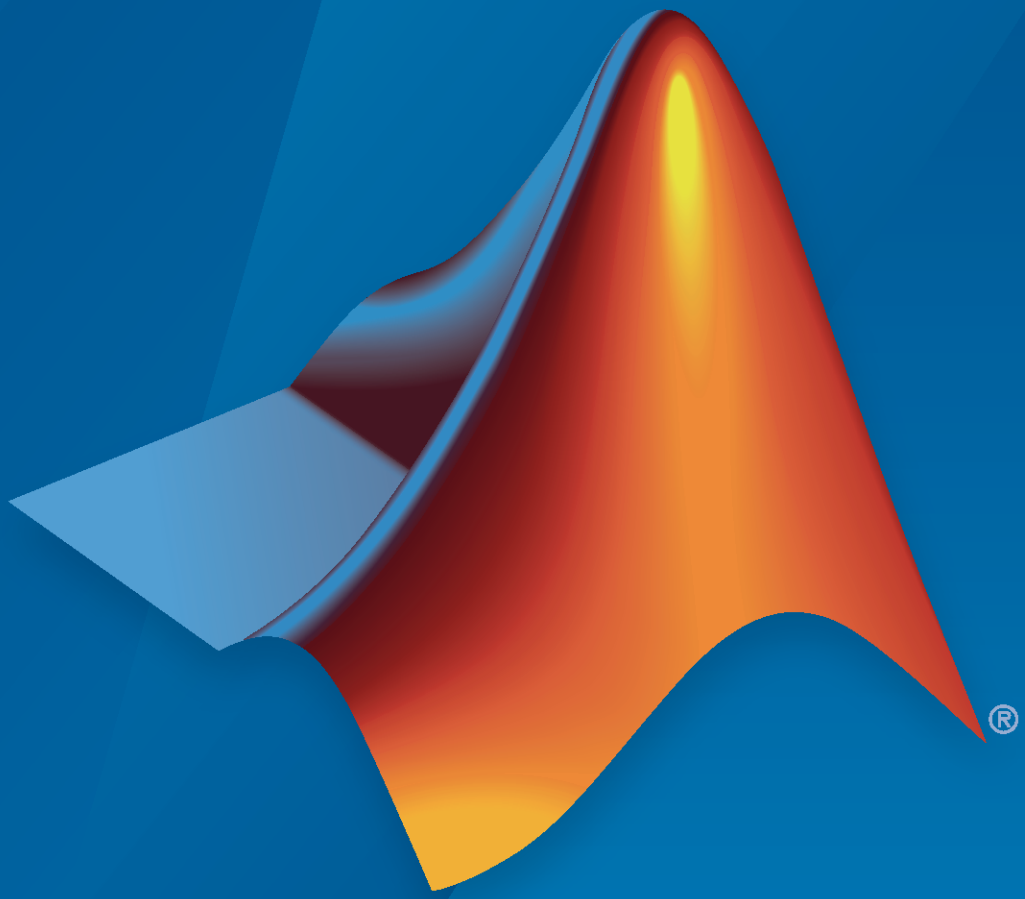
Deep Learning HDL Toolbox™

Reference



MATLAB®

R2020b

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

Revision History

September 2020	Online only	New for Version 1.0 (R2020b)

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

`dlhdl.Workflow` creates a workflow configuration object for you to specify the workflow to deploy your trained series network.

`dlhdl.Workflow (Name,Value)` creates a workflow configuration object for you to specify the workflow to deploy your trained deep learning network, with additional options specified by one or more name-value pair arguments.

Properties

`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.

Example: `'Bitstream','myfile.bit'` specifies that you want to deploy the trained network using your custom bitstream file `myfile.bit` which is in your current working directory.

Example: `'Bitstream','C:\myfolder\myfile.bit'` specifies that you want to deploy the trained network using your custom bitstream file `myfile.bit` that is located in the folder `'C:\myFolder'`.

`Network` — Name of the pretrained deep learning network that you want to import or the name of the quantized network object

' ' (default)

Deep learning network name specified as a variable

Example: `'network'`, `snet` creates a workflow object for the saved pretrained network, `snet`. To specify `snet`, you can import any of the existing supported pretrained networks or use transfer learning to adapt the network to your problem. For information on supported networks, see "Supported Pretrained Networks".

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 using the `dlquantizer` class. For information on supported networks, see "Supported Pretrained Networks".

Assign `VGG-19` to `snet`:

```
snet = vgg19;
```

`'ProcessorConfig'` — Optional arguments when estimating workflow object without a bitstream

`ProcessorConfig` (default) | `dlhdl.ProcessorConfig`

Custom processor configuration object specified as `dlhdl.ProcessorConfig` object

Example: `'ProcessorConfig',hPC`

```
hPC = dlhdl.ProcessorConfig()
hW = dlhdl.Workflow('network', alexnet,'ProcessorConfig',hPC);
```

`'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);
```

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 Using Custom Bitstream

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

Create Workflow Object with Quantized Network Object

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

See Also

Functions
`activations` | `compile` | `deploy` | `estimate` | `predict`

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

Topics
“Prototype Deep Learning Networks on FPGA and SoCs Workflow”
“Quantization of Deep Neural Networks”

Introduced in R2020b

activations

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

Retrieve intermediate layer results for deployed deep learning network

Syntax

```
activations(imIn,layername)
activations(imIn,layername, Name,Value)
```

Description

`activations(imIn,layername)` returns intermediate layer activation data results for the image data in `imIn`, 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`.

`activations(imIn,layername, Name,Value)`returns intermediate layer activation data results for the image data in `imIn`, 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`.

Examples

Retrieve Layer Activation Results

Retrieve the activation results of the `LogoNet` `maxpool_3` layer for a given input image.

Create a file in your current working directory called `getLogoNetwork.m`. Enter these lines into the file:

```
function net = getLogoNetwork()
    data = getLogoData();
    net  = data.convnet;
end

function data = getLogoData()
    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');
end


snet = getLogoNetwork();
hT = dlhdl.Target('Xilinx');
hW = dlhdl.Workflow('Network','snet','Bitstream','zcu102_single','target',hT);
image = imread('heineken.png');
inputImg = imresize(image, [227, 227]);
imIn = single(inputImg);
results = hW.activations(imIn,'maxpool_3','Profiler','on');
```

The result of the code execution is a 25-by-25-by-384 matrix for `results` and

```
Deep Learning Processor Profiler Performance Results
```

```
                    LastLayerLatency(cycles)    LastLayerLatency(seconds)       FramesNum      Tota
                                 -------------               -------------                ---------       --
Network                           32497812                     0.14772                     1          324
    conv_module                   32497812                     0.14772
        conv_1                     6953894                     0.03161
        maxpool_1                  3305128                     0.01502
        conv_2                    10397281                     0.04726
        maxpool_2                  1207938                     0.00549
        conv_3                     9267269                     0.04212
        maxpool_3                  1366383                     0.00621
 * The clock frequency of the DL processor is: 220MHz
```

Input Arguments

`imIn` — Input resized image to deep learning network

single

Input image resized to match the input image layer image size of the deep learning network.

Example: To read an input image, resize it to 227x227, and convert it to single use:

Use this image to run the code:


```
image = imread('heineken.png');
inputImg = imresize(image, [227, 227]);
imIn = single(inputImg);
```

Example: `imIn`

`layername` — Name of layer in deployed deep learning network
'' (default) | character vector

Name of the layer in the deployed deep learning network whose results are retrieved for the image specified in `imIn`.

The layer has to be of the type `Convolution`, `Fully Connected`, `Max Pooling`, `ReLU`, or `Dropout`. `Convolution` and `Fully Connected` layers are allowed as long as they are not followed by a `ReLU` layer.

Example: 'maxpool_3'

Name-Value Pair Arguments

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`.

`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'`

See Also

`compile` | `deploy` | `estimate` | `predict`

Introduced in R2020b

compile

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

Compile workflow object

Syntax

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

Description

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

`compile(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.

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.

```
snet = alexnet;
hT = dlhdl.Target('Xilinx');
hW = dlhdl.Workflow('network', snet, 'Bitstream', 'zcu102_single','Target',hT);
hW.compile('InputFrameNumberLimit',30);
```

2. The result of the code execution is:

```
        offset_name          offset_address     allocated_space
    _____________________    ______________    _________________

    "InputDataOffset"          "0x00000000"     "24.0 MB"
    "OutputResultOffset"       "0x01800000"     "4.0 MB"
    "SystemBufferOffset"       "0x01c00000"     "28.0 MB"
    "InstructionDataOffset"    "0x03800000"     "4.0 MB"
    "ConvWeightDataOffset"     "0x03c00000"     "16.0 MB"
    "FCWeightDataOffset"       "0x04c00000"     "224.0 MB"
    "EndOffset"                "0x12c00000"     "Total: 300.0 MB"
```

Input Arguments

Name-Value Pair Arguments

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`.

`InputFrameNumberLimit` — Maximum input frame number limit
integer

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

Example: `'InputFrameNumberLimit',30`

See Also

`deploy` | `estimate` | `predict`

Topics
“Compiler Output”

Introduced in R2020b

deploy

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

Deploy the specified neural network to the target FPGA board

Syntax

```
deploy
```

Description

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

Examples

Deploy LogoNet to Intel Arria 10 SoC Development Kit

Note Before you run the `deploy` function, make sure that your host computer is connected to the Intel Arria 10 SoC board. For more information, see “Check Host Computer Connection to FPGA Boards”

Deploy VGG-19 to the Intel Arria 10 SoC development kit that has `single` data types.

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

### Programming FPGA bitstream using JTAG ...
### Programming FPGA bitstream has completed successfully.

tableOut =
          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"

### Loading weights to FC Processor.
### 4% finished, current time is 14-Jun-2020 18:31:07.
### 8% finished, current time is 14-Jun-2020 18:31:32.
### 12% finished, current time is 14-Jun-2020 18:31:58.
### 16% finished, current time is 14-Jun-2020 18:32:23.
### 20% finished, current time is 14-Jun-2020 18:32:48.
### 24% finished, current time is 14-Jun-2020 18:33:13.
### 28% finished, current time is 14-Jun-2020 18:33:39.
### 32% finished, current time is 14-Jun-2020 18:34:04.
### 36% finished, current time is 14-Jun-2020 18:34:30.
### 40% finished, current time is 14-Jun-2020 18:34:56.
```

```
### 44% finished, current time is 14-Jun-2020 18:35:21.
### 48% finished, current time is 14-Jun-2020 18:35:46.
### 52% finished, current time is 14-Jun-2020 18:36:11.
### 56% finished, current time is 14-Jun-2020 18:36:36.
### 60% finished, current time is 14-Jun-2020 18:37:02.
### 64% finished, current time is 14-Jun-2020 18:37:27.
### 68% finished, current time is 14-Jun-2020 18:37:52.
### 72% finished, current time is 14-Jun-2020 18:38:17.
### 76% finished, current time is 14-Jun-2020 18:38:43.
### 80% finished, current time is 14-Jun-2020 18:39:08.
### 84% finished, current time is 14-Jun-2020 18:39:33.
### 88% finished, current time is 14-Jun-2020 18:39:58.
### 92% finished, current time is 14-Jun-2020 18:40:23.
### 96% finished, current time is 14-Jun-2020 18:40:48.
### FC Weights loaded. Current time is 14-Jun-2020 18:41:06
```

The `deploy` function starts programming the FPGA device, displays progress messages, and the time it takes to deploy the network.

Deploy Quantized LogoNet to Xilinx ZCU102 Development Kit

Create a file in your current working directory called `getLogoNetwork.m`. Enter these lines into the file:

```
function net = getLogoNetwork()
    data = getLogoData();
    net  = data.convnet;
end

function data = getLogoData()
    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');
end
```

Use this image to run the code:

To quantize the network, you need the products listed under FPGA in “Quantization Workflow Prerequisites”.

```
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',dlquantizeObj,'Bitstream','zcu102_int8','Target',hTarget);
hW.deploy

### Programming FPGA Bitstream using Ethernet...
Downloading target FPGA device configuration over Ethernet to SD card ...
# Copied /tmp/hdlcoder_rd to /mnt/hdlcoder_rd
# Copying Bitstream hdlcoder_system.bit to /mnt/hdlcoder_rd
# Set Bitstream to hdlcoder_rd/hdlcoder_system.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'

Downloading target FPGA device configuration over Ethernet to SD card done. The system will now reboot for persistent changes to take ef


System is rebooting . . . . . .
### Programming the FPGA bitstream has been completed successfully.
          offset_name          offset_address     allocated_space
    _______________________    ______________    _________________

    "InputDataOffset"           "0x00000000"     "48.0 MB"
    "OutputResultOffset"        "0x03000000"     "4.0 MB"
    "SystemBufferOffset"        "0x03400000"     "60.0 MB"
    "InstructionDataOffset"     "0x07000000"     "8.0 MB"
    "ConvWeightDataOffset"      "0x07800000"     "8.0 MB"
    "FCWeightDataOffset"        "0x08000000"     "12.0 MB"
    "EndOffset"                 "0x08c00000"     "Total: 140.0 MB"

### Loading weights to FC Processor.
### FC Weights loaded. Current time is 12-Jun-2020 13:17:56
```

See Also

`calibrate` | `compile` | `dlquantizationOptions` | `dlquantizer` | `estimate` | `predict` | `validate`

Introduced in R2020b

estimate

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

Estimate performance of specified deep learning network and bitstream for target device board

Syntax

```
estimate(estimateparameter)
```

Description

`estimate(estimateparameter)` estimates and predicts the performance of the deep learning network that you specified as part of the `dlhdl.Workflow` class for deployment to the specified target device boards.

Examples

Estimate Performance of VGG-19 Running On ZCU102 Bitstream

Determine the average number of cycles executed per input image.

Create an example object by using `dlhdl.Workflow` and then use the `estimate` function to estimate the `Performance`.

```
snet = vgg19;
hW = dlhdl.Workflow('network', snet, 'Bitstream', 'zcu102_single');
hW.estimate('Performance')
```

Once the code is executed the result is

```
             Deep Learning Processor Estimator Performance Results

                  LastLayerLatency(cycles)   LastLayerLatency(seconds)       FramesNum      Total Latency     Frames/s
                       -------------              -------------               ---------        ---------       ---------
Network                  204460802                 0.92937                      1          204460802             1.1
    conv_module          158812469                 0.72187
        conv1_1            2022004                 0.00919
        conv1_2           15855549                 0.07207
        pool1              2334753                 0.01061
        conv2_1            7536365                 0.03426
        conv2_2           14837392                 0.06744
        pool2              1446960                 0.00658
        conv3_1            7950445                 0.03614
        conv3_2           14365933                 0.06530
        conv3_3           14365933                 0.06530
        conv3_4           14365933                 0.06530
        pool3               930145                 0.00423
        conv4_1            7073684                 0.03215
        conv4_2           13761300                 0.06255
        conv4_3           13761300                 0.06255
        conv4_4           13761300                 0.06255
        pool4               572644                 0.00260
        conv5_1            3432645                 0.01560
        conv5_2            3432645                 0.01560
        conv5_3            3432645                 0.01560
        conv5_4            3432645                 0.01560
        pool5               140249                 0.00064
    fc_module            45648333                 0.20749
```

```
        fc6               37940907                 0.17246
        fc7                6194731                 0.02816
        fc8                1512695                 0.00688
 * The clock frequency of the DL processor is: 220MHz
```

Estimate Performance of Quantized LogoNet Running On ZCU102 Bitstream

Determine the average number of cycles executed per input image.

Create an example object by using `dlhdl.Workflow` and then use the `estimate` function to estimate the `Performance`.

Create a file in your current working directory called `getLogoNetwork.m`. Enter these lines into the file:

```
function net = getLogoNetwork()
    data = getLogoData();
    net  = data.convnet;
end

function data = getLogoData()
    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');
end
```

Use this image to run the code:

To quantize the network, you need the products listed under FPGA in “Quantization Workflow Prerequisites”.

```
snet = getLogoNetwork();
dlquantObj = dlquantizer(snet,'ExecutionEnvironment','FPGA');
Image = imageDatastore('heineken.png','Labels','Heineken');
dlquantObj.calibrate(Image);
hW = dlhdl.Workflow('Network',dlquantObj,'Bitstream','zcu102_int8');
hW.estimate('Performance');
```

Once the code is executed the result is :

```
            Deep Learning Processor Estimator Performance Results


                   LastLayerLatency(cycles)   LastLayerLatency(seconds)       FramesNum      Total Latency     Frames/s
                        -------------              -------------               ---------        ---------       ---------
Network                  13923758                 0.04641                       1           13923758            21.5
    conv_module          12737303                 0.04246
        conv_1            3327693                 0.01109
        maxpool_1         1876824                 0.00626
        conv_2            2936929                 0.00979
        maxpool_2          723536                 0.00241
        conv_3            2456212                 0.00819
        maxpool_3          882032                 0.00294
        conv_4             520052                 0.00173
        maxpool_4           14025                 0.00005
    fc_module             1186455                 0.00395
        fc_1               708503                 0.00236
        fc_2               453111                 0.00151
        fc_3                24841                 0.00008
 * The clock frequency of the DL processor is: 300MHz
```

Estimate Performance of Deep Learning Network by Using Custom Processor Configuration

Estimate the throughput and initial latency for a given trained deep learning network running on a custom processor configuration.

1. Create a custom processor configuration object of class `dlhdl.ProcessorConfig`.
2. Create an object of class `workflow` by using the `dlhdl.Workflow` class.
3. Call the `estimate` function for the workflow object.
4. For example:

```
hPC = dlhdl.ProcessorConfig;
snet = vgg19;
hW = dlhdl.Workflow('Network', snet, 'ProcessorConfig',hPC);
result = hW.estimate('Performance');
```

The result of the estimation is:

```
            Deep Learning Processor Estimator Performance Results

                   LastLayerLatency(cycles)   LastLayerLatency(seconds)       FramesNum      Total Latency     Frames/s
                        -------------              -------------               ---------        ---------       ---------
Network                  202770372                 1.01385                       1           202770372            1.0
    conv_module          158812469                 0.79406
        conv1_1            2022004                 0.01011
        conv1_2           15855549                 0.07928
        pool1              2334753                 0.01167
        conv2_1            7536365                 0.03768
        conv2_2           14837392                 0.07419
        pool2              1446960                 0.00723
        conv3_1            7950445                 0.03975
        conv3_2           14365933                 0.07183
        conv3_3           14365933                 0.07183
        conv3_4           14365933                 0.07183
        pool3               930145                 0.00465
        conv4_1            7073684                 0.03537
        conv4_2           13761300                 0.06881
        conv4_3           13761300                 0.06881
        conv4_4           13761300                 0.06881
```

```
        pool4               572644                  0.00286
        conv5_1            3432645                  0.01716
        conv5_2            3432645                  0.01716
        conv5_3            3432645                  0.01716
        conv5_4            3432645                  0.01716
        pool5               140249                  0.00070
    fc_module            43957903                  0.21979
        fc6              36535923                  0.18268
        fc7               5965299                  0.02983
        fc8               1456681                  0.00728
* The clock frequency of the DL processor is: 200MHz
```

Input Arguments

`estimateparameter` — Parameter for estimation

'Performance'

Parameter for estimation, specified as a character vector. If you specify `Performance`, then the function estimates the performance for `FramesPerSecond`, `LatencyInCycles`, and `Time(s)` in a table format.

See Also

`calibrate` | `compile` | `deploy` | `dlquantizationOptions` | `dlquantizer` | `predict` | `validate`

Introduced in R2020b

predict

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

Run inference on deployed network and profile speed of neural network deployed on specified target device

Syntax

```
predict(imds)
predict(imds, Name,Value)
```

Description

`predict(imds)` predicts responses for the image data in `imds` by using the deep learning network that you specified in the `dlhdl.Workflow` class for deployment on the specified target board and returns the results.

`predict(imds, Name,Value)` predicts responses for the image data in `imds` by using the deep learning network that you specified by using the `dlhdl.Workflow` class for deployment on the specified target boards and returns the results, with one or more arguments specified by optional name-value pair arguments.

Examples

Predict Outcome and Profile Results

Note Before you run the `predict` function, make sure that your host computer is connected to the target device board. For more information, see “Configure Board-Specific Setup Information” .

Use this image to run the code:

```
% Save the pretrained SeriesNetwork object
snet = vgg19;

% Create a Target object and define the interface to the target board
hTarget = dlhdl.Target('Intel');

% Create a workflow object for the SeriesNetwork and using the FPFA bitstream
hW = dlhdl.Workflow('Network', snet, 'Bitstream', 'arria10soc_single','Target',hTarget);

% Load input images and resize them according to the network specifications
image = imread('zebra.jpeg');
inputImg = imresize(image, [224, 224]);
imshow(inputImg);
imIn = single(inputImg);
% Deploy the workflow object
hW.deploy;
% Predict the outcome and optionally profile the results to measure performance.
[prediction, speed] = hW.predict(imIn,'Profile','on');
[val, idx] = max(prediction);
snet.Layers(end).ClassNames{idx}


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


             Deep Learning Processor Profiler Performance Results

                     LastLayerLatency(cycles)   LastLayerLatency(seconds)       FramesNum      Total Latency     Frames/s
                     ------------------------   -------------------------       ---------      -------------     ---------
Network                          166206640                     1.10804                1          166206873             0.9
    conv_module                  156100737                     1.04067
        conv1_1                    2174602                     0.01450
        conv1_2                   15580687                     0.10387
        pool1                      1976185                     0.01317
        conv2_1                    7534356                     0.05023
        conv2_2                   14623885                     0.09749
        pool2                      1171628                     0.00781
        conv3_1                    7540868                     0.05027
        conv3_2                   14093791                     0.09396
        conv3_3                   14093717                     0.09396
        conv3_4                   14094381                     0.09396
        pool3                       766669                     0.00511
        conv4_1                    6999620                     0.04666
        conv4_2                   13725380                     0.09150
        conv4_3                   13724671                     0.09150
        conv4_4                   13725125                     0.09150
        pool4                       465360                     0.00310
        conv5_1                    3424060                     0.02283
        conv5_2                    3423759                     0.02283
        conv5_3                    3424758                     0.02283
        conv5_4                    3424461                     0.02283
        pool5                       113010                     0.00075
    fc_module                     10105903                     0.06737
        fc6                        8397997                     0.05599
        fc7                        1370215                     0.00913
        fc8                         337689                     0.00225
 * The clock frequency of the DL processor is: 150MHz


ans =

    'zebra'
```

Obtain Prediction Results for Quantized LogoNet Network

Note Before you run the `predict` function, make sure that your host computer is connected to the target device board. For more information, see “Configure Board-Specific Setup Information” .

Create a file in your current working directory called `getLogoNetwork.m`. Enter these lines into the file:

```
function net = getLogoNetwork()
    data = getLogoData();
    net  = data.convnet;
end

function data = getLogoData()
    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');
end
```

Use this image to run the code:

To quantize the network, you need the products listed under FPGA in “Quantization Workflow Prerequisites”.

```
% Save the pretrained SeriesNetwork object
snet = getLogoNetwork();

% Create a Target object and define the interface to the target board
hTarget = dlhdl.Target('Xilinx','Interface','Ethernet');

% Create a Quantized Network Object

dlquantObj = dlquantizer(snet,'ExecutionEnvironment','FPGA');
Image = imageDatastore('heineken.png','Labels','Heineken');
dlquantObj.calibrate(Image);

% Create a workflow object for the SeriesNetwork and using the FPFA bitstream
hW = dlhdl.Workflow('Network', dlquantObj, 'Bitstream', 'zcu102_int8','Target',hTarget);

% Load input images and resize them according to the network specifications
image = imread('heineken.png');
inputImg = imresize(image, [227, 227]);
imshow(inputImg);
imIn = single(inputImg);
% Deploy the workflow object
hW.deploy;
```

```
% Predict the outcome and optionally profile the results to measure performance.
[prediction, speed] = hW.predict(imIn,'Profile','on');
[val, idx] = max(prediction);
snet.Layers(end).ClassNames{idx}

### Loading weights to FC Processor.
### FC Weights loaded. Current time is 12-Jun-2020 16:55:34
### FPGA bitstream programming has been skipped as the same bitstream is already loaded on the target FPGA.
### Deep learning network programming has been skipped as the same network is already loaded on the target FPGA.
### Finished writing input activations.
### Running single input activations.


              Deep Learning Processor Profiler Performance Results

                     LastLayerLatency(cycles)   LastLayerLatency(seconds)       FramesNum      Total Latency     Frames/s
                     ------------             ------------               ---------        ---------       ---------
Network                   13604105                 0.04535                     1           13604146            22.1
    conv_module           12033763                 0.04011
        conv_1             3339984                 0.01113
        maxpool_1          1490805                 0.00497
        conv_2             2866483                 0.00955
        maxpool_2           574102                 0.00191
        conv_3             2432474                 0.00811
        maxpool_3           700552                 0.00234
        conv_4              617505                 0.00206
        maxpool_4            11951                 0.00004
    fc_module              1570342                 0.00523
        fc_1                937715                 0.00313
        fc_2                599341                 0.00200
        fc_3                 33284                 0.00011
 * The clock frequency of the DL processor is: 300MHz
```

Input Arguments

`imds` — Input resized image to deep learning network

single

Input image resized to match the image input layer size of the deep learning network.

Example: To read an input image, resize it to 227x227, and convert it to single use:

Use this image to run the code:


```
image = imread('heineken.png');
inputImg = imresize(image, [227, 227]);
imIn = single(inputImg)
```

Example: `imIn`

Name-Value Pair Arguments

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`.

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.

Example: `'Profile', 'On'`

See Also

`calibrate` | `compile` | `deploy` | `dlquantizationOptions` | `dlquantizer` | `estimate` | `validate`

Topics

“Profile Inference Run”
“Profile Network for Performance Improvement”

Introduced in R2020b

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'`

Target device vendor name, specified as a character vector.

Example: `'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'`

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.

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
```

See Also

Functions
`release` | `validateConnection`

Objects
`dlhdl.Workflow`

Introduced in R2020b

release

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

Release the connection to the target device

Syntax

```
release
```

Description

`release` releases the connection to the target board.

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.

```
snet = alexnet;
hW = dlhdl.Workflow('Network',snet,'BitstreamName','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, [227, 227]);
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;
```

See Also

`validateConnection`

Introduced in R2020b

validateConnection

Validate SSH connection and deployed bitstream

Syntax

`validateConnection`

Description

`validateConnection`:

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.

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,'BitstreamName','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.

```
snet = alexnet;
hW = dlhdl.Workflow('Network',snet,'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
```

See Also

`release`

Introduced in R2020b

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.

Properties

`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`

`TargetFrequency` — Target frequency in MHz

100 (default) | integer

Specify the target board frequency in MHz.

Example: `180`

`TargetPlatform` — Name of the target board

'Xilinx Zynq Ultrascale+ MPSoC ZCU 102 Evaluation Kit' (default) | character vector

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

Example: `'Xilinx Zynq ZC706 evaluation kit'`

Examples

Create a `ProcessorConfig` Object

```
hPC = dlhdl.ProcessorConfig
hPC =

                    Processing Module "conv"
                            ConvThreadNumber: 16
                             InputMemorySize: [227  227    3]
                            OutputMemorySize: [227  227    3]
                            FeatureSizeLimit: 1024

                      Processing Module "fc"
                              FCThreadNumber: 4
```

```
                InputMemorySize: 25088
               OutputMemorySize: 4096

         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 the `ProcessorConfig` Object

```
hPC.TargetPlatform = 'Xilinx Zynq ZC706 evaluation kit';
>> hPC.SynthesisTool = 'Xilinx Vivado';
>> hPC.TargetFrequency = 180;
```

The result of the properties modification is:

```
hPC
hPC =

           Processing Module "conv"
                  ConvThreadNumber: 16
                   InputMemorySize: [227  227    3]
                  OutputMemorySize: [227  227    3]
                  FeatureSizeLimit: 1024

            Processing Module "fc"
                   FCThreadNumber: 4
                   InputMemorySize: 25088
                  OutputMemorySize: 4096

           System Level Properties
                    TargetPlatform: 'Xilinx Zynq ZC706 evaluation kit'
                   TargetFrequency: 180
                     SynthesisTool: 'Xilinx Vivado'
                   ReferenceDesign: 'AXI-Stream DDR Memory Access : 3-AXIM'
           SynthesisToolChipFamily: 'Zynq UltraScale+'
           SynthesisToolDeviceName: 'xczu9eg-ffvb1156-2-e'
         SynthesisToolPackageName: ''
           SynthesisToolSpeedValue: ''
```

See Also

Functions

`getModuleProperty` | `setModuleProperty`

Objects

`dlhdl.buildProcessor`

Topics

“Custom Processor Configuration Workflow”
“Deep Learning Processor IP Core”

Introduced in R2020b

getModuleProperty

Class: `dlhdl.ProcessorConfig`
Package: `dlhdl`

Use the `getModuleProperty` method to get values of module properties within the `dlhdl.ProcessorConfig` object

Syntax

```
getModuleProperty(ModuleName,ModuleProperty)
```

Description

The `getModuleProperty(ModuleName,ModuleProperty)` method returns the value of the module property for modules within the `dlhdl.ProcessorConfig` object.

Examples

Retrieve the `ConvThreadNumber` for the `conv` Module Inside the `dlhdl.ProcessorConfig` Object

1. Create an example object by using `dlhdl.ProcessorConfig` class, and then use the `getModuleProperty` method to obtain the `ConvThreadNumber`.

```
hPC = dlhdl.ProcessorConfig;
hPC.getModuleProperty('conv','ConvThreadNumber')
```

2. Once the code is executed, the result is:

```
ans =

    16
```

Retrieve the `InputMemorySize` for the `fc` Module Inside the `dlhdl.ProcessorConfig` Object

1. Create an example object by using `dlhdl.ProcessorConfig` class, and then use the `getModuleProperty` method to obtain the `ConvThreadNumber`.

```
hPC = dlhdl.ProcessorConfig;
hPC.getModuleProperty('fc','InputMemorySize')
```

2. Once the code is executed, the result is:

```
ans =

       25088
```

Input Arguments

`ModuleName` — Name of the module whose parameters are to be retrieved

'' (default) | 'conv' | character vector

The `dlhdl.ProcessorConfig` object module name, specified as a character vector.

Module Properties

`conv` Module Properties

`ConvThreadNumber` — Number of parallel convolution processor kernel threads

16 (default) | 4 | 9 | 16 | 25 | 36 | 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

512 (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.

`fc` Module Properties

`FCThreadNumber` — Number of parallel fully connected (fc) MAC threads

4 (default) | 4 | 8 | 16 | 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

9216 (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.

See Also

`setModuleProperty`

Topics

“Deep Learning Processor Architecture”

Introduced in R2020b

setModuleProperty

Class: `dlhdl.ProcessorConfig`
Package: `dlhdl`

Use the `setModuleProperty` method to set properties of modules within the `dlhdl.ProcessorConfig` object

Syntax

```
setModuleProperty(ModuleName,Name,Value)
```

Description

The `setModuleProperty(ModuleName,Name,Value)` method sets the properties of the module mentioned in `ModuleName` by using the values specified as `Name,Value` pairs.

Examples

Set the Value for `ConvThreadNumber` for `conv` Module Within the `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 the code is executed, the result is:

```
hPC =

                    Processing Module "conv"
                           ConvThreadNumber: 25
                            InputMemorySize: [227  227    3]
                           OutputMemorySize: [227  227    3]
                           FeatureSizeLimit: 512

                      Processing Module "fc"
                              FCThreadNumber: 4
                            InputMemorySize: 25088
                           OutputMemorySize: 4096

                     System Level Properties
                              TargetPlatform: 'Xilinx Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit'
                            TargetFrequency: 200
                              SynthesisTool: 'Xilinx Vivado'
                           ReferenceDesign: 'AXI-Stream DDR Memory Access : 5-AXIM'
                    SynthesisToolChipFamily: 'Zynq UltraScale+'
                    SynthesisToolDeviceName: 'xczu9eg-ffvb1156-2-e'
                   SynthesisToolPackageName: ''
                    SynthesisToolSpeedValue: ''
```

Set the Value for `InputMemorySize` for `fc` Module Within the `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 the code is executed, the result is:

```
hPC =

                    Processing Module "conv"
                           ConvThreadNumber: 25
                            InputMemorySize: [227  227    3]
                           OutputMemorySize: [227  227    3]
                           FeatureSizeLimit: 512

                      Processing Module "fc"
                              FCThreadNumber: 4
                            InputMemorySize: 25060
                           OutputMemorySize: 4096

                     System Level Properties
                             TargetPlatform: 'Xilinx Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit'
                            TargetFrequency: 200
                              SynthesisTool: 'Xilinx Vivado'
                           ReferenceDesign: 'AXI-Stream DDR Memory Access : 5-AXIM'
                     SynthesisToolChipFamily: 'Zynq UltraScale+'
                     SynthesisToolDeviceName: 'xczu9eg-ffvb1156-2-e'
                    SynthesisToolPackageName: ''
                     SynthesisToolSpeedValue: ''
```

Input Arguments

`ModuleName` — Name of the module whose parameters are to be set

'' (default) | 'fc' | character vector

The `dlhdl.ProcessorConfig` object module name, specified as a character vector.

Module Properties

`conv` Module Properties

`ConvThreadNumber` — Number of parallel convolution processor kernel threads

16 (default) | 4 | 9 | 16 | 25 | 36 | 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', 25`

`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
512 (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`

Example: `'KernelDataType','single'`

`fc` Module Properties

`FCThreadNumber` — Number of parallel fully connected (fc) MAC threads
4 (default) | 4 | 8 | 16 | 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', 8`

`InputMemorySize` — Cache block RAM (BRAM) sizes
9216 (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', 25088`

`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`

Example: `'KernelDataType','single'`

See Also
`getModuleProperty`

Topics
“Deep Learning Processor Architecture”

Introduced in R2020b

dlhdl.buildProcessor class

Package: `dlhdl`

Build and generate custom processor IP

Description

Use the `dlhdl.buildProcessor` class to generate code for the custom processor configured by using the `dlhdl.ProcessorConfig` object.

Creation

The `dlhdl.buildProcessor` class generates code for the object generated from the `dlhdl.ProcessorConfig` object .

Properties

`processorconfigobject` — Name of the object generated using`dlhdl.ProcessorConfig`
(default) | variable

Name of the custom processor configuration object, specified as a variable of type `dlhdl.Processorconfig`.

Generate code for an object created by using `dlhdl.ProcessorConfig`:

```
hPC = dlhdl.ProcessorConfig;
dlhdl.buildProcessor(hPC)
```

See Also

Topics
“Deep Learning Processor IP Core”
“Generate Custom Bitstream”
“Generate Custom Processor IP”

Introduced in R2020b

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 18.1 executable file. You must have already installed Altera® Quartus II.

```
hdlsetuptoolpath('ToolName','Altera Quartus II','ToolPath',...
 'C:\intel\18.1\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 19.2 executable file. You must have already installed Intel Quartus Pro.

```
hdlsetuptoolpath('ToolName','Intel Quartus Pro','ToolPath',...
 'C:\intel\19.2_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 2019.1 batch file. You must have already installed Xilinx Vivado.

```
hdlsetuptoolpath('ToolName','Xilinx Vivado','ToolPath',...
 'C:\Xilinx\Vivado\2019.1\bin\vivado.bat');
```

Set Up Microsemi Libero SoC

The following command sets the synthesis tool path to point to an installed Microsemi® Libero® Design Suite batch file. You must have already installed Microsemi Libero SoC.

```
hdlsetuptoolpath('ToolName','Microsemi Libero SoC','ToolPath',...
 'C:\Microsemi\Libero_SoC_v11.8\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\2018.3\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.

See Also

`setenv` | `startup`

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)

Introduced in R2011a

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.

Creation

Syntax

```
quantObj = dlquantizer(net)
quantObj = dlquantizer(net,Name,Value)
```

Description

`quantObj = dlquantizer(net)` creates a `dlquantizer` object for the specified network.

`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.

Use `dlquantizer` to create an quantized network for FPGA or GPU deployment. To learn about the products required to quantize and deploy the deep learning network to an FPGA or GPU environment, see “Quantization Workflow Prerequisites” .

Input Arguments

`net` — Pretrained neural network
`DAGNetwork` object | `SeriesNetwork` object | `yolov2ObjectDetector` object | `ssdObjectDetector` object

Pretrained neural network, specified as a `DAGNetwork`, `SeriesNetwork`, `yolov2ObjectDetector`, or a `ssdObjectDetector` object.

Quantization of `ssdObjectDetector` networks requires the `ExecutionEnvironment` property to be set to `'FPGA'`.

Properties

`NetworkObject` — Pretrained neural network
`DAGNetwork` object | `SeriesNetwork` object | `yolov2ObjectDetector` object | `ssdObjectDetector` object

Pretrained neural network, specified as a `DAGNetwork`, `SeriesNetwork`, `yolov2ObjectDetector`, or a `ssdObjectDetector` object.

Quantization of `ssdObjectDetector` networks requires the `ExecutionEnvironment` property to be set to `'FPGA'`.

`ExecutionEnvironment` — Execution environment
'GPU' (default) | 'FPGA'

Specify the execution environment for the quantized network. When this parameter is not specified the default execution environment is GPU. To learn about the products required to quantize and deploy the deep learning network to an FPGA or GPU environment, see “Quantization Workflow Prerequisites” .

Example: `'ExecutionEnvironment','FPGA'`

Object Functions

calibrate	Simulate and collect ranges of a deep neural network
validate	Quantize and validate a deep neural network

Examples

Specify FPGA Execution Environment

- This example shows how to specify an FPGA execution environment.

```
net = vgg19;
quantobj = dlquantizer(net,'ExecutionEnvironment','FPGA');
```

Quantize a Neural Network

This example shows how to quantize learnable parameters in the convolution layers of a neural network, 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
```

```
net =

  DAGNetwork with properties:

         Layers: [68x1 nnet.cnn.layer.Layer]
    Connections: [75x2 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. Save this function in a local file.

```
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 =

  95x5 table

                Optimized Layer Name                    Network Layer Name       Learnables / Activations    MinValue      MaxVal
    ____________________________________________    ____________________________    ____________________    __________    ______

    {'conv1_relu_conv1_Weights'                }    {'relu_conv1'           }         "Weights"               -0.91985       0.8
    {'conv1_relu_conv1_Bias'                   }    {'relu_conv1'           }         "Bias"                  -0.07925       0.2
```

```
    {'fire2-squeeze1x1_fire2-relu_squeeze1x1_Weights'}    {'fire2-relu_squeeze1x1'}         "Weights"                -1.38         1.
    {'fire2-squeeze1x1_fire2-relu_squeeze1x1_Bias'   }    {'fire2-relu_squeeze1x1'}         "Bias"                 -0.11641        0.2
    {'fire2-expand1x1_fire2-relu_expand1x1_Weights'  }    {'fire2-relu_expand1x1' }         "Weights"                -0.7406       0.9
    {'fire2-expand1x1_fire2-relu_expand1x1_Bias'     }    {'fire2-relu_expand1x1' }         "Bias"                 -0.060056       0.1
    {'fire2-expand3x3_fire2-relu_expand3x3_Weights'  }    {'fire2-relu_expand3x3' }         "Weights"                -0.74397      0.6
    {'fire2-expand3x3_fire2-relu_expand3x3_Bias'     }    {'fire2-relu_expand3x3' }         "Bias"                 -0.051778      0.07
    {'fire3-squeeze1x1_fire3-relu_squeeze1x1_Weights'}    {'fire3-relu_squeeze1x1'}         "Weights"                -0.77263      0.6
    {'fire3-squeeze1x1_fire3-relu_squeeze1x1_Bias'   }    {'fire3-relu_squeeze1x1'}         "Bias"                 -0.10141        0.3
    {'fire3-expand1x1_fire3-relu_expand1x1_Weights'  }    {'fire3-relu_expand1x1' }         "Weights"                -0.72131      0.9
    {'fire3-expand1x1_fire3-relu_expand1x1_Bias'     }    {'fire3-relu_expand1x1' }         "Bias"                 -0.067043       0.3
    {'fire3-expand3x3_fire3-relu_expand3x3_Weights'  }    {'fire3-relu_expand3x3' }         "Weights"                -0.61196      0.7
    {'fire3-expand3x3_fire3-relu_expand3x3_Bias'     }    {'fire3-relu_expand3x3' }         "Bias"                 -0.053612       0.1
    {'fire4-squeeze1x1_fire4-relu_squeeze1x1_Weights'}    {'fire4-relu_squeeze1x1'}         "Weights"                -0.74145       1.
    {'fire4-squeeze1x1_fire4-relu_squeeze1x1_Bias'   }    {'fire4-relu_squeeze1x1'}         "Bias"                 -0.10886        0.1
...
```

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: [1x1 struct]
```

Examine the `MetricResults.Result` field of the validation output to see the performance of the quantized network.

```
valResults.MetricResults.Result
```

```
ans =

  2x3 table

    NetworkImplementation    MetricOutput    LearnableParameterMemory(bytes)
    _____________________    ____________    _______________________________

     {'Floating-Point'}           1                   2.9003e+06
     {'Quantized'     }           1                   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 FPGA Execution Environment

This example shows how to quantize learnable parameters in the convolution layers of a neural network, and explore the behavior of the quantized network. In this example, you quantize the `LogoNet` neural network. Quantization helps reduce the memory requirement of a deep neural network by quantizing weights, biases and activations of network layers to 8-bit scaled integer data types. Use MATLAB® to retrieve the prediction results from the target device.

To run this example, you need the products listed under FPGA in “Quantization Workflow Prerequisites”.

For additional requirements, see “Quantization Workflow Prerequisites”.

Create a file in your current working directory called `getLogoNetwork.m`. Enter these lines into the file:

```
function net = getLogoNetwork()
    data = getLogoData();
    net  = data.convnet;
end

function data = getLogoData()
    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');
end
```

Load the pretrained network.

```
snet = getLogoNetwork();
```

```
snet =

  SeriesNetwork with properties:

         Layers: [22×1 nnet.cnn.layer.Layer]
     InputNames: {'imageinput'}
    OutputNames: {'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.

This example uses the images in the `logos_dataset` data set. Define an `augmentedImageDatastore` object to resize the data for the network. Then, split the data into calibration and validation data sets.

```
curDir = pwd;
newDir = fullfile(matlabroot,'examples','deeplearning_shared','data','logos_dataset.zip');
copyfile(newDir,curDir);
unzip('logos_dataset.zip');
imageData = imageDatastore(fullfile(curDir,'logos_dataset'),...
 'IncludeSubfolders',true,'FileExtensions','.JPG','LabelSource','foldernames');
[calibrationData, validationData] = splitEachLabel(imageData, 0.5,'randomized');
```

Create a `dlquantizer` object and specify the network to quantize.

```
dlQuantObj = dlquantizer(snet,'ExecutionEnvironment','FPGA');
```

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.

```
 dlQuantObj.calibrate(calibrationData)
```

```
ans =
        Optimized Layer Name        Network Layer Name    Learnables / Activations     MinValue       MaxValue
```

```
    _____________________________    _______________    ____________________    _________    _________

    {'conv_1_Weights'          }     {'conv_1'    }     "Weights"                -0.048978     0.039352
    {'conv_1_Bias'             }     {'conv_1'    }     "Bias"                     0.99996       1.0028
    {'conv_2_Weights'          }     {'conv_2'    }     "Weights"                -0.055518     0.061901
    {'conv_2_Bias'             }     {'conv_2'    }     "Bias"                  -0.00061171      0.00227
    {'conv_3_Weights'          }     {'conv_3'    }     "Weights"                -0.045942     0.046927
    {'conv_3_Bias'             }     {'conv_3'    }     "Bias"                   -0.0013998    0.0015218
    {'conv_4_Weights'          }     {'conv_4'    }     "Weights"                -0.045967         0.051
    {'conv_4_Bias'             }     {'conv_4'    }     "Bias"                     -0.00164    0.0037892
    {'fc_1_Weights'            }     {'fc_1'      }     "Weights"                -0.051394     0.054344
    {'fc_1_Bias'               }     {'fc_1'      }     "Bias"                   -0.00052319   0.00084454
    {'fc_2_Weights'            }     {'fc_2'      }     "Weights"                  -0.05016     0.051557
    {'fc_2_Bias'               }     {'fc_2'      }     "Bias"                   -0.0017564    0.0018502
    {'fc_3_Weights'            }     {'fc_3'      }     "Weights"                -0.050706       0.04678
    {'fc_3_Bias'               }     {'fc_3'      }     "Bias"                     -0.02951     0.024855
    {'imageinput'              }     {'imageinput'}     "Activations"                     0          255
    {'imageinput_normalization'}     {'imageinput'}     "Activations"              -139.34       198.72
```

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 create the target object, enter:

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

Define a metric function to use to compare the behavior of the network before and after quantization. Save this function in a local file.

```
function accuracy = hComputeAccuracy(predictionScores, net, dataStore)
%% hComputeAccuracy test helper function computes model level accuracy statistics

% Copyright 2020 The MathWorks, Inc.

    % Load ground truth
    groundTruth = dataStore.Labels;

    % 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.

```
options = dlquantizationOptions('MetricFcn', ...
    {@(x)hComputeModelAccuracy(x, snet, validationData)},'Bitstream','arria10soc_int8',...
'Target',hTarget);
```

To compile and deploy the quantized network, run the `validate` function of the `dlquantizer` object. Use the `validate` function to quantize the learnable parameters in the convolution layers of the network and exercise the network. 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 checks for the Intel Quartus tool and the supported tool version. It then starts programming the FPGA device by using the sof file, displays progress messages, and the time it takes to deploy the network. The function uses the metric function defined in the `dlquantizationOptions` object to compare the results of the network before and after quantization.

```
prediction = dlQuantObj.validate(validationData,options);
```

```
          offset_name          offset_address     allocated_space
    ___________________________    ______________    _______________

    "InputDataOffset"              "0x00000000"      "48.0 MB"
```

```
    "OutputResultOffset"        "0x03000000"     "4.0 MB"
    "SystemBufferOffset"        "0x03400000"     "60.0 MB"
    "InstructionDataOffset"     "0x07000000"     "8.0 MB"
    "ConvWeightDataOffset"      "0x07800000"     "8.0 MB"
    "FCWeightDataOffset"        "0x08000000"     "12.0 MB"
    "EndOffset"                 "0x08c00000"     "Total: 140.0 MB"

### Programming FPGA Bitstream using JTAG...
### Programming the FPGA bitstream has been completed successfully.
### Loading weights to Conv Processor.
### Conv Weights loaded. Current time is 16-Jul-2020 12:45:10
### Loading weights to FC Processor.
### FC Weights loaded. Current time is 16-Jul-2020 12:45:26
### Finished writing input activations.
### Running single input activations.


             Deep Learning Processor Profiler Performance Results

                  LastLayerLatency(cycles)   LastLayerLatency(seconds)       FramesNum      Total Latency     Frames/s
                  ------------               ------------                    ---------        ---------       ---------
Network                  13570959                    0.09047                    30          380609145            11.8
    conv_module          12667786                    0.08445
        conv_1            3938907                    0.02626
        maxpool_1         1544560                    0.01030
        conv_2            2910954                    0.01941
        maxpool_2          577524                    0.00385
        conv_3            2552707                    0.01702
        maxpool_3          676542                    0.00451
        conv_4             455434                    0.00304
        maxpool_4           11251                    0.00008
    fc_module               903173                    0.00602
        fc_1                536164                    0.00357
        fc_2                342643                    0.00228
        fc_3                 24364                    0.00016
 * The clock frequency of the DL processor is: 150MHz


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


             Deep Learning Processor Profiler Performance Results

                  LastLayerLatency(cycles)   LastLayerLatency(seconds)       FramesNum      Total Latency     Frames/s
                  ------------               ------------                    ---------        ---------       ---------
Network                  13570364                    0.09047                    30          380612682            11.8
    conv_module          12667103                    0.08445
        conv_1            3939296                    0.02626
        maxpool_1         1544371                    0.01030
        conv_2            2910747                    0.01940
        maxpool_2          577654                    0.00385
        conv_3            2551829                    0.01701
        maxpool_3          676548                    0.00451
        conv_4             455396                    0.00304
        maxpool_4           11355                    0.00008
    fc_module               903261                    0.00602
        fc_1                536206                    0.00357
        fc_2                342688                    0.00228
        fc_3                 24365                    0.00016
 * The clock frequency of the DL processor is: 150MHz


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


             Deep Learning Processor Profiler Performance Results

                  LastLayerLatency(cycles)   LastLayerLatency(seconds)       FramesNum      Total Latency     Frames/s
                  ------------               ------------                    ---------        ---------       ---------
Network                  13571561                    0.09048                    30          380608338            11.8
    conv_module          12668340                    0.08446
        conv_1            3939070                    0.02626
        maxpool_1         1545327                    0.01030
        conv_2            2911061                    0.01941
        maxpool_2          577557                    0.00385
        conv_3            2552082                    0.01701
        maxpool_3          676506                    0.00451
        conv_4             455582                    0.00304
        maxpool_4           11248                    0.00007
```

```
    fc_module               903221                 0.00602
        fc_1                536167                 0.00357
        fc_2                342643                 0.00228
        fc_3                 24409                 0.00016
 * The clock frequency of the DL processor is: 150MHz


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


             Deep Learning Processor Profiler Performance Results

                  LastLayerLatency(cycles)   LastLayerLatency(seconds)       FramesNum      Total Latency     Frames/s
                        -------------              -------------               ---------        ---------       ---------
Network                  13569862                 0.09047                     30          380613327            11.8
    conv_module          12666756                 0.08445
        conv_1            3939212                 0.02626
        maxpool_1         1543267                 0.01029
        conv_2            2911184                 0.01941
        maxpool_2          577275                 0.00385
        conv_3            2552868                 0.01702
        maxpool_3          676438                 0.00451
        conv_4             455353                 0.00304
        maxpool_4           11252                 0.00008
    fc_module              903106                 0.00602
        fc_1               536050                 0.00357
        fc_2               342645                 0.00228
        fc_3                24409                 0.00016
 * The clock frequency of the DL processor is: 150MHz


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


             Deep Learning Processor Profiler Performance Results

                  LastLayerLatency(cycles)   LastLayerLatency(seconds)       FramesNum      Total Latency     Frames/s
                        -------------              -------------               ---------        ---------       ---------
Network                  13570823                 0.09047                     30          380619836            11.8
    conv_module          12667607                 0.08445
        conv_1            3939074                 0.02626
        maxpool_1         1544519                 0.01030
        conv_2            2910636                 0.01940
        maxpool_2          577769                 0.00385
        conv_3            2551800                 0.01701
        maxpool_3          676795                 0.00451
        conv_4             455859                 0.00304
        maxpool_4           11248                 0.00007
    fc_module              903216                 0.00602
        fc_1               536165                 0.00357
        fc_2               342643                 0.00228
        fc_3                24406                 0.00016
 * The clock frequency of the DL processor is: 150MHz

          offset_name          offset_address     allocated_space
    _______________________    ______________    _________________

    "InputDataOffset"           "0x00000000"     "48.0 MB"
    "OutputResultOffset"        "0x03000000"     "4.0 MB"
    "SystemBufferOffset"        "0x03400000"     "60.0 MB"
    "InstructionDataOffset"     "0x07000000"     "8.0 MB"
    "ConvWeightDataOffset"      "0x07800000"     "8.0 MB"
    "FCWeightDataOffset"        "0x08000000"     "12.0 MB"
    "EndOffset"                 "0x08c00000"     "Total: 140.0 MB"

### FPGA bitstream programming has been skipped as the same bitstream is already loaded on the target FPGA.
### Deep learning network programming has been skipped as the same network is already loaded on the target FPGA.
### Finished writing input activations.
### Running single input activations.


             Deep Learning Processor Profiler Performance Results

                  LastLayerLatency(cycles)   LastLayerLatency(seconds)       FramesNum      Total Latency     Frames/s
                        -------------              -------------               ---------        ---------       ---------
Network                  13572329                 0.09048                     10          127265075            11.8
    conv_module          12669135                 0.08446
        conv_1            3939559                 0.02626
```

```
        maxpool_1          1545378                0.01030
        conv_2             2911243                0.01941
        maxpool_2           577422                0.00385
        conv_3             2552064                0.01701
        maxpool_3           676678                0.00451
        conv_4              455657                0.00304
        maxpool_4            11227                0.00007
    fc_module               903194                0.00602
        fc_1                536140                0.00357
        fc_2                342688                0.00228
        fc_3                 24364                0.00016
 * The clock frequency of the DL processor is: 150MHz


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


              Deep Learning Processor Profiler Performance Results

                   LastLayerLatency(cycles)   LastLayerLatency(seconds)       FramesNum      Total Latency     Frames/s
                         -------------              -------------               ---------        ---------       ---------
Network                   13572527                0.09048                     10          127266427            11.8
    conv_module           12669266                0.08446
        conv_1             3939776                0.02627
        maxpool_1          1545632                0.01030
        conv_2             2911169                0.01941
        maxpool_2           577592                0.00385
        conv_3             2551613                0.01701
        maxpool_3           676811                0.00451
        conv_4              455418                0.00304
        maxpool_4            11348                0.00008
    fc_module               903261                0.00602
        fc_1                536205                0.00357
        fc_2                342689                0.00228
        fc_3                 24365                0.00016
 * The clock frequency of the DL processor is: 150MHz
```

Examine the `MetricResults.Result` field of the validation output to see the performance of the quantized network.

```
validateOut = prediction.MetricResults.Result
```

```
ans =
    NetworkImplementation    MetricOutput
    _____________________    ____________

     {'Floating-Point'}          0.9875
     {'Quantized'     }          0.9875
```

Examine the `QuantizedNetworkFPS` field of the validation output to see the frames per second performance of the quantized network.

```
prediction.QuantizedNetworkFPS
```

```
ans = 11.8126
```

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.

See Also

Apps
Deep Network Quantizer

Functions
`calibrate` | `dlquantizationOptions` | `validate`

Topics
“Quantization of Deep Neural Networks”

Introduced in R2020a

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. Use the `dlquantizationOptions` object to define the metric function to use that compares the accuracy of the network before and after quantization.

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 as `Name, Value` pair arguments.

Properties

`MetricFcn` — Function to use for calculating validation metrics

cell array of function handles

Cