex-Ultrascale KCU105 evaluation board';

% Tool information
hRD.SupportedToolVersion = {'2020.1','2020.2'};

% Add custom design files
```

```
% add custom Vivado design
hRD.addCustomVivadoDesign( ...
  'CustomBlockDesignTcl', 'system_top.tcl',...
  'VivadoBoardPart',      'xilinx.com:kcu105:part0:1.0');

% Add HDL Verifier JTAG as AXI Master IP from support package
hRD.addIPRepository( ...
  'IPListFunction','hdlverifier.fpga.vivado.iplist', ...
  'NotExistMessage', 'IP Repository not found.');

% Add interfaces

% add clock interface
hRD.addClockInterface( ...
     'ClockConnection',      'system_0/clk_out1', ...
     'ResetConnection',      'system_0/peripheral_aresetn',...
     'DefaultFrequencyMHz',  125,...
     'MinFrequencyMHz',      10,...
     'MaxFrequencyMHz',      250,...
     'ClockNumber',          1,...
     'ClockModuleInstance',  'system_0/clk_wiz_0');

% add AXI4 and AXI4-Lite slave interfaces
% This slave interface is used for intracting between DDR4 and Deep Learning IP
hRD.addAXI4SlaveInterface( ...
    'InterfaceConnection', 'system_0/M_AXI', ...
    'BaseAddress',         '0x44A00000',...
    'MasterAddressSpace',  'system_0/hdlverifier_axi_manager_0/axi4m',...
    'InterfaceType',       'AXI4');

% AXI4 Master Interface for the layer activation data with max data bit-width of 512
hRD.addAXI4MasterInterface(...
        'InterfaceID',         'AXI4 Master Activation Data', ...
        'ReadSupport',          true, ...
        'WriteSupport',         true, ...
        'MaxDataWidth',         512, ...
        'AddrWidth',            32, ...
        'InterfaceConnection', 'axi_interconnect_0/S01_AXI',...
        'TargetAddressSegments', {{'ddr4_0/C0_DDR4_MEMORY_MAP/C0_DDR4_ADDRESS_BLOCK',hex2dec('80

% AXI4 Master Interface for the layer weight data with max data bit-width of 512
hRD.addAXI4MasterInterface(...
        'InterfaceID',         'AXI4 Master Weight Data', ...
        'ReadSupport',          true, ...
        'WriteSupport',         true, ...
        'MaxDataWidth',         512, ...
        'AddrWidth',            32, ...
        'InterfaceConnection', 'axi_interconnect_0/S02_AXI',...
        'TargetAddressSegments', {{'ddr4_0/C0_DDR4_MEMORY_MAP/C0_DDR4_ADDRESS_BLOCK',hex2dec('80

% AXI4 Master Interface for the debugger with max data bit-width of 512
hRD.addAXI4MasterInterface(...
        'InterfaceID',         'AXI4 Master Debug', ...
        'ReadSupport',          true, ...
        'WriteSupport',         true, ...
        'MaxDataWidth',         512, ...
        'AddrWidth',            32, ...
        'InterfaceConnection', 'axi_interconnect_0/S03_AXI',...
        'TargetAddressSegments', {{'ddr4_0/C0_DDR4_MEMORY_MAP/C0_DDR4_ADDRESS_BLOCK',hex2dec('80
```

3. The reference design plugin file must contain information about the target interface and the deep learning processor IP core, the memory address space for the deep learning processor IP core, and a command to validate the reference design. The file also requires information on the resources consumed by the reference design. This information is used during resource estimation. Add the deep learning processor information to the reference design file:

```
% Deep learning specific properties
hRD.registerDeepLearningTargetInterface("JTAG");
hRD.registerDeepLearningMemoryAddressSpace(0x80000000, 0x80000000); % 2GB

% Resource utilization information
hRD.ResourcesUsed.LogicElements = 30500;
hRD.ResourcesUsed.DSP = 3;
hRD.ResourcesUsed.RAM = 26.5;
```

Performance Estimation

Reduce the time required to design and deploy a custom deep learning network that meets performance requirements by analyzing the layer-level latencies before deploying the network.

Estimate the performance of network for your custom board by collecting calibration data from the custom board, by:

1. Generating a calibration bitstream
2. Deploying the calibration bitstream to the target custom board
3. Retrieving the external to internal memory transaction latencies

Create a Processor Configuration object.

```
hPC = dlhdl.ProcessorConfig;
```

Specify the `TargetPlatform`. This automatically sets the `SynthesisToolChipFamily`, `SynthesisToolDeviceName`, and `ReferenceDesign` properties.

```
hPC.TargetPlatform = 'Xilinx Kintex-Ultrascale KCU105 evaluation board';
```

Set the target frequency.

```
hPC.TargetFrequency = 100;
```

This example uses a ResNet-18 pretrained network. For more details, see `resnet18`. Set the deep learning network:

```
net = resnet18;
```

To fit this design onto the target, reduce the number of parallel convolution processor kernel threads for the `conv` module to 9.

```
setModuleProperty(hPC, 'conv', 'ConvThreadNumber', 9);
```

Set the Xilinx Vivado toolpath to your design tool using the `hdlsetuptoolpath` function, then build the calibration bitstream.

```
hdlsetuptoolpath('ToolName','Xilinx Vivado','ToolPath','C:\Xilinx\Vivado\2020.2\bin\vivado.bat');
bitstreamPath = buildCalibrationBitstream(hPC);
```

Deploy the bitstream to the hardware and obtain the external- to-internal memory transaction latencies. You can use these values to get better estimates for the layer-level latencies.

```
deployCalibrationBitstream(hPC, bitstreamPath);
```

The `deployCalibrationBitstream` saves the calibration data from the hardware as a structure in the `CalibrationData` property of the `dlhdl.ProcessorConfig` object. The function also saves the calibration data as a MAT-file with the name `calibrationData.mat`. You can load this data into a new `dlhdl.ProcessorConfig` object by entering:

```
load('calibrationData.mat');
hPC.CalibrationData = calData;
```

Estimate the performance of the network for the custom processor configuration.

```
estimatePerformance(hPC, net);
```

```
            Deep Learning Processor Estimator Performance Results

                 LastFrameLatency(cycles)   LastFrameLatency(seconds)       FramesNum      Total Latency     Frames/s
                      -------------              -------------              ---------        ---------      ---------
Network                   34817713                    0.34818                   1           34817713            2.9
    ____data_norm_add       117907                    0.00118
    ____data_norm           117907                    0.00118
    ____conv1              3356175                    0.03356
    ____pool1               635032                    0.00635
    ____res2a_branch2a     1850730                    0.01851
    ____res2a_branch2b     1850730                    0.01851
    ____res2a               162115                    0.00162
    ____res2b_branch2a     1850730                    0.01851
    ____res2b_branch2b     1850730                    0.01851
    ____res2b               162115                    0.00162
    ____res3a_branch1      1012642                    0.01013
    ____res3a_branch2a     1011307                    0.01011
    ____res3a_branch2b     1599289                    0.01599
    ____res3a                79207                    0.00079
    ____res3b_branch2a     1599289                    0.01599
    ____res3b_branch2b     1599289                    0.01599
    ____res3b                79207                    0.00079
    ____res4a_branch1        823109                    0.00823
    ____res4a_branch2a       825037                    0.00825
    ____res4a_branch2b      1552695                    0.01553
    ____res4a                39605                    0.00040
    ____res4b_branch2a      1552695                    0.01553
    ____res4b_branch2b      1552695                    0.01553
    ____res4b                39605                    0.00040
    ____res5a_branch1       1182129                    0.01182
    ____res5a_branch2a      1186237                    0.01186
    ____res5a_branch2b      2284751                    0.02285
    ____res5a                19703                    0.00020
    ____res5b_branch2a      2284751                    0.02285
    ____res5b_branch2b      2284751                    0.02285
    ____res5b                19703                    0.00020
    ____pool5                47376                    0.00047
    ____fc1000              188470                    0.00188
 * The clock frequency of the DL processor is: 100MHz
```

Resource Estimation

Verify that the generated bistream and network fit on your target custom board, by using `estimateResources` to estimate the resource utilization. To learn how to estimate the resource

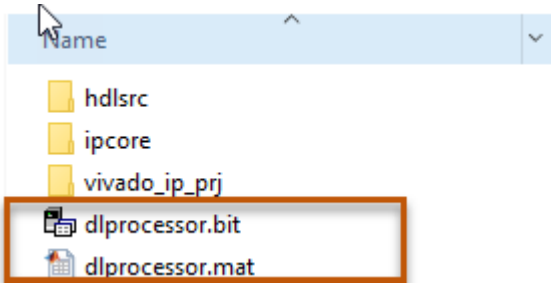
utilization for your custom boards, see “Estimate Resource Utilization for Custom Board and Reference Design”.

Generate Custom Bitstream for Custom Processor Configuration

Generate a bitstream for the custom processor configuration `hPC`.

```
dlhdl.buildProcessor(hPC);
```

Locate the bitstream file and associated MAT file at `cwd\dlhdl_prj\`, where `cwd` is your current working folder. The name of the bitstream file is `dlprocessor.bit`. The name of the MAT file is `dlprocessor.mat`. To use the generated bitstream for the supported Xilinx boards, copy the `dlprocessor.bit` and `dlprocessor.mat` files to the current working folder.



Deploy the Custom Bitstream and Run Predictions on the Network

After you generate the bitstream, deploy the network and run the predictions on the network. For more information, refer to the “Prototype Deep Learning Networks on FPGA and SoC Devices” page. For an example on prototyping, see “Bicyclist and Pedestrian Classification by Using FPGA”.

Create Target Object

Create a target object with the vendor name of the target device. Specify the interface to connect the target device to the host using the `Interface` name-value pair. This example connects to the target using the JTAG interface.

```
hT = dlhdl.Target('Xilinx', 'Interface', 'JTAG')
```

Create Workflow Object for ResNet-18 Network

Create an object of the `dlhdl.Workflow` class. Specify the network, the bitstream name, and the target object.

```
hW = dlhdl.Workflow('Network', net, 'Bitstream', 'dlprocessor.bit', 'Target', hT);
```

Compile the Network

Run the `compile` function of the `dlhdl.Workflow` object.

```
compile(hW)
```

Deploy the Bitstream to the FPGA

To deploy the network on the Xilinx KCU105 Kintex hardware, run the `deploy` function of the `dlhdl.Workflow` object.

```
deploy(hW)
```

```
### Programming FPGA Bitstream using JTAG...
### Programming the FPGA bitstream has been completed successfully.
### Loading weights to Conv Processor.
### Conv Weights loaded. Current time is 07-Jun-2022 17:44:19
### Loading weights to FC Processor.
### FC Weights loaded. Current time is 07-Jun-2022 17:44:27
```

Run Prediction for the Network

Load the sample image.

```
img = imread('sampleImage1.png');
imshow(img);
```

Run a prediction on the image. The `result` output argument contains the output of the layer preceding the `ClassificationOutputLayer` and `speed` contains the profiler table.

```
[result, speed] = predict(hW, img, 'Profile', 'on');
```

```
              Deep Learning Processor Profiler Performance Results

                    LastFrameLatency(cycles)    LastFrameLatency(seconds)       FramesNum       Total Latency     Frames/s
                          -------------               -------------             ---------         ---------      ---------
Network                     34197670                     0.34198                    1             34200121            2.9
    data_norm_add             157975                     0.00158
    data_norm                 157887                     0.00158
    conv1                    2984413                     0.02984
    pool1                     516698                     0.00517
    res2a_branch2a           1734527                     0.01735
    res2a_branch2b           1734368                     0.01734
    res2a                     224302                     0.00224
    res2b_branch2a           1734604                     0.01735
    res2b_branch2b           1734348                     0.01734
    res2b                     224312                     0.00224
    res3a_branch1             979411                     0.00979
    res3a_branch2a            974770                     0.00975
    res3a_branch2b           1561164                     0.01561
    res3a                     109624                     0.00110
    res3b_branch2a           1561119                     0.01561
    res3b_branch2b           1561136                     0.01561
    res3b                     109624                     0.00110
    res4a_branch1             817239                     0.00817
    res4a_branch2a            819776                     0.00820
    res4a_branch2b           1555224                     0.01555
    res4a                      54841                     0.00055
    res4b_branch2a           1555469                     0.01555
    res4b_branch2b           1555284                     0.01555
    res4b                      54831                     0.00055
    res5a_branch1            1197847                     0.01198
    res5a_branch2a           1199724                     0.01200
    res5a_branch2b           2331422                     0.02331
    res5a                      27324                     0.00027
    res5b_branch2a           2331507                     0.02332
    res5b_branch2b           2331802                     0.02332
    res5b                      27274                     0.00027
    pool5                      81101                     0.00081
    fc1000                    196536                     0.00197
 * The clock frequency of the DL processor is: 100MHz
```

Get the output class from the prediction.

```
[value,idx] = max(result);
classNames = net.Layers(end).Classes;
classNames(idx)
```

```
ans =

  categorical

     monitor
```

Limitations

When you estimate the performance Deep Learning HDL Toolbox assumes that the deep learning processor is connected to the DDR memory dedicated for the programmable logic (PL) portion of the FPGA or SoC and does not account for the DDR memory communications arbitration with other components.

Tips

To obtain the performance estimation for a `dlquantizer` object, set the `dlhdl.ProcessorConfig` object `ProcessorDataType` to `int8`.

Version History

Introduced in R2021a

See Also

`buildCalibrationBitstream` | `deployCalibrationBitstream` | `estimateResources` | `getModuleProperty` | `optimizeConfigurationForNetwork` | `setModuleProperty`

Topics

“Estimate Performance of Deep Learning Network”

estimateResources

Class: `dlhdl.ProcessorConfig`
Package: `dlhdl`

Return estimated resources used by custom bitstream configuration

Syntax

```
estimateResources(processorConfigObject)
resources = estimateResources(processorConfigObject)
estimateResources(processorConfigObject,Name,Value)
resources = estimateResources(processorConfigObject,Name,Value)
```

Description

`estimateResources(processorConfigObject)` returns the estimated resources used by the custom bitstream configuration.

`resources = estimateResources(processorConfigObject)` returns a table containing the estimated resources used by the custom bitstream configuration.

`estimateResources(processorConfigObject,Name,Value)` returns the estimated resources used by the custom bitstream configuration, with additional options specified by one or more name-value arguments.

“Estimate Resource Utilization for Custom Board and Reference Design” on page 1-201

`resources = estimateResources(processorConfigObject,Name,Value)` returns the estimated resources used by the custom bitstream configuration, with additional options specified by one or more name-value arguments.

Input Arguments

`processorConfigObject` — Processor configuration

`dlhdl.ProcessorConfig` object

Processor configuration, specified as a `dlhdl.ProcessorConfig` object.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

`IncludeReferenceDesign` — Display reference design resource utilization

false (default) | false | true | logical

Flag to enable the display of reference design resource utilization specified as a `logical` data type.

Example: `'IncludeReferenceDesign',true`

Output Arguments

`resources` — Custom processor configuration object resource utilization

`table`

Resources used by the custom bitstream configuration, returned as a table.

Examples

Estimate Resources Used by Default Custom Processor Configuration

1. Create a default custom processor configuration object. Use the `dlhdl.ProcessorConfig` class.

   ```
   hPC = dlhdl.ProcessorConfig;
   ```
2. To retrieve the resources used by the custom processor configuration, call the `estimateResources` method.

   ```
   hPC.estimateResources;
   ```
3. Calling `estimateResources` returns these results:

```
              Deep Learning Processor Estimator Resource Results

                           DSPs          Block RAM*     LUTs(CLB/ALUT)
                      -------------    -------------    -------------
Available                   2520             912           274080
                      -------------    -------------    -------------
DL_Processor               381( 16%)        508( 56%)     216119( 79%)
* Block RAM represents Block RAM tiles in Xilinx devices and Block RAM bits in Intel devices
```

Estimate Resource Utilization for Custom Board and Reference Design

Rapidly prototype the deployment of deep learning networks to your custom board by using the `estimateResources` function. Estimate the resource utilization of the deep learning processor configuration for your custom board. Optimize the integration of custom IP cores and reference design into your system by using the `estimateResources` function to estimate the resource utilization of your reference design. The synthesis tool that you use must be in the list of tools supported by the `SynthesisTool` property of the `dlhdl.ProcessorConfig` object. For a list of supported tools and device families, see “SynthesisTool” on page 1-0 and “SynthesisToolChipFamily” on page 1-0 .

In this example, estimate the resource utilization for your custom board that has the Kintex® Ultrascale+™ chip family. Also estimate the resource utilization of the reference design for the Xilinx® Zynq® Ultrascale+™ MPSoC ZCU102 board.

Prerequisites

- Deep Learning HDL Toolbox™
- Deep Learning HDL Toolbox™ Support Package for Xilinx FPGA and SoC
- HDL Coder™

Estimate Resource Utilization for Kintex® Ultrascale™ Board

To estimate the resource utilization for your custom board that has a Kintex® Ultrascale™ chip family , use the `estimateResource` function of the `dlhdl.ProcessorConfig` object.

1. Add the `dlhdl_device_registration.m` file to the MATLAB® search path.
2. Create a `dlhdl.ProcessorConfig` object.
3. Update the `SynthesisToolChipFamily` and `SynthesisToolDeviceName` properties of the `dlhdl.ProcessorConfig` object.
4. Use the `estimateResources` function to retrieve the resource utilization for your custom board.

Deep Learning HDL Toolbox™ does not support lookup table (LUT) estimation for custom boards.

```
hPC = dlhdl.ProcessorConfig;
hPC.SynthesisToolChipFamily = 'KintexU';
hPC.SynthesisToolDeviceName = 'xcku040-ffva1156-2-e';
hPC.estimateResources
```

```
Warning: Device family "KintexU" is not currently supported for LUT Estimation. Supported familie

              Deep Learning Processor Estimator Resource Results

                           DSPs          Block RAM*     LUTs(CLB/ALUT)
                       ------------    ------------    ------------
Available                   1920             600           242400
                       ------------    ------------    ------------
DL_Processor               381( 20%)        508( 85%)          0(  0%)
* Block RAM represents Block RAM tiles in Xilinx devices and Block RAM bits in Intel devices
```

Estimate Resource Utilization for Custom Reference Design

Estimate the resource utilization for a reference design that you want to integrate into your system that has a Xilinx® Zynq® Ultrascale+™ MPSoC ZCU102 board. Use the `estimateResource` function with the `IncludeReferenceDesign` name-value argument. The `estimateResources` function uses the `ResourcesUsed.LogicElements`, `ResourcesUsed.DSP`, and `ResourcesUsed.RAM` information in the reference design plugin file to perform the resource estimation. To estimate resource utilization for your custom reference design, you must populate your reference design file with values for `ResourcesUsed.LogicElements`, `ResourcesUsed.DSP`, and `ResourcesUsed.RAM`. See “ResourcesUsed” on page 1-0 . The reference design used in this code is located at `$supportpackageinstallationfolder/Xilinx/boards/+DLZCU102/+matlab_libiio_3axi4_master_2019_1/plugin_rd.m`.

```
hPC_referencedesign = dlhdl.ProcessorConfig;
hPC_referencedesign.estimateResources('IncludeReferenceDesign',true)
```

```
              Deep Learning Processor Estimator Resource Results

                           DSPs          Block RAM*     LUTs(CLB/ALUT)
                       ------------    ------------    ------------
Available                   2520             912           274080
                       ------------    ------------    ------------
Total                      384( 16%)        586( 65%)     251119( 92%)
ReferenceDesign              3(  1%)         78(  9%)      35000( 13%)
DL_Processor               381( 16%)        508( 56%)     216119( 79%)
* Block RAM represents Block RAM tiles in Xilinx devices and Block RAM bits in Intel devices
```

The `estimateResources` function returns the resource utilization for the reference design and for the deep learning processor configuration.

Supporting Files

Device Registration File

Use the `dlhdl_device_registration.m` file to register a custom device family. Estimate the resource utilization of the custom device by using the `estimateResources` function.

```
type dlhdl_device_registration.m
```

```
function hFPGADeviceFamily = dlhdl_device_registration
% Register a new device family by providing the following details:
% 1. Device Family Name
% 2. Vendor(Intel/Xilinx)
% 3. DSP Width
% 4. RAM Width
% 5. RAM Depth
% 6. SplitDSP Width(Optional) - alternative DSP Width supported by the DSP macro
% 7. SplitRAM Width(Optional) - alternative RAM Width supported by the RAM macro

hFPGADeviceFamily = { ...
    kintex_ultrascale();...
    };
end

function hFPGADeviceFamily = kintex_ultrascale()
    % Datasheets :
    % https://www.xilinx.com/support/documentation/user_guides/ug579-ultrascale-dsp.pdf
    % https://www.xilinx.com/support/documentation/user_guides/ug573-ultrascale-memory-resources
    hFPGADeviceFamily = hdlcoder.FPGADeviceInfo('Name', 'KintexU');
    hFPGADeviceFamily.Vendor = 'Xilinx';
    hFPGADeviceFamily.DSPWidth = [27, 18];
    hFPGADeviceFamily.RAMWidth = 36;
    hFPGADeviceFamily.SplitRAMWidth = 18;
    hFPGADeviceFamily.RAMDepth = 1024;
end
```

Version History

Introduced in R2021a

See Also

`estimatePerformance` | `getModuleProperty` | `optimizeConfigurationForNetwork` | `setModuleProperty`

Topics

“Estimate Resource Utilization for Custom Processor Configuration”

getModuleProperty

Class: `dlhdl.ProcessorConfig`
Package: `dlhdl`

Use the `getModuleProperty` method to get values of module properties within the `dlhdl.ProcessorConfig` object

Syntax

```
getModuleProperty(processorConfigObject,ModuleName,ModulePropertyName)
```

Description

The `getModuleProperty(processorConfigObject,ModuleName,ModulePropertyName)` method returns the value of the module property for modules within the `dlhdl.ProcessorConfig` object.

Input Arguments

`processorConfigObject` — Processor configuration
`dlhdl.ProcessorConfig` object

Processor configuration, specified as a `dlhdl.ProcessorConfig` object.

`ModuleName` — Name of the module whose parameters are to be retrieved
"conv" | "fc" | "custom" | 'conv' | 'fc' | 'custom' | character vector | string

The `dlhdl.ProcessorConfig` object module name, specified as a character vector or string.

`ModulePropertyName` — Name of the module property whose value is to be retrieved
character vector or string

`'conv'`, `'fc'`, or `'custom'` module properties specified as character vector.

Example: "ConvThreadNumber" or 'ConvThreadNumber'

This table lists module names and module property names.

Module Name	Module Property Name
conv	“ModuleGeneration” on page 1-0
conv	“LRNBlockGeneration” on page 1-0
conv	“SegmentationBlockGeneration” on page 1-0
conv	“ConvThreadNumber” on page 1-0
conv	“InputMemorySize” on page 1-0
conv	“OutputMemorySize” on page 1-0
conv	“FeatureSizeLimit” on page 1-0

`fc`	“ModuleGeneration” on page 1-0
`fc`	“SoftmaxBlockGeneration” on page 1-0
`fc`	“SigmoidBlockGeneration” on page 1-0
`fc`	“FCThreadNumber” on page 1-0
`fc`	“InputMemorySize” on page 1-0
`fc`	“OutputMemorySize” on page 1-0
`custom`	“ModuleGeneration” on page 1-0
`custom`	“Addition” on page 1-0
`custom`	“Multiplication” on page 1-0
`custom`	“Resize2D” on page 1-0
`custom`	“Sigmoid” on page 1-0
`custom`	“TanhLayer” on page 1-0
`custom`	“InputMemorySize” on page 1-0
`custom`	“OutputMemorySize” on page 1-0

Examples

Retrieve `ConvThreadNumber` for `conv` Module Inside `dlhdl.ProcessorConfig` Object

1. Create an example object by using the `dlhdl.ProcessorConfig` class, and then use the `getModuleProperty` method to obtain the `ConvThreadNumber`.

```
hPC = dlhdl.ProcessorConfig;
hPC.getModuleProperty("conv","ConvThreadNumber")
```

2. Once you execute the code, the result is:

```
ans =

    16
```

Retrieve `InputMemorySize` for `fc` Module Inside `dlhdl.ProcessorConfig` Object

1. Create an example object by using the `dlhdl.ProcessorConfig` class, and then use the `getModuleProperty` method to obtain the `ConvThreadNumber`.

```
hPC = dlhdl.ProcessorConfig;
hPC.getModuleProperty("fc","InputMemorySize")
```

2. Once you execute the code, the result is:

```
ans =

       25088
```

Version History

Introduced in R2020b

See Also

`estimatePerformance` | `estimateResources` | `optimizeConfigurationForNetwork` | `setModuleProperty`

Topics

“Deep Learning Processor IP Core Architecture”
“Estimate Performance of Deep Learning Network”
“Estimate Resource Utilization for Custom Processor Configuration”

openCustomLayerModel

Class: `dlhdl.ProcessorConfig`
Package: `dlhdl`

Open a generated custom layer verification model to verify your custom layers

Syntax

```
openCustomLayerModel(processorConfigObject)
openCustomLayerModel(processorConfigObject,Name = Value)
```

Description

`openCustomLayerModel(processorConfigObject)` opens a generated custom layer verification model to verify your custom layers.

`openCustomLayerModel(processorConfigObject,Name = Value)` opens a generated custom layer verification model to verify your custom layers, with one or more arguments specified by optional name-value arguments.

Input Arguments

`processorConfigObject` — Processor configuration

`dlhdl.ProcessorConfig` object

Processor configuration, specified as a `dlhdl.ProcessorConfig` object.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example:

`Network` — Network object

`SeriesNetwork` (default) | `SeriesNetwork` | `DAGNetwork` object | `dlquantizer` object

Name of the network object that contains your registered custom layer. If you do not include the network argument, Deep Learning HDL Toolbox generates a custom network that includes your registered custom layer.

Example: Network = net

`InputImages` — Input image

m-by-*n*-by-*k* numeric array

Input image, specified as a m-by-n-by-k numeric array. m, n, and k must match the dimensions of the deep learning network input image layer. For example, for the ResNet-18 network, resize the input images to a 224-by-224-by-3 array.

Example: InputImages = image

Data Types: `single`

Examples

Generate and Open Model for Custom Signum Layer

1. Before you can generate a test bench model, you must create and register a signum layer. For instructions, see “Register Custom Signum Layer” on page 1-219.
2. Generate a test bench model for your custom layer.

```
image = randi(255, [2,2,4]);
% To generate a test network that contains your custom layer do not specify
%  the Network name-value pair argument
openCustomLayerModel(hPC, InputImages = single(image));
% To test your actual network with custom layer specify the Network name-value
% pair argument. For example, if your custom network name is myNet, at the command line enter:
%   openCustomLayerModel(hPC, Network = myNet, InputImages = single(image));
```

Version History

Introduced in R2022a

See Also

`estimatePerformance` | `getModuleProperty` | `optimizeConfigurationForNetwork` | `registerCustomLayer` | `setModuleProperty`

optimizeConfigurationForNetwork

Class: `dlhdl.ProcessorConfig`
Package: `dlhdl`

Update network-specific deep learning processor configuration with optimized deep learning processor configuration

Syntax

```
newprocessorConfigObject = optimizeConfigurationForNetwork(
processorConfigObject,network)
newprocessorConfigObject = optimizeConfigurationForNetwork( ___ ,Name,Value)
```

Description

`newprocessorConfigObject = optimizeConfigurationForNetwork( processorConfigObject,network)` updates the processor configuration object specified by `processorConfigObject` with the optimized deep learning processor configuration, optimized for the `network` and returns the updated processor configuration object.

`newprocessorConfigObject = optimizeConfigurationForNetwork( ___ ,Name,Value)` updates the processor configuration, with options specified by one or more name-value arguments.

Input Arguments

`processorConfigObject` — Processor configuration

`dlhdl.ProcessorConfig` object

Processor configuration, specified as a `dlhdl.ProcessorConfig` object.

`network` — Name of network

`SeriesNetwork` object | `DAGNetwork` object | `dlnetwork` object

Name of network for optimized deep learning processor configuration, specified as a `SeriesNetwork`, `DAGNetwork`, or `dlnetwork`.

Example: `optimizeConfigurationForNetwork(snet)`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

`FramesPerSecond` — Target frames per second

0 (default) | double

Target frames per second for the network, specified as a `double` data type.

Example: `FramesPerSecond=10`

Output Arguments

`newprocessorConfigObject` — Processor configuration

`dlhdl.ProcessorConfig` object

Processor configuration specified as a `dlhdl.ProcessorConfig` object.

Examples

Generate Optimized Processor Configuration for `MobileNetV2` Network

Retrieve an optimized `MobileNetV2` network-specific processor configuration, by using the `optimizeConfigurationForNetwork` method.

Create a `dlhdl.ProcessorConfig` object.

```
net = mobilenetv2;
hPC = dlhdl.ProcessorConfig;
```

To retrieve an optimized processor configuration, call the `optimizeConfigurationForNetwork` method.

```
optimizeConfigurationForNetwork(hPC,net)
```

```
### Optimizing processor configuration for deep learning network...
### Note: Processing module "conv" property "InputMemorySize" changed from "[227 227 3]" to "[224 224 3]".
### Note: Processing module "conv" property "OutputMemorySize" changed from "[227 227 3]" to "[112 112 32]".
### Note: Processing module "conv" property "FeatureSizeLimit" changed from "2048" to "1280".
### Note: Processing module "conv" property "SegmentationBlockGeneration" changed from "true" to "false".
### Note: Processing module "fc" property "InputMemorySize" changed from "25088" to "1280".
### Note: Processing module "fc" property "OutputMemorySize" changed from "4096" to "1000".
### Note: Processing module "fc" property "SoftmaxBlockGeneration" changed from "false" to "true".

                   Processing Module "conv"
                          ModuleGeneration: 'on'
                         LRNBlockGeneration: 'off'
                SegmentationBlockGeneration: 'off'
                          ConvThreadNumber: 16
                           InputMemorySize: [224 224 3]
                          OutputMemorySize: [112 112 32]
                          FeatureSizeLimit: 1280

                      Processing Module "fc"
                          ModuleGeneration: 'on'
                      SoftmaxBlockGeneration: 'on'
                      SigmoidBlockGeneration: 'off'
                             FCThreadNumber: 4
                           InputMemorySize: 1280
                          OutputMemorySize: 1000

                  Processing Module "custom"
                          ModuleGeneration: 'on'
                                  Addition: 'on'
                            Multiplication: 'on'
                                  Resize2D: 'off'
                                   Sigmoid: 'off'
                                 TanhLayer: 'off'
                           InputMemorySize: 40
                          OutputMemorySize: 120

             Processor Top Level Properties
                            RunTimeControl: 'register'
                             RunTimeStatus: 'register'
                        InputStreamControl: 'register'
                       OutputStreamControl: 'register'
                              SetupControl: 'register'
```

```
                        ProcessorDataType: 'single'

                    System Level Properties
                             TargetPlatform: 'Xilinx Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit'
                            TargetFrequency: 200
                              SynthesisTool: 'Xilinx Vivado'
                            ReferenceDesign: 'AXI-Stream DDR Memory Access : 3-AXIM'
                     SynthesisToolChipFamily: 'Zynq UltraScale+'
                     SynthesisToolDeviceName: 'xczu9eg-ffvb1156-2-e'
                    SynthesisToolPackageName: ''
                     SynthesisToolSpeedValue: ''

### Optimizing processor configuration for deep learning network complete.

ans =

                   Processing Module "conv"
                           ModuleGeneration: 'on'
                         LRNBlockGeneration: 'off'
                SegmentationBlockGeneration: 'off'
                           ConvThreadNumber: 16
                            InputMemorySize: [224 224 3]
                           OutputMemorySize: [112 112 32]
                           FeatureSizeLimit: 1280

                        Processing Module "fc"
                           ModuleGeneration: 'on'
                        SoftmaxBlockGeneration: 'on'
                        SigmoidBlockGeneration: 'off'
                            FCThreadNumber: 4
                           InputMemorySize: 1280
                           OutputMemorySize: 1000

                    Processing Module "custom"
                           ModuleGeneration: 'on'
                                   Addition: 'on'
                             Multiplication: 'on'
                                   Resize2D: 'off'
                                    Sigmoid: 'off'
                                  TanhLayer: 'off'
                           InputMemorySize: 40
                           OutputMemorySize: 120

            Processor Top Level Properties
                             RunTimeControl: 'register'
                              RunTimeStatus: 'register'
                         InputStreamControl: 'register'
                        OutputStreamControl: 'register'
                               SetupControl: 'register'
                         ProcessorDataType: 'single'

                    System Level Properties
                             TargetPlatform: 'Xilinx Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit'
                            TargetFrequency: 200
                              SynthesisTool: 'Xilinx Vivado'
                            ReferenceDesign: 'AXI-Stream DDR Memory Access : 3-AXIM'
                     SynthesisToolChipFamily: 'Zynq UltraScale+'
                     SynthesisToolDeviceName: 'xczu9eg-ffvb1156-2-e'
                    SynthesisToolPackageName: ''
                     SynthesisToolSpeedValue: ''
```

Optimize Deep Learning Processor Configuration for Network Performance

This example shows how to generate a deep learning processor configuration and estimate the performance of a pretrained network. Generate a deep learning processor configuration optimized for the target frames-per-second value of the network, then generate a custom bitstream by using the optimized processor configuration.

Load Pretrained Network and Create Processor Configuration

To load a pretrained ResNet-18 network, enter:

```
net = resnet18;
```

Create a custom deep learning processor configuration. For more information, see `dlhdl.ProcessorConfig`.

```
hPC = dlhdl.ProcessorConfig;
```

Estimate Network Performance

Establish the baseline performance of the network, by estimating the performance of the ResNet-18 network. Estimate the performance, by using the `estimatePerformance` method of the `dlhdl.ProcessorConfig` object. The method returns the estimated layer latency, network latency, and network performance in frames per second.

```
estimatePerformance(hPC,net);
```

```
### Optimizing network: Fused 'nnet.cnn.layer.BatchNormalizationLayer' into 'nnet.cnn.layer.Convo
### Notice: The layer 'data' of type 'ImageInputLayer' is split into an image input layer 'data',
### The network includes the following layers:
     1   'data'                                Image Input                 224×224×3 images with
     2   'conv1'                               2-D Convolution             64 7×7×3 convolutions w
     3   'conv1_relu'                          ReLU                        ReLU
     4   'pool1'                               2-D Max Pooling             3×3 max pooling with st
     5   'res2a_branch2a'                      2-D Convolution             64 3×3×64 convolutions
     6   'res2a_branch2a_relu'                 ReLU                        ReLU
     7   'res2a_branch2b'                      2-D Convolution             64 3×3×64 convolutions
     8   'res2a'                               Addition                    Element-wise addition o
     9   'res2a_relu'                          ReLU                        ReLU
    10   'res2b_branch2a'                      2-D Convolution             64 3×3×64 convolutions
    11   'res2b_branch2a_relu'                 ReLU                        ReLU
    12   'res2b_branch2b'                      2-D Convolution             64 3×3×64 convolutions
    13   'res2b'                               Addition                    Element-wise addition o
    14   'res2b_relu'                          ReLU                        ReLU
    15   'res3a_branch2a'                      2-D Convolution             128 3×3×64 convolutions
    16   'res3a_branch2a_relu'                 ReLU                        ReLU
    17   'res3a_branch2b'                      2-D Convolution             128 3×3×128 convolution
    18   'res3a_branch1'                       2-D Convolution             128 1×1×64 convolutions
    19   'res3a'                               Addition                    Element-wise addition o
    20   'res3a_relu'                          ReLU                        ReLU
    21   'res3b_branch2a'                      2-D Convolution             128 3×3×128 convolution
    22   'res3b_branch2a_relu'                 ReLU                        ReLU
    23   'res3b_branch2b'                      2-D Convolution             128 3×3×128 convolution
    24   'res3b'                               Addition                    Element-wise addition o
    25   'res3b_relu'                          ReLU                        ReLU
    26   'res4a_branch2a'                      2-D Convolution             256 3×3×128 convolution
    27   'res4a_branch2a_relu'                 ReLU                        ReLU
    28   'res4a_branch2b'                      2-D Convolution             256 3×3×256 convolution
    29   'res4a_branch1'                       2-D Convolution             256 1×1×128 convolution
    30   'res4a'                               Addition                    Element-wise addition o
    31   'res4a_relu'                          ReLU                        ReLU
    32   'res4b_branch2a'                      2-D Convolution             256 3×3×256 convolution
    33   'res4b_branch2a_relu'                 ReLU                        ReLU
    34   'res4b_branch2b'                      2-D Convolution             256 3×3×256 convolution
    35   'res4b'                               Addition                    Element-wise addition o
    36   'res4b_relu'                          ReLU                        ReLU
    37   'res5a_branch2a'                      2-D Convolution             512 3×3×256 convolution
    38   'res5a_branch2a_relu'                 ReLU                        ReLU
    39   'res5a_branch2b'                      2-D Convolution             512 3×3×512 convolution
    40   'res5a_branch1'                       2-D Convolution             512 1×1×256 convolution
    41   'res5a'                               Addition                    Element-wise addition o
```

```
    42   'res5a_relu'                        ReLU                          ReLU
    43   'res5b_branch2a'                    2-D Convolution               512 3×3×512 convolutio
    44   'res5b_branch2a_relu'               ReLU                          ReLU
    45   'res5b_branch2b'                    2-D Convolution               512 3×3×512 convolutio
    46   'res5b'                             Addition                      Element-wise addition o
    47   'res5b_relu'                        ReLU                          ReLU
    48   'pool5'                             2-D Global Average Pooling    2-D global average poo
    49   'fc1000'                            Fully Connected               1000 fully connected la
    50   'prob'                              Softmax                       softmax
    51   'ClassificationLayer_predictions'   Classification Output        crossentropyex with 'te

### Notice: The layer 'prob' with type 'nnet.cnn.layer.SoftmaxLayer' is implemented in software.
### Notice: The layer 'ClassificationLayer_predictions' with type 'nnet.cnn.layer.ClassificationO


              Deep Learning Processor Estimator Performance Results

                    LastFrameLatency(cycles)   LastFrameLatency(seconds)       FramesNum      Tota
                          -------------              -------------              ---------        --
Network                    21328236                    0.10664                     1           213
    ____data_norm_add        210750                    0.00105
    ____data_norm            210750                    0.00105
    ____conv1               2164124                    0.01082
    ____pool1                515064                    0.00258
    ____res2a_branch2a       966221                    0.00483
    ____res2a_branch2b       966221                    0.00483
    ____res2a                210750                    0.00105
    ____res2b_branch2a       966221                    0.00483
    ____res2b_branch2b       966221                    0.00483
    ____res2b                210750                    0.00105
    ____res3a_branch1        540861                    0.00270
    ____res3a_branch2a       540749                    0.00270
    ____res3a_branch2b       919117                    0.00460
    ____res3a                105404                    0.00053
    ____res3b_branch2a       919117                    0.00460
    ____res3b_branch2b       919117                    0.00460
    ____res3b                105404                    0.00053
    ____res4a_branch1        503405                    0.00252
    ____res4a_branch2a       509261                    0.00255
    ____res4a_branch2b       905421                    0.00453
    ____res4a                 52724                    0.00026
    ____res4b_branch2a       905421                    0.00453
    ____res4b_branch2b       905421                    0.00453
    ____res4b                 52724                    0.00026
    ____res5a_branch1        744525                    0.00372
    ____res5a_branch2a       751693                    0.00376
    ____res5a_branch2b      1415373                    0.00708
    ____res5a                 26368                    0.00013
    ____res5b_branch2a      1415373                    0.00708
    ____res5b_branch2b      1415373                    0.00708
    ____res5b                 26368                    0.00013
    ____pool5                 54594                    0.00027
    ____fc1000               207351                    0.00104
 * The clock frequency of the DL processor is: 200MHz
```

The estimated frames-per-second performance is 9.4 frames per second. To improve the network performance, you can modify the properties of the custom deep learning processor configuration hPC

or use the `optimizeConfigurationForNetwork` method. In this example, you use the `optimizeConfigurationForNetwork` method. To learn about modifying the properties manually, see “Effects of Custom Deep Learning Processor Parameters on Performance and Resource Utilization”.

Generate Optimized Processor Configuration

Optimize the processor configuration by using the `optimizeConfigurationForNetwork` method. Use the optional `FramesPerSecond` name-value argument.

```
hPC_optimized = optimizeConfigurationForNetwork(hPC,net,FramesPerSecond=10);
```

```
### Optimizing processor configuration for deep learning network...


              Deep Learning Processor Estimator Resource Results

                               DSPs          Block RAM*     LUTs(CLB/ALUT)
                           -------------    -------------    -------------
Available                       2520              912           274080
                           -------------    -------------    -------------
Total                       438( 18%)        600( 66%)     270396( 99%)
ReferenceDesign               3(  1%)         78(  9%)      35000( 13%)
DL_Processor                435( 18%)        522( 58%)     235396( 86%)
* Block RAM represents Block RAM tiles in Xilinx devices and Block RAM bits in Intel devices
### Note: Processing module "conv" property "InputMemorySize" changed from "[227 227 3]" to "[21
### Note: Processing module "conv" property "OutputMemorySize" changed from "[227 227 3]" to "[2
### Note: Processing module "conv" property "SegmentationBlockGeneration" changed from "true" to
### Note: Processing module "fc" property "FCThreadNumber" changed from "4" to "8".
### Note: Processing module "fc" property "WeightAXIDataBitwidth" changed from "128" to "256".
### Note: Processing module "fc" property "SoftmaxBlockGeneration" changed from "false" to "true"

                      Processing Module "conv"
                              ModuleGeneration: 'on'
                            LRNBlockGeneration: 'off'
                   SegmentationBlockGeneration: 'off'
                              ConvThreadNumber: 16
                               InputMemorySize: [217 217 3]
                              OutputMemorySize: [217 217 3]
                              FeatureSizeLimit: 2048

                           Processing Module "fc"
                                 ModuleGeneration: 'on'
                              SoftmaxBlockGeneration: 'on'
                              SigmoidBlockGeneration: 'off'
                                      FCThreadNumber: 8
                                     InputMemorySize: 25088
                                     OutputMemorySize: 4096

                         Processing Module "custom"
                                 ModuleGeneration: 'on'
                                          Addition: 'on'
                                    Multiplication: 'on'
                                          Resize2D: 'off'
                                           Sigmoid: 'off'
                                         TanhLayer: 'off'
                                     InputMemorySize: 40
                                     OutputMemorySize: 120
```

```
              Processor Top Level Properties
                              RunTimeControl: 'register'
                               RunTimeStatus: 'register'
                           InputStreamControl: 'register'
                          OutputStreamControl: 'register'
                                 SetupControl: 'register'
                            ProcessorDataType: 'single'

                     System Level Properties
                               TargetPlatform: 'Xilinx Zynq UltraScale+ MPSoC ZCU102 Evaluation K
                              TargetFrequency: 200
                                SynthesisTool: 'Xilinx Vivado'
                              ReferenceDesign: 'AXI-Stream DDR Memory Access : 3-AXIM'
                       SynthesisToolChipFamily: 'Zynq UltraScale+'
                       SynthesisToolDeviceName: 'xczu9eg-ffvb1156-2-e'
                     SynthesisToolPackageName: ''
                       SynthesisToolSpeedValue: ''

### Optimizing processor configuration for deep learning network complete.
```

Estimate performance of the ResNet-18 network by using the new optimized deep learning processor configuration.

```
estimatePerformance(hPC_optimized,net);
```

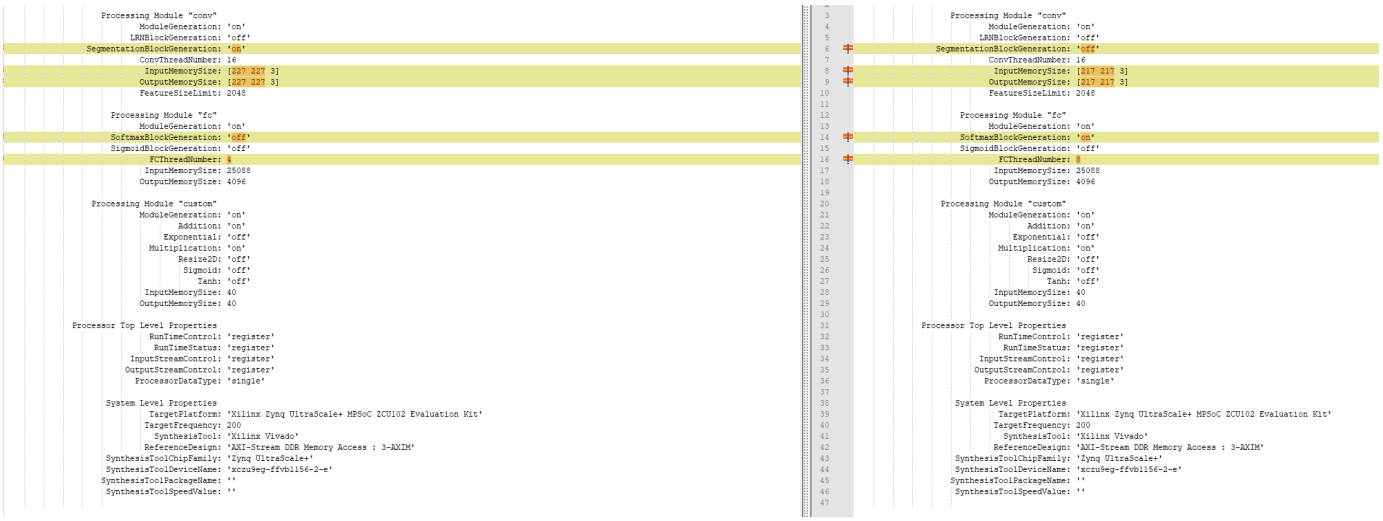
```
### Optimizing network: Fused 'nnet.cnn.layer.BatchNormalizationLayer' into 'nnet.cnn.layer.Convo
### Notice: The layer 'data' of type 'ImageInputLayer' is split into an image input layer 'data'
### The network includes the following layers:
     1   'data'                              Image Input                 224×224×3 images with
     2   'conv1'                             2-D Convolution             64 7×7×3 convolutions w
     3   'conv1_relu'                        ReLU                        ReLU
     4   'pool1'                             2-D Max Pooling             3×3 max pooling with st
     5   'res2a_branch2a'                    2-D Convolution             64 3×3×64 convolutions
     6   'res2a_branch2a_relu'               ReLU                        ReLU
     7   'res2a_branch2b'                    2-D Convolution             64 3×3×64 convolutions
     8   'res2a'                             Addition                    Element-wise addition o
     9   'res2a_relu'                        ReLU                        ReLU
    10   'res2b_branch2a'                    2-D Convolution             64 3×3×64 convolutions
    11   'res2b_branch2a_relu'               ReLU                        ReLU
    12   'res2b_branch2b'                    2-D Convolution             64 3×3×64 convolutions
    13   'res2b'                             Addition                    Element-wise addition o
    14   'res2b_relu'                        ReLU                        ReLU
    15   'res3a_branch2a'                    2-D Convolution             128 3×3×64 convolutions
    16   'res3a_branch2a_relu'               ReLU                        ReLU
    17   'res3a_branch2b'                    2-D Convolution             128 3×3×128 convolution
    18   'res3a_branch1'                     2-D Convolution             128 1×1×64 convolutions
    19   'res3a'                             Addition                    Element-wise addition o
    20   'res3a_relu'                        ReLU                        ReLU
    21   'res3b_branch2a'                    2-D Convolution             128 3×3×128 convolution
    22   'res3b_branch2a_relu'               ReLU                        ReLU
    23   'res3b_branch2b'                    2-D Convolution             128 3×3×128 convolution
    24   'res3b'                             Addition                    Element-wise addition o
    25   'res3b_relu'                        ReLU                        ReLU
    26   'res4a_branch2a'                    2-D Convolution             256 3×3×128 convolution
    27   'res4a_branch2a_relu'               ReLU                        ReLU
    28   'res4a_branch2b'                    2-D Convolution             256 3×3×256 convolution
    29   'res4a_branch1'                     2-D Convolution             256 1×1×128 convolution
```

```
    30   'res4a'                          Addition                    Element-wise addition c
    31   'res4a_relu'                     ReLU                        ReLU
    32   'res4b_branch2a'                 2-D Convolution             256 3×3×256 convolution
    33   'res4b_branch2a_relu'            ReLU                        ReLU
    34   'res4b_branch2b'                 2-D Convolution             256 3×3×256 convolution
    35   'res4b'                          Addition                    Element-wise addition c
    36   'res4b_relu'                     ReLU                        ReLU
    37   'res5a_branch2a'                 2-D Convolution             512 3×3×256 convolution
    38   'res5a_branch2a_relu'            ReLU                        ReLU
    39   'res5a_branch2b'                 2-D Convolution             512 3×3×512 convolution
    40   'res5a_branch1'                  2-D Convolution             512 1×1×256 convolution
    41   'res5a'                          Addition                    Element-wise addition c
    42   'res5a_relu'                     ReLU                        ReLU
    43   'res5b_branch2a'                 2-D Convolution             512 3×3×512 convolution
    44   'res5b_branch2a_relu'            ReLU                        ReLU
    45   'res5b_branch2b'                 2-D Convolution             512 3×3×512 convolution
    46   'res5b'                          Addition                    Element-wise addition c
    47   'res5b_relu'                     ReLU                        ReLU
    48   'pool5'                          2-D Global Average Pooling  2-D global average pool
    49   'fc1000'                         Fully Connected             1000 fully connected la
    50   'prob'                           Softmax                     softmax
    51   'ClassificationLayer_predictions'   Classification Output       crossentropyex with 'te

### Notice: The layer 'ClassificationLayer_predictions' with type 'nnet.cnn.layer.ClassificationO


              Deep Learning Processor Estimator Performance Results

                    LastFrameLatency(cycles)   LastFrameLatency(seconds)       FramesNum      Tota
                          -------------              -------------               ---------        --
Network                    19966252                    0.09983                     1           199
    ____data_norm_add        210750                    0.00105
    ____data_norm            210750                    0.00105
    ____conv1               2224339                    0.01112
    ____pool1                632402                    0.00316
    ____res2a_branch2a      1038708                    0.00519
    ____res2a_branch2b      1038708                    0.00519
    ____res2a                210750                    0.00105
    ____res2b_branch2a      1038708                    0.00519
    ____res2b_branch2b      1038708                    0.00519
    ____res2b                210750                    0.00105
    ____res3a_branch1        630228                    0.00315
    ____res3a_branch2a       625092                    0.00313
    ____res3a_branch2b       919117                    0.00460
    ____res3a                105404                    0.00053
    ____res3b_branch2a       919117                    0.00460
    ____res3b_branch2b       919117                    0.00460
    ____res3b                105404                    0.00053
    ____res4a_branch1        503405                    0.00252
    ____res4a_branch2a       509261                    0.00255
    ____res4a_branch2b       905421                    0.00453
    ____res4a                 52724                    0.00026
    ____res4b_branch2a       905421                    0.00453
    ____res4b_branch2b       905421                    0.00453
    ____res4b                 52724                    0.00026
    ____res5a_branch1        506957                    0.00253
    ____res5a_branch2a       514125                    0.00257
    ____res5a_branch2b       940237                    0.00470
```

```
    ____res5a                 26368                    0.00013
    ____res5b_branch2a       940237                    0.00470
    ____res5b_branch2b       940237                    0.00470
    ____res5b                 26368                    0.00013
    ____pool5                 54594                    0.00027
    ____fc1000               103438                    0.00052
    ____prob                   1262                    0.00001
 * The clock frequency of the DL processor is: 200MHz
```

The new estimated frames per second performance is 10 frames per second.

This image shows the comparison between the original processor configuration and the optimized processor configuration:



The optimized processor configuration has:

- `SegmentationBlockGeneration` turned off.
- `InputMemorySize` and `OutputMemorySize` reduced to [217 217 3].
- `SoftMaxBlockGeneration` turned on.
- `FCThreadNumber` increased to 8.

Generate Optimized Custom Bitstream

Use the optimized custom deep learning processor configuration to build and generate a custom bitstream. Use the custom bitstream to deploy the pretrained ResNet-18 network to your target FPGA board.

```
hdlsetuptoolpath('ToolName', 'Xilinx Vivado', 'ToolPath', 'C:\Xilinx\Vivado\2020.2\bin\vivado.ba
dlhdl.buildProcessor(hPC_optimized);
```

Version History

Introduced in R2021b

R2022b: Added `FramesPerSecond` name-value argument

You can specify the target frames per second for the network by using the optional `FramesPerSecond` name-value argument.

See Also

`estimatePerformance` | `estimateResources` | `getModuleProperty` | `setModuleProperty`

Topics

“Generate Custom Bitstream to Meet Custom Deep Learning Network Requirements”

registerCustomLayer

Class: `dlhdl.ProcessorConfig`
Package: `dlhdl`

Register the custom layer definition and Simulink model representation of the custom layer

Syntax

```
registerCustomLayer(processorConfigObject, 'Layer', Layer, 'Model', Model)
```

Description

`registerCustomLayer(processorConfigObject, 'Layer', Layer, 'Model', Model)` registers a custom layer specified by the `Layer` argument and the Simulink® model representation of the custom layer, specified by the `Model` argument.

Input Arguments

`processorConfigObject` — Processor configuration

`dlhdl.ProcessorConfig` object

Processor configuration, specified as a `dlhdl.ProcessorConfig` object.

`Layer` — Instance of custom layer definition

`nnet.layer.Layer`

Class definition of the custom layer object, specified as an `nnet.layer.Layer` object.

Example: `Layer = hSig`

`Model` — Simulink model representing custom layer

`''` (default)

Simulink model representing the custom layer, specified as a file name of the Simulink model on the MATLAB® path or absolute or relative path to the Simulink model.

Example: `Model = 'myfile.slx'`

Example: `Model = 'C:\myfolder\myfile.slx'`

Examples

Register Custom Signum Layer

1. Create a function that represents the custom signum layer. Save the function definition as `SignumLayer.m`.

```
classdef SignumLayer < nnet.layer.Layer
    % Example custom Signum layer.

    properties
        testPropertyValue1 single = 3;
```

```
        testPropertyValue2 single = 4;
    end

    methods
        function layer = SignumLayer(name)
            % Set layer name.
            layer.Name = name;
            % Set layer description.
            layer.Description = "custom signum layer";
        end

        function Z = predict(layer, X)
            % Z = predict(layer, X) forwards the input data X through the
            % layer and outputs the result Z.

            Z = sign(X) + layer.testPropertyValue1 + layer.testPropertyValue2;

        end
    end
end
```

2. Create a variable `hSig`. Assign the custom signum layer function definition to `hSig`.

   ```
   hSig = SignumLayer('sLayer');
   ```

3. Create a Simulink model that represents the custom signum layer. Save the Simulink model as `SignumLayer.slx`.
4. Create a custom deep learning processor configuration object by using the `dlhdl.ProcessorConfig` class. Save the custom deep learning processor configuration as `hPC`.

   ```
   hPC = dlhdl.ProcessorConfig;
   ```

5. Use the `registerCustomLayer` method to register the custom signum layer definition and Simulink model.

   ```
   % If the Simulink model is on the MATLAB path, use:
   hPC.registerCustomLayer(Layer = hSig, Model = 'SignumLayer.slx');
   % If the Simulink model is in a folder called myLayers on your C drive, use:
   %   hPC.registerCustomLayer( Layer = hSig, Model = 'C:\myLayers\SignumLayer.slx');
   ```

Version History

Introduced in R2022a

See Also

`estimatePerformance` | `getModuleProperty` | `openCustomLayerModel` | `optimizeConfigurationForNetwork` | `setModuleProperty`

setModuleProperty

Class: `dlhdl.ProcessorConfig`
Package: `dlhdl`

Use the `setModuleProperty` method to set properties of modules within the `dlhdl.ProcessorConfig` object

Syntax

```
setModuleProperty(processorConfigObject,ModuleName,Name,Value)
```

Description

The `setModuleProperty(processorConfigObject,ModuleName,Name,Value)` method sets the properties of the module mentioned in `ModuleName` by using the values specified as `Name,Value` pairs.

Input Arguments

`processorConfigObject` — Processor configuration

`dlhdl.ProcessorConfig` object

Processor configuration, specified as a `dlhdl.ProcessorConfig` object.

`ModuleName` — Name of the module whose parameters are to be set

"conv" | "fc" | "custom" | 'conv' | 'fc' | 'custom' | string | character vector

The `dlhdl.ProcessorConfig` object module name, specified as a character vector or string.

Name-Value Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example:

conv module parameters

`ModuleGeneration` — Enable or disable convolution module generation as a part of the deep learning processor configuration

'on' (default) | 'off' | character vector

Use this parameter to control generation of the convolution module as a part of the deep learning processor configuration.

Example: `'ModuleGeneration', 'on'`

`LRNBlockGeneration` — Enable or disable local response normalization (LRN) block generation as a part of the convolution module of the deep learning processor configuration
'off' (default) | 'on' | character vector

Use this parameter to control generation of the LRN block as a part of the convolution module of the deep learning processor configuration.

Example: `'LRNBlockGeneration', 'on'`

`SegmentationBlockGeneration` — Enable or disable segmentation block generation as a part of the convolution module of the deep learning processor configuration
'on' (default) | 'off' | character vector

Use this parameter to control generation of the segmentation block as a part of the convolution module of the deep learning processor configuration.

`ConvThreadNumber` — Number of parallel convolution processor kernel threads
16 (default) | 4 | 9 | 16 | 25 | 36 | 64 | 256 | unsigned integer

This parameter is the number of parallel 3-by-3 convolution kernel threads that are a part of the `conv` module within the `dlhdl.ProcessorConfig` object.

Example: `'ConvThreadNumber', 64`

`InputMemorySize` — Cache block RAM (BRAM) sizes
[227 227 3] (default) | 3D positive integer array

This parameter is a 3D matrix representing input image size limited by the `conv` module BRAM size within the `dlhdl.ProcessorConfig` object.

Example: `'InputMemorySize', [227 227 3]`

`OutputMemorySize` — Cache block RAM (BRAM) sizes
[227 227 3] (default) | 3D positive integer array

This parameter is a 3D matrix representing output image size limited by the `conv` module BRAM size within the `dlhdl.ProcessorConfig` object.

Example: `'OutputMemorySize', [227 227 3]`

`FeatureSizeLimit` — Maximum input and output feature size
2048 (default) | positive integer

This parameter is a positive integer representing the maximum input and output feature size as a part of the `conv` module within the `dlhdl.ProcessorConfig` object.

Example: `'FeatureSizeLimit', 512`

fc module parameters

`ModuleGeneration` — Enable or disable fully connected module generation as a part of the deep learning processor configuration
'on' (default) | 'off' | character vector

Use this parameter to control generation of the fully connected module as a part of the deep learning processor configuration.

Example: `'ModuleGeneration', 'on'`

`SoftmaxBlockGeneration` — Enable or disable Softmax block generation as a part of the fully connected module of the deep learning processor configuration
'off' (default) | 'on' | character vector

Use this parameter to control generation of the Softmax block as a part of the fully connected module of the deep learning processor configuration. When you set this property to `off`, the Softmax layer is still implemented in software.

Example: `'SoftmaxBlockGeneration', 'on'`

`SigmoidBlockGeneration` — Enable or disable sigmoid block generation as a part of the fully connected module of the deep learning processor configuration
'off' (default) | 'on' | character vector

Use this parameter to control generation of the sigmoid block as a part of the fully connected module of the deep learning processor configuration.

`FCThreadNumber` — Number of parallel fully connected (fc) MAC threads
4 (default) | 4 | 8 | 16 | 32 | 64 | unsigned integer

This parameter is the number of parallel fc MAC threads that are a part of the `fc` module within the `dlhdl.ProcessorConfig` object.

Example: `'FCThreadNumber', 16`

`InputMemorySize` — Cache block RAM (BRAM) sizes
25088 (default) | unsigned integer

This parameter is an unsigned integer representing cache BRAM size limited by the `fc` module BRAM size within the `dlhdl.ProcessorConfig` object.

Example: `'InputMemorySize', 9216`

`OutputMemorySize` — Cache block RAM (BRAM) sizes
4096 (default) | unsigned integer

This parameter is an unsigned integer representing cache BRAM size limited by the `fc` module BRAM size within the `dlhdl.ProcessorConfig` object.

Example: `'OutputMemorySize', 4096`

custom module properties

`ModuleGeneration` — Enable or disable adder module generation as a part of the deep learning processor configuration
'on' (default) | 'off' | character vector

Use this parameter to control generation of the adder module as a part of the deep learning processor configuration.

Example: `'ModuleGeneration', 'on'`

`Addition` — Enable or disable addition layer generation as a part of the custom module of the deep learning processor configuration
'on' (default) | 'off' | character vector

Use this parameter to control generation of the addition layer as a part of the custom module of the deep learning processor configuration.

`Multiplication` — Enable or disable multiplication layer generation as a part of the custom module of the deep learning processor configuration
'on' (default) | 'off' | character vector

Use this parameter to control generation of the multiplication layer as a part of the custom module of the deep learning processor configuration.

`Resize2D` — Enable or disable `resize2d` layer generation as a part of the custom module of the deep learning processor configuration
'off' (default) | 'on' | character vector

Use this parameter to control generation of the `resize2d` layer as a part of the custom module of the deep learning processor configuration.

`Sigmoid` — Enable or disable sigmoid layer generation as a part of the custom module of the deep learning processor configuration
'off' (default) | 'on' | character vector

Use this parameter to control generation of the sigmoid layer as a part of the custom module of the deep learning processor configuration.

`TanhLayer` — Enable or disable `tanh` layer generation as a part of the custom module of the deep learning processor configuration
'off' (default) | 'on' | character vector

Use this parameter to control generation of the `tanh` layer as a part of the custom module of the deep learning processor configuration.

`InputMemorySize` — Cache block RAM (BRAM) sizes
40 (default) | unsigned integer

This parameter is an unsigned integer representing cache BRAM size limited by the `adder` module BRAM size within the `dlhdl.ProcessorConfig` object.

Example: `'InputMemorySize', 40`

`OutputMemorySize` — Cache block RAM (BRAM) sizes
120 (default) | unsigned integer

This parameter is an unsigned integer representing cache BRAM size limited by the `adder` module BRAM size within the `dlhdl.ProcessorConfig` object.

Example: `'OutputMemorySize', 40`

Examples

Set Value for `ConvThreadNumber` Within `dlhdl.ProcessorConfig` Object

1 Create an example object by using the `dlhdl.ProcessorConfig` class, and then use the `setModuleProperty` method to set the value for `convThreadNumber`.

```
hPC = dlhdl.ProcessorConfig;
hPC.setModuleProperty("conv","ConvThreadNumber",25)
hPC
```

2 Once you execute the code, the result is:

```
hPC =

                   Processing Module "conv"
                           ModuleGeneration: 'on'
                         LRNBlockGeneration: 'off'
                SegmentationBlockGeneration: 'on'
                           ConvThreadNumber: 25
                            InputMemorySize: [227 227 3]
                           OutputMemorySize: [227 227 3]
                           FeatureSizeLimit: 2048

                     Processing Module "fc"
                           ModuleGeneration: 'on'
                        SoftmaxBlockGeneration: 'off'
                        SigmoidBlockGeneration: 'off'
                                  FCThreadNumber: 4
                           InputMemorySize: 25088
                           OutputMemorySize: 4096

                 Processing Module "custom"
                           ModuleGeneration: 'on'
                                   Addition: 'on'
                             Multiplication: 'on'
                                   Resize2D: 'off'
                                    Sigmoid: 'off'
                                  TanhLayer: 'off'
                           InputMemorySize: 40
                           OutputMemorySize: 40

           Processor Top Level Properties
                             RunTimeControl: 'register'
                              RunTimeStatus: 'register'
                         InputStreamControl: 'register'
                        OutputStreamControl: 'register'
                               SetupControl: 'register'
                          ProcessorDataType: 'single'

                  System Level Properties
                           TargetPlatform: 'Xilinx Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit'
                          TargetFrequency: 200
                            SynthesisTool: 'Xilinx Vivado'
                          ReferenceDesign: 'AXI-Stream DDR Memory Access : 3-AXIM'
                   SynthesisToolChipFamily: 'Zynq UltraScale+'
                   SynthesisToolDeviceName: 'xczu9eg-ffvb1156-2-e'
                  SynthesisToolPackageName: ''
                   SynthesisToolSpeedValue: ''
```

Set Value for `InputMemorySize` Within `dlhdl.ProcessorConfig` Object

1 Create an example object by using the `dlhdl.ProcessorConfig` class, and then use the `setModuleProperty` method to set the value for `InputMemorySize`.

```
hPC = dlhdl.ProcessorConfig;
hPC.setModuleProperty("fc","InputMemorySize",25060)
hPC
```

2 Once you execute the code, the result is:

```
hPC =

                   Processing Module "conv"
                           ModuleGeneration: 'on'
                         LRNBlockGeneration: 'off'
                SegmentationBlockGeneration: 'on'
                           ConvThreadNumber: 16
                            InputMemorySize: [227 227 3]
                           OutputMemorySize: [227 227 3]
```

```
                    FeatureSizeLimit: 2048

                Processing Module "fc"
                      ModuleGeneration: 'on'
                SoftmaxBlockGeneration: 'off'
                SigmoidBlockGeneration: 'off'
                        FCThreadNumber: 4
                       InputMemorySize: 25060
                      OutputMemorySize: 4096

            Processing Module "custom"
                      ModuleGeneration: 'on'
                              Addition: 'on'
                        Multiplication: 'on'
                              Resize2D: 'off'
                               Sigmoid: 'off'
                             TanhLayer: 'off'
                       InputMemorySize: 40
                      OutputMemorySize: 40

        Processor Top Level Properties
                        RunTimeControl: 'register'
                         RunTimeStatus: 'register'
                    InputStreamControl: 'register'
                   OutputStreamControl: 'register'
                          SetupControl: 'register'
                     ProcessorDataType: 'single'

           System Level Properties
                         TargetPlatform: 'Xilinx Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit'
                        TargetFrequency: 200
                          SynthesisTool: 'Xilinx Vivado'
                        ReferenceDesign: 'AXI-Stream DDR Memory Access : 3-AXIM'
                SynthesisToolChipFamily: 'Zynq UltraScale+'
                SynthesisToolDeviceName: 'xczu9eg-ffvb1156-2-e'
               SynthesisToolPackageName: ''
                SynthesisToolSpeedValue: ''
```

Set Value for `InputMemorySize` Within `dlhdl.ProcessorConfig` Object

1. Create an example object by using the `dlhdl.ProcessorConfig` class, and then use the `setModuleProperty` method to set the value for `InputMemorySize`.

```
hPC = dlhdl.ProcessorConfig;
hPC.setModuleProperty("custom","InputMemorySize",80)
hPC
```

2. Once you execute the code, the result is:

```
hPC =

              Processing Module "conv"
                      ModuleGeneration: 'on'
                    LRNBlockGeneration: 'off'
           SegmentationBlockGeneration: 'on'
                      ConvThreadNumber: 16
                       InputMemorySize: [227 227 3]
                      OutputMemorySize: [227 227 3]
                      FeatureSizeLimit: 2048

                Processing Module "fc"
                      ModuleGeneration: 'on'
                SoftmaxBlockGeneration: 'off'
                SigmoidBlockGeneration: 'off'
                        FCThreadNumber: 4
                       InputMemorySize: 25088
                      OutputMemorySize: 4096

            Processing Module "custom"
                      ModuleGeneration: 'on'
                              Addition: 'on'
                        Multiplication: 'on'
                              Resize2D: 'off'
                               Sigmoid: 'off'
                             TanhLayer: 'off'
                       InputMemorySize: 80
```

```
                OutputMemorySize: 40

Processor Top Level Properties
                  RunTimeControl: 'register'
                   RunTimeStatus: 'register'
                InputStreamControl: 'register'
               OutputStreamControl: 'register'
                     SetupControl: 'register'
                ProcessorDataType: 'single'

         System Level Properties
                   TargetPlatform: 'Xilinx Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit'
                  TargetFrequency: 200
                    SynthesisTool: 'Xilinx Vivado'
                  ReferenceDesign: 'AXI-Stream DDR Memory Access : 3-AXIM'
          SynthesisToolChipFamily: 'Zynq UltraScale+'
          SynthesisToolDeviceName: 'xczu9eg-ffvb1156-2-e'
        SynthesisToolPackageName: ''
          SynthesisToolSpeedValue: ''
```

Turn off conv module Within `dlhdl.ProcessorConfig` Object

1. Create an example object by using the `dlhdl.ProcessorConfig` class, and then use the `setModuleProperty` method to set the value for `ModuleGeneration`.

   ```
   hPC = dlhdl.ProcessorConfig;
   hPC.setModuleProperty("conv","ModuleGeneration", "off")
   hPC
   ```

2. Once you execute the code, the result is:

```
hPC =

         Processing Module "conv"
                 ModuleGeneration: 'off'

           Processing Module "fc"
                 ModuleGeneration: 'on'
           SoftmaxBlockGeneration: 'off'
           SigmoidBlockGeneration: 'off'
                  FCThreadNumber: 4
                 InputMemorySize: 25088
                OutputMemorySize: 4096

      Processing Module "custom"
                ModuleGeneration: 'on'
                        Addition: 'on'
                  Multiplication: 'on'
                        Resize2D: 'off'
                         Sigmoid: 'off'
                       TanhLayer: 'off'
                 InputMemorySize: 40
                OutputMemorySize: 40

Processor Top Level Properties
                  RunTimeControl: 'register'
                   RunTimeStatus: 'register'
                InputStreamControl: 'register'
               OutputStreamControl: 'register'
                     SetupControl: 'register'
                ProcessorDataType: 'single'

         System Level Properties
                   TargetPlatform: 'Xilinx Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit'
                  TargetFrequency: 200
                    SynthesisTool: 'Xilinx Vivado'
                  ReferenceDesign: 'AXI-Stream DDR Memory Access : 3-AXIM'
          SynthesisToolChipFamily: 'Zynq UltraScale+'
          SynthesisToolDeviceName: 'xczu9eg-ffvb1156-2-e'
        SynthesisToolPackageName: ''
          SynthesisToolSpeedValue: ''
```

Version History

Introduced in R2020b

See Also

`estimatePerformance` | `estimateResources` | `getModuleProperty` | `optimizeConfigurationForNetwork`

Topics

“Deep Learning Processor IP Core Architecture”
“Estimate Performance of Deep Learning Network”
“Estimate Resource Utilization for Custom Processor Configuration”

verifyCustomLayerModel

Class: `dlhdl.ProcessorConfig`
Package: `dlhdl`

Verify the functionality and accuracy of the custom layer by using the generated custom layer verification model

Syntax

```
verifyCustomLayerModel(processorConfigObject)
```

Description

`verifyCustomLayerModel(processorConfigObject)` verifies the functionality of the custom layer and prediction accuracy of the network with the custom layer by using the custom layer verification model, which you generated by using the `openCustomLayerModel` method.

Input Arguments

`processorConfigObject` — Processor configuration

`dlhdl.ProcessorConfig` object

Processor configuration, specified as a `dlhdl.ProcessorConfig` object.

Examples

Verify Custom Signum Layer Model

1. Generate the verification model for your custom signum layer. For instructions, see “Generate and Open Model for Custom Signum Layer” on page 1-208. Leave the generated model file open.
2. Verify the generated verification model for the custom signum layer by using the `verifyCustomLayerModel` method.

```
verifyCustomLayerModel(hPC)
```

```
### Custom layer verification model simulation and validation begin.
### Compiling Simulink model 'dnnfpgaCustomLayerVerificationModel' ...
### Complete Simulink model 'dnnfpgaCustomLayerVerificationModel' compilation.
Verification passed.
### Custom layer verification model simulation and validation complete.
```

Version History

Introduced in R2022a

See Also

`estimatePerformance` | `getModuleProperty` | `optimizeConfigurationForNetwork` | `openCustomLayerModel` | `registerCustomLayer` | `setModuleProperty`

dlhdl.buildProcessor

Build and generate custom processor IP

Syntax

```
dlhdl.buildProcessor(processorconfigobject)
dlhdl.buildProcessor(processorconfigobject,Name,Value)
```

Description

`dlhdl.buildProcessor(processorconfigobject)` generates a bitstream for the `processorconfigobject` object.

`dlhdl.buildProcessor(processorconfigobject,Name,Value)` generates a bitstream for the `processorconfigobject` object, with additional options specified by one or more `Name,Value` arguments.

Examples

Generate Custom Bitstream for Custom Processor Configuration

Create a custom processor configuration. Generate a bitstream for the custom processor configuration.

Create a `dlhdl.ProcessorConfig` object. Save the object in `hPC`.

```
hPC = dlhdl.ProcessorConfig
```

Generate a custom bitstream for `hPC`

```
dlhdl.buildProcessor(hPC)
```

Generate Custom Bitstream and Specify Project Folder Name, Deep Learning Processor IP Core Name, and Target Code Generation language

Create a custom deep learning processor configuration. When you generate code specify the project folder name, deep learning processor IP core name, and target code generation language.

```
hPC = dlhdl.ProcessorConfig;
dlhdl.buildProcessor(hPC,'ProjectFolder','fconlyprocessor_prj',...
'ProcessorName','fconlyprocessor','HDLCoderConfig',{'TargetLanguage','Verilog'});
```

Input Arguments

`processorconfigobject` — Name of the object generated by using `dlhdl.buildProcessor`

(default) | variable

Name of the custom processor configuration object, specified as a variable of type `dlhdl.Processorconfig`.

Example: `dlhdl.buildProcessor(hPC)`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example:

`ProjectFolder` — Project folder name
'dlhdl_prj' (default)

Name of project folder where generated files are saved

Example: `'ProjectFolder', 'fconlyprocessor_prj'`

`ProcessorName` — Name of generated deep learning processor IP core
'' (default) | character vector or string

Name of generated deep learning processor IP core

Example: `'ProcessorName', 'fconlyprocessor'`

`TargetLanguage` — Target language
'VHDL' (default) | 'Verilog' | character vector or string

Specify whether to generate VHDL or Verilog code.

Example: `'HDLCoderConfig',{'TargetLanguage','Verilog'}`

`OverrideResourceCheck` — Flag to display resource utilization and warning for over consumption of resources
'false' (default) | 'true'

Flag to display resource utilization and warning for over consumption of resources specified as a `logical` datatype. When you set `OverrideResourceCheck` to `false`, the `buildProcessor` function displays the estimated resource consumption and a warning if the resource consumption exceeds 100 percent of the resources of the target board. When you set `OverrideResourceCheck` to `true`, the `buildProcessor` function does not display the estimated resource consumption or a warning if the resource consumption exceeds 100 percent of the resources of the target board.

Example: `'OverrideResourceCheck', 'false'`

Version History
Introduced in R2020b

See Also
`dlhdl.ProcessorConfig`

Topics
“Deep Learning Processor IP Core”
“Generate Custom Bitstream”
“Generate Custom Processor IP”

hdlsetuptoolpath

Set up system environment to access FPGA synthesis software

Syntax

```
hdlsetuptoolpath('ToolName',TOOLNAME,'ToolPath',TOOLPATH)
```

Description

`hdlsetuptoolpath('ToolName',TOOLNAME,'ToolPath',TOOLPATH)` adds a third-party FPGA synthesis tool to your system path. It sets up the system environment variables for the synthesis tool. To configure one or more supported third-party FPGA synthesis tools to use with HDL Coder™, use the `hdlsetuptoolpath` function.

Before opening the HDL Workflow Advisor, add the tool to your system path. If you already have the HDL Workflow Advisor open, see "Add Synthesis Tool for Current HDL Workflow Advisor Session" (HDL Coder).

Examples

Set Up Intel Quartus Prime

The following command sets the synthesis tool path to point to an installed Intel Quartus® Prime Standard Edition 20.1 executable file. You must have already installed Altera® Quartus II.

```
hdlsetuptoolpath('ToolName','Altera Quartus II','ToolPath',...
 'C:\intel\20.1\quartus\bin\quartus.exe');
```

Set Up Intel Quartus Pro

The following command sets the synthesis tool path to point to an installed Intel Quartus Pro 21.3 executable file. You must have already installed Intel Quartus Pro.

```
hdlsetuptoolpath('ToolName','Intel Quartus Pro','ToolPath',...
 'C:\intel\21.3_pro\quartus\bin64\qpro.exe');
```

Note An installation of Quartus Pro contains both `quartus.exe` and `qpro.exe` executable files. When both tools are added to the path by using `hdlsetuptoolpath`, HDL Coder checks the tool availability before running the HDL Workflow Advisor.

Set Up Xilinx ISE

The following command sets the synthesis tool path to point to an installed Xilinx ISE 14.7 executable file. You must have already installed Xilinx ISE.

```
hdlsetuptoolpath('ToolName','Xilinx ISE','ToolPath',...
 'C:\Xilinx\14.7\ISE_DS\ISE\bin\nt64\ise.exe');
```

Set Up Xilinx Vivado

The following command sets the synthesis tool path to point to an installed Vivado® Design Suite 2020.2 batch file. You must have already installed Xilinx Vivado.

```
hdlsetuptoolpath('ToolName','Xilinx Vivado','ToolPath',...
 'C:\Xilinx\Vivado\2020.2\bin\vivado.bat');
```

Set Up Microchip Libero SoC

The following command sets the synthesis tool path to point to an installed Microchip Libero® Design Suite batch file. You must have already installed Microchip Libero SoC.

```
hdlsetuptoolpath('ToolName','Microsemi Libero SoC','ToolPath',...
 'C:\Microsemi\Libero_SoC_v12.6\Designer\bin\libero.exe');
```

Input Arguments

`TOOLNAME` — Synthesis tool name

character vector

Synthesis tool name, specified as a character vector.

Example: `'Xilinx Vivado'`

`TOOLPATH` — Full path to the synthesis tool executable or batch file

character vector

Full path to the synthesis tool executable or batch file, specified as a character vector.

Example: `'C:\Xilinx\Vivado\2020.2\bin\vivado.bat'`

Tips

- If you have an icon for the tool on your Windows® desktop, you can find the full path to the synthesis tool.
  1. Right-click the icon and select **Properties**.
  2. Click the **Shortcut** tab.
- The `hdlsetuptoolpath` function changes the system path and system environment variables for only the current MATLAB session. To execute `hdlsetuptoolpath` programmatically when MATLAB starts, add `hdlsetuptoolpath` to your `startup.m` script.

Version History

Introduced in R2011a

See Also

`startup` | `setenv`

Topics

“HDL Language Support and Supported Third-Party Tools and Hardware” (HDL Coder)
“Tool Setup” (HDL Coder)

“Add Synthesis Tool for Current HDL Workflow Advisor Session” (HDL Coder)

dlquantizer

Quantize a deep neural network to 8-bit scaled integer data types

Description

Use the `dlquantizer` object to reduce the memory requirement of a deep neural network by quantizing weights, biases, and activations to 8-bit scaled integer data types. You can create and verify the behavior of a quantized network for GPU, FPGA, CPU deployment, or explore the quantized network in MATLAB.

For CPU and GPU deployment, the software generates code for a convolutional deep neural network by quantizing the weights, biases, and activations of the convolution layers to 8-bit scaled integer data types. The quantization is performed by providing the calibration result file produced by the `calibrate` function to the `codegen` command.

Code generation does not support quantized deep neural networks produced by the `quantize` function.

This object requires Deep Learning Toolbox Model Quantization Library. To learn about the products required to quantize a deep neural network, see “Quantization Workflow Prerequisites”.

Creation

Syntax

```
quantObj = dlquantizer(net)
quantObj = dlquantizer(net,Name,Value)
```

Description

`quantObj = dlquantizer(net)` creates a `dlquantizer` object for the specified deep neural network, `net`.

`quantObj = dlquantizer(net,Name,Value)` creates a `dlquantizer` object for the specified network, with additional options specified by one or more name-value pair arguments.

Input Arguments

net — Pretrained neural network
`DAGNetwork` object | `dlnetwork` object | `SeriesNetwork` object | `yolov2ObjectDetector` object | `yolov3ObjectDetector` object | `yolov4ObjectDetector` object | `ssdObjectDetector` object

Pretrained neural network, specified as a `DAGNetwork`, `dlnetwork`, `SeriesNetwork`, `yolov2ObjectDetector`, `yolov3ObjectDetector`, `yolov4ObjectDetector`, or a `ssdObjectDetector` object.

Properties

`NetworkObject` — Pretrained neural network
`DAGNetwork` object | `dlnetwork` object | `SeriesNetwork` object | `yolov2ObjectDetector` object | `yolov3ObjectDetector` | `yolov4ObjectDetector` | `ssdObjectDetector` object

This property is read-only.

Pre-trained neural network, specified as a `DAGNetwork`, `dlnetwork`, `SeriesNetwork`, `yolov2ObjectDetector`, `yolov3ObjectDetector`, `yolov4ObjectDetector`, or a `ssdObjectDetector` object.

`ExecutionEnvironment` — Execution environment
`'GPU'` (default) | `'FPGA'` | `'CPU'` | `'MATLAB'`

Execution environment for the quantized network, specified as `'GPU'`, `'FPGA'`, `'CPU'`, or `'MATLAB'`. How the network is quantized depends on the choice of execution environment.

The `'MATLAB'` execution environment indicates a target-agnostic quantization of the neural network will be performed. This option does not require you to have target hardware in order to explore the quantized network in MATLAB.

Example: `'ExecutionEnvironment','FPGA'`

`Simulation` — Validate behavior of network quantized for FPGA environment using simulation
`'off'` (default) | `'on'`

Whether to validate behavior of network quantized for FPGA using simulation, specified as one of these values:

- `'on'` — Validate the behavior of the quantized network by simulating the quantized network in MATLAB and comparing the prediction results of the original single-precision network to the simulated prediction results of the quantized network.
- `'off'` — Generate code and validate the behavior of the quantized network on the target hardware.

Note This option is only valid when ExecutionEnvironment is set to `'FPGA'`.

Note Alternatively, you can use the `quantize` method to create a simulatable quantized network. The simulatable quantized network enables visibility of the quantized layers, weights, and biases of the network, as well as simulatable quantized inference behavior.

Example: `'Simulation', 'on'`

Object Functions

calibrate	Simulate and collect ranges of a deep neural network
validate	Quantize and validate a deep neural network
quantize	Quantize deep neural network

Examples

Quantize a Neural Network for GPU Target

This example shows how to quantize learnable parameters in the convolution layers of a neural network for GPU and explore the behavior of the quantized network. In this example, you quantize the squeezenet neural network after retraining the network to classify new images according to the "Train Deep Learning Network to Classify New Images" example. In this example, the memory required for the network is reduced approximately 75% through quantization while the accuracy of the network is not affected.

Load the pretrained network. `net` is the output network of the Train Deep Learning Network to Classify New Images example.

```
load squeezenetmerch
net
```

```
net =
  DAGNetwork with properties:

         Layers: [68×1 nnet.cnn.layer.Layer]
    Connections: [75×2 table]
     InputNames: {'data'}
    OutputNames: {'new_classoutput'}
```

Define calibration and validation data to use for quantization.

The calibration data is used to collect the dynamic ranges of the weights and biases in the convolution and fully connected layers of the network and the dynamic ranges of the activations in all layers of the network. For the best quantization results, the calibration data must be representative of inputs to the network.

The validation data is used to test the network after quantization to understand the effects of the limited range and precision of the quantized convolution layers in the network.

In this example, use the images in the `MerchData` data set. Define an `augmentedImageDatastore` object to resize the data for the network. Then, split the data into calibration and validation data sets.

```
unzip('MerchData.zip');
imds = imageDatastore('MerchData', ...
    'IncludeSubfolders',true, ...
    'LabelSource','foldernames');
[calData, valData] = splitEachLabel(imds, 0.7, 'randomized');
aug_calData = augmentedImageDatastore([227 227], calData);
aug_valData = augmentedImageDatastore([227 227], valData);
```

Create a `dlquantizer` object and specify the network to quantize.

```
quantObj = dlquantizer(net);
```

Define a metric function to use to compare the behavior of the network before and after quantization. This example uses the `hComputeModelAccuracy` metric function.

```
function accuracy = hComputeModelAccuracy(predictionScores, net, dataStore)
%% Computes model-level accuracy statistics
```

```
    % Load ground truth
    tmp = readall(dataStore);
    groundTruth = tmp.response;

    % Compare with predicted label with actual ground truth
    predictionError = {};
    for idx=1:numel(groundTruth)
        [~, idy] = max(predictionScores(idx,:));
        yActual = net.Layers(end).Classes(idy);
        predictionError{end+1} = (yActual == groundTruth(idx)); %#ok
    end

    % Sum all prediction errors.
    predictionError = [predictionError{:}];
    accuracy = sum(predictionError)/numel(predictionError);
end
```

Specify the metric function in a `dlquantizationOptions` object.

```
quantOpts = dlquantizationOptions('MetricFcn',{@(x)hComputeModelAccuracy(x, net, aug_valData)});
```

Use the `calibrate` function to exercise the network with sample inputs and collect range information. The `calibrate` function exercises the network and collects the dynamic ranges of the weights and biases in the convolution and fully connected layers of the network and the dynamic ranges of the activations in all layers of the network. The function returns a table. Each row of the table contains range information for a learnable parameter of the optimized network.

```
calResults = calibrate(quantObj, aug_calData)
```

```
calResults=121×5 table
        Optimized Layer Name         Network Layer Name     Learnables / Activations    MinValue
    _____________________________    ___________________    ________________________    _________

    {'conv1_Weights'           }    {'conv1'           }          "Weights"              -0.91985
    {'conv1_Bias'              }    {'conv1'           }          "Bias"                 -0.07925
    {'fire2-squeeze1x1_Weights'}    {'fire2-squeeze1x1'}          "Weights"                 -1.38
    {'fire2-squeeze1x1_Bias'   }    {'fire2-squeeze1x1'}          "Bias"                 -0.11641
    {'fire2-expand1x1_Weights' }    {'fire2-expand1x1' }          "Weights"               -0.7406
    {'fire2-expand1x1_Bias'    }    {'fire2-expand1x1' }          "Bias"                -0.060056
    {'fire2-expand3x3_Weights' }    {'fire2-expand3x3' }          "Weights"              -0.74397
    {'fire2-expand3x3_Bias'    }    {'fire2-expand3x3' }          "Bias"                -0.051778
    {'fire3-squeeze1x1_Weights'}    {'fire3-squeeze1x1'}          "Weights"               -0.7712
    {'fire3-squeeze1x1_Bias'   }    {'fire3-squeeze1x1'}          "Bias"                 -0.10138
    {'fire3-expand1x1_Weights' }    {'fire3-expand1x1' }          "Weights"              -0.72035
    {'fire3-expand1x1_Bias'    }    {'fire3-expand1x1' }          "Bias"                -0.067029
    {'fire3-expand3x3_Weights' }    {'fire3-expand3x3' }          "Weights"              -0.61443
    {'fire3-expand3x3_Bias'    }    {'fire3-expand3x3' }          "Bias"                -0.053613
    {'fire4-squeeze1x1_Weights'}    {'fire4-squeeze1x1'}          "Weights"               -0.7422
    {'fire4-squeeze1x1_Bias'   }    {'fire4-squeeze1x1'}          "Bias"                 -0.10885
      ⋮
```

Use the `validate` function to quantize the learnable parameters in the convolution layers of the network and exercise the network. The function uses the metric function defined in the `dlquantizationOptions` object to compare the results of the network before and after quantization.

```
valResults = validate(quantObj, aug_valData, quantOpts)
```

```
valResults = struct with fields:
       NumSamples: 20
    MetricResults: [1×1 struct]
       Statistics: [2×2 table]
```

Examine the validation output to see the performance of the quantized network.

```
valResults.MetricResults.Result
```

```
ans=2×2 table
    NetworkImplementation    MetricOutput
    _____________________    ____________

     {'Floating-Point'}           1
     {'Quantized'     }           1
```

```
valResults.Statistics
```

```
ans=2×2 table
    NetworkImplementation    LearnableParameterMemory(bytes)
    _____________________    _______________________________

     {'Floating-Point'}               2.9003e+06
     {'Quantized'     }               7.3393e+05
```

In this example, the memory required for the network was reduced approximately 75% through quantization. The accuracy of the network is not affected.

The weights, biases, and activations of the convolution layers of the network specified in the dlquantizer object now use scaled 8-bit integer data types.

Quantize a Neural Network for CPU Target

This example shows how to quantize and validate a neural network for a CPU target. This workflow is similar to other execution environments, but before validating you must establish a `raspi` connection and specify it as target using `dlquantizationOptions`.

First, load your network. This example uses the pretrained network `squeezenet`.

```
load squeezenetmerch
net
```

```
net =
  DAGNetwork with properties:

         Layers: [68×1 nnet.cnn.layer.Layer]
    Connections: [75×2 table]
     InputNames: {'data'}
    OutputNames: {'new_classoutput'}
```

Then define your calibration and validation data, `calDS` and `valDS` respectively.

```
unzip('MerchData.zip');
imds = imageDatastore('MerchData', ...
    'IncludeSubfolders',true, ...
    'LabelSource','foldernames');
[calData, valData] = splitEachLabel(imds, 0.7, 'randomized');
aug_calData = augmentedImageDatastore([227 227],calData);
aug_valData = augmentedImageDatastore([227 227],valData);
```

Create the `dlquantizer` object and specify a CPU execution environment.

```
dq =  dlquantizer(net,'ExecutionEnvironment','CPU')
```

```
dq =
  dlquantizer with properties:

           NetworkObject: [1×1 DAGNetwork]
    ExecutionEnvironment: 'CPU'
```

Calibrate the network.

```
calResults = calibrate(dq,aug_calData,'UseGPU','off')
```

```
calResults=122×5 table
        Optimized Layer Name         Network Layer Name     Learnables / Activations    MinValue
    ____________________________    ____________________    ________________________    ________

    {'conv1_Weights'           }    {'conv1'           }           "Weights"             -0.91985
    {'conv1_Bias'              }    {'conv1'           }           "Bias"                -0.07925
    {'fire2-squeeze1x1_Weights'}    {'fire2-squeeze1x1'}           "Weights"                -1.38
    {'fire2-squeeze1x1_Bias'   }    {'fire2-squeeze1x1'}           "Bias"                -0.11641
    {'fire2-expand1x1_Weights' }    {'fire2-expand1x1' }           "Weights"              -0.7400
    {'fire2-expand1x1_Bias'    }    {'fire2-expand1x1' }           "Bias"                -0.06005
    {'fire2-expand3x3_Weights' }    {'fire2-expand3x3' }           "Weights"              -0.7439
    {'fire2-expand3x3_Bias'    }    {'fire2-expand3x3' }           "Bias"                -0.05177
    {'fire3-squeeze1x1_Weights'}    {'fire3-squeeze1x1'}           "Weights"              -0.7712
    {'fire3-squeeze1x1_Bias'   }    {'fire3-squeeze1x1'}           "Bias"                -0.10138
    {'fire3-expand1x1_Weights' }    {'fire3-expand1x1' }           "Weights"              -0.7203
    {'fire3-expand1x1_Bias'    }    {'fire3-expand1x1' }           "Bias"                -0.06702
    {'fire3-expand3x3_Weights' }    {'fire3-expand3x3' }           "Weights"              -0.6144
    {'fire3-expand3x3_Bias'    }    {'fire3-expand3x3' }           "Bias"                -0.05361
    {'fire4-squeeze1x1_Weights'}    {'fire4-squeeze1x1'}           "Weights"              -0.7422
    {'fire4-squeeze1x1_Bias'   }    {'fire4-squeeze1x1'}           "Bias"                -0.10885
      ⋮
```

Use the MATLAB Support Package for Raspberry Pi Hardware function, `raspi`, to create a connection to the Raspberry Pi. In the following code, replace:

- `raspiname` with the name or address of your Raspberry Pi
- `username` with your user name
- `password` with your password

```
% r = raspi('raspiname','username','password')
```

For example,

```
r = raspi('gpucoder-raspberrypi-7','pi','matlab')

r =
  raspi with properties:

         DeviceAddress: 'gpucoder-raspberrypi-7'
                  Port: 18734
             BoardName: 'Raspberry Pi 3 Model B+'
         AvailableLEDs: {'led0'}
  AvailableDigitalPins: [2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27]
  AvailableSPIChannels: {}
     AvailableI2CBuses: {}
      AvailableWebcams: {}
           I2CBusSpeed:
AvailableCANInterfaces: {}

  Supported peripherals
```

Specify `raspi` object as the target for the quantized network.

```
opts = dlquantizationOptions('Target',r)

opts =
  dlquantizationOptions with properties:

    MetricFcn: {}
    Bitstream: ''
       Target: [1×1 raspi]
```

Validate the quantized network with the `validate` function.

```
valResults = validate(dq,aug_valData,opts)

### Starting application: 'codegen\lib\validate_predict_int8\pil\validate_predict_int8.elf'
    To terminate execution: clear validate_predict_int8_pil
### Launching application validate_predict_int8.elf...
### Host application produced the following standard output (stdout) and standard error (stderr)

valResults = struct with fields:
       NumSamples: 20
    MetricResults: [1×1 struct]
       Statistics: []
```

Examine the validation output to see the performance of the quantized network.

```
valResults.MetricResults.Result

ans=2×2 table
    NetworkImplementation    MetricOutput
    _____________________    ____________

     {'Floating-Point'}          0.95
     {'Quantized'     }          0.95
```

Quantize Network for FPGA Deployment

Reduce the memory footprint of a deep neural network by quantizing the weights, biases, and activations of convolution layers to 8-bit scaled integer data types. This example shows how to use Deep Learning Toolbox Model Quantization Library and Deep Learning HDL Toolbox to deploy the `int8` network to a target FPGA board.

For this example, you need:

- Deep Learning Toolbox ™
- Deep Learning HDL Toolbox ™
- Deep Learning Toolbox Model Quantization Library
- Deep Learning HDL Toolbox Support Package for Xilinx FPGA and SoC Devices
- MATLAB Coder Interface for Deep Learning.

Load Pretrained Network

Load the pretrained LogoNet network and analyze the network architecture.

```
snet = getLogoNetwork;
deepNetworkDesigner(snet);
```

Load Data

This example uses the logos_dataset data set. The data set consists of 320 images. Each image is 227-by-227 in size and has three color channels (RGB). Create an `augmentedImageDatastore` object for calibration and validation. Expedite calibration and validation by reducing the calibration data set to 20 images. The MATLAB simulation workflow has a maximum limit of five images when validating the quantized network. Reduce the validation data set sizes to five images. The FPGA validation workflow has a maximum limit of one image when validating the quantized network. Reduce the FPGA validation data set to a single image.

```
curDir = pwd;
unzip("logos_dataset.zip");
imageData = imageDatastore(fullfile(curDir,'logos_dataset'),...
'IncludeSubfolders',true,'FileExtensions','.JPG','LabelSource','foldernames');
[calibrationData, validationData] = splitEachLabel(imageData, 0.5,'randomized');
calibrationData_reduced = calibrationData.subset(1:20);
validationData_simulation = validationData.subset(1:5);
validationData_FPGA = validationData.subset(1:1);
```

Generate Calibration Result File for the Network

Create a `dlquantizer` object and specify the network to quantize. Specify the execution environment as FPGA.

```
dlQuantObj_simulation = dlquantizer(snet,'ExecutionEnvironment',"FPGA",'Simulation','on');
dlQuantObj_FPGA = dlquantizer(snet,'ExecutionEnvironment',"FPGA");
```

Use the `calibrate` function to exercise the network with sample inputs and collect the range information. The `calibrate` function collects the dynamic ranges of the weights and biases. The calibrate function returns a table. Each row of the table contains range information for a learnable parameter of the quantized network.

```
calibrate(dlQuantObj_simulation,calibrationData_reduced)
```

```
ans=35×5 table
        Optimized Layer Name        Network Layer Name    Learnables / Activations     MinValue
    ____________________________    __________________    ________________________    __________

    {'conv_1_Weights'          }      {'conv_1'    }          "Weights"                -0.04897
    {'conv_1_Bias'             }      {'conv_1'    }          "Bias"                     0.9999
    {'conv_2_Weights'          }      {'conv_2'    }          "Weights"                -0.05551
    {'conv_2_Bias'             }      {'conv_2'    }          "Bias"                  -0.000611
    {'conv_3_Weights'          }      {'conv_3'    }          "Weights"                -0.04594
    {'conv_3_Bias'             }      {'conv_3'    }          "Bias"                   -0.00139
    {'conv_4_Weights'          }      {'conv_4'    }          "Weights"                -0.04596
    {'conv_4_Bias'             }      {'conv_4'    }          "Bias"                    -0.0016
    {'fc_1_Weights'            }      {'fc_1'      }          "Weights"                -0.05139
    {'fc_1_Bias'               }      {'fc_1'      }          "Bias"                  -0.000523
    {'fc_2_Weights'            }      {'fc_2'      }          "Weights"                 -0.0501
    {'fc_2_Bias'               }      {'fc_2'      }          "Bias"                   -0.00175
    {'fc_3_Weights'            }      {'fc_3'      }          "Weights"                -0.05070
    {'fc_3_Bias'               }      {'fc_3'      }          "Bias"                    -0.0295
    {'imageinput'              }      {'imageinput'}          "Activations"                   0
    {'imageinput_normalization'}      {'imageinput'}          "Activations"              -139.34
      ⋮
```

```
calibrate(dlQuantObj_FPGA,calibrationData_reduced)
```

```
ans=35×5 table
        Optimized Layer Name        Network Layer Name    Learnables / Activations     MinValue
    ____________________________    __________________    ________________________    __________

    {'conv_1_Weights'          }      {'conv_1'    }          "Weights"                -0.04897
    {'conv_1_Bias'             }      {'conv_1'    }          "Bias"                     0.9999
    {'conv_2_Weights'          }      {'conv_2'    }          "Weights"                -0.05551
    {'conv_2_Bias'             }      {'conv_2'    }          "Bias"                  -0.000611
    {'conv_3_Weights'          }      {'conv_3'    }          "Weights"                -0.04594
    {'conv_3_Bias'             }      {'conv_3'    }          "Bias"                   -0.00139
    {'conv_4_Weights'          }      {'conv_4'    }          "Weights"                -0.04596
    {'conv_4_Bias'             }      {'conv_4'    }          "Bias"                    -0.0016
    {'fc_1_Weights'            }      {'fc_1'      }          "Weights"                -0.05139
    {'fc_1_Bias'               }      {'fc_1'      }          "Bias"                  -0.000523
    {'fc_2_Weights'            }      {'fc_2'      }          "Weights"                 -0.0501
    {'fc_2_Bias'               }      {'fc_2'      }          "Bias"                   -0.00175
    {'fc_3_Weights'            }      {'fc_3'      }          "Weights"                -0.05070
    {'fc_3_Bias'               }      {'fc_3'      }          "Bias"                    -0.0295
    {'imageinput'              }      {'imageinput'}          "Activations"                   0
    {'imageinput_normalization'}      {'imageinput'}          "Activations"              -139.34
      ⋮
```

Create Target Object

Create a target object with a custom name for your target device and an interface to connect your target device to the host computer. Interface options are JTAG and Ethernet. To use JTAG, install Xilinx™ Vivado™ Design Suite 2020.2. To set the Xilinx Vivado toolpath, enter:

```
% hdlsetuptoolpath('ToolName', 'Xilinx Vivado', 'ToolPath', 'C:\Xilinx\Vivado
\2020.2\bin\vivado.bat');
```

To create the target object, enter:

```
hTarget = dlhdl.Target('Xilinx','Interface','Ethernet');
```

Alternatively, you can also use the JTAG interface.

```
% hTarget = dlhdl.Target('Xilinx', 'Interface', 'JTAG');
```

Create `dlQuantizationOptions` Object

Create a `dlquantizationOptions` object. Specify the target bitstream and target board interface. The default metric function is a Top-1 accuracy metric function.

```
options_FPGA = dlquantizationOptions('Bitstream','zcu102_int8','Target',hTarget);
options_simulation = dlquantizationOptions;
```

To use a custom metric function, specify the metric function in the `dlquantizationOptions` object.

```
options_FPGA = dlquantizationOptions('MetricFcn',{@(x)hComputeAccuracy(x,snet,validationData_FPGA
options_simulation = dlquantizationOptions('MetricFcn',{@(x)hComputeAccuracy(x,snet,validationDat
```

Validate Quantized Network

Use the `validate` function to quantize the learnable parameters in the convolution layers of the network. The `validate` function simulates the quantized network in MATLAB. The `validate` function uses the metric function defined in the `dlquantizationOptions` object to compare the results of the single-data-type network object to the results of the quantized network object.

```
prediction_simulation = dlQuantObj_simulation.validate(validationData_simulation,options_simulati
```

```
Compiling leg: conv_1>>relu_4 ...
Compiling leg: conv_1>>relu_4 ... complete.
Compiling leg: maxpool_4 ...
Compiling leg: maxpool_4 ... complete.
Compiling leg: fc_1>>fc_3 ...
Compiling leg: fc_1>>fc_3 ... complete.

prediction_simulation = struct with fields:
       NumSamples: 5
    MetricResults: [1×1 struct]
       Statistics: []
```

For validation on an FPGA, the validate function:

- Programs the FPGA board by using the output of the `compile` method and the programming file
- Downloads the network weights and biases
- Compares the performance of the network before and after quantization

```
prediction_FPGA = dlQuantObj_FPGA.validate(validationData_FPGA,options_FPGA)
```

```
### Compiling network for Deep Learning FPGA prototyping ...
### Targeting FPGA bitstream zcu102_int8.
### The network includes the following layers:
     1   'imageinput'    Image Input             227×227×3 images with 'zerocenter' normalizatio
     2   'conv_1'        Convolution             96 5×5×3 convolutions with stride [1  1] and pa
     3   'relu_1'        ReLU                    ReLU
     4   'maxpool_1'     Max Pooling             3×3 max pooling with stride [2  2] and padding
     5   'conv_2'        Convolution             128 3×3×96 convolutions with stride [1  1] and
```

```
     6   'relu_2'        ReLU                    ReLU
     7   'maxpool_2'     Max Pooling             3×3 max pooling with stride [2  2] and padding
     8   'conv_3'        Convolution             384 3×3×128 convolutions with stride [1  1] and
     9   'relu_3'        ReLU                    ReLU
    10   'maxpool_3'     Max Pooling             3×3 max pooling with stride [2  2] and padding
    11   'conv_4'        Convolution             128 3×3×384 convolutions with stride [2  2] and
    12   'relu_4'        ReLU                    ReLU
    13   'maxpool_4'     Max Pooling             3×3 max pooling with stride [2  2] and padding
    14   'fc_1'          Fully Connected         2048 fully connected layer
    15   'relu_5'        ReLU                    ReLU
    16   'dropout_1'     Dropout                 50% dropout
    17   'fc_2'          Fully Connected         2048 fully connected layer
    18   'relu_6'        ReLU                    ReLU
    19   'dropout_2'     Dropout                 50% dropout
    20   'fc_3'          Fully Connected         32 fully connected layer
    21   'softmax'       Softmax                 softmax
    22   'classoutput'   Classification Output   crossentropyex with 'adidas' and 31 other classe

### Notice: The layer 'imageinput' with type 'nnet.cnn.layer.ImageInputLayer' is implemented in s
### Notice: The layer 'softmax' with type 'nnet.cnn.layer.SoftmaxLayer' is implemented in softwar
### Notice: The layer 'classoutput' with type 'nnet.cnn.layer.ClassificationOutputLayer' is imple
### Compiling layer group: conv_1>>relu_4 ...
### Compiling layer group: conv_1>>relu_4 ... complete.
### Compiling layer group: maxpool_4 ...
### Compiling layer group: maxpool_4 ... complete.
### Compiling layer group: fc_1>>fc_3 ...
### Compiling layer group: fc_1>>fc_3 ... complete.

### Allocating external memory buffers:

          offset_name          offset_address    allocated_space
    ___________________________    ______________    _______________

    "InputDataOffset"           "0x00000000"     "12.0 MB"
    "OutputResultOffset"        "0x00c00000"     "4.0 MB"
    "SchedulerDataOffset"       "0x01000000"     "4.0 MB"
    "SystemBufferOffset"        "0x01400000"     "36.0 MB"
    "InstructionDataOffset"     "0x03800000"     "8.0 MB"
    "ConvWeightDataOffset"      "0x04000000"     "12.0 MB"
    "FCWeightDataOffset"        "0x04c00000"     "12.0 MB"
    "EndOffset"                 "0x05800000"     "Total: 88.0 MB"

### Network compilation complete.

### FPGA bitstream programming has been skipped as the same bitstream is already loaded on the ta
### Deep learning network programming has been skipped as the same network is already loaded on t
### Finished writing input activations.
### Running single input activation.


              Deep Learning Processor Bitstream Build Info

Resource                   Utilized           Total        Percentage
------------------        ----------      ------------    ------------
LUTs (CLB/ALM)*              248358            274080           90.62
DSPs                            384              2520           15.24
Block RAM                       581               912           63.71
* LUT count represents Configurable Logic Block(CLB) utilization in Xilinx devices and Adaptive L
```

```
### Notice: The layer 'imageinput' of type 'ImageInputLayer' is split into an image input layer
### Notice: The layer 'softmax' with type 'nnet.cnn.layer.SoftmaxLayer' is implemented in softwa
### Notice: The layer 'classoutput' with type 'nnet.cnn.layer.ClassificationOutputLayer' is imple


              Deep Learning Processor Estimator Performance Results

                     LastFrameLatency(cycles)   LastFrameLatency(seconds)       FramesNum      Tota
                           -------------              -------------               ---------        --
Network                     40142478                  0.18247                       1           401
    ____imageinput_norm       216472                  0.00098
    ____conv_1               6825671                  0.03103
    ____maxpool_1            3755088                  0.01707
    ____conv_2              10440701                  0.04746
    ____maxpool_2            1447840                  0.00658
    ____conv_3               9405685                  0.04275
    ____maxpool_3            1765856                  0.00803
    ____conv_4               1819636                  0.00827
    ____maxpool_4              28098                  0.00013
    ____fc_1                 2651288                  0.01205
    ____fc_2                 1696632                  0.00771
    ____fc_3                   89511                  0.00041
 * The clock frequency of the DL processor is: 220MHz



              Deep Learning Processor Bitstream Build Info

Resource                   Utilized           Total        Percentage
------------------        ----------      ------------    ------------
LUTs (CLB/ALM)*              168645            274080           61.53
DSPs                            800              2520           31.75
Block RAM                       453               912           49.67
* LUT count represents Configurable Logic Block(CLB) utilization in Xilinx devices and Adaptive L

### Finished writing input activations.
### Running single input activation.

prediction_FPGA = struct with fields:
       NumSamples: 1
    MetricResults: [1×1 struct]
       Statistics: [2×7 table]
```

View Performance of Quantized Neural Network

Display the accuracy of the quantized network.

```
prediction_simulation.MetricResults.Result
```

```
ans=2×2 table
    NetworkImplementation    MetricOutput
    _____________________    ____________

     {'Floating-Point'}           1
     {'Quantized'     }           1
```

```
prediction_FPGA.MetricResults.Result
```

```
ans=2×2 table
    NetworkImplementation    MetricOutput
    _____________________    ____________

     {'Floating-Point'}           1
     {'Quantized'     }           1
```

Display the performance of the quantized network in frames per second.

```
prediction_FPGA.Statistics.FramesPerSecond(2)
```

```
ans = 19.0828
```

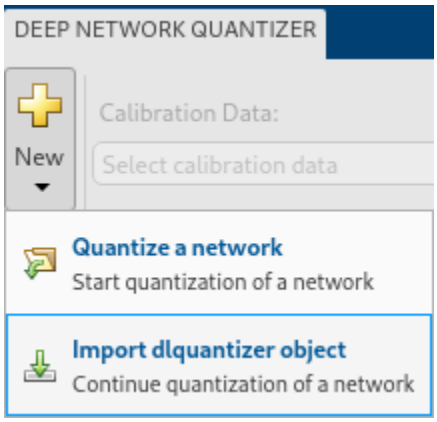
Import a `dlquantizer` Object into the Deep Network Quantizer App

This example shows you how to import a `dlquantizer` object from the base workspace into the **Deep Network Quantizer** app. This allows you to begin quantization of a deep neural network using the command line or the app, and resume your work later in the app.

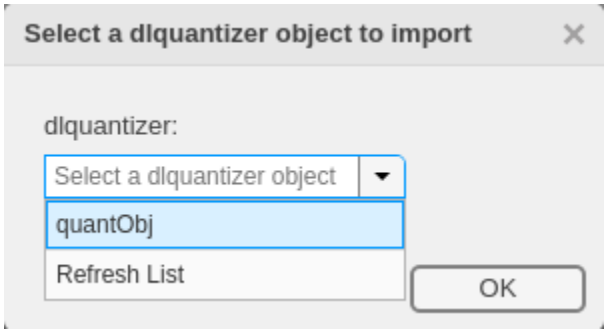
Open the **Deep Network Quantizer** app.

```
deepNetworkQuantizer
```

In the app, click **New** and select `Import dlquantizer object`.

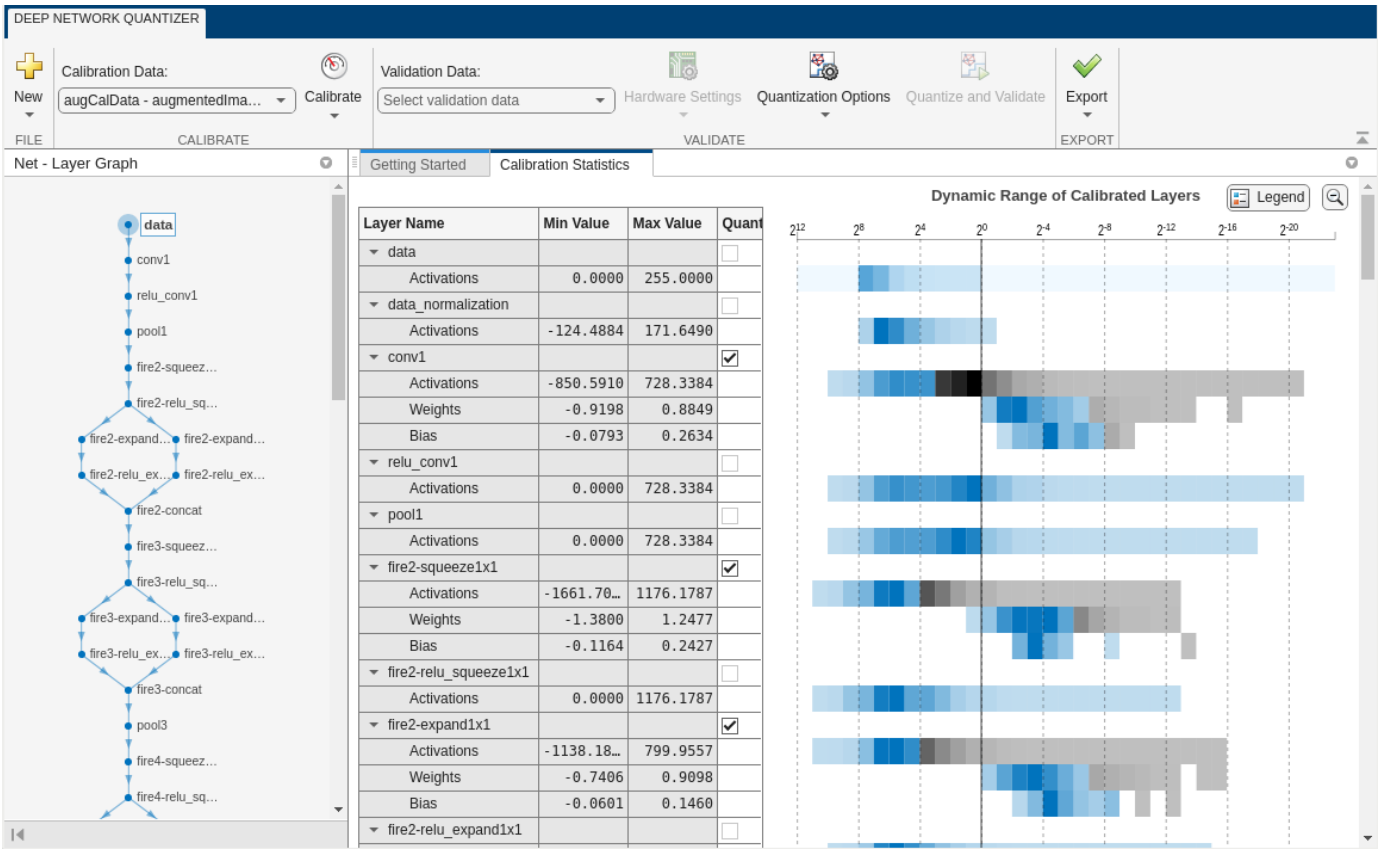


In the dialog, select the `dlquantizer` object to import from the base workspace. For this example, use `quantObj` that you create in the above example Quantize a Neural Network for GPU Target.



The app imports any data contained in the `dlquantizer` object that was collected at the command line. This data can include the network to quantize, calibration data, validation data, and calibration statistics.

The app displays a table containing the calibration data contained in the imported `dlquantizer` object, `quantObj`. To the right of the table, the app displays histograms of the dynamic ranges of the parameters. The gray regions of the histograms indicate data that cannot be represented by the quantized representation. For more information on how to interpret these histograms, see “Quantization of Deep Neural Networks”.



Emulate Target Agnostic Quantized Network

This example shows how to create a target agnostic, simulatable quantized deep neural network in MATLAB.

Target agnostic quantization allows you to see the effect quantization has on your neural network without target hardware or target-specific quantization schemes. Creating a target agnostic quantized network is useful if you:

- Do not have access to your target hardware.
- Want to preview whether or not your network is suitable for quantization.
- Want to find layers that are sensitive to quantization.

Quantized networks emulate quantized behavior for quantization-compatible layers. Network architecture like layers and connections are the same as the original network, but inference behavior uses limited precision types. Once you have quantized your network, you can use the quantizationDetails function to retrieve details on what was quantized.

Load the pretrained network. `net` is a SqueezeNet network that has been retrained using transfer learning to classify images in the `MerchData` data set.

```
load squeezenetmerch
net
```

```
net =
  DAGNetwork with properties:

         Layers: [68×1 nnet.cnn.layer.Layer]
    Connections: [75×2 table]
     InputNames: {'data'}
    OutputNames: {'new_classoutput'}
```

You can use the `quantizationDetails` function to see that the network is not quantized.

```
qDetailsOriginal = quantizationDetails(net)
```

```
qDetailsOriginal = struct with fields:
           IsQuantized: 0
         TargetLibrary: ""
    QuantizedLayerNames: [0×0 string]
    QuantizedLearnables: [0×3 table]
```

Unzip and load the `MerchData` images as an image datastore.

```
unzip('MerchData.zip')
imds = imageDatastore('MerchData', ...
    'IncludeSubfolders',true, ...
    'LabelSource','foldernames');
```

Define calibration and validation data to use for quantization. The output size of the images are changed for both calibration and validation data according to network requirements.

```
[calData,valData] = splitEachLabel(imds,0.7,'randomized');
augCalData = augmentedImageDatastore([227 227],calData);
augValData = augmentedImageDatastore([227 227],valData);
```

Create `dlquantizer` object and specify the network to quantize. Set the execution environment to MATLAB. How the network is quantized depends on the execution environment. The MATLAB execution environment is agnostic to the target hardware and allows you to prototype quantized behavior.

```
quantObj = dlquantizer(net,'ExecutionEnvironment','MATLAB');
```

Use the `calibrate` function to exercise the network with sample inputs and collect range information. The `calibrate` function exercises the network and collects the dynamic ranges of the weights and biases in the convolution and fully connected layers of the network and the dynamic ranges of the activations in all layers of the network. The function returns a table. Each row of the table contains range information for a learnable parameter of the optimized network.

```
calResults = calibrate(quantObj,augCalData);
```

Use the `quantize` method to quantize the network object and return a simulatable quantized network.

```
qNet = quantize(quantObj)
```

```
qNet =
Quantized DAGNetwork with properties:

         Layers: [68×1 nnet.cnn.layer.Layer]
    Connections: [75×2 table]
     InputNames: {'data'}
    OutputNames: {'new_classoutput'}

Use the quantizationDetails method to extract quantization details.
```

You can use the `quantizationDetails` function to see that the network is now quantized.

```
qDetailsQuantized = quantizationDetails(qNet)
```

```
qDetailsQuantized = struct with fields:
            IsQuantized: 1
          TargetLibrary: "none"
    QuantizedLayerNames: [26×1 string]
    QuantizedLearnables: [52×3 table]
```

Make predictions using the original, single-precision floating-point network, and the quantized INT8 network.

```
predOriginal = classify(net,augValData);       % Predictions for the non-quantized network
predQuantized = classify(qNet,augValData);     % Predictions for the quantized network
```

Compute the relative accuracy of the quantized network as compared to the original network.

```
ccrQuantized = mean(predQuantized == valData.Labels)*100
```

```
ccrQuantized = 100
```

```
ccrOriginal = mean(predOriginal == valData.Labels)*100
```

```
ccrOriginal = 100
```

For this validation data set, the quantized network gives the same predictions as the floating-point network.

Version History

Introduced in R2020a

R2023a: `dlnetwork` support

`dlquantizer` now supports `dlnetwork` objects for quantization using the `quantize` function.

R2022b: `dlnetwork` support

`dlquantizer` now supports `dlnetwork` objects for quantization using the `calibrate` and `validate` functions.

R2022a: Validate the performance of quantized network for CPU target

You can now use the `dlquantizer` object and the `validate` function to quantize a network and generate code for CPU targets.

R2022a: Quantize neural networks without a specific target

Specify `MATLAB` as the `ExecutionEnvironment` to quantize your neural networks without generating code or committing to a specific target for code deployment. This can be useful if you:

- Do not have access to your target hardware.
- Want to inspect your quantized network without generating code.

Your quantized network implements `int8` data instead of `single` data. It keeps the same layers and connections as the original network, and it has the same inference behavior as it would when running on hardware.

Once you have quantized your network, you can use the `quantizationDetails` function to inspect your quantized network. Additionally, you also have the option to deploy the code to a GPU target.

R2023a: Quantize `yolov3ObjectDetector` and `yolov4ObjectDetector` using `dlquantizer`

`yolov3ObjectDetector` and `yolov4ObjectDetector` objects can now be quantized using `dlquantizer`.

See Also

Apps
Deep Network Quantizer

Functions
`calibrate` | `quantize` | `validate` | `dlquantizationOptions` | `quantizationDetails` | `estimateNetworkMetrics`

Topics
“Quantization of Deep Neural Networks”
“Quantize Residual Network Trained for Image Classification and Generate CUDA Code”
“Quantize Layers in Object Detectors and Generate CUDA Code”
“Quantize Network for FPGA Deployment”
“Generate INT8 Code for Deep Learning Network on Raspberry Pi” (MATLAB Coder)
“Parameter Pruning and Quantization of Image Classification Network”

dlquantizationOptions

Options for quantizing a trained deep neural network

Description

The `dlquantizationOptions` object provides options for quantizing a trained deep neural network to scaled 8-bit integer data types.

This object requires Deep Learning Toolbox Model Quantization Library. To learn about the products required to quantize a deep neural network, see “Quantization Workflow Prerequisites”.

Creation

Syntax

```
quantOpts = dlquantizationOptions
quantOpts = dlquantizationOptions(Name,Value)
```

Description

`quantOpts = dlquantizationOptions` creates a `dlquantizationOptions` object with default property values.

`quantOpts = dlquantizationOptions(Name,Value)` creates a `dlquantizationOptions` object with additional properties specified by one or more name-value pair arguments.

Properties

`MetricFcn` — Metric function to use for validation of quantized network

cell array of function handles

Metric function to use for validation of quantized network, specified as a cell array of one or more function handles.

Example: `options = dlquantizationOptions('MetricFcn', {@(x)hComputeModelAccuracy(x,net,groundTruth)});`

Data Types: `cell`

Execution Environment Options

`Bitstream` — Name of FPGA bitstream

`'zcu102_int8'` | `'zc706_int8'` | `'arria10soc_int8'`

This property is valid only when the `'ExecutionEnvironment` property of the `dlquantizer` object is set to `'FPGA'`.

Name of the FPGA bitstream specified, as one of these values:

Bitstream	Target Board
'zcu102_int8'	Xilinx Zynq UltraScale™ ZCU102
'zc706_int8'	Xilinx Zynq-7000 ZC706
'arria10soc_int8'	Intel Arria 10 SoC development kit

Example: `quantOpts = dlquantizationOptions('Bitstream','zcu102_int8')`

`Target` — Target for quantized network
`dlhdl.Target` object | `raspi` object

Target for quantized network, specified as one of the following:

Target	Execution Environment for Quantized Network	Example
Target CPU board specified as a `raspi` object	Set `Target` property as a `raspi` object only when `'ExecutionEnvironment` property of the `dlquantizer` object is set to `'CPU'`	`r = raspi('hostname','User Name','Password'); quantOpts = dlquantizationOptions('Target',r)`
Target FPGA board vendor name and interface, specified as a `dlhdl.Target` object	Set `Target` property as a `dlhdl.Target` object only when `'ExecutionEnvironment` property of the `dlquantizer` object is set to `'FPGA'`	`hTarget = dlhdl.Target('Intel','Interface','JTAG'); quantOpts = dlquantizationOptions('Target',hTarget)`

Examples

Quantize a Neural Network for GPU Target

This example shows how to quantize learnable parameters in the convolution layers of a neural network for GPU and explore the behavior of the quantized network. In this example, you quantize the squeezenet neural network after retraining the network to classify new images according to the “Train Deep Learning Network to Classify New Images” example. In this example, the memory required for the network is reduced approximately 75% through quantization while the accuracy of the network is not affected.

Load the pretrained network. `net` is the output network of the Train Deep Learning Network to Classify New Images example.

```
load squeezenetmerch
net
```

```
net =
  DAGNetwork with properties:
```

```
     Layers: [68×1 nnet.cnn.layer.Layer]
Connections: [75×2 table]
 InputNames: {'data'}
OutputNames: {'new_classoutput'}
```

Define calibration and validation data to use for quantization.

The calibration data is used to collect the dynamic ranges of the weights and biases in the convolution and fully connected layers of the network and the dynamic ranges of the activations in all layers of the network. For the best quantization results, the calibration data must be representative of inputs to the network.

The validation data is used to test the network after quantization to understand the effects of the limited range and precision of the quantized convolution layers in the network.

In this example, use the images in the `MerchData` data set. Define an `augmentedImageDatastore` object to resize the data for the network. Then, split the data into calibration and validation data sets.

```
unzip('MerchData.zip');
imds = imageDatastore('MerchData', ...
    'IncludeSubfolders',true, ...
    'LabelSource','foldernames');
[calData, valData] = splitEachLabel(imds, 0.7, 'randomized');
aug_calData = augmentedImageDatastore([227 227], calData);
aug_valData = augmentedImageDatastore([227 227], valData);
```

Create a `dlquantizer` object and specify the network to quantize.

```
quantObj = dlquantizer(net);
```

Define a metric function to use to compare the behavior of the network before and after quantization. This example uses the `hComputeModelAccuracy` metric function.

```
function accuracy = hComputeModelAccuracy(predictionScores, net, dataStore)
%% Computes model-level accuracy statistics

    % Load ground truth
    tmp = readall(dataStore);
    groundTruth = tmp.response;

    % Compare with predicted label with actual ground truth
    predictionError = {};
    for idx=1:numel(groundTruth)
        [~, idy] = max(predictionScores(idx,:));
        yActual = net.Layers(end).Classes(idy);
        predictionError{end+1} = (yActual == groundTruth(idx)); %#ok
    end

    % Sum all prediction errors.
    predictionError = [predictionError{:}];
    accuracy = sum(predictionError)/numel(predictionError);
end
```

Specify the metric function in a `dlquantizationOptions` object.

```
quantOpts = dlquantizationOptions('MetricFcn',{@(x)hComputeModelAccuracy(x, net, aug_valData)});
```

Use the `calibrate` function to exercise the network with sample inputs and collect range information. The `calibrate` function exercises the network and collects the dynamic ranges of the weights and biases in the convolution and fully connected layers of the network and the dynamic ranges of the activations in all layers of the network. The function returns a table. Each row of the table contains range information for a learnable parameter of the optimized network.

```
calResults = calibrate(quantObj, aug_calData)
```

```
calResults=121×5 table
        Optimized Layer Name         Network Layer Name     Learnables / Activations    MinValue
    _____________________________    ___________________    ________________________    _________

    {'conv1_Weights'           }    {'conv1'           }           "Weights"             -0.91985
    {'conv1_Bias'              }    {'conv1'           }           "Bias"                -0.07925
    {'fire2-squeeze1x1_Weights'}    {'fire2-squeeze1x1'}           "Weights"                -1.38
    {'fire2-squeeze1x1_Bias'   }    {'fire2-squeeze1x1'}           "Bias"                -0.11641
    {'fire2-expand1x1_Weights' }    {'fire2-expand1x1' }           "Weights"              -0.7406
    {'fire2-expand1x1_Bias'    }    {'fire2-expand1x1' }           "Bias"                -0.06005
    {'fire2-expand3x3_Weights' }    {'fire2-expand3x3' }           "Weights"             -0.74397
    {'fire2-expand3x3_Bias'    }    {'fire2-expand3x3' }           "Bias"                -0.05177
    {'fire3-squeeze1x1_Weights'}    {'fire3-squeeze1x1'}           "Weights"              -0.7712
    {'fire3-squeeze1x1_Bias'   }    {'fire3-squeeze1x1'}           "Bias"                -0.10138
    {'fire3-expand1x1_Weights' }    {'fire3-expand1x1' }           "Weights"             -0.72035
    {'fire3-expand1x1_Bias'    }    {'fire3-expand1x1' }           "Bias"                -0.06702
    {'fire3-expand3x3_Weights' }    {'fire3-expand3x3' }           "Weights"             -0.61443
    {'fire3-expand3x3_Bias'    }    {'fire3-expand3x3' }           "Bias"                -0.05361
    {'fire4-squeeze1x1_Weights'}    {'fire4-squeeze1x1'}           "Weights"              -0.7422
    {'fire4-squeeze1x1_Bias'   }    {'fire4-squeeze1x1'}           "Bias"                -0.10885
      ⋮
```

Use the `validate` function to quantize the learnable parameters in the convolution layers of the network and exercise the network. The function uses the metric function defined in the `dlquantizationOptions` object to compare the results of the network before and after quantization.

```
valResults = validate(quantObj, aug_valData, quantOpts)
```

```
valResults = struct with fields:
       NumSamples: 20
    MetricResults: [1×1 struct]
       Statistics: [2×2 table]
```

Examine the validation output to see the performance of the quantized network.

```
valResults.MetricResults.Result
```

```
ans=2×2 table
    NetworkImplementation    MetricOutput
    _____________________    ____________

     {'Floating-Point'}           1
     {'Quantized'     }           1
```

```
valResults.Statistics
```

```
ans=2×2 table
    NetworkImplementation    LearnableParameterMemory(bytes)
    _____________________    _______________________________

     {'Floating-Point'}                2.9003e+06
     {'Quantized'     }                7.3393e+05
```

In this example, the memory required for the network was reduced approximately 75% through quantization. The accuracy of the network is not affected.

The weights, biases, and activations of the convolution layers of the network specified in the dlquantizer object now use scaled 8-bit integer data types.

Quantize Network for FPGA Deployment

Reduce the memory footprint of a deep neural network by quantizing the weights, biases, and activations of convolution layers to 8-bit scaled integer data types. This example shows how to use Deep Learning Toolbox Model Quantization Library and Deep Learning HDL Toolbox to deploy the `int8` network to a target FPGA board.

For this example, you need:

- Deep Learning Toolbox ™
- Deep Learning HDL Toolbox ™
- Deep Learning Toolbox Model Quantization Library
- Deep Learning HDL Toolbox Support Package for Xilinx FPGA and SoC Devices
- MATLAB Coder Interface for Deep Learning.

Load Pretrained Network

Load the pretrained LogoNet network and analyze the network architecture.

```
snet = getLogoNetwork;
deepNetworkDesigner(snet);
```

Load Data

This example uses the logos_dataset data set. The data set consists of 320 images. Each image is 227-by-227 in size and has three color channels (RGB). Create an `augmentedImageDatastore` object for calibration and validation. Expedite calibration and validation by reducing the calibration data set to 20 images. The MATLAB simulation workflow has a maximum limit of five images when validating the quantized network. Reduce the validation data set sizes to five images. The FPGA validation workflow has a maximum limit of one image when validating the quantized network. Reduce the FPGA validation data set to a single image.

```
curDir = pwd;
unzip("logos_dataset.zip");
imageData = imageDatastore(fullfile(curDir,'logos_dataset'),...
'IncludeSubfolders',true,'FileExtensions','.JPG','LabelSource','foldernames');
[calibrationData, validationData] = splitEachLabel(imageData, 0.5,'randomized');
calibrationData_reduced = calibrationData.subset(1:20);
```

```
validationData_simulation = validationData.subset(1:5);
validationData_FPGA = validationData.subset(1:1);
```

Generate Calibration Result File for the Network

Create a `dlquantizer` object and specify the network to quantize. Specify the execution environment as FPGA.

```
dlQuantObj_simulation = dlquantizer(snet,'ExecutionEnvironment',"FPGA",'Simulation','on');
dlQuantObj_FPGA = dlquantizer(snet,'ExecutionEnvironment',"FPGA");
```

Use the `calibrate` function to exercise the network with sample inputs and collect the range information. The `calibrate` function collects the dynamic ranges of the weights and biases. The calibrate function returns a table. Each row of the table contains range information for a learnable parameter of the quantized network.

```
calibrate(dlQuantObj_simulation,calibrationData_reduced)
```

```
ans=35×5 table
        Optimized Layer Name        Network Layer Name    Learnables / Activations     MinValue
    ____________________________    __________________    ________________________    __________

    {'conv_1_Weights'          }      {'conv_1'    }          "Weights"                -0.04897
    {'conv_1_Bias'             }      {'conv_1'    }          "Bias"                     0.9999
    {'conv_2_Weights'          }      {'conv_2'    }          "Weights"                -0.05551
    {'conv_2_Bias'             }      {'conv_2'    }          "Bias"                  -0.000611
    {'conv_3_Weights'          }      {'conv_3'    }          "Weights"                -0.04594
    {'conv_3_Bias'             }      {'conv_3'    }          "Bias"                   -0.00139
    {'conv_4_Weights'          }      {'conv_4'    }          "Weights"                -0.04596
    {'conv_4_Bias'             }      {'conv_4'    }          "Bias"                    -0.0016
    {'fc_1_Weights'            }      {'fc_1'      }          "Weights"                -0.05139
    {'fc_1_Bias'               }      {'fc_1'      }          "Bias"                  -0.000523
    {'fc_2_Weights'            }      {'fc_2'      }          "Weights"                 -0.0501
    {'fc_2_Bias'               }      {'fc_2'      }          "Bias"                   -0.00175
    {'fc_3_Weights'            }      {'fc_3'      }          "Weights"                -0.05070
    {'fc_3_Bias'               }      {'fc_3'      }          "Bias"                    -0.0295
    {'imageinput'              }      {'imageinput'}          "Activations"                   0
    {'imageinput_normalization'}      {'imageinput'}          "Activations"              -139.3
      ⋮
```

```
calibrate(dlQuantObj_FPGA,calibrationData_reduced)
```

```
ans=35×5 table
        Optimized Layer Name        Network Layer Name    Learnables / Activations     MinValue
    ____________________________    __________________    ________________________    __________

    {'conv_1_Weights'          }      {'conv_1'    }          "Weights"                -0.04897
    {'conv_1_Bias'             }      {'conv_1'    }          "Bias"                     0.9999
    {'conv_2_Weights'          }      {'conv_2'    }          "Weights"                -0.05551
    {'conv_2_Bias'             }      {'conv_2'    }          "Bias"                  -0.000611
    {'conv_3_Weights'          }      {'conv_3'    }          "Weights"                -0.04594
    {'conv_3_Bias'             }      {'conv_3'    }          "Bias"                   -0.00139
    {'conv_4_Weights'          }      {'conv_4'    }          "Weights"                -0.04596
    {'conv_4_Bias'             }      {'conv_4'    }          "Bias"                    -0.0016
    {'fc_1_Weights'            }      {'fc_1'      }          "Weights"                -0.05139
    {'fc_1_Bias'               }      {'fc_1'      }          "Bias"                  -0.000523
    {'fc_2_Weights'            }      {'fc_2'      }          "Weights"                 -0.0501
```

```
    {'fc_2_Bias'               }      {'fc_2'      }          "Bias"                  -0.001756
    {'fc_3_Weights'            }      {'fc_3'      }          "Weights"                -0.050706
    {'fc_3_Bias'               }      {'fc_3'      }          "Bias"                    -0.0295
    {'imageinput'              }      {'imageinput'}          "Activations"                    0
    {'imageinput_normalization'}      {'imageinput'}          "Activations"             -139.34
      ⋮
```

Create Target Object

Create a target object with a custom name for your target device and an interface to connect your target device to the host computer. Interface options are JTAG and Ethernet. To use JTAG, install Xilinx™ Vivado™ Design Suite 2020.2. To set the Xilinx Vivado toolpath, enter:

```
% hdlsetuptoolpath('ToolName', 'Xilinx Vivado', 'ToolPath', 'C:\Xilinx\Vivado
\2020.2\bin\vivado.bat');
```

To create the target object, enter:

```
hTarget = dlhdl.Target('Xilinx','Interface','Ethernet');
```

Alternatively, you can also use the JTAG interface.

```
% hTarget = dlhdl.Target('Xilinx', 'Interface', 'JTAG');
```

Create `dlQuantizationOptions` Object

Create a `dlquantizationOptions` object. Specify the target bitstream and target board interface. The default metric function is a Top-1 accuracy metric function.

```
options_FPGA = dlquantizationOptions('Bitstream','zcu102_int8','Target',hTarget);
options_simulation = dlquantizationOptions;
```

To use a custom metric function, specify the metric function in the `dlquantizationOptions` object.

```
options_FPGA = dlquantizationOptions('MetricFcn',{@(x)hComputeAccuracy(x,snet,validationData_FPGA
options_simulation = dlquantizationOptions('MetricFcn',{@(x)hComputeAccuracy(x,snet,validationDat
```

Validate Quantized Network

Use the `validate` function to quantize the learnable parameters in the convolution layers of the network. The `validate` function simulates the quantized network in MATLAB. The `validate` function uses the metric function defined in the `dlquantizationOptions` object to compare the results of the single-data-type network object to the results of the quantized network object.

```
prediction_simulation = dlQuantObj_simulation.validate(validationData_simulation,options_simulati
```

```
Compiling leg: conv_1>>relu_4 ...
Compiling leg: conv_1>>relu_4 ... complete.
Compiling leg: maxpool_4 ...
Compiling leg: maxpool_4 ... complete.
Compiling leg: fc_1>>fc_3 ...
Compiling leg: fc_1>>fc_3 ... complete.

prediction_simulation = struct with fields:
       NumSamples: 5
    MetricResults: [1×1 struct]
```

```
       Statistics: []
```

For validation on an FPGA, the validate function:

- Programs the FPGA board by using the output of the `compile` method and the programming file
- Downloads the network weights and biases
- Compares the performance of the network before and after quantization

```
prediction_FPGA = dlQuantObj_FPGA.validate(validationData_FPGA,options_FPGA)
```

```
### Compiling network for Deep Learning FPGA prototyping ...
### Targeting FPGA bitstream zcu102_int8.
### The network includes the following layers:
     1   'imageinput'    Image Input             227×227×3 images with 'zerocenter' normalizatio
     2   'conv_1'        Convolution             96 5×5×3 convolutions with stride [1  1] and pa
     3   'relu_1'        ReLU                    ReLU
     4   'maxpool_1'     Max Pooling             3×3 max pooling with stride [2  2] and padding
     5   'conv_2'        Convolution             128 3×3×96 convolutions with stride [1  1] and
     6   'relu_2'        ReLU                    ReLU
     7   'maxpool_2'     Max Pooling             3×3 max pooling with stride [2  2] and padding
     8   'conv_3'        Convolution             384 3×3×128 convolutions with stride [1  1] and
     9   'relu_3'        ReLU                    ReLU
    10   'maxpool_3'     Max Pooling             3×3 max pooling with stride [2  2] and padding
    11   'conv_4'        Convolution             128 3×3×384 convolutions with stride [2  2] and
    12   'relu_4'        ReLU                    ReLU
    13   'maxpool_4'     Max Pooling             3×3 max pooling with stride [2  2] and padding
    14   'fc_1'          Fully Connected         2048 fully connected layer
    15   'relu_5'        ReLU                    ReLU
    16   'dropout_1'     Dropout                 50% dropout
    17   'fc_2'          Fully Connected         2048 fully connected layer
    18   'relu_6'        ReLU                    ReLU
    19   'dropout_2'     Dropout                 50% dropout
    20   'fc_3'          Fully Connected         32 fully connected layer
    21   'softmax'       Softmax                 softmax
    22   'classoutput'   Classification Output   crossentropyex with 'adidas' and 31 other classe

### Notice: The layer 'imageinput' with type 'nnet.cnn.layer.ImageInputLayer' is implemented in s
### Notice: The layer 'softmax' with type 'nnet.cnn.layer.SoftmaxLayer' is implemented in softwa
### Notice: The layer 'classoutput' with type 'nnet.cnn.layer.ClassificationOutputLayer' is imple
### Compiling layer group: conv_1>>relu_4 ...
### Compiling layer group: conv_1>>relu_4 ... complete.
### Compiling layer group: maxpool_4 ...
### Compiling layer group: maxpool_4 ... complete.
### Compiling layer group: fc_1>>fc_3 ...
### Compiling layer group: fc_1>>fc_3 ... complete.

### Allocating external memory buffers:

          offset_name          offset_address    allocated_space
    _______________________    ______________    _______________

    "InputDataOffset"           "0x00000000"     "12.0 MB"
    "OutputResultOffset"        "0x00c00000"     "4.0 MB"
    "SchedulerDataOffset"       "0x01000000"     "4.0 MB"
    "SystemBufferOffset"        "0x01400000"     "36.0 MB"
    "InstructionDataOffset"     "0x03800000"     "8.0 MB"
```

```
    "ConvWeightDataOffset"      "0x04000000"     "12.0 MB"
    "FCWeightDataOffset"        "0x04c00000"     "12.0 MB"
    "EndOffset"                 "0x05800000"     "Total: 88.0 MB"

### Network compilation complete.

### FPGA bitstream programming has been skipped as the same bitstream is already loaded on the ta
### Deep learning network programming has been skipped as the same network is already loaded on t
### Finished writing input activations.
### Running single input activation.


             Deep Learning Processor Bitstream Build Info

Resource                   Utilized           Total        Percentage
------------------        ----------      ------------    ------------
LUTs (CLB/ALM)*              248358            274080           90.62
DSPs                            384              2520           15.24
Block RAM                       581               912           63.71
* LUT count represents Configurable Logic Block(CLB) utilization in Xilinx devices and Adaptive L

### Notice: The layer 'imageinput' of type 'ImageInputLayer' is split into an image input layer
### Notice: The layer 'softmax' with type 'nnet.cnn.layer.SoftmaxLayer' is implemented in softwar
### Notice: The layer 'classoutput' with type 'nnet.cnn.layer.ClassificationOutputLayer' is imple


             Deep Learning Processor Estimator Performance Results

                     LastFrameLatency(cycles)   LastFrameLatency(seconds)       FramesNum      Tota
                           -------------             -------------              ---------        --
Network                    40142478                   0.18247                      1           401
    ____imageinput_norm     216472                    0.00098
    ____conv_1             6825671                   0.03103
    ____maxpool_1          3755088                   0.01707
    ____conv_2            10440701                   0.04746
    ____maxpool_2          1447840                   0.00658
    ____conv_3             9405685                   0.04275
    ____maxpool_3          1765856                   0.00803
    ____conv_4             1819636                   0.00827
    ____maxpool_4            28098                   0.00013
    ____fc_1               2651288                   0.01205
    ____fc_2               1696632                   0.00771
    ____fc_3                 89511                   0.00041
 * The clock frequency of the DL processor is: 220MHz



             Deep Learning Processor Bitstream Build Info

Resource                   Utilized           Total        Percentage
------------------        ----------      ------------    ------------
LUTs (CLB/ALM)*              168645            274080           61.53
DSPs                            800              2520           31.75
Block RAM                       453               912           49.67
* LUT count represents Configurable Logic Block(CLB) utilization in Xilinx devices and Adaptive L
```

```
### Finished writing input activations.
### Running single input activation.

prediction_FPGA = struct with fields:
       NumSamples: 1
    MetricResults: [1×1 struct]
       Statistics: [2×7 table]
```

View Performance of Quantized Neural Network

Display the accuracy of the quantized network.

```
prediction_simulation.MetricResults.Result
```

```
ans=2×2 table
    NetworkImplementation    MetricOutput
    _____________________    ____________

     {'Floating-Point'}           1
     {'Quantized'     }           1
```

```
prediction_FPGA.MetricResults.Result
```

```
ans=2×2 table
    NetworkImplementation    MetricOutput
    _____________________    ____________

     {'Floating-Point'}           1
     {'Quantized'     }           1
```

Display the performance of the quantized network in frames per second.

```
prediction_FPGA.Statistics.FramesPerSecond(2)
```

```
ans = 19.0828
```

Quantize a Neural Network for CPU Target

This example shows how to quantize and validate a neural network for a CPU target. This workflow is similar to other execution environments, but before validating you must establish a `raspi` connection and specify it as target using `dlquantizationOptions`.

First, load your network. This example uses the pretrained network `squeezenet`.

```
load squeezenetmerch
net
```

```
net =
  DAGNetwork with properties:

         Layers: [68×1 nnet.cnn.layer.Layer]
    Connections: [75×2 table]
     InputNames: {'data'}
    OutputNames: {'new_classoutput'}
```

Then define your calibration and validation data, `calDS` and `valDS` respectively.

```
unzip('MerchData.zip');
imds = imageDatastore('MerchData', ...
    'IncludeSubfolders',true, ...
    'LabelSource','foldernames');
[calData, valData] = splitEachLabel(imds, 0.7, 'randomized');
aug_calData = augmentedImageDatastore([227 227],calData);
aug_valData = augmentedImageDatastore([227 227],valData);
```

Create the `dlquantizer` object and specify a CPU execution environment.

```
dq =  dlquantizer(net,'ExecutionEnvironment','CPU')
```

```
dq =
  dlquantizer with properties:

           NetworkObject: [1×1 DAGNetwork]
    ExecutionEnvironment: 'CPU'
```

Calibrate the network.

```
calResults = calibrate(dq,aug_calData,'UseGPU','off')
```

```
calResults=122×5 table
        Optimized Layer Name         Network Layer Name     Learnables / Activations    MinValue
    _____________________________    ___________________    ________________________    ________

    {'conv1_Weights'           }    {'conv1'           }           "Weights"             -0.91985
    {'conv1_Bias'              }    {'conv1'           }           "Bias"                -0.07925
    {'fire2-squeeze1x1_Weights'}    {'fire2-squeeze1x1'}           "Weights"                -1.38
    {'fire2-squeeze1x1_Bias'   }    {'fire2-squeeze1x1'}           "Bias"                -0.11641
    {'fire2-expand1x1_Weights' }    {'fire2-expand1x1' }           "Weights"              -0.7400
    {'fire2-expand1x1_Bias'    }    {'fire2-expand1x1' }           "Bias"                -0.06005
    {'fire2-expand3x3_Weights' }    {'fire2-expand3x3' }           "Weights"              -0.7439
    {'fire2-expand3x3_Bias'    }    {'fire2-expand3x3' }           "Bias"                -0.05177
    {'fire3-squeeze1x1_Weights'}    {'fire3-squeeze1x1'}           "Weights"              -0.7712
    {'fire3-squeeze1x1_Bias'   }    {'fire3-squeeze1x1'}           "Bias"                -0.10138
    {'fire3-expand1x1_Weights' }    {'fire3-expand1x1' }           "Weights"              -0.7203
    {'fire3-expand1x1_Bias'    }    {'fire3-expand1x1' }           "Bias"                -0.06702
    {'fire3-expand3x3_Weights' }    {'fire3-expand3x3' }           "Weights"              -0.6144
    {'fire3-expand3x3_Bias'    }    {'fire3-expand3x3' }           "Bias"                -0.05361
    {'fire4-squeeze1x1_Weights'}    {'fire4-squeeze1x1'}           "Weights"              -0.7422
    {'fire4-squeeze1x1_Bias'   }    {'fire4-squeeze1x1'}           "Bias"                -0.10885
      ⋮
```

Use the MATLAB Support Package for Raspberry Pi Hardware function, `raspi`, to create a connection to the Raspberry Pi. In the following code, replace:

- `raspiname` with the name or address of your Raspberry Pi
- `username` with your user name
- `password` with your password

```
% r = raspi('raspiname','username','password')
```

For example,

```
r = raspi('gpucoder-raspberrypi-7','pi','matlab')

r =
  raspi with properties:

         DeviceAddress: 'gpucoder-raspberrypi-7'
                  Port: 18734
             BoardName: 'Raspberry Pi 3 Model B+'
         AvailableLEDs: {'led0'}
  AvailableDigitalPins: [2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27]
  AvailableSPIChannels: {}
     AvailableI2CBuses: {}
      AvailableWebcams: {}
           I2CBusSpeed:
AvailableCANInterfaces: {}

  Supported peripherals
```

Specify `raspi` object as the target for the quantized network.

```
opts = dlquantizationOptions('Target',r)

opts =
  dlquantizationOptions with properties:

    MetricFcn: {}
    Bitstream: ''
       Target: [1×1 raspi]
```

Validate the quantized network with the `validate` function.

```
valResults = validate(dq,aug_valData,opts)

### Starting application: 'codegen\lib\validate_predict_int8\pil\validate_predict_int8.elf'
    To terminate execution: clear validate_predict_int8_pil
### Launching application validate_predict_int8.elf...
### Host application produced the following standard output (stdout) and standard error (stderr)

valResults = struct with fields:
       NumSamples: 20
    MetricResults: [1×1 struct]
       Statistics: []
```

Examine the validation output to see the performance of the quantized network.

```
valResults.MetricResults.Result

ans=2×2 table
    NetworkImplementation    MetricOutput
    _____________________    ____________

     {'Floating-Point'}          0.95
     {'Quantized'     }          0.95
```

Version History

Introduced in R2020a

R2023a: Specify Raspberry Pi as quantization target
Behavior changed in R2023a

You can now specify a `raspi` object as the target for quantization using the `Target` property when `dlquantizer` `Execution Environment` is set to `CPU`.

See Also

Apps
Deep Network Quantizer

Functions
`dlquantizer` | `calibrate` | `validate` | `quantize` | `quantizationDetails` | `estimateNetworkMetrics`

Topics
“Quantization of Deep Neural Networks”
“Quantize Layers in Object Detectors and Generate CUDA Code”
“Classify Images on FPGA Using Quantized Neural Network”
“Generate INT8 Code for Deep Learning Network on Raspberry Pi” (MATLAB Coder)

calibrate

Simulate and collect ranges of a deep neural network

Syntax

```
calResults = calibrate(quantObj,calData)
calResults = calibrate(quantObj,calData,Name,Value)
```

Description

`calResults = calibrate(quantObj,calData)` exercises the network and collects the dynamic ranges of the weights and biases in the convolution and fully connected layers of the network and the dynamic ranges of the activations in all layers of the network specified by `dlquantizer` object, `quantObj`, using the data specified by `calData`.

`calResults = calibrate(quantObj,calData,Name,Value)` calibrates the network with additional options specified by one or more name-value pair arguments.

This function requires Deep Learning Toolbox Model Quantization Library. To learn about the products required to quantize a deep neural network, see “Quantization Workflow Prerequisites”.

Examples

Quantize a Neural Network for GPU Target

This example shows how to quantize learnable parameters in the convolution layers of a neural network for GPU and explore the behavior of the quantized network. In this example, you quantize the squeezenet neural network after retraining the network to classify new images according to the “Train Deep Learning Network to Classify New Images” example. In this example, the memory required for the network is reduced approximately 75% through quantization while the accuracy of the network is not affected.

Load the pretrained network. `net` is the output network of the Train Deep Learning Network to Classify New Images example.

```
load squeezenetmerch
net
```

```
net =
  DAGNetwork with properties:

         Layers: [68×1 nnet.cnn.layer.Layer]
    Connections: [75×2 table]
     InputNames: {'data'}
    OutputNames: {'new_classoutput'}
```

Define calibration and validation data to use for quantization.

The calibration data is used to collect the dynamic ranges of the weights and biases in the convolution and fully connected layers of the network and the dynamic ranges of the activations in all

layers of the network. For the best quantization results, the calibration data must be representative of inputs to the network.

The validation data is used to test the network after quantization to understand the effects of the limited range and precision of the quantized convolution layers in the network.

In this example, use the images in the `MerchData` data set. Define an `augmentedImageDatastore` object to resize the data for the network. Then, split the data into calibration and validation data sets.

```
unzip('MerchData.zip');
imds = imageDatastore('MerchData', ...
    'IncludeSubfolders',true, ...
    'LabelSource','foldernames');
[calData, valData] = splitEachLabel(imds, 0.7, 'randomized');
aug_calData = augmentedImageDatastore([227 227], calData);
aug_valData = augmentedImageDatastore([227 227], valData);
```

Create a `dlquantizer` object and specify the network to quantize.

```
quantObj = dlquantizer(net);
```

Define a metric function to use to compare the behavior of the network before and after quantization. This example uses the `hComputeModelAccuracy` metric function.

```
function accuracy = hComputeModelAccuracy(predictionScores, net, dataStore)
%% Computes model-level accuracy statistics

    % Load ground truth
    tmp = readall(dataStore);
    groundTruth = tmp.response;

    % Compare with predicted label with actual ground truth
    predictionError = {};
    for idx=1:numel(groundTruth)
        [~, idy] = max(predictionScores(idx,:));
        yActual = net.Layers(end).Classes(idy);
        predictionError{end+1} = (yActual == groundTruth(idx)); %#ok
    end

    % Sum all prediction errors.
    predictionError = [predictionError{:}];
    accuracy = sum(predictionError)/numel(predictionError);
end
```

Specify the metric function in a `dlquantizationOptions` object.

```
quantOpts = dlquantizationOptions('MetricFcn',{@(x)hComputeModelAccuracy(x, net, aug_valData)});
```

Use the `calibrate` function to exercise the network with sample inputs and collect range information. The `calibrate` function exercises the network and collects the dynamic ranges of the weights and biases in the convolution and fully connected layers of the network and the dynamic ranges of the activations in all layers of the network. The function returns a table. Each row of the table contains range information for a learnable parameter of the optimized network.

```
calResults = calibrate(quantObj, aug_calData)
```

```
calResults=121×5 table
        Optimized Layer Name         Network Layer Name     Learnables / Activations    MinValue
```

```
    _____________________________    ____________________    ____________________________    ________

    {'conv1_Weights'           }    {'conv1'           }           "Weights"             -0.91985
    {'conv1_Bias'              }    {'conv1'           }           "Bias"                -0.07925
    {'fire2-squeeze1x1_Weights'}    {'fire2-squeeze1x1'}           "Weights"                -1.38
    {'fire2-squeeze1x1_Bias'   }    {'fire2-squeeze1x1'}           "Bias"                -0.11641
    {'fire2-expand1x1_Weights' }    {'fire2-expand1x1' }           "Weights"              -0.7400
    {'fire2-expand1x1_Bias'    }    {'fire2-expand1x1' }           "Bias"               -0.060056
    {'fire2-expand3x3_Weights' }    {'fire2-expand3x3' }           "Weights"              -0.74397
    {'fire2-expand3x3_Bias'    }    {'fire2-expand3x3' }           "Bias"               -0.051778
    {'fire3-squeeze1x1_Weights'}    {'fire3-squeeze1x1'}           "Weights"              -0.7712
    {'fire3-squeeze1x1_Bias'   }    {'fire3-squeeze1x1'}           "Bias"                -0.10138
    {'fire3-expand1x1_Weights' }    {'fire3-expand1x1' }           "Weights"              -0.72035
    {'fire3-expand1x1_Bias'    }    {'fire3-expand1x1' }           "Bias"               -0.067029
    {'fire3-expand3x3_Weights' }    {'fire3-expand3x3' }           "Weights"              -0.61443
    {'fire3-expand3x3_Bias'    }    {'fire3-expand3x3' }           "Bias"               -0.053613
    {'fire4-squeeze1x1_Weights'}    {'fire4-squeeze1x1'}           "Weights"              -0.7422
    {'fire4-squeeze1x1_Bias'   }    {'fire4-squeeze1x1'}           "Bias"                -0.10885
      ⋮
```

Use the `validate` function to quantize the learnable parameters in the convolution layers of the network and exercise the network. The function uses the metric function defined in the `dlquantizationOptions` object to compare the results of the network before and after quantization.

```
valResults = validate(quantObj, aug_valData, quantOpts)
```

```
valResults = struct with fields:
       NumSamples: 20
    MetricResults: [1×1 struct]
       Statistics: [2×2 table]
```

Examine the validation output to see the performance of the quantized network.

```
valResults.MetricResults.Result
```

```
ans=2×2 table
    NetworkImplementation    MetricOutput
    _____________________    ____________

     {'Floating-Point'}           1
     {'Quantized'     }           1
```

```
valResults.Statistics
```

```
ans=2×2 table
    NetworkImplementation    LearnableParameterMemory(bytes)
    _____________________    _______________________________

     {'Floating-Point'}                2.9003e+06
     {'Quantized'     }                7.3393e+05
```

In this example, the memory required for the network was reduced approximately 75% through quantization. The accuracy of the network is not affected.

The weights, biases, and activations of the convolution layers of the network specified in the dlquantizer object now use scaled 8-bit integer data types.

Quantize Network for FPGA Deployment

Reduce the memory footprint of a deep neural network by quantizing the weights, biases, and activations of convolution layers to 8-bit scaled integer data types. This example shows how to use Deep Learning Toolbox Model Quantization Library and Deep Learning HDL Toolbox to deploy the `int8` network to a target FPGA board.

For this example, you need:

- Deep Learning Toolbox ™
- Deep Learning HDL Toolbox ™
- Deep Learning Toolbox Model Quantization Library
- Deep Learning HDL Toolbox Support Package for Xilinx FPGA and SoC Devices
- MATLAB Coder Interface for Deep Learning.

Load Pretrained Network

Load the pretrained LogoNet network and analyze the network architecture.

```
snet = getLogoNetwork;
deepNetworkDesigner(snet);
```

Load Data

This example uses the logos_dataset data set. The data set consists of 320 images. Each image is 227-by-227 in size and has three color channels (RGB). Create an `augmentedImageDatastore` object for calibration and validation. Expedite calibration and validation by reducing the calibration data set to 20 images. The MATLAB simulation workflow has a maximum limit of five images when validating the quantized network. Reduce the validation data set sizes to five images. The FPGA validation workflow has a maximum limit of one image when validating the quantized network. Reduce the FPGA validation data set to a single image.

```
curDir = pwd;
unzip("logos_dataset.zip");
imageData = imageDatastore(fullfile(curDir,'logos_dataset'),...
'IncludeSubfolders',true,'FileExtensions','.JPG','LabelSource','foldernames');
[calibrationData, validationData] = splitEachLabel(imageData, 0.5,'randomized');
calibrationData_reduced = calibrationData.subset(1:20);
validationData_simulation = validationData.subset(1:5);
validationData_FPGA = validationData.subset(1:1);
```

Generate Calibration Result File for the Network

Create a `dlquantizer` object and specify the network to quantize. Specify the execution environment as FPGA.

```
dlQuantObj_simulation = dlquantizer(snet,'ExecutionEnvironment',"FPGA",'Simulation','on');
dlQuantObj_FPGA = dlquantizer(snet,'ExecutionEnvironment',"FPGA");
```

Use the `calibrate` function to exercise the network with sample inputs and collect the range information. The `calibrate` function collects the dynamic ranges of the weights and biases. The calibrate function returns a table. Each row of the table contains range information for a learnable parameter of the quantized network.

```
calibrate(dlQuantObj_simulation,calibrationData_reduced)
```

```
ans=35×5 table
        Optimized Layer Name        Network Layer Name    Learnables / Activations     MinValue
    ____________________________    __________________    ________________________    __________

    {'conv_1_Weights'          }      {'conv_1'    }          "Weights"                -0.04897
    {'conv_1_Bias'             }      {'conv_1'    }          "Bias"                    0.9999
    {'conv_2_Weights'          }      {'conv_2'    }          "Weights"                -0.05551
    {'conv_2_Bias'             }      {'conv_2'    }          "Bias"                  -0.000611
    {'conv_3_Weights'          }      {'conv_3'    }          "Weights"                -0.04594
    {'conv_3_Bias'             }      {'conv_3'    }          "Bias"                  -0.001399
    {'conv_4_Weights'          }      {'conv_4'    }          "Weights"                -0.04596
    {'conv_4_Bias'             }      {'conv_4'    }          "Bias"                   -0.0016
    {'fc_1_Weights'            }      {'fc_1'      }          "Weights"                -0.05139
    {'fc_1_Bias'               }      {'fc_1'      }          "Bias"                  -0.000523
    {'fc_2_Weights'            }      {'fc_2'      }          "Weights"                 -0.0501
    {'fc_2_Bias'               }      {'fc_2'      }          "Bias"                  -0.001756
    {'fc_3_Weights'            }      {'fc_3'      }          "Weights"                -0.05070
    {'fc_3_Bias'               }      {'fc_3'      }          "Bias"                   -0.0295
    {'imageinput'              }      {'imageinput'}          "Activations"                  0
    {'imageinput_normalization'}      {'imageinput'}          "Activations"            -139.34
      ⋮
```

```
calibrate(dlQuantObj_FPGA,calibrationData_reduced)
```

```
ans=35×5 table
        Optimized Layer Name        Network Layer Name    Learnables / Activations     MinValue
    ____________________________    __________________    ________________________    __________

    {'conv_1_Weights'          }      {'conv_1'    }          "Weights"                -0.04897
    {'conv_1_Bias'             }      {'conv_1'    }          "Bias"                    0.9999
    {'conv_2_Weights'          }      {'conv_2'    }          "Weights"                -0.05551
    {'conv_2_Bias'             }      {'conv_2'    }          "Bias"                  -0.000611
    {'conv_3_Weights'          }      {'conv_3'    }          "Weights"                -0.04594
    {'conv_3_Bias'             }      {'conv_3'    }          "Bias"                  -0.001399
    {'conv_4_Weights'          }      {'conv_4'    }          "Weights"                -0.04596
    {'conv_4_Bias'             }      {'conv_4'    }          "Bias"                   -0.0016
    {'fc_1_Weights'            }      {'fc_1'      }          "Weights"                -0.05139
    {'fc_1_Bias'               }      {'fc_1'      }          "Bias"                  -0.000523
    {'fc_2_Weights'            }      {'fc_2'      }          "Weights"                 -0.0501
    {'fc_2_Bias'               }      {'fc_2'      }          "Bias"                  -0.001756
    {'fc_3_Weights'            }      {'fc_3'      }          "Weights"                -0.05070
    {'fc_3_Bias'               }      {'fc_3'      }          "Bias"                   -0.0295
    {'imageinput'              }      {'imageinput'}          "Activations"                  0
    {'imageinput_normalization'}      {'imageinput'}          "Activations"            -139.34
      ⋮
```

Create Target Object

Create a target object with a custom name for your target device and an interface to connect your target device to the host computer. Interface options are JTAG and Ethernet. To use JTAG, install Xilinx™ Vivado™ Design Suite 2020.2. To set the Xilinx Vivado toolpath, enter:

```
% hdlsetuptoolpath('ToolName', 'Xilinx Vivado', 'ToolPath', 'C:\Xilinx\Vivado
\2020.2\bin\vivado.bat');
```

To create the target object, enter:

```
hTarget = dlhdl.Target('Xilinx','Interface','Ethernet');
```

Alternatively, you can also use the JTAG interface.

```
% hTarget = dlhdl.Target('Xilinx', 'Interface', 'JTAG');
```

Create `dlQuantizationOptions` Object

Create a `dlquantizationOptions` object. Specify the target bitstream and target board interface. The default metric function is a Top-1 accuracy metric function.

```
options_FPGA = dlquantizationOptions('Bitstream','zcu102_int8','Target',hTarget);
options_simulation = dlquantizationOptions;
```

To use a custom metric function, specify the metric function in the `dlquantizationOptions` object.

```
options_FPGA = dlquantizationOptions('MetricFcn',{@(x)hComputeAccuracy(x,snet,validationData_FPGA
options_simulation = dlquantizationOptions('MetricFcn',{@(x)hComputeAccuracy(x,snet,validationDa
```

Validate Quantized Network

Use the `validate` function to quantize the learnable parameters in the convolution layers of the network. The `validate` function simulates the quantized network in MATLAB. The `validate` function uses the metric function defined in the `dlquantizationOptions` object to compare the results of the single-data-type network object to the results of the quantized network object.

```
prediction_simulation = dlQuantObj_simulation.validate(validationData_simulation,options_simulati
```

```
Compiling leg: conv_1>>relu_4 ...
Compiling leg: conv_1>>relu_4 ... complete.
Compiling leg: maxpool_4 ...
Compiling leg: maxpool_4 ... complete.
Compiling leg: fc_1>>fc_3 ...
Compiling leg: fc_1>>fc_3 ... complete.

prediction_simulation = struct with fields:
       NumSamples: 5
    MetricResults: [1×1 struct]
       Statistics: []
```

For validation on an FPGA, the validate function:

- Programs the FPGA board by using the output of the `compile` method and the programming file
- Downloads the network weights and biases

- Compares the performance of the network before and after quantization

```
prediction_FPGA = dlQuantObj_FPGA.validate(validationData_FPGA,options_FPGA)

### Compiling network for Deep Learning FPGA prototyping ...
### Targeting FPGA bitstream zcu102_int8.
### The network includes the following layers:
     1   'imageinput'    Image Input             227×227×3 images with 'zerocenter' normalizatio
     2   'conv_1'        Convolution             96 5×5×3 convolutions with stride [1  1] and pa
     3   'relu_1'        ReLU                    ReLU
     4   'maxpool_1'     Max Pooling             3×3 max pooling with stride [2  2] and padding
     5   'conv_2'        Convolution             128 3×3×96 convolutions with stride [1  1] and
     6   'relu_2'        ReLU                    ReLU
     7   'maxpool_2'     Max Pooling             3×3 max pooling with stride [2  2] and padding
     8   'conv_3'        Convolution             384 3×3×128 convolutions with stride [1  1] and
     9   'relu_3'        ReLU                    ReLU
    10   'maxpool_3'     Max Pooling             3×3 max pooling with stride [2  2] and padding
    11   'conv_4'        Convolution             128 3×3×384 convolutions with stride [2  2] and
    12   'relu_4'        ReLU                    ReLU
    13   'maxpool_4'     Max Pooling             3×3 max pooling with stride [2  2] and padding
    14   'fc_1'          Fully Connected         2048 fully connected layer
    15   'relu_5'        ReLU                    ReLU
    16   'dropout_1'     Dropout                 50% dropout
    17   'fc_2'          Fully Connected         2048 fully connected layer
    18   'relu_6'        ReLU                    ReLU
    19   'dropout_2'     Dropout                 50% dropout
    20   'fc_3'          Fully Connected         32 fully connected layer
    21   'softmax'       Softmax                 softmax
    22   'classoutput'   Classification Output   crossentropyex with 'adidas' and 31 other classe

### Notice: The layer 'imageinput' with type 'nnet.cnn.layer.ImageInputLayer' is implemented in s
### Notice: The layer 'softmax' with type 'nnet.cnn.layer.SoftmaxLayer' is implemented in softwa
### Notice: The layer 'classoutput' with type 'nnet.cnn.layer.ClassificationOutputLayer' is imple
### Compiling layer group: conv_1>>relu_4 ...
### Compiling layer group: conv_1>>relu_4 ... complete.
### Compiling layer group: maxpool_4 ...
### Compiling layer group: maxpool_4 ... complete.
### Compiling layer group: fc_1>>fc_3 ...
### Compiling layer group: fc_1>>fc_3 ... complete.

### Allocating external memory buffers:

          offset_name          offset_address    allocated_space
    _______________________    ______________    _______________

    "InputDataOffset"           "0x00000000"     "12.0 MB"
    "OutputResultOffset"        "0x00c00000"     "4.0 MB"
    "SchedulerDataOffset"       "0x01000000"     "4.0 MB"
    "SystemBufferOffset"        "0x01400000"     "36.0 MB"
    "InstructionDataOffset"     "0x03800000"     "8.0 MB"
    "ConvWeightDataOffset"      "0x04000000"     "12.0 MB"
    "FCWeightDataOffset"        "0x04c00000"     "12.0 MB"
    "EndOffset"                 "0x05800000"     "Total: 88.0 MB"

### Network compilation complete.

### FPGA bitstream programming has been skipped as the same bitstream is already loaded on the ta
### Deep learning network programming has been skipped as the same network is already loaded on t
```

```
### Finished writing input activations.
### Running single input activation.


              Deep Learning Processor Bitstream Build Info

Resource                   Utilized           Total        Percentage
------------------        ----------      ------------    ------------
LUTs (CLB/ALM)*              248358            274080           90.62
DSPs                            384              2520           15.24
Block RAM                       581               912           63.71
* LUT count represents Configurable Logic Block(CLB) utilization in Xilinx devices and Adaptive L

### Notice: The layer 'imageinput' of type 'ImageInputLayer' is split into an image input layer
### Notice: The layer 'softmax' with type 'nnet.cnn.layer.SoftmaxLayer' is implemented in softwar
### Notice: The layer 'classoutput' with type 'nnet.cnn.layer.ClassificationOutputLayer' is imple


              Deep Learning Processor Estimator Performance Results

                     LastFrameLatency(cycles)   LastFrameLatency(seconds)       FramesNum      Tota
                    ------------              ------------                    ---------        --
Network                    40142478                   0.18247                       1           40
    ____imageinput_norm      216472                   0.00098
    ____conv_1              6825671                   0.03103
    ____maxpool_1           3755088                   0.01707
    ____conv_2             10440701                   0.04746
    ____maxpool_2           1447840                   0.00658
    ____conv_3              9405685                   0.04275
    ____maxpool_3           1765856                   0.00803
    ____conv_4              1819636                   0.00827
    ____maxpool_4             28098                   0.00013
    ____fc_1                2651288                   0.01205
    ____fc_2                1696632                   0.00771
    ____fc_3                  89511                   0.00041
 * The clock frequency of the DL processor is: 220MHz



              Deep Learning Processor Bitstream Build Info

Resource                   Utilized           Total        Percentage
------------------        ----------      ------------    ------------
LUTs (CLB/ALM)*              168645            274080           61.53
DSPs                            800              2520           31.75
Block RAM                       453               912           49.67
* LUT count represents Configurable Logic Block(CLB) utilization in Xilinx devices and Adaptive L

### Finished writing input activations.
### Running single input activation.

prediction_FPGA = struct with fields:
       NumSamples: 1
    MetricResults: [1×1 struct]
       Statistics: [2×7 table]
```

View Performance of Quantized Neural Network

Display the accuracy of the quantized network.

```
prediction_simulation.MetricResults.Result
```

```
ans=2×2 table
    NetworkImplementation    MetricOutput
    _____________________    ____________

     {'Floating-Point'}           1
     {'Quantized'     }           1
```

```
prediction_FPGA.MetricResults.Result
```

```
ans=2×2 table
    NetworkImplementation    MetricOutput
    _____________________    ____________

     {'Floating-Point'}           1
     {'Quantized'     }           1
```

Display the performance of the quantized network in frames per second.

```
prediction_FPGA.Statistics.FramesPerSecond(2)
```

```
ans = 19.0828
```

Quantize a Neural Network for CPU Target

This example shows how to quantize and validate a neural network for a CPU target. This workflow is similar to other execution environments, but before validating you must establish a `raspi` connection and specify it as target using `dlquantizationOptions`.

First, load your network. This example uses the pretrained network `squeezenet`.

```
load squeezenetmerch
net
```

```
net =
  DAGNetwork with properties:

         Layers: [68×1 nnet.cnn.layer.Layer]
    Connections: [75×2 table]
     InputNames: {'data'}
    OutputNames: {'new_classoutput'}
```

Then define your calibration and validation data, `calDS` and `valDS` respectively.

```
unzip('MerchData.zip');
imds = imageDatastore('MerchData', ...
    'IncludeSubfolders',true, ...
    'LabelSource','foldernames');
[calData, valData] = splitEachLabel(imds, 0.7, 'randomized');
```

```
aug_calData = augmentedImageDatastore([227 227],calData);
aug_valData = augmentedImageDatastore([227 227],valData);
```

Create the `dlquantizer` object and specify a CPU execution environment.

```
dq =  dlquantizer(net,'ExecutionEnvironment','CPU')
```

```
dq =
  dlquantizer with properties:

           NetworkObject: [1×1 DAGNetwork]
    ExecutionEnvironment: 'CPU'
```

Calibrate the network.

```
calResults = calibrate(dq,aug_calData,'UseGPU','off')
```

```
calResults=122×5 table
        Optimized Layer Name         Network Layer Name     Learnables / Activations    MinValue
    _____________________________    ___________________    ________________________    _________

    {'conv1_Weights'           }    {'conv1'           }           "Weights"             -0.91985
    {'conv1_Bias'              }    {'conv1'           }           "Bias"                -0.07925
    {'fire2-squeeze1x1_Weights'}    {'fire2-squeeze1x1'}           "Weights"                -1.38
    {'fire2-squeeze1x1_Bias'   }    {'fire2-squeeze1x1'}           "Bias"                -0.11641
    {'fire2-expand1x1_Weights' }    {'fire2-expand1x1' }           "Weights"              -0.7406
    {'fire2-expand1x1_Bias'    }    {'fire2-expand1x1' }           "Bias"               -0.060056
    {'fire2-expand3x3_Weights' }    {'fire2-expand3x3' }           "Weights"              -0.7439
    {'fire2-expand3x3_Bias'    }    {'fire2-expand3x3' }           "Bias"               -0.051778
    {'fire3-squeeze1x1_Weights'}    {'fire3-squeeze1x1'}           "Weights"              -0.7712
    {'fire3-squeeze1x1_Bias'   }    {'fire3-squeeze1x1'}           "Bias"                -0.10138
    {'fire3-expand1x1_Weights' }    {'fire3-expand1x1' }           "Weights"              -0.7203
    {'fire3-expand1x1_Bias'    }    {'fire3-expand1x1' }           "Bias"               -0.067029
    {'fire3-expand3x3_Weights' }    {'fire3-expand3x3' }           "Weights"              -0.6144
    {'fire3-expand3x3_Bias'    }    {'fire3-expand3x3' }           "Bias"               -0.053613
    {'fire4-squeeze1x1_Weights'}    {'fire4-squeeze1x1'}           "Weights"              -0.7422
    {'fire4-squeeze1x1_Bias'   }    {'fire4-squeeze1x1'}           "Bias"                -0.10885
      ⋮
```

Use the MATLAB Support Package for Raspberry Pi Hardware function, `raspi`, to create a connection to the Raspberry Pi. In the following code, replace:

- `raspiname` with the name or address of your Raspberry Pi
- `username` with your user name
- `password` with your password

```
% r = raspi('raspiname','username','password')
```

For example,

```
r = raspi('gpucoder-raspberrypi-7','pi','matlab')
```

```
r =
  raspi with properties:

         DeviceAddress: 'gpucoder-raspberrypi-7'
```

```
                Port: 18734
           BoardName: 'Raspberry Pi 3 Model B+'
       AvailableLEDs: {'led0'}
  AvailableDigitalPins: [2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27]
  AvailableSPIChannels: {}
     AvailableI2CBuses: {}
      AvailableWebcams: {}
           I2CBusSpeed:
AvailableCANInterfaces: {}

  Supported peripherals
```

Specify `raspi` object as the target for the quantized network.

```
opts = dlquantizationOptions('Target',r)
```

```
opts =
  dlquantizationOptions with properties:

    MetricFcn: {}
    Bitstream: ''
       Target: [1×1 raspi]
```

Validate the quantized network with the `validate` function.

```
valResults = validate(dq,aug_valData,opts)
```

```
### Starting application: 'codegen\lib\validate_predict_int8\pil\validate_predict_int8.elf'
    To terminate execution: clear validate_predict_int8_pil
### Launching application validate_predict_int8.elf...
### Host application produced the following standard output (stdout) and standard error (stderr)

valResults = struct with fields:
       NumSamples: 20
    MetricResults: [1×1 struct]
       Statistics: []
```

Examine the validation output to see the performance of the quantized network.

```
valResults.MetricResults.Result
```

```
ans=2×2 table
    NetworkImplementation    MetricOutput
    _____________________    ____________

     {'Floating-Point'}          0.95
     {'Quantized'     }          0.95
```

Quantize YOLO v3 Object Detector

This example shows how to quantize a `yolov3ObjectDetector` (Computer Vision Toolbox) object using preprocessed calibration and validation data.

First, download a pretrained YOLO v3 object detector.

```
detector = downloadPretrainedNetwork();
```

This example uses a small labeled data set that contains one or two labeled instances of a vehicle. Many of these images come from the Caltech Cars 1999 and 2001 data sets, created by Pietro Perona and used with permission.

Unzip the vehicle images and load the vehicle ground truth data.

```
unzip vehicleDatasetImages.zip
data = load('vehicleDatasetGroundTruth.mat');
vehicleDataset = data.vehicleDataset;
```

Add the full path to the local vehicle data folder.

```
vehicleDataset.imageFilename = fullfile(pwd, vehicleDataset.imageFilename);
```

Create an `imageDatastore` for loading the images and a `boxLabelDatastore` (Computer Vision Toolbox) for the ground truth bounding boxes.

```
imds = imageDatastore(vehicleDataset.imageFilename);
blds = boxLabelDatastore(vehicleDataset(:,2));
```

Use the `combine` function to combine both the datastores into a `CombinedDatastore`.

```
combinedDS = combine(imds, blds);
```

Split the data into calibration and validation data.

```
calData = combinedDS.subset(1:32);
valData = combinedDS.subset(33:64);
```

Use the `preprocess` (Computer Vision Toolbox) method of `yolov3ObjectDetector` (Computer Vision Toolbox) object with `transform` function to prepare the data for calibration and validation.

The `transform` function returns a `TransformedDatastore` object.

```
processedCalData = transform(calData, @(data)preprocess(detector,data));
processedValData = transform(valData, @(data)preprocess(detector,data));
```

Create the `dlquantizer` object.

```
dq = dlquantizer(detector, 'ExecutionEnvironment', 'MATLAB');
```

Calibrate the network.

```
calResults = calibrate(dq, processedCalData,'UseGPU','off')
```

```
calResults=135×5 table
        Optimized Layer Name         Network Layer Name     Learnables / Activations    MinValue
    _____________________________    ___________________    ________________________    _________

    {'conv1_Weights'           }    {'conv1'           }           "Weights"             -0.92189
    {'conv1_Bias'              }    {'conv1'           }           "Bias"               -0.096271
    {'fire2-squeeze1x1_Weights'}    {'fire2-squeeze1x1'}           "Weights"              -1.3751
    {'fire2-squeeze1x1_Bias'   }    {'fire2-squeeze1x1'}           "Bias"                -0.12068
    {'fire2-expand1x1_Weights' }    {'fire2-expand1x1' }           "Weights"             -0.75275
```

```
    {'fire2-expand1x1_Bias'    }    {'fire2-expand1x1' }          "Bias"              -0.05925
    {'fire2-expand3x3_Weights' }    {'fire2-expand3x3' }          "Weights"            -0.7527
    {'fire2-expand3x3_Bias'    }    {'fire2-expand3x3' }          "Bias"              -0.06221
    {'fire3-squeeze1x1_Weights'}    {'fire3-squeeze1x1'}          "Weights"              -0.758
    {'fire3-squeeze1x1_Bias'   }    {'fire3-squeeze1x1'}          "Bias"               -0.1020
    {'fire3-expand1x1_Weights' }    {'fire3-expand1x1' }          "Weights"            -0.7156
    {'fire3-expand1x1_Bias'    }    {'fire3-expand1x1' }          "Bias"              -0.06931
    {'fire3-expand3x3_Weights' }    {'fire3-expand3x3' }          "Weights"            -0.6007
    {'fire3-expand3x3_Bias'    }    {'fire3-expand3x3' }          "Bias"              -0.05804
    {'fire4-squeeze1x1_Weights'}    {'fire4-squeeze1x1'}          "Weights"              -0.738
    {'fire4-squeeze1x1_Bias'   }    {'fire4-squeeze1x1'}          "Bias"               -0.1118
      ⋮
```

Validate the quantized network with the `validate` function.

```
valResults = validate(dq, processedValData)
```

```
valResults = struct with fields:
       NumSamples: 32
    MetricResults: [1×1 struct]
       Statistics: []
```

```
function detector = downloadPretrainedNetwork()
   pretrainedURL = 'https://ssd.mathworks.com/supportfiles/vision/data/yolov3SqueezeNetVehicleExa
   websave('yolov3SqueezeNetVehicleExample_21aSPKG.zip', pretrainedURL);

   unzip('yolov3SqueezeNetVehicleExample_21aSPKG.zip');

   pretrained = load("yolov3SqueezeNetVehicleExample_21aSPKG.mat");
   detector = pretrained.detector;
end
```

Input Arguments

`quantObj` — Network to quantize

`dlquantizer` object

Network to quantize, specified as a `dlquantizer` object.

`calData` — Data to use for calibration of quantized network

`imageDatastore` object | `augmentedImageDatastore` object | `pixelLabelImageDatastore` object | `CombinedDatastore` object | `TransformedDatastore` object

Data to use for calibration of quantized network, specified as an `imageDatastore` object, an `augmentedImageDatastore` object, a `pixelLabelImageDatastore` object, a `CombinedDatastore` object, or a `TransformedDatastore` object.

You must preprocess the data used for calibration of a `yolov3ObjectDetector` object using the `preprocess` function. For an example of using preprocessed data for calibration of a `yolov3ObjectDetector`, see “Quantize YOLO v3 Object Detector”.

For more information on valid datastores, see “Prepare Data for Quantizing Networks”.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `calResults = calibrate(quantObj,calData,'UseGPU','on')`

`MiniBatchSize` — Size of mini-batches

32 (default) | positive integer

Size of the mini-batches to use for calibration, specified as a positive integer. Larger mini-batch sizes require more memory, but can lead to faster calibration.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

`UseGPU` — Whether to use host GPU for calibration

'auto' (default) | `'on'` | `'off`

Whether to use host GPU for calibration, specified as one of the following:

- `'auto'` — Use host GPU for calibration if one is available. Otherwise, use host CPU for calibration.
- `'on'` — Use host GPU for calibration.
- `'off'` — Use host CPU for calibration.

Data Types: `char`

Output Arguments

`calResults` — Dynamic ranges of network

table

Dynamic ranges of layers of the network, returned as a table. Each row in the table displays the minimum and maximum values of a learnable parameter of a convolution layer of the optimized network. The software uses these minimum and maximum values to determine the scaling for the data type of the quantized parameter.

Version History

Introduced in R2020a

R2022b: Calibrate on host GPU or host CPU

You can now choose whether to calibrate your network using the host GPU or host CPU. By default, the `calibrate` function and the **Deep Network Quantizer** app will calibrate on the host GPU if one is available.

In previous versions, it was required that the execution environment was the same as the instrumentation environment used for the calibration step of quantization.

R2022b: Specify mini-batch size to use for calibration

Use `MiniBatchSize` to specify the size of mini-batches to use for calibration.

R2021a: ARM Cortex-A calibration support

The Deep Learning Toolbox Model Quantization Library now supports calibration of a network for quantization and deployment on ARM® Cortex®-A microcontrollers.

See Also

Apps
Deep Network Quantizer

Functions
`validate` | `dlquantizer` | `dlquantizationOptions` | `quantize` | `quantizationDetails` | `estimateNetworkMetrics`

Topics
“Quantization Workflow Prerequisites”
“Prepare Data for Quantizing Networks”
“Quantization of Deep Neural Networks”
“Quantize Residual Network Trained for Image Classification and Generate CUDA Code”
“Quantize Network for FPGA Deployment”
“Generate int8 Code for Deep Learning Networks” (MATLAB Coder)

validate

Quantize and validate a deep neural network

Syntax

```
valResults = validate(quantObj,valData)
valResults = validate(quantObj,valData,quantOpts)
```

Description

`valResults = validate(quantObj,valData)` quantizes the weights, biases, and activations in the convolution layers of the network, and validates the network specified by `dlquantizer` object, `quantObj`, using the data specified by `valData`.

`valResults = validate(quantObj,valData,quantOpts)` quantizes and validates the network with additional options specified by `quantOpts`.

This function requires Deep Learning Toolbox Model Quantization Library. To learn about the products required to quantize a deep neural network, see “Quantization Workflow Prerequisites”.

Examples

Quantize a Neural Network for GPU Target

This example shows how to quantize learnable parameters in the convolution layers of a neural network for GPU and explore the behavior of the quantized network. In this example, you quantize the squeezenet neural network after retraining the network to classify new images according to the “Train Deep Learning Network to Classify New Images” example. In this example, the memory required for the network is reduced approximately 75% through quantization while the accuracy of the network is not affected.

Load the pretrained network. `net` is the output network of the Train Deep Learning Network to Classify New Images example.

```
load squeezenetmerch
net
```

```
net =
  DAGNetwork with properties:

         Layers: [68×1 nnet.cnn.layer.Layer]
    Connections: [75×2 table]
     InputNames: {'data'}
    OutputNames: {'new_classoutput'}
```

Define calibration and validation data to use for quantization.

The calibration data is used to collect the dynamic ranges of the weights and biases in the convolution and fully connected layers of the network and the dynamic ranges of the activations in all

layers of the network. For the best quantization results, the calibration data must be representative of inputs to the network.

The validation data is used to test the network after quantization to understand the effects of the limited range and precision of the quantized convolution layers in the network.

In this example, use the images in the `MerchData` data set. Define an `augmentedImageDatastore` object to resize the data for the network. Then, split the data into calibration and validation data sets.

```
unzip('MerchData.zip');
imds = imageDatastore('MerchData', ...
    'IncludeSubfolders',true, ...
    'LabelSource','foldernames');
[calData, valData] = splitEachLabel(imds, 0.7, 'randomized');
aug_calData = augmentedImageDatastore([227 227], calData);
aug_valData = augmentedImageDatastore([227 227], valData);
```

Create a `dlquantizer` object and specify the network to quantize.

```
quantObj = dlquantizer(net);
```

Define a metric function to use to compare the behavior of the network before and after quantization. This example uses the `hComputeModelAccuracy` metric function.

```
function accuracy = hComputeModelAccuracy(predictionScores, net, dataStore)
%% Computes model-level accuracy statistics

    % Load ground truth
    tmp = readall(dataStore);
    groundTruth = tmp.response;

    % Compare with predicted label with actual ground truth
    predictionError = {};
    for idx=1:numel(groundTruth)
        [~, idy] = max(predictionScores(idx,:));
        yActual = net.Layers(end).Classes(idy);
        predictionError{end+1} = (yActual == groundTruth(idx)); %#ok
    end

    % Sum all prediction errors.
    predictionError = [predictionError{:}];
    accuracy = sum(predictionError)/numel(predictionError);
end
```

Specify the metric function in a `dlquantizationOptions` object.

```
quantOpts = dlquantizationOptions('MetricFcn',{@(x)hComputeModelAccuracy(x, net, aug_valData)});
```

Use the `calibrate` function to exercise the network with sample inputs and collect range information. The `calibrate` function exercises the network and collects the dynamic ranges of the weights and biases in the convolution and fully connected layers of the network and the dynamic ranges of the activations in all layers of the network. The function returns a table. Each row of the table contains range information for a learnable parameter of the optimized network.

```
calResults = calibrate(quantObj, aug_calData)
```

```
calResults=121×5 table
        Optimized Layer Name         Network Layer Name     Learnables / Activations    MinValue
```

```
    _____________________________    ____________________    ____________________________    ________

    {'conv1_Weights'           }    {'conv1'           }           "Weights"             -0.91985
    {'conv1_Bias'              }    {'conv1'           }           "Bias"                -0.07925
    {'fire2-squeeze1x1_Weights'}    {'fire2-squeeze1x1'}           "Weights"                -1.38
    {'fire2-squeeze1x1_Bias'   }    {'fire2-squeeze1x1'}           "Bias"                -0.11641
    {'fire2-expand1x1_Weights' }    {'fire2-expand1x1' }           "Weights"              -0.7400
    {'fire2-expand1x1_Bias'    }    {'fire2-expand1x1' }           "Bias"               -0.060056
    {'fire2-expand3x3_Weights' }    {'fire2-expand3x3' }           "Weights"              -0.74397
    {'fire2-expand3x3_Bias'    }    {'fire2-expand3x3' }           "Bias"               -0.051778
    {'fire3-squeeze1x1_Weights'}    {'fire3-squeeze1x1'}           "Weights"              -0.7712
    {'fire3-squeeze1x1_Bias'   }    {'fire3-squeeze1x1'}           "Bias"                -0.10138
    {'fire3-expand1x1_Weights' }    {'fire3-expand1x1' }           "Weights"              -0.72035
    {'fire3-expand1x1_Bias'    }    {'fire3-expand1x1' }           "Bias"               -0.067029
    {'fire3-expand3x3_Weights' }    {'fire3-expand3x3' }           "Weights"              -0.61443
    {'fire3-expand3x3_Bias'    }    {'fire3-expand3x3' }           "Bias"               -0.053613
    {'fire4-squeeze1x1_Weights'}    {'fire4-squeeze1x1'}           "Weights"              -0.7422
    {'fire4-squeeze1x1_Bias'   }    {'fire4-squeeze1x1'}           "Bias"                -0.10885
      ⋮
```

Use the `validate` function to quantize the learnable parameters in the convolution layers of the network and exercise the network. The function uses the metric function defined in the `dlquantizationOptions` object to compare the results of the network before and after quantization.

```
valResults = validate(quantObj, aug_valData, quantOpts)
```

```
valResults = struct with fields:
       NumSamples: 20
    MetricResults: [1×1 struct]
       Statistics: [2×2 table]
```

Examine the validation output to see the performance of the quantized network.

```
valResults.MetricResults.Result
```

```
ans=2×2 table
    NetworkImplementation    MetricOutput
    _____________________    ____________

     {'Floating-Point'}           1
     {'Quantized'     }           1
```

```
valResults.Statistics
```

```
ans=2×2 table
    NetworkImplementation    LearnableParameterMemory(bytes)
    _____________________    _______________________________

     {'Floating-Point'}                2.9003e+06
     {'Quantized'     }                7.3393e+05
```

In this example, the memory required for the network was reduced approximately 75% through quantization. The accuracy of the network is not affected.

The weights, biases, and activations of the convolution layers of the network specified in the dlquantizer object now use scaled 8-bit integer data types.

Quantize Network for FPGA Deployment

Reduce the memory footprint of a deep neural network by quantizing the weights, biases, and activations of convolution layers to 8-bit scaled integer data types. This example shows how to use Deep Learning Toolbox Model Quantization Library and Deep Learning HDL Toolbox to deploy the `int8` network to a target FPGA board.

For this example, you need:

- Deep Learning Toolbox ™
- Deep Learning HDL Toolbox ™
- Deep Learning Toolbox Model Quantization Library
- Deep Learning HDL Toolbox Support Package for Xilinx FPGA and SoC Devices
- MATLAB Coder Interface for Deep Learning.

Load Pretrained Network

Load the pretrained LogoNet network and analyze the network architecture.

```
snet = getLogoNetwork;
deepNetworkDesigner(snet);
```

Load Data

This example uses the logos_dataset data set. The data set consists of 320 images. Each image is 227-by-227 in size and has three color channels (RGB). Create an `augmentedImageDatastore` object for calibration and validation. Expedite calibration and validation by reducing the calibration data set to 20 images. The MATLAB simulation workflow has a maximum limit of five images when validating the quantized network. Reduce the validation data set sizes to five images. The FPGA validation workflow has a maximum limit of one image when validating the quantized network. Reduce the FPGA validation data set to a single image.

```
curDir = pwd;
unzip("logos_dataset.zip");
imageData = imageDatastore(fullfile(curDir,'logos_dataset'),...
'IncludeSubfolders',true,'FileExtensions','.JPG','LabelSource','foldernames');
[calibrationData, validationData] = splitEachLabel(imageData, 0.5,'randomized');
calibrationData_reduced = calibrationData.subset(1:20);
validationData_simulation = validationData.subset(1:5);
validationData_FPGA = validationData.subset(1:1);
```

Generate Calibration Result File for the Network

Create a `dlquantizer` object and specify the network to quantize. Specify the execution environment as FPGA.

```
dlQuantObj_simulation = dlquantizer(snet,'ExecutionEnvironment',"FPGA",'Simulation','on');
dlQuantObj_FPGA = dlquantizer(snet,'ExecutionEnvironment',"FPGA");
```

Use the `calibrate` function to exercise the network with sample inputs and collect the range information. The `calibrate` function collects the dynamic ranges of the weights and biases. The calibrate function returns a table. Each row of the table contains range information for a learnable parameter of the quantized network.

```
calibrate(dlQuantObj_simulation,calibrationData_reduced)
```

```
ans=35×5 table
        Optimized Layer Name        Network Layer Name    Learnables / Activations     MinValue
    ____________________________    __________________    ________________________    __________

    {'conv_1_Weights'          }      {'conv_1'    }          "Weights"                -0.04897
    {'conv_1_Bias'             }      {'conv_1'    }          "Bias"                    0.9999
    {'conv_2_Weights'          }      {'conv_2'    }          "Weights"                -0.05551
    {'conv_2_Bias'             }      {'conv_2'    }          "Bias"                  -0.000611
    {'conv_3_Weights'          }      {'conv_3'    }          "Weights"                -0.04594
    {'conv_3_Bias'             }      {'conv_3'    }          "Bias"                   -0.00139
    {'conv_4_Weights'          }      {'conv_4'    }          "Weights"                -0.04596
    {'conv_4_Bias'             }      {'conv_4'    }          "Bias"                    -0.0016
    {'fc_1_Weights'            }      {'fc_1'      }          "Weights"                -0.05139
    {'fc_1_Bias'               }      {'fc_1'      }          "Bias"                  -0.000523
    {'fc_2_Weights'            }      {'fc_2'      }          "Weights"                 -0.0501
    {'fc_2_Bias'               }      {'fc_2'      }          "Bias"                   -0.00175
    {'fc_3_Weights'            }      {'fc_3'      }          "Weights"                -0.05070
    {'fc_3_Bias'               }      {'fc_3'      }          "Bias"                    -0.0295
    {'imageinput'              }      {'imageinput'}          "Activations"                   0
    {'imageinput_normalization'}      {'imageinput'}          "Activations"             -139.34
      ⋮
```

```
calibrate(dlQuantObj_FPGA,calibrationData_reduced)
```

```
ans=35×5 table
        Optimized Layer Name        Network Layer Name    Learnables / Activations     MinValue
    ____________________________    __________________    ________________________    __________

    {'conv_1_Weights'          }      {'conv_1'    }          "Weights"                -0.04897
    {'conv_1_Bias'             }      {'conv_1'    }          "Bias"                    0.9999
    {'conv_2_Weights'          }      {'conv_2'    }          "Weights"                -0.05551
    {'conv_2_Bias'             }      {'conv_2'    }          "Bias"                  -0.000611
    {'conv_3_Weights'          }      {'conv_3'    }          "Weights"                -0.04594
    {'conv_3_Bias'             }      {'conv_3'    }          "Bias"                   -0.00139
    {'conv_4_Weights'          }      {'conv_4'    }          "Weights"                -0.04596
    {'conv_4_Bias'             }      {'conv_4'    }          "Bias"                    -0.0016
    {'fc_1_Weights'            }      {'fc_1'      }          "Weights"                -0.05139
    {'fc_1_Bias'               }      {'fc_1'      }          "Bias"                  -0.000523
    {'fc_2_Weights'            }      {'fc_2'      }          "Weights"                 -0.0501
    {'fc_2_Bias'               }      {'fc_2'      }          "Bias"                   -0.00175
    {'fc_3_Weights'            }      {'fc_3'      }          "Weights"                -0.05070
    {'fc_3_Bias'               }      {'fc_3'      }          "Bias"                    -0.0295
    {'imageinput'              }      {'imageinput'}          "Activations"                   0
    {'imageinput_normalization'}      {'imageinput'}          "Activations"             -139.34
      ⋮
```

Create Target Object

Create a target object with a custom name for your target device and an interface to connect your target device to the host computer. Interface options are JTAG and Ethernet. To use JTAG, install Xilinx™ Vivado™ Design Suite 2020.2. To set the Xilinx Vivado toolpath, enter:

```
% hdlsetuptoolpath('ToolName', 'Xilinx Vivado', 'ToolPath', 'C:\Xilinx\Vivado
\2020.2\bin\vivado.bat');
```

To create the target object, enter:

```
hTarget = dlhdl.Target('Xilinx','Interface','Ethernet');
```

Alternatively, you can also use the JTAG interface.

```
% hTarget = dlhdl.Target('Xilinx', 'Interface', 'JTAG');
```

Create `dlQuantizationOptions` Object

Create a `dlquantizationOptions` object. Specify the target bitstream and target board interface. The default metric function is a Top-1 accuracy metric function.

```
options_FPGA = dlquantizationOptions('Bitstream','zcu102_int8','Target',hTarget);
options_simulation = dlquantizationOptions;
```

To use a custom metric function, specify the metric function in the `dlquantizationOptions` object.

```
options_FPGA = dlquantizationOptions('MetricFcn',{@(x)hComputeAccuracy(x,snet,validationData_FPGA
options_simulation = dlquantizationOptions('MetricFcn',{@(x)hComputeAccuracy(x,snet,validationDa
```

Validate Quantized Network

Use the `validate` function to quantize the learnable parameters in the convolution layers of the network. The `validate` function simulates the quantized network in MATLAB. The `validate` function uses the metric function defined in the `dlquantizationOptions` object to compare the results of the single-data-type network object to the results of the quantized network object.

```
prediction_simulation = dlQuantObj_simulation.validate(validationData_simulation,options_simulati
```

```
Compiling leg: conv_1>>relu_4 ...
Compiling leg: conv_1>>relu_4 ... complete.
Compiling leg: maxpool_4 ...
Compiling leg: maxpool_4 ... complete.
Compiling leg: fc_1>>fc_3 ...
Compiling leg: fc_1>>fc_3 ... complete.

prediction_simulation = struct with fields:
       NumSamples: 5
    MetricResults: [1×1 struct]
       Statistics: []
```

For validation on an FPGA, the validate function:

- Programs the FPGA board by using the output of the `compile` method and the programming file
- Downloads the network weights and biases

- Compares the performance of the network before and after quantization

```
prediction_FPGA = dlQuantObj_FPGA.validate(validationData_FPGA,options_FPGA)

### Compiling network for Deep Learning FPGA prototyping ...
### Targeting FPGA bitstream zcu102_int8.
### The network includes the following layers:
     1   'imageinput'    Image Input             227×227×3 images with 'zerocenter' normalizatio
     2   'conv_1'        Convolution             96 5×5×3 convolutions with stride [1  1] and pa
     3   'relu_1'        ReLU                    ReLU
     4   'maxpool_1'     Max Pooling             3×3 max pooling with stride [2  2] and padding
     5   'conv_2'        Convolution             128 3×3×96 convolutions with stride [1  1] and
     6   'relu_2'        ReLU                    ReLU
     7   'maxpool_2'     Max Pooling             3×3 max pooling with stride [2  2] and padding
     8   'conv_3'        Convolution             384 3×3×128 convolutions with stride [1  1] and
     9   'relu_3'        ReLU                    ReLU
    10   'maxpool_3'     Max Pooling             3×3 max pooling with stride [2  2] and padding
    11   'conv_4'        Convolution             128 3×3×384 convolutions with stride [2  2] and
    12   'relu_4'        ReLU                    ReLU
    13   'maxpool_4'     Max Pooling             3×3 max pooling with stride [2  2] and padding
    14   'fc_1'          Fully Connected         2048 fully connected layer
    15   'relu_5'        ReLU                    ReLU
    16   'dropout_1'     Dropout                 50% dropout
    17   'fc_2'          Fully Connected         2048 fully connected layer
    18   'relu_6'        ReLU                    ReLU
    19   'dropout_2'     Dropout                 50% dropout
    20   'fc_3'          Fully Connected         32 fully connected layer
    21   'softmax'       Softmax                 softmax
    22   'classoutput'   Classification Output   crossentropyex with 'adidas' and 31 other classe

### Notice: The layer 'imageinput' with type 'nnet.cnn.layer.ImageInputLayer' is implemented in s
### Notice: The layer 'softmax' with type 'nnet.cnn.layer.SoftmaxLayer' is implemented in softwa
### Notice: The layer 'classoutput' with type 'nnet.cnn.layer.ClassificationOutputLayer' is imple
### Compiling layer group: conv_1>>relu_4 ...
### Compiling layer group: conv_1>>relu_4 ... complete.
### Compiling layer group: maxpool_4 ...
### Compiling layer group: maxpool_4 ... complete.
### Compiling layer group: fc_1>>fc_3 ...
### Compiling layer group: fc_1>>fc_3 ... complete.

### Allocating external memory buffers:

          offset_name          offset_address    allocated_space
    _______________________    ______________    _______________

    "InputDataOffset"           "0x00000000"     "12.0 MB"
    "OutputResultOffset"        "0x00c00000"     "4.0 MB"
    "SchedulerDataOffset"       "0x01000000"     "4.0 MB"
    "SystemBufferOffset"        "0x01400000"     "36.0 MB"
    "InstructionDataOffset"     "0x03800000"     "8.0 MB"
    "ConvWeightDataOffset"      "0x04000000"     "12.0 MB"
    "FCWeightDataOffset"        "0x04c00000"     "12.0 MB"
    "EndOffset"                 "0x05800000"     "Total: 88.0 MB"

### Network compilation complete.

### FPGA bitstream programming has been skipped as the same bitstream is already loaded on the ta
### Deep learning network programming has been skipped as the same network is already loaded on t
```

```
### Finished writing input activations.
### Running single input activation.


              Deep Learning Processor Bitstream Build Info

Resource                   Utilized           Total        Percentage
------------------        ----------      ------------    ------------
LUTs (CLB/ALM)*              248358            274080           90.62
DSPs                            384              2520           15.24
Block RAM                       581               912           63.71
* LUT count represents Configurable Logic Block(CLB) utilization in Xilinx devices and Adaptive L

### Notice: The layer 'imageinput' of type 'ImageInputLayer' is split into an image input layer
### Notice: The layer 'softmax' with type 'nnet.cnn.layer.SoftmaxLayer' is implemented in softwar
### Notice: The layer 'classoutput' with type 'nnet.cnn.layer.ClassificationOutputLayer' is imple


              Deep Learning Processor Estimator Performance Results

                     LastFrameLatency(cycles)   LastFrameLatency(seconds)       FramesNum      Tota
                    ------------              ------------                    ---------        --
Network                    40142478                   0.18247                       1           401
    ____imageinput_norm      216472                   0.00098
    ____conv_1              6825671                   0.03103
    ____maxpool_1           3755088                   0.01707
    ____conv_2             10440701                   0.04746
    ____maxpool_2           1447840                   0.00658
    ____conv_3              9405685                   0.04275
    ____maxpool_3           1765856                   0.00803
    ____conv_4              1819636                   0.00827
    ____maxpool_4             28098                   0.00013
    ____fc_1                2651288                   0.01205
    ____fc_2                1696632                   0.00771
    ____fc_3                  89511                   0.00041
 * The clock frequency of the DL processor is: 220MHz



              Deep Learning Processor Bitstream Build Info

Resource                   Utilized           Total        Percentage
------------------        ----------      ------------    ------------
LUTs (CLB/ALM)*              168645            274080           61.53
DSPs                            800              2520           31.75
Block RAM                       453               912           49.67
* LUT count represents Configurable Logic Block(CLB) utilization in Xilinx devices and Adaptive L

### Finished writing input activations.
### Running single input activation.

prediction_FPGA = struct with fields:
       NumSamples: 1
    MetricResults: [1×1 struct]
       Statistics: [2×7 table]
```

View Performance of Quantized Neural Network

Display the accuracy of the quantized network.

```
prediction_simulation.MetricResults.Result
```

```
ans=2×2 table
    NetworkImplementation    MetricOutput
    _____________________    ____________

     {'Floating-Point'}           1
     {'Quantized'     }           1
```

```
prediction_FPGA.MetricResults.Result
```

```
ans=2×2 table
    NetworkImplementation    MetricOutput
    _____________________    ____________

     {'Floating-Point'}           1
     {'Quantized'     }           1
```

Display the performance of the quantized network in frames per second.

```
prediction_FPGA.Statistics.FramesPerSecond(2)
```

```
ans = 19.0828
```

Quantize a Neural Network for CPU Target

This example shows how to quantize and validate a neural network for a CPU target. This workflow is similar to other execution environments, but before validating you must establish a `raspi` connection and specify it as target using `dlquantizationOptions`.

First, load your network. This example uses the pretrained network `squeezenet`.

```
load squeezenetmerch
net
```

```
net =
  DAGNetwork with properties:

         Layers: [68×1 nnet.cnn.layer.Layer]
    Connections: [75×2 table]
     InputNames: {'data'}
    OutputNames: {'new_classoutput'}
```

Then define your calibration and validation data, `calDS` and `valDS` respectively.

```
unzip('MerchData.zip');
imds = imageDatastore('MerchData', ...
    'IncludeSubfolders',true, ...
    'LabelSource','foldernames');
[calData, valData] = splitEachLabel(imds, 0.7, 'randomized');
```

```
aug_calData = augmentedImageDatastore([227 227],calData);
aug_valData = augmentedImageDatastore([227 227],valData);
```

Create the `dlquantizer` object and specify a CPU execution environment.

```
dq =  dlquantizer(net,'ExecutionEnvironment','CPU')
```

```
dq =
  dlquantizer with properties:

           NetworkObject: [1×1 DAGNetwork]
    ExecutionEnvironment: 'CPU'
```

Calibrate the network.

```
calResults = calibrate(dq,aug_calData,'UseGPU','off')
```

```
calResults=122×5 table
        Optimized Layer Name         Network Layer Name     Learnables / Activations    MinValue
    _____________________________    ___________________    ________________________    _________

    {'conv1_Weights'           }    {'conv1'           }           "Weights"             -0.91985
    {'conv1_Bias'              }    {'conv1'           }           "Bias"                -0.07925
    {'fire2-squeeze1x1_Weights'}    {'fire2-squeeze1x1'}           "Weights"                -1.38
    {'fire2-squeeze1x1_Bias'   }    {'fire2-squeeze1x1'}           "Bias"                -0.11641
    {'fire2-expand1x1_Weights' }    {'fire2-expand1x1' }           "Weights"              -0.7406
    {'fire2-expand1x1_Bias'    }    {'fire2-expand1x1' }           "Bias"                -0.06005
    {'fire2-expand3x3_Weights' }    {'fire2-expand3x3' }           "Weights"              -0.7439
    {'fire2-expand3x3_Bias'    }    {'fire2-expand3x3' }           "Bias"                -0.05177
    {'fire3-squeeze1x1_Weights'}    {'fire3-squeeze1x1'}           "Weights"              -0.7712
    {'fire3-squeeze1x1_Bias'   }    {'fire3-squeeze1x1'}           "Bias"                -0.10138
    {'fire3-expand1x1_Weights' }    {'fire3-expand1x1' }           "Weights"              -0.7203
    {'fire3-expand1x1_Bias'    }    {'fire3-expand1x1' }           "Bias"                -0.06702
    {'fire3-expand3x3_Weights' }    {'fire3-expand3x3' }           "Weights"              -0.6144
    {'fire3-expand3x3_Bias'    }    {'fire3-expand3x3' }           "Bias"                -0.05361
    {'fire4-squeeze1x1_Weights'}    {'fire4-squeeze1x1'}           "Weights"              -0.7422
    {'fire4-squeeze1x1_Bias'   }    {'fire4-squeeze1x1'}           "Bias"                -0.10885
      ⋮
```

Use the MATLAB Support Package for Raspberry Pi Hardware function, `raspi`, to create a connection to the Raspberry Pi. In the following code, replace:

- `raspiname` with the name or address of your Raspberry Pi
- `username` with your user name
- `password` with your password

```
% r = raspi('raspiname','username','password')
```

For example,

```
r = raspi('gpucoder-raspberrypi-7','pi','matlab')
```

```
r =
  raspi with properties:

         DeviceAddress: 'gpucoder-raspberrypi-7'
```

```
                Port: 18734
           BoardName: 'Raspberry Pi 3 Model B+'
       AvailableLEDs: {'led0'}
  AvailableDigitalPins: [2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27]
  AvailableSPIChannels: {}
     AvailableI2CBuses: {}
      AvailableWebcams: {}
           I2CBusSpeed:
AvailableCANInterfaces: {}

  Supported peripherals
```

Specify `raspi` object as the target for the quantized network.

```
opts = dlquantizationOptions('Target',r)
```

```
opts =
  dlquantizationOptions with properties:

    MetricFcn: {}
    Bitstream: ''
       Target: [1×1 raspi]
```

Validate the quantized network with the `validate` function.

```
valResults = validate(dq,aug_valData,opts)
```

```
### Starting application: 'codegen\lib\validate_predict_int8\pil\validate_predict_int8.elf'
    To terminate execution: clear validate_predict_int8_pil
### Launching application validate_predict_int8.elf...
### Host application produced the following standard output (stdout) and standard error (stderr)

valResults = struct with fields:
       NumSamples: 20
    MetricResults: [1×1 struct]
       Statistics: []
```

Examine the validation output to see the performance of the quantized network.

```
valResults.MetricResults.Result
```

```
ans=2×2 table
    NetworkImplementation    MetricOutput
    _____________________    ____________

     {'Floating-Point'}          0.95
     {'Quantized'     }          0.95
```

Quantize YOLO v3 Object Detector

This example shows how to quantize a `yolov3ObjectDetector` (Computer Vision Toolbox) object using preprocessed calibration and validation data.

First, download a pretrained YOLO v3 object detector.

```
detector = downloadPretrainedNetwork();
```

This example uses a small labeled data set that contains one or two labeled instances of a vehicle. Many of these images come from the Caltech Cars 1999 and 2001 data sets, created by Pietro Perona and used with permission.

Unzip the vehicle images and load the vehicle ground truth data.

```
unzip vehicleDatasetImages.zip
data = load('vehicleDatasetGroundTruth.mat');
vehicleDataset = data.vehicleDataset;
```

Add the full path to the local vehicle data folder.

```
vehicleDataset.imageFilename = fullfile(pwd, vehicleDataset.imageFilename);
```

Create an `imageDatastore` for loading the images and a `boxLabelDatastore` (Computer Vision Toolbox) for the ground truth bounding boxes.

```
imds = imageDatastore(vehicleDataset.imageFilename);
blds = boxLabelDatastore(vehicleDataset(:,2));
```

Use the `combine` function to combine both the datastores into a `CombinedDatastore`.

```
combinedDS = combine(imds, blds);
```

Split the data into calibration and validation data.

```
calData = combinedDS.subset(1:32);
valData = combinedDS.subset(33:64);
```

Use the `preprocess` (Computer Vision Toolbox) method of `yolov3ObjectDetector` (Computer Vision Toolbox) object with `transform` function to prepare the data for calibration and validation.

The `transform` function returns a `TransformedDatastore` object.

```
processedCalData = transform(calData, @(data)preprocess(detector,data));
processedValData = transform(valData, @(data)preprocess(detector,data));
```

Create the `dlquantizer` object.

```
dq = dlquantizer(detector, 'ExecutionEnvironment', 'MATLAB');
```

Calibrate the network.

```
calResults = calibrate(dq, processedCalData,'UseGPU','off')
```

```
calResults=135×5 table
        Optimized Layer Name         Network Layer Name     Learnables / Activations    MinValue
    _____________________________    ___________________    ________________________    _________

    {'conv1_Weights'           }    {'conv1'           }           "Weights"             -0.92189
    {'conv1_Bias'              }    {'conv1'           }           "Bias"               -0.096271
    {'fire2-squeeze1x1_Weights'}    {'fire2-squeeze1x1'}           "Weights"              -1.3751
    {'fire2-squeeze1x1_Bias'   }    {'fire2-squeeze1x1'}           "Bias"                -0.12068
    {'fire2-expand1x1_Weights' }    {'fire2-expand1x1' }           "Weights"             -0.75275
```

```
    {'fire2-expand1x1_Bias'    }    {'fire2-expand1x1' }          "Bias"              -0.05925
    {'fire2-expand3x3_Weights' }    {'fire2-expand3x3' }          "Weights"            -0.7527
    {'fire2-expand3x3_Bias'    }    {'fire2-expand3x3' }          "Bias"              -0.06221
    {'fire3-squeeze1x1_Weights'}    {'fire3-squeeze1x1'}          "Weights"              -0.758
    {'fire3-squeeze1x1_Bias'   }    {'fire3-squeeze1x1'}          "Bias"               -0.1020
    {'fire3-expand1x1_Weights' }    {'fire3-expand1x1' }          "Weights"            -0.7156
    {'fire3-expand1x1_Bias'    }    {'fire3-expand1x1' }          "Bias"              -0.06931
    {'fire3-expand3x3_Weights' }    {'fire3-expand3x3' }          "Weights"            -0.6007
    {'fire3-expand3x3_Bias'    }    {'fire3-expand3x3' }          "Bias"              -0.05804
    {'fire4-squeeze1x1_Weights'}    {'fire4-squeeze1x1'}          "Weights"              -0.738
    {'fire4-squeeze1x1_Bias'   }    {'fire4-squeeze1x1'}          "Bias"               -0.1118
      ⋮
```

Validate the quantized network with the `validate` function.

```
valResults = validate(dq, processedValData)
```

```
valResults = struct with fields:
       NumSamples: 32
    MetricResults: [1×1 struct]
       Statistics: []
```

```
function detector = downloadPretrainedNetwork()
   pretrainedURL = 'https://ssd.mathworks.com/supportfiles/vision/data/yolov3SqueezeNetVehicleExa
   websave('yolov3SqueezeNetVehicleExample_21aSPKG.zip', pretrainedURL);

   unzip('yolov3SqueezeNetVehicleExample_21aSPKG.zip');

   pretrained = load("yolov3SqueezeNetVehicleExample_21aSPKG.mat");
   detector = pretrained.detector;
end
```

Input Arguments

`quantObj` — Network to quantize

`dlquantizer` object

Network to quantize, specified as a `dlquantizer` object.

`valData` — Data to use for validation of quantized network

`imageDatastore` object | `augmentedImageDatastore` object | `pixelLabelImageDatastore` object | `CombinedDatastore` object | `TransformedDatastore` object

Data to use for validation of quantized network, specified as an `imageDatastore` object, an `augmentedImageDatastore` object, a `pixelLabelImageDatastore` object, a `CombinedDatastore` object, or a `TransformedDatastore` object.

You must preprocess the data used for validation of a quantized `yolov3ObjectDetector` object using the `preprocess` function. For an example of using preprocessed data for validation of a `yolov3ObjectDetector`, see “Quantize YOLO v3 Object Detector”.

`validate` accepts a `CombinedDatastore` or `TransformedDatastore` object as input data for validating quantized `yolov3ObjectDetector` and `yolov4ObjectDetector` objects. The

`CombinedDatastore` and `TransformedDatastore` used for validation must contain an `imageDatastore` or `augmentedImageDatastore` as the first datastore and a `boxLabelDatastore` as the second datastore. For more information on valid datastores, see “Prepare Data for Quantizing Networks”.

`quantOpts` — Options for quantizing network
`dlquantizationOptions` object

Options for quantizing the network, specified as a `dlquantizationOptions` object.

Output Arguments

`valResults` — Performance of quantized network
struct

Performance of quantized network, returned as a struct. The struct contains these fields.

- `NumSamples` — The number of sample inputs used to validate the network, specified by `valData`.
- `MetricResults` — Struct containing results of the metric function defined in the `dlquantizationOptions` object. When more than one metric function is specified in the `dlquantizationOptions` object, `MetricResults` is an array of structs.

  `MetricResults` contains these fields:

Field	Description
`MetricFunction`	Metric function used to determine the performance of the quantized network, specified in the `dlquantizationOptions` object.
`Result`	Table indicating the results of the metric function before and after quantization. The first row in the table, `'Floating-Point'`, contains information for the original floating-point implementation. The second row, `'Quantized'`, contains information for the quantized implementation. The output of the metric function is displayed in the `MetricOutput` column.

- `Statistics` — Table indicating the learnable parameter memory used, in bytes, by the original floating-point implementation of the network and the quantized implementation.

  When the `ExecutionEnvironment` for the `dlquantizer` object is set to `FPGA`, the `Statistics` table is a table indicating these values for the original floating-point and quantized network implementations:

  - Frames per second
  - Number of convolution threads
  - Number of fully connected threads
  - Lookup table (LUT) resource utilization percentage

- Block RAM resource utilization percentage
- DSP resource utilization percentage

Algorithms

The `validate` function determines the default metric function to use for the validation based on the type of network that is being quantized.

Type of Network	Metric Function
Classification	Top-1 Accuracy — Accuracy of the network
Object Detection	Average Precision — Average precision over all detection results. See `evaluateDetectionPrecision`.
Regression	MSE — Mean squared error of the network
Semantic Segmentation	`evaluateSemanticSegmentation` — Evaluate semantic segmentation data set against ground truth
Single Shot Detector (SSD)	WeightedIOU — Average IoU of each class, weighted by the number of pixels in that class

When the `'Simulation'` property of the `dlquantizer` object is set to `'on'`, the default metric function uses at most 5 files from the validation datastore. Custom metric functions, specified using a `dlquantizationOptions` object, do not have this restriction.

Version History

Introduced in R2020a

R2022a: Validate the performance of quantized network for CPU target

You can now use the `dlquantizer` object and the `validate` function to quantize a network and generate code for CPU targets.

See Also

Apps
Deep Network Quantizer

Functions
`calibrate` | `dlquantizer` | `dlquantizationOptions` | `quantize` | `quantizationDetails` | `estimateNetworkMetrics`

Topics
“Quantization Workflow Prerequisites”
“Prepare Data for Quantizing Networks”
“Quantization of Deep Neural Networks”
“Quantize Residual Network Trained for Image Classification and Generate CUDA Code”
“Quantize Network for FPGA Deployment”

“Generate int8 Code for Deep Learning Networks” (MATLAB Coder)

dlhdl.Simulator class

Package: `dlhdl`

Create an object that retrieves intermediate layer results and validate deep learning network prediction accuracy

Description

Use the `dlhdl.Simulator` class to creates a simulator object that you use to retrieve intermediate layer results and verify the accuracy of prediction results for your deep learning network without the need for hardware targets.

Creation

`simObj=dlhdl.Simulator('Network',Network,'ProcessorConfig',ProcessorConfig)` creates a `dlhdl.Simulator` object that you use to retrieve intermediate layer results and verify the accuracy of prediction results for your deep learning network without the need for hardware targets.

Input Arguments

`Network` — Network object

`SeriesNetwork` object | `DAGNetwork` object | `dlquantizer` object

Name of the deep learning network object.

Example: `'network',net` creates a workflow object for the saved pretrained network `net`. To specify `net`, you can import any of the existing supported pretrained networks or use transfer learning to adapt the network to your issue. See “Supported Pretrained Networks”.

```
net = resnet18;
hPC = dlhdl.ProcessorConfig;
simObj = dlhdl.Simulator('Network',net,'Bitstream',hPC);
```

Example: `'network',dlquantizeObj` creates a workflow object for the quantized network object `dlquantizeObj`. To specify `dlquantizeObj`, you can import any of the supported existing pretrained networks and create an object by using the `dlquantizer` class. For information on supported networks, see “Supported Pretrained Networks”.

```
net = resnet18;
dlquantObj = dlquantizer(net,'ExecutionEnvironment','FPGA');
dlquantObj.calibrate(imdsTrain);
simObj = dlhdl.Simulator('Network',dlquantObj,'Bitstream',hPC);
```

`ProcessorConfig` — `dlhdl.ProcessorConfig` object

`hPC`

Deep learning processor configuration object, specified as a `dlhdl.ProcessorConfig` object

Example: `'ProcessorConfig',hPC`

```
hPC = dlhdl.ProcessorConfig;
simObj = dlhdl.Simulator('Network',resnet18,'ProcessorConfig',hPC);
```

Methods

Public Methods

activations	Retrieve intermediate layers results for dlhdl.Simulator object
predict	Retrieve prediction results for dlhdl.Simulator object

Examples

Create a `dlhdl.Simulator` Object for the ResNet-18 Network

1. Retrieve the deep learning processor configuration for the `zcu102_single` bitstream and save to `hPC`.

```
hPC = dlhdl.ProcessorConfig('Bitstream','zcu102_single');
```

2. Create a `dlhdl.Simulator` object with resnet18 as the network and `hPC` as the `ProcessorConfig` object.

```
simObj = dlhdl.Simulator('Network',resnet18,'ProcessorConfig',hPC);
```

Create a `dlhdl.Simulator` Object for the resnet18 Network and int8 data type Deep Learning Processor Configuration

1. Create a deep learning processor configuration that has the int8 data type and save it to `hPC`.

```
hPC = dlhdl.ProcessorConfig;
hPC.ProcessorDataType = 'int8';
```

2. Create a `dlquantizer` object with ResNet-18 as the network and FPGA execution environment. Calibrate the quantized network object by using the `calibrate` function.

```
net = resnet18;
dlQuantObj = dlquantizer(net,'ExecutionEnvironment','FPGA');
dlQuantObj.calibrate(imageDataStore);
```

3. Create a `dlhdl.Simulator` object with `dlQuantObj` as the network and `hPC` as the `ProcessorConfig` object.

```
simObj = dlhdl.Simulator('Network',dlQuantObj,'ProcessorConfig',hPC);
```

Version History

Introduced in R2021b

See Also

`dlhdl.Workflow` | `dlhdl.Target`

Topics

“Prototype and Verify Deep Learning Networks Without Target Hardware”

activations

Class: `dlhdl.Simulator`
Package: `dlhdl`

Retrieve intermediate layers results for `dlhdl.Simulator` object

Syntax

```
activations(simulatorObject,image,layername)
result = activations(simulatorObject,image,layername)
```

Description

`activations(simulatorObject,image,layername)` returns intermediate layer activation data results for the image data in `image` and the name of the layer specified in `layername`. The result size depends on the output size of the layer. The layer output size can be retrieved by using the `analyzeNetwork` function.

`result = activations(simulatorObject,image,layername)` stores the intermediate layer activation data results for the image data in `image` and the name of the layer specified in `layername` in `result`. The result size depends on the output size of the layer. The layer output size can be retrieved by using the `analyzeNetwork` function.

Input Arguments

`simulatorObject` — Instance of simulator object

`dlhdl.Simulator` object

Instance of the simulator object, specified as an `dlhdl.Simulator` object.

`image` — Input image

m-by-*n*-by-*k* numeric array

Input image, specified as a *m*-by-*n*-by-*k* numeric array. *m* , *n*, and *k* must match the dimensions of the deep learning network input image layer. For example, for the LogoNet network, resize the input images to a 227-by-227-by-3 array.

Data Types: `single`

`layername` — Layer to extract activations from

'' (default) | character vector

Layer to extract activations from, specified as a character vector.

To compute the activations of a `SeriesNetwork` object, specify the layer as a character vector corresponding to the layer name.

To compute the activations of a `DAGNetwork` object, specify the layer as a character vector corresponding to the layer name. If the layer has multiple outputs, specify the layer and output as the layer name, followed by the character `"/"`, followed by the name of the layer output. That is, `layername` of the form `'layerName/outputName'`.

These limitations apply when extracting activations from layers:

- A `maxPooling2DLayer` which has the `HasUnpoolingIndices` argument set to `true` has three outputs `out`, `indices`, and `size`. Activations are supported only for the output `out`.
- Activations for a `ReLU`layer in the branch of a DAG network are supported when the layer is not preceded by a `Convolution`, `Fully Connected`, or `Addition` layer.
- Activations for the `dropout` layer is not supported.

Example: `'conv1'`

Example: `'mpool/out'`

Output Arguments

`result` — Intermediate layer activation data

`array of single`

Intermediate layer activation data, returned as an array of singles. The array size depends on the layer output size. For example, for the ResNet-18 network `pool1` layer, the size of the returned result array is 56-by-56-by-64.

Examples

Prototype and Verify Deep Learning Networks Without Target Hardware

Rapidly prototype your custom deep learning network and bitstream by visualizing intermediate layer activation results and verifying prediction accuracy without target hardware by emulating the network and bitstream. To emulate the network and bitstream, create a `dlhdl.Simulator` object. Use the `dlhdl.Simulator` object to:

- Retrieve intermediate layer results by using the `activations` function.
- Verify prediction accuracy by using the `predict` function.

In this example, retrieve the intermediate layer activation results and verify the prediction accuracy for the ResNet-18 network and deep learning processor configuration for the `zcu102_single` bitstream.

Prerequisites

- Deep Learning Toolbox ™
- Deep Learning HDL Toolbox ™
- Deep Learning Toolbox Model for ResNet-18 Network
- Deep Learning HDL Toolbox Support Package for Xilinx FPGA and SoC Devices
- Image Processing Toolbox ™
- MATLAB® Coder™ Interface for Deep learning

Load Pretrained SeriesNetwork

To load the pretrained network ResNet-18, enter:

```
snet = resnet18;
```

To view the layers of the pretrained network, enter:

```
analyzeNetwork(snet);
```

The first layer, the image input layer, requires input images of size 224-by-224-by-3, where 3 is the number of color channels.

```
inputSize = snet.Layers(1).InputSize;
```

Define Training and Validation Data Sets

This example uses the MathWorks MerchData data set. This is a small data set containing 75 images of MathWorks merchandise, belonging to five different classes (cap, cube, playing cards, screwdriver, and torch).

```
curDir = pwd;
unzip('MerchData.zip');
imds = imageDatastore('MerchData', ...
'IncludeSubfolders',true, ...
'LabelSource','foldernames');
[imdsTrain,imdsValidation] = splitEachLabel(imds,0.7,'randomized');
```

Replace Final Layers

The fully connected layer and the classification layer of the pretrained network `net` are configured for 1000 classes. These two layers `fc1000` and `ClassificationLayer_predictions` in ResNet-18 contain information on how to combine the features that the network extracts into class probabilities and predicted labels. These layers must be fine-tuned for the new classification problem. Extract all the layers, except the last two layers, from the pretrained network.

```
lgraph = layerGraph(snet)
```

```
lgraph =
  LayerGraph with properties:

     InputNames: {'data'}
    OutputNames: {'ClassificationLayer_predictions'}
         Layers: [71×1 nnet.cnn.layer.Layer]
    Connections: [78×2 table]
```

```
numClasses = numel(categories(imdsTrain.Labels))
```

```
numClasses = 5
```

```
newLearnableLayer = fullyConnectedLayer(numClasses, ...
'Name','new_fc', ...
'WeightLearnRateFactor',10, ...
'BiasLearnRateFactor',10);
lgraph = replaceLayer(lgraph,'fc1000',newLearnableLayer);
newClassLayer = classificationLayer('Name','new_classoutput');
lgraph = replaceLayer(lgraph,'ClassificationLayer_predictions',newClassLayer);
```

Train Network

The network requires input images of size 224-by-224-by-3, but the images in the image datastores have different sizes. Use an augmented image datastore to automatically resize the training images. Specify additional augmentation operations to perform on the training images, such as randomly

flipping the training images along the vertical axis and randomly translating them up to 30 pixels horizontally and vertically. Data augmentation helps prevent the network from overfitting and memorizing the exact details of the training images.

```
pixelRange = [-30 30];
imageAugmenter = imageDataAugmenter( ...
'RandXReflection',true, ...
'RandXTranslation',pixelRange, ...
'RandYTranslation',pixelRange);
```

To automatically resize the validation images without performing further data augmentation, use an augmented image datastore without specifying any additional preprocessing operations.

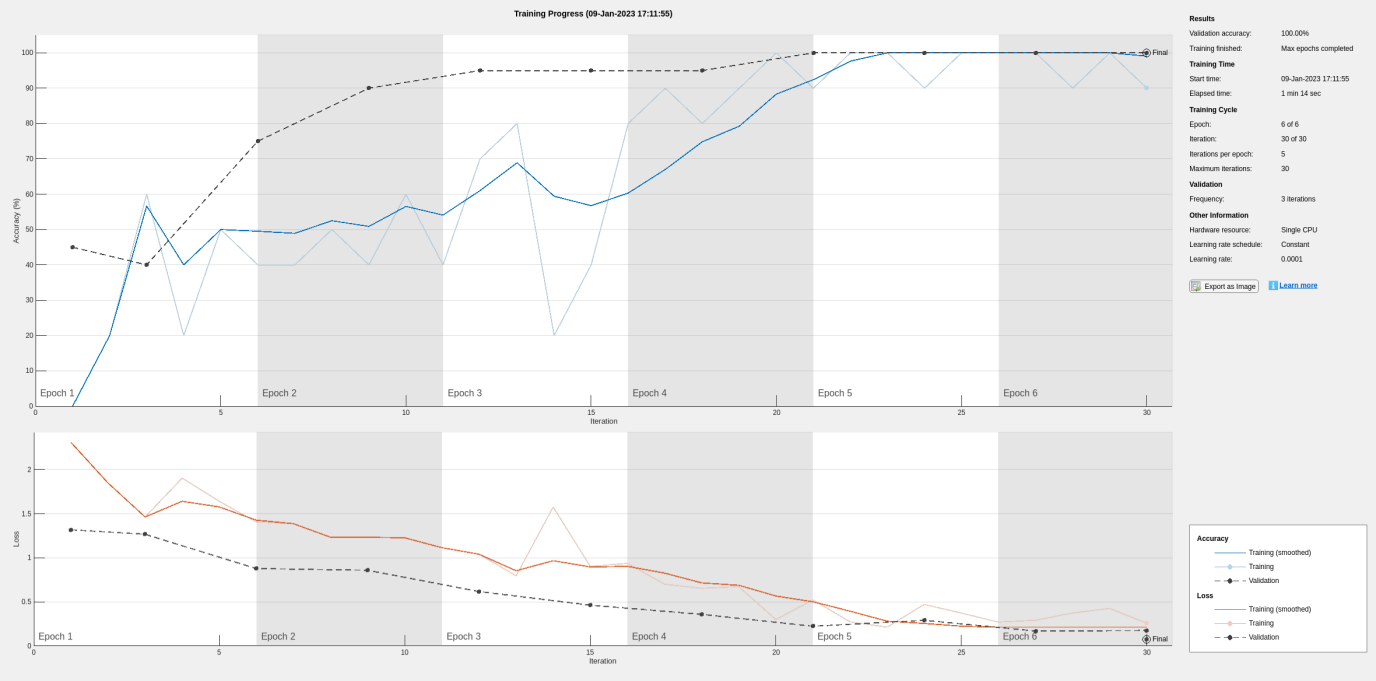
```
augimdsTrain = augmentedImageDatastore(inputSize(1:2),imdsTrain, ...
'DataAugmentation',imageAugmenter);
augimdsValidation = augmentedImageDatastore(inputSize(1:2),imdsValidation);
```

Specify the training options. For transfer learning, keep the features from the early layers of the pretrained network (the transferred layer weights). To slow down learning in the transferred layers, set the initial learning rate to a small value. Specify the mini-batch size and validation data. The software validates the network for every `ValidationFrequency` iteration during training.

```
options = trainingOptions('sgdm', ...
'MiniBatchSize',10, ...
'MaxEpochs',6, ...
'InitialLearnRate',1e-4, ...
'Shuffle','every-epoch', ...
'ValidationData',augimdsValidation, ...
'ValidationFrequency',3, ...
'Verbose',false, ...
'Plots','training-progress');
```

Train the network that consists of the transferred and new layers. By default, `trainNetwork` uses a GPU if one is available (requires Parallel Computing Toolbox™ and a supported GPU device. See “GPU Computing Requirements” (Parallel Computing Toolbox)). Otherwise, the network uses a CPU (requires MATLAB Coder Interface for Deep learning). You can also specify the execution environment by using the `'ExecutionEnvironment'` name-value argument of `trainingOptions`.

```
netTransfer = trainNetwork(augimdsTrain,lgraph,options);
```



Retrieve Deep Learning Processor Configuration

Use the `dlhdl.ProcessorConfig` object to retrieve the deep learning processor configuration for the `zcu102_single` bitstream.

```
hPC = dlhdl.ProcessorConfig('Bitstream','zcu102_single');
```

Create Simulator Object

Create a `dlhdl.Simulator` object with ResNet-18 as the network and `hPC` as the `ProcessorConfig` object.

```
simObj = dlhdl.Simulator('Network',netTransfer,'ProcessorConfig',hPC);
```

```
### Optimizing network: Fused 'nnet.cnn.layer.BatchNormalizationLayer' into 'nnet.cnn.layer.Conv
Compiling leg: conv1>>pool1 ...
Compiling leg: conv1>>pool1 ... complete.
Compiling leg: res2a_branch2a>>res2a_branch2b ...
Compiling leg: res2a_branch2a>>res2a_branch2b ... complete.
Compiling leg: res2b_branch2a>>res2b_branch2b ...
Compiling leg: res2b_branch2a>>res2b_branch2b ... complete.
Compiling leg: res3a_branch1 ...
Compiling leg: res3a_branch1 ... complete.
Compiling leg: res3a_branch2a>>res3a_branch2b ...
Compiling leg: res3a_branch2a>>res3a_branch2b ... complete.
Compiling leg: res3b_branch2a>>res3b_branch2b ...
Compiling leg: res3b_branch2a>>res3b_branch2b ... complete.
Compiling leg: res4a_branch1 ...
Compiling leg: res4a_branch1 ... complete.
Compiling leg: res4a_branch2a>>res4a_branch2b ...
Compiling leg: res4a_branch2a>>res4a_branch2b ... complete.
Compiling leg: res4b_branch2a>>res4b_branch2b ...
Compiling leg: res4b_branch2a>>res4b_branch2b ... complete.
```

```
Compiling leg: res5a_branch1 ...
Compiling leg: res5a_branch1 ... complete.
Compiling leg: res5a_branch2a>>res5a_branch2b ...
Compiling leg: res5a_branch2a>>res5a_branch2b ... complete.
Compiling leg: res5b_branch2a>>res5b_branch2b ...
Compiling leg: res5b_branch2a>>res5b_branch2b ... complete.
Compiling leg: pool5 ...
Compiling leg: pool5 ... complete.
Compiling leg: new_fc ...
Compiling leg: new_fc ... complete.
```

Load Image for Prediction and Intermediate Layer Activation Results

Load the example image. Save it's size for future use.

```
imgFile = fullfile(pwd,'MerchData','MathWorks Cube','MathWorks cube_0.jpg');
inputImg = imresize(imread(imgFile),inputSize(1:2));
imshow(inputImg)
```

Show Activations of First Maxpool Layer

Investigate features by observing which areas in the convolution layers activate on an image. Compare that image to the corresponding areas in the original images. Each layer of a convolutional neural network consists of many 2-D arrays called *channels*. Pass the image through the network and examine the output activations of the `pool1` layer.

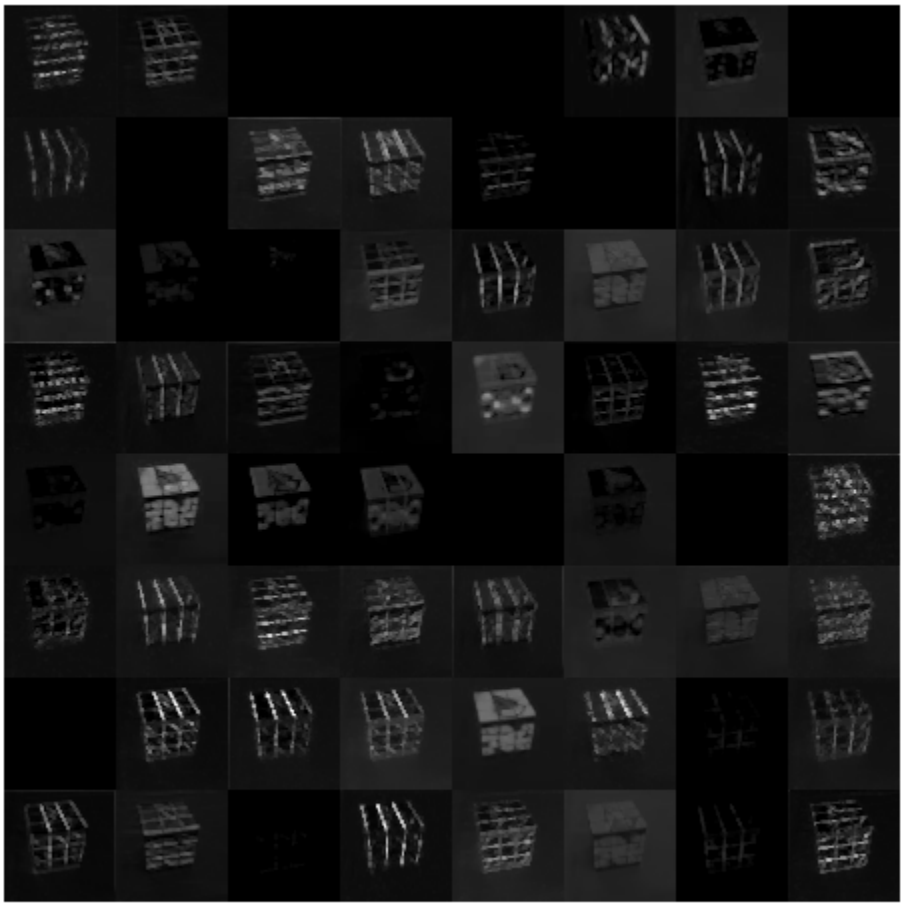
```
act1 = simObj.activations(single(inputImg),'pool1');
```

The activations are returned as a 3-D array, with the third dimension indexing the channel on the `pool1` layer. To show these activations by using the `imtile` function, reshape the array to 4-D. The third dimension in the input to `imtile` represents the image color. Set the third dimension to have size 1 because the activations do not have color. The fourth dimension indexes the channel.

```
sz = size(act1);
act1 = reshape(act1,[sz(1) sz(2) 1 sz(3)]);
```

Display the activations. Each activation can take any value, so normalize the output by using the `mat2gray`. All activations are scaled so that the minimum activation is 0 and the maximum activation is 1. Display the 64 images on an 8-by-8 grid, one for each channel in the layer.

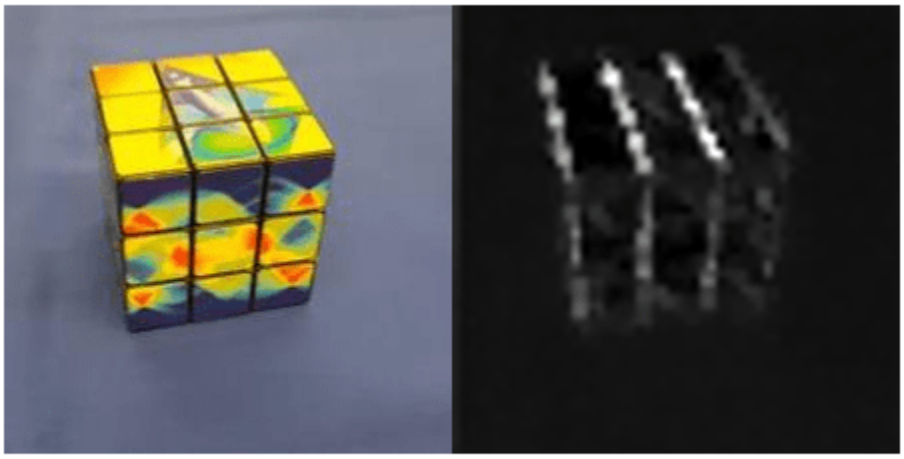
```
I = imtile(mat2gray(act1),'GridSize',[8 8]);
imshow(I)
```



Find Strongest Activation Channel

Find the strongest channels by programmatically investigating channels with large activations. Find the channel that has the largest activation by using the `max` function, resize the channel output, and display the activations.

```
[maxValue,maxValueIndex] = max(max(max(act1)));
act1chMax = act1(:,:,:,maxValueIndex);
act1chMax = mat2gray(act1chMax);
act1chMax = imresize(act1chMax,inputSize(1:2));

I = imtile({inputImg,act1chMax});
imshow(I)
```



Compare the strongest activation channel image to the original image. This channel activates on edges. It activates positively on light left/dark right edges and negatively on dark left/light right edges.

Verify Prediction Results

Verify and display the prediction results of the `dlhdl.Simulator` object by using the `predict` function.

```
prediction = simObj.predict(single(inputImg));
[val, idx] = max(prediction);
netTransfer.Layers(end).ClassNames{idx}
```

```
ans =
'MathWorks Cube'
```

Version History

Introduced in R2021b

See Also

`dlhdl.Simulator` | `predict`

Topics

“Prototype and Verify Deep Learning Networks Without Target Hardware”

predict

Class: `dlhdl.Simulator`
Package: `dlhdl`

Retrieve prediction results for `dlhdl.Simulator` object

Syntax

```
prediction = predict(simulatorObject,image)
```

Description

`prediction = predict(simulatorObject,image)` returns a table containing the percentage prediction values for the input image specified by `image`.

Input Arguments

`simulatorObject` — Instance of simulator object

`dlhdl.Simulator` object

Instance of the simulator object, specified as an `dlhdl.Simulator` object.

`image` — Input image

m-by-*n*-by-*k* numeric array

Input image, specified as a *m*-by-*n*-by-*k* numeric array. *m*, *n*, and *k* must match the dimensions of the deep learning network input image layer. For example, for the LogoNet network, resize the input images to a 227-by-227-by-3 array.

Data Types: `single`

Output Arguments

`prediction` — Network prediction for input image

`table`

Deep learning network prediction for the input image specified by `image`.

Examples

Prototype and Verify Deep Learning Networks Without Target Hardware

Rapidly prototype your custom deep learning network and bitstream by visualizing intermediate layer activation results and verifying prediction accuracy without target hardware by emulating the network and bitstream. To emulate the network and bitstream, create a `dlhdl.Simulator` object. Use the `dlhdl.Simulator` object to:

- Retrieve intermediate layer results by using the `activations` function.

- Verify prediction accuracy by using the `predict` function.

In this example, retrieve the intermediate layer activation results and verify the prediction accuracy for the ResNet-18 network and deep learning processor configuration for the `zcu102_single` bitstream.

Prerequisites

- Deep Learning Toolbox ™
- Deep Learning HDL Toolbox ™
- Deep Learning Toolbox Model for ResNet-18 Network
- Deep Learning HDL Toolbox Support Package for Xilinx FPGA and SoC Devices
- Image Processing Toolbox ™
- MATLAB® Coder™ Interface for Deep learning

Load Pretrained SeriesNetwork

To load the pretrained network ResNet-18, enter:

```
snet = resnet18;
```

To view the layers of the pretrained network, enter:

```
analyzeNetwork(snet);
```

The first layer, the image input layer, requires input images of size 224-by-224-by-3, where 3 is the number of color channels.

```
inputSize = snet.Layers(1).InputSize;
```

Define Training and Validation Data Sets

This example uses the MathWorks MerchData data set. This is a small data set containing 75 images of MathWorks merchandise, belonging to five different classes (cap, cube, playing cards, screwdriver, and torch).

```
curDir = pwd;
unzip('MerchData.zip');
imds = imageDatastore('MerchData', ...
'IncludeSubfolders',true, ...
'LabelSource','foldernames');
[imdsTrain,imdsValidation] = splitEachLabel(imds,0.7,'randomized');
```

Replace Final Layers

The fully connected layer and the classification layer of the pretrained network `net` are configured for 1000 classes. These two layers `fc1000` and `ClassificationLayer_predictions` in ResNet-18 contain information on how to combine the features that the network extracts into class probabilities and predicted labels. These layers must be fine-tuned for the new classification problem. Extract all the layers, except the last two layers, from the pretrained network.

```
lgraph = layerGraph(snet)
```

```
lgraph =
  LayerGraph with properties:
```

```
     InputNames: {'data'}
    OutputNames: {'ClassificationLayer_predictions'}
         Layers: [71×1 nnet.cnn.layer.Layer]
    Connections: [78×2 table]
```

```
numClasses = numel(categories(imdsTrain.Labels))
```

```
numClasses = 5
```

```
newLearnableLayer = fullyConnectedLayer(numClasses, ...
'Name','new_fc', ...
'WeightLearnRateFactor',10, ...
'BiasLearnRateFactor',10);
lgraph = replaceLayer(lgraph,'fc1000',newLearnableLayer);
newClassLayer = classificationLayer('Name','new_classoutput');
lgraph = replaceLayer(lgraph,'ClassificationLayer_predictions',newClassLayer);
```

Train Network

The network requires input images of size 224-by-224-by-3, but the images in the image datastores have different sizes. Use an augmented image datastore to automatically resize the training images. Specify additional augmentation operations to perform on the training images, such as randomly flipping the training images along the vertical axis and randomly translating them up to 30 pixels horizontally and vertically. Data augmentation helps prevent the network from overfitting and memorizing the exact details of the training images.

```
pixelRange = [-30 30];
imageAugmenter = imageDataAugmenter( ...
'RandXReflection',true, ...
'RandXTranslation',pixelRange, ...
'RandYTranslation',pixelRange);
```

To automatically resize the validation images without performing further data augmentation, use an augmented image datastore without specifying any additional preprocessing operations.

```
augimdsTrain = augmentedImageDatastore(inputSize(1:2),imdsTrain, ...
'DataAugmentation',imageAugmenter);
augimdsValidation = augmentedImageDatastore(inputSize(1:2),imdsValidation);
```

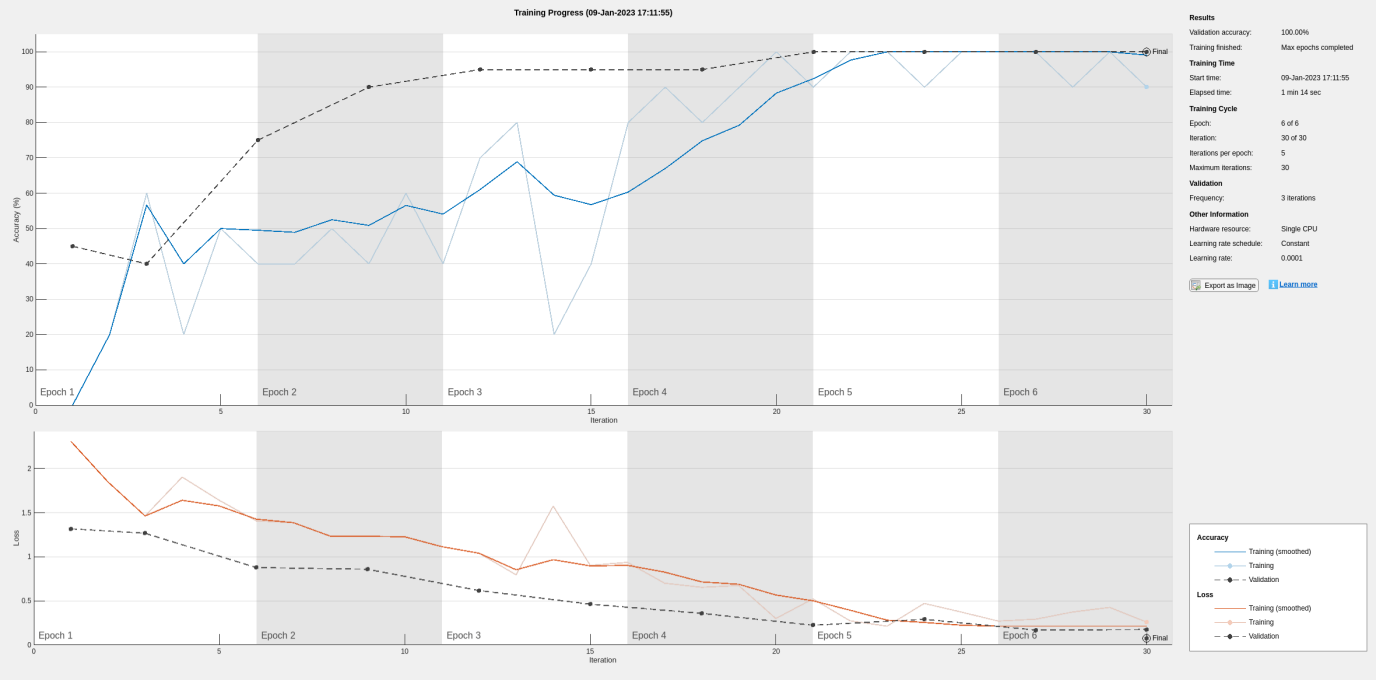
Specify the training options. For transfer learning, keep the features from the early layers of the pretrained network (the transferred layer weights). To slow down learning in the transferred layers, set the initial learning rate to a small value. Specify the mini-batch size and validation data. The software validates the network for every `ValidationFrequency` iteration during training.

```
options = trainingOptions('sgdm', ...
'MiniBatchSize',10, ...
'MaxEpochs',6, ...
'InitialLearnRate',1e-4, ...
'Shuffle','every-epoch', ...
'ValidationData',augimdsValidation, ...
'ValidationFrequency',3, ...
'Verbose',false, ...
'Plots','training-progress');
```

Train the network that consists of the transferred and new layers. By default, `trainNetwork` uses a GPU if one is available (requires Parallel Computing Toolbox™ and a supported GPU device. See “GPU Computing Requirements” (Parallel Computing Toolbox)). Otherwise, the network uses a CPU

(requires MATLAB Coder Interface for Deep learning). You can also specify the execution environment by using the `'ExecutionEnvironment'` name-value argument of `trainingOptions`.

```
netTransfer = trainNetwork(augimdsTrain,lgraph,options);
```



Retrieve Deep Learning Processor Configuration

Use the `dlhdl.ProcessorConfig` object to retrieve the deep learning processor configuration for the `zcu102_single` bitstream.

```
hPC = dlhdl.ProcessorConfig('Bitstream','zcu102_single');
```

Create Simulator Object

Create a `dlhdl.Simulator` object with ResNet-18 as the network and `hPC` as the `ProcessorConfig` object.

```
simObj = dlhdl.Simulator('Network',netTransfer,'ProcessorConfig',hPC);
```

```
### Optimizing network: Fused 'nnet.cnn.layer.BatchNormalizationLayer' into 'nnet.cnn.layer.Convo
Compiling leg: conv1>>pool1 ...
Compiling leg: conv1>>pool1 ... complete.
Compiling leg: res2a_branch2a>>res2a_branch2b ...
Compiling leg: res2a_branch2a>>res2a_branch2b ... complete.
Compiling leg: res2b_branch2a>>res2b_branch2b ...
Compiling leg: res2b_branch2a>>res2b_branch2b ... complete.
Compiling leg: res3a_branch1 ...
Compiling leg: res3a_branch1 ... complete.
Compiling leg: res3a_branch2a>>res3a_branch2b ...
Compiling leg: res3a_branch2a>>res3a_branch2b ... complete.
Compiling leg: res3b_branch2a>>res3b_branch2b ...
Compiling leg: res3b_branch2a>>res3b_branch2b ... complete.
Compiling leg: res4a_branch1 ...
```

```
Compiling leg: res4a_branch1 ... complete.
Compiling leg: res4a_branch2a>>res4a_branch2b ...
Compiling leg: res4a_branch2a>>res4a_branch2b ... complete.
Compiling leg: res4b_branch2a>>res4b_branch2b ...
Compiling leg: res4b_branch2a>>res4b_branch2b ... complete.
Compiling leg: res5a_branch1 ...
Compiling leg: res5a_branch1 ... complete.
Compiling leg: res5a_branch2a>>res5a_branch2b ...
Compiling leg: res5a_branch2a>>res5a_branch2b ... complete.
Compiling leg: res5b_branch2a>>res5b_branch2b ...
Compiling leg: res5b_branch2a>>res5b_branch2b ... complete.
Compiling leg: pool5 ...
Compiling leg: pool5 ... complete.
Compiling leg: new_fc ...
Compiling leg: new_fc ... complete.
```

Load Image for Prediction and Intermediate Layer Activation Results

Load the example image. Save it's size for future use.

```
imgFile = fullfile(pwd,'MerchData','MathWorks Cube','MathWorks cube_0.jpg');
inputImg = imresize(imread(imgFile),inputSize(1:2));
imshow(inputImg)
```

Show Activations of First Maxpool Layer

Investigate features by observing which areas in the convolution layers activate on an image. Compare that image to the corresponding areas in the original images. Each layer of a convolutional neural network consists of many 2-D arrays called *channels*. Pass the image through the network and examine the output activations of the `pool1` layer.

```
act1 = simObj.activations(single(inputImg),'pool1');
```

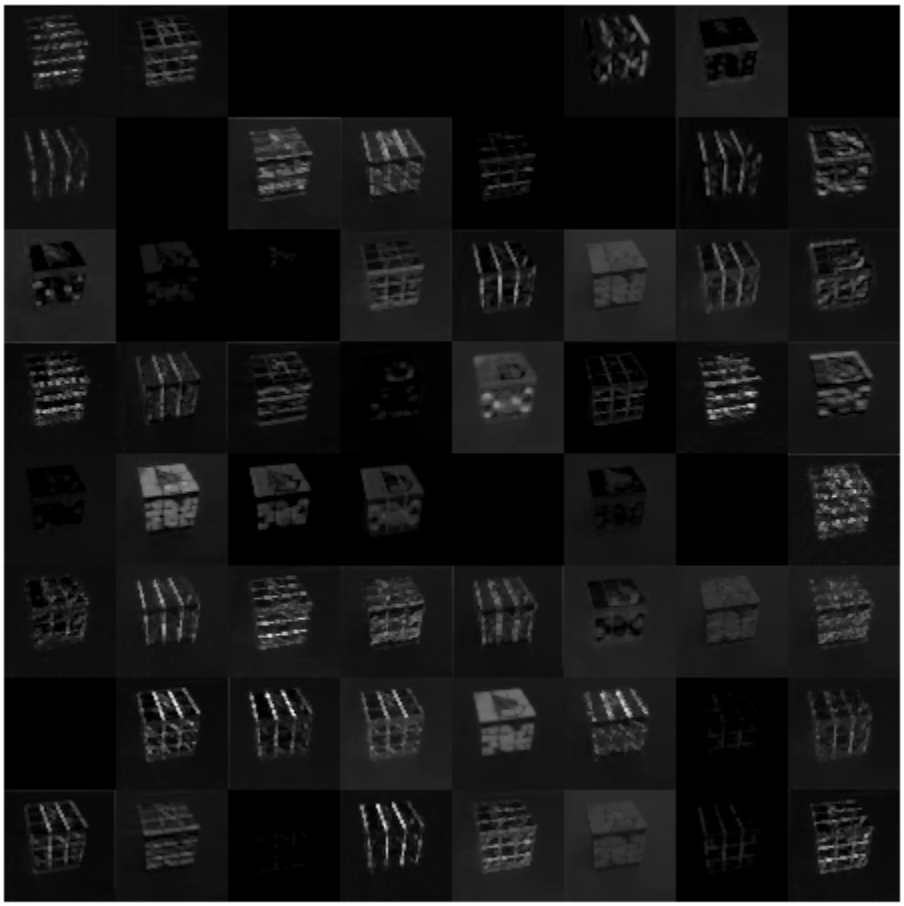
The activations are returned as a 3-D array, with the third dimension indexing the channel on the `pool1` layer. To show these activations by using the `imtile` function, reshape the array to 4-D. The

third dimension in the input to `imtile` represents the image color. Set the third dimension to have size 1 because the activations do not have color. The fourth dimension indexes the channel.

```
sz = size(act1);
act1 = reshape(act1,[sz(1) sz(2) 1 sz(3)]);
```

Display the activations. Each activation can take any value, so normalize the output by using the `mat2gray`. All activations are scaled so that the minimum activation is 0 and the maximum activation is 1. Display the 64 images on an 8-by-8 grid, one for each channel in the layer.

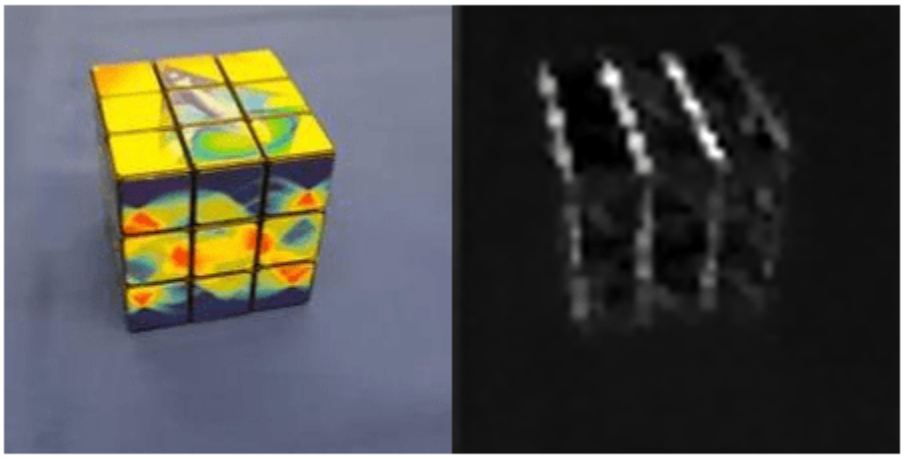
```
I = imtile(mat2gray(act1),'GridSize',[8 8]);
imshow(I)
```



Find Strongest Activation Channel

Find the strongest channels by programmatically investigating channels with large activations. Find the channel that has the largest activation by using the `max` function, resize the channel output, and display the activations.

```
[maxValue,maxValueIndex] = max(max(max(act1)));
act1chMax = act1(:,:,:,maxValueIndex);
act1chMax = mat2gray(act1chMax);
act1chMax = imresize(act1chMax,inputSize(1:2));

I = imtile({inputImg,act1chMax});
imshow(I)
```



Compare the strongest activation channel image to the original image. This channel activates on edges. It activates positively on light left/dark right edges and negatively on dark left/light right edges.

Verify Prediction Results

Verify and display the prediction results of the `dlhdl.Simulator` object by using the `predict` function.

```
prediction = simObj.predict(single(inputImg));
[val, idx] = max(prediction);
netTransfer.Layers(end).ClassNames{idx}
```

```
ans =
'MathWorks Cube'
```

Version History

Introduced in R2021b

See Also

`activations` | `dlhdl.Simulator`

Topics

“Prototype and Verify Deep Learning Networks Without Target Hardware”

hdlcoder.ReferenceDesign class

Package: `hdlcoder`

Reference design registration object that describes SoC reference design

Description

`refdesign = hdlcoder.ReferenceDesign('SynthesisTool', toolname)` creates a reference design object that you use to register a custom reference design for an SoC platform.

To specify the characteristics of your reference design, set the properties of the reference design object.

Use a reference design tool version that is compatible with the supported tool version. If you choose a different tool version, it is possible that HDL Coder is unable to create the reference design project for IP core integration.

Creation

`refdesign = hdlcoder.ReferenceDesign('SynthesisTool',toolname)` creates a reference design object that you use to register a custom reference design for an SoC platform.

Input Arguments

`toolname` — Synthesis tool name
`Xilinx Vivado` (default) | `Altera Quartus II` | `Xilinx ISE` | `Microchip Libero SoC`

Synthesis tool name, specified as a character vector.

Example: `'Altera Quartus II'`

Properties

`ReferenceDesignName` — Reference design name
`''` (default) | character vector

Reference design name, specified as a character vector. In the HDL Workflow Advisor, this name appears in the **Reference design** drop-down list.

Example: `'Default system (Vivado 2015.4)'`

`BoardName` — Board name
`''` (default) | character vector

Board associated with this reference design, specified as a character vector.

Example: `'Enclustra Mars ZX3 with PM3 base board'`

`SupportedToolVersion` — Supported tool version
`{}` (default) | cell array of character vectors

One or more tool versions that work with this reference design, specified as a cell array of character vectors.

Example: `{'2020.2'}`

Example: `{'13.7','14.0'}`

`CustomConstraints` — Design constraint file (optional)
`{}` (default) | cell array of character vectors

One or more design constraint files, specified as a cell array of character vectors. This property is optional.

Example: `{'MarsZX3_PM3.xdc'}`

Example: `{'MyDesign.qsf'}`

`CustomFiles` — Relative path to required file or folder (optional)
`{}` (default) | cell array of character vectors

One or more relative paths to files or folders that the reference design requires, specified as a cell array of character vectors. This property is optional.

Examples of required files or folders:

- Existing IP core used in the reference design.

  For example, if the IP core, *`my_ip_core`*, is in the reference design folder, set `CustomFiles` to `{'my_ip_core']`
- PS7 definition XML file.

  For example, to include a PS7 definition XML file, *`ps7_system_prj.xml`*, in a folder, *`data`*, set `CustomFiles` to `{fullfile('data', 'ps7_system_prj.xml')}`
- Folder containing existing IP cores used in the reference design. HDL Coder supports only a specific IP core folder name for each synthesis tool:
  - For Altera Qsys, IP core files must be in a folder named `ip`. Set `CustomFiles` to `{'ip'}`.
  - For Xilinx Vivado, IP core files, or a zip file containing the IP core files, must be in a folder named `ipcore`. Set `CustomFiles` to `{'ipcore'}`.
  - For Xilinx EDK, IP core files must be in a folder named `pcores`. Set `CustomFiles` to `{'pcores'}`.

Note To add IP modules to the reference design, it is recommended to create an IP repository folder that contains these IP modules, and then use the `addIPRepository` method.

Example: `{'my_ip_core'}`

Example: `{fullfile('data', 'ps7_system_prj.xml')}`

Example: `{'ip'}`

Example: `{'ipcore'}`

Example: `{'pcores'}`

`DeviceTreeName` — Linux device tree name
character vector

Specify the device tree file name. For an example that shows how to use different device tree file names when mapping the DUT ports to different AXI4-Stream channels, see “Dynamically Create Reference Design with Master Only or Slave Only AXI4-Stream Interface” (HDL Coder).

Example: `'devicetree_axistream_iio.dtb'`

`AddMATLABAXIManagerParameter` — Control visibility of Insert AXI Manager parameter
`'true'` (default) | `'false'` | logical data type

Control visibility of the **Insert AXI Manager (HDL Verifier required)** parameter in the **Set Target Reference Design** task of the HDL Workflow Advisor. By default, the property value is `'true'`, which means that the parameter is visible in the **Set Target Reference Design** task. To disable the parameter, set the property value to `'false'`.

After you enable this property, to specify whether you want the code generator to insert the AXI manager IP, use the `MATLABAXIManagerDefaultValue` property.

This property is optional.

Example: `'false'`

`MATLABAXIManagerDefaultValue` — Specify whether to insert AXI manager IP
`'off'` (default) | `'JTAG'` | `'Ethernet'` | character vector

Specify whether you want the code generator to insert the AXI manager IP. The values that you specify are the choices for the **Insert AXI Manager (HDL Verifier required)** drop-down in the **Set Target Reference Design** task of the HDL Workflow Advisor. To specify insertion of the AXI manager IP automatically, before you set this property, set the `AddMATLABAXIManagerParameter` property to `'true'`.

This property is optional. Set this property to one of these values.

- `'off'` — Disable insertion of the AXI manager IP.
- `'JTAG'` — Enable AXI manager IP insertion for the JTAG connection. This value inserts the AXI Manager IP into your reference design.
- `'Ethernet'` — Enable AXI manager IP insertion for the Ethernet connection. This value inserts the UDP AXI Manager IP into your reference design.

Example: `'JTAG'`

`IPCacheZipFile` — IP cache file to include in the project
`''` (default) | `'ipcache.zip'` | character vector

Specify the IP cache zip file to include in your project. When you run the `IP Core Generation` workflow in the HDL Workflow Advisor, the code generator extracts this file in the **Create Project** task. The **Build FPGA Bitstream** task reuses the IP cache, which accelerates reference design synthesis.

This property is optional.

Example: `'ipcache.zip'`

`ReportTimingFailure` — Report timing failures as warnings or errors
`'hdlcoder.ReportTiming.Warning'` (default) | `'hdlcoder.ReportTiming.Error'`

Specify whether you want the code generator to report timing failures in the **Build FPGA Bitstream** task as warnings or errors. When you run the `IP Core Generation` workflow in the HDL Workflow Advisor, by default, the code generator reports any timing failures as error. If you have implemented the custom logic to resolve timing failures, you can specify these failures to be reported as warning instead of error. To learn more, see "Resolve Timing Failures in IP Core Generation and Simulink Real-Time FPGA I/O Workflows" (HDL Coder).

This property is optional.

Example: `'hdlcoder.ReportTiming.Warning'`

`HasProcessingSystem` — Specify if reference design has existing Processing System (PS)
`true` (default) | `false` | logical data type

Specify if the reference design has an existing PS.

Example: `'false'`

`GenerateIPCoreDeviceTreeNodes` — Enable generation of device tree nodes for HDL Coder IP core
`false` (default) | `true` | logical data type

Enable generation of device tree nodes for an HDL Coder generated IP core, and then insert the nodes into the device tree. To enable the generation of device tree nodes for the IP core, `HasProcessingSystem` must be set to `true`.

Do not enable this property if you do not need any additional device tree nodes to be inserted into the registered device tree for the generated IP core.

Example: `'true'`

`ResourcesUsed` — Board resources used by reference design
structure

Board resources used by reference design, returned as a structure with the fields:

`LogicElements` — Reference design resources utilized by FPGA lookup tables (LUTs)
0 (default)

Reference design resources utilized by FPGA lookup tables (LUTs), specified as a number.

Example: `hRD.ResourcesUsed.LogicElements = 100`

`DSP` — Reference design resources utilized by FPGA DSP slices
0 (default)

Reference design resources utilized by FPGA DSP slices, specified as a number.

Example: `hRD.ResourcesUsed.DSP = 3`

`RAM` — Reference design resources utilized by FPGA board RAM resources
0 (default)

Reference design resources utilized by FPGA board RAM resources, specified as a number.

Example: `hRD.ResourcesUsed.RAM = 32000`

Methods

Public Methods

CallbackCustomProgrammingMethod	Function handle for custom callback function that gets executed during Program Target Device task in the Workflow Advisor
CustomizeReferenceDesignFcn	Function handle for callback function that gets executed before Set Target Interface task in the HDL Workflow Advisor
EmbeddedCoderSupportPackage	Specify whether to use an Embedded Coder support package
PostBuildBitstreamFcn	Function handle for callback function that gets executed after Build FPGA Bitstream task in the HDL Workflow Advisor
PostCreateProjectFcn	Function handle for callback function that gets executed after Create Project task in the HDL Workflow Advisor
PostSWInterfaceFcn	Function handle for custom callback function that gets executed after Generate Software Interface task in the HDL Workflow Advisor
PostTargetInterfaceFcn	Function handle for callback function that gets executed after Set Target Interface task in the HDL Workflow Advisor
PostTargetReferenceDesignFcn	Function handle for callback function that gets executed after Set Target Reference Design task in the HDL Workflow Advisor
addAXI4MasterInterface	Add and define AXI4 Master interface
addAXI4SlaveInterface	Add and define AXI4 slave interface
addAXI4StreamInterface	Add AXI4-Stream interface
addAXI4StreamVideoInterface	Add AXI4-Stream Video interface
addClockInterface	Add clock and reset interface
addCustomEDKDesign	Specify Xilinx EDK MHS project file
addCustomQsysDesign	Specify Altera Qsys project file
addCustomVivadoDesign	Specify Xilinx Vivado exported block design Tcl file
addCustomLiberoDesign	Specify Microchip Libero SoC exported block design Tcl file
addDeviceTree	Add device tree for reference design object
addDeviceTreeIncludeDirectory	Specify the path of an include file to compile the device tree against
addIPRepository	Include IP modules from your IP repository folder in your custom reference design
addInternalIOInterface	Add and define internal IO interface between generated IP core and existing IP cores
addParameter	Add and define custom parameters for your reference design
validateReferenceDesign	Check property values in reference design object

Version History

Introduced in R2015a

See Also

`hdlcoder.Board`

Topics

“Define Custom Board and Reference Design for Zynq Workflow” (HDL Coder)

“Define Custom Board and Reference Design for Intel SoC Workflow” (HDL Coder)
“Define Custom Board and Reference Design for Microchip Workflow” (HDL Coder)
“Register a Custom Board” (HDL Coder)
“Register a Custom Reference Design” (HDL Coder)
“Define Custom Parameters and Callback Functions for Custom Reference Design” (HDL Coder)
“Board and Reference Design Registration System” (HDL Coder)

registerDeepLearningMemoryAddressSpace

Class: `hdlcoder.ReferenceDesign`
Package: `hdlcoder`

Add memory address space to reference design

Syntax

```
registerDeepLearningMemoryAddressSpace(baseAddr,addrRange)
```

Description

`registerDeepLearningMemoryAddressSpace(baseAddr,addrRange)` registers memory address space accessible by the deep learning processor IP core to your reference design. The deep learning processor IP core uses this registered memory space to store the inputs to the deep learning network, network weights, and intermediate computations performed by the deep learning processor.

Input Arguments

`baseAddr` — Base address of the deep learning memory address space

0x0 (default) | uint32

Base address of the deep learning memory address space, in bytes as `uint32`.

Example: 0x80000000

`addrRange` — Address range of the deep learning memory address space

0x0 (default) | uint32

Address range of the deep learning memory address space, in bytes as `uint32`.

Example: 0x40000000

Requirements

The registered memory address space must be accessible by the AXI4 Master interfaces in your reference design.

Tips

- A minimum of 32 MB of memory address space is required to run any deep learning network. To run a variety of deep learning networks, it is recommended to register a minimum of 512 MB of memory address space.
- The `compile` method of the `dlhdl.Workflow` object generates the memory address space for a deep learning network. See `compile`. Determine the memory address space for your deep learning network by using the `compile` method of the `dlhdl.Workflow` object.

Version History

Introduced in R2021b

See Also

`registerDeepLearningTargetInterface` | `validateReferenceDesignForDeepLearning`

registerDeepLearningTargetInterface

Class: `hdlcoder.ReferenceDesign`
Package: `hdlcoder`

Add and register a target interface

Syntax

```
registerDeepLearningTargetInterface(interfaceType)
```

Description

`registerDeepLearningTargetInterface(interfaceType)` registers a target interface to an `hdlcoder.ReferenceDesign` object. Use the registered target interface to interact with the generated deep learning processor IP core by using MATLAB.

Input Arguments

`interfaceType` — Target interface type

'JTAG' (default) | 'JTAG' | string | character vector

Target interface type, specified as a string or character vector.

Example: 'JTAG'

Limitations

To register a JTAG target interface to your reference design, your reference design must meet these requirements.

- The reference design must include JTAG MATLAB as AXI Master IP.
- The JTAG MATLAB as AXI Master IP must be one of the masters to the AXI4-Slave interface in the reference design.
- The JTAG MATLAB as AXI Master IP must have access to the same memory as the AXI4-Master interfaces in the reference design.

Version History

Introduced in R2021b

See Also

`registerDeepLearningMemoryAddressSpace` | `validateReferenceDesignForDeepLearning`

validateReferenceDesignForDeepLearning

Class: `hdlcoder.ReferenceDesign`
Package: `hdlcoder`

Checks property values in reference design object

Syntax

```
validateReferenceDesignForDeepLearning
```

Description

`validateReferenceDesignForDeepLearning` checks that the `hdlcoder.ReferenceDesign` object is compatible with the deep learning processor IP core generation workflow.

Reference Design Requirements

The `validateReferenceDesign` method checks your `hdlcoder.ReferenceDesign` object for these requirements:

- Your reference design must have an AXI4 Slave interface. You use this interface to access registers in the deep learning processor IP core.
- Your reference design must have three AXI4 Master interfaces with `InterfaceID` values of `"AXI4 Master Activation Data"`, `"AXI4 Master Weight Data"`, and `"AXI4 Master Debug"`. The deep learning processor IP core uses these AXI4 Master interfaces to access memory for network storage and intermediate calculations.
- Your reference design must have a registered memory address space. Use the `registerDeepLearningMemoryAddressSpace` to register a memory address space. To run a variety of deep learning networks on your deep learning processor IP core, it is recommended to register a minimum of 512 MB of memory address space.

Tips

Register a target interface for your reference design by using the `registerDeepLearningTargetInterface` method. You can use the registered target interface to run your deep learning network on the generated deep learning processor IP core by using MATLAB and a `dlhdl.Workflow` object.

Version History

Introduced in R2021b

See Also

`registerDeepLearningMemoryAddressSpace` | `registerDeepLearningTargetInterface`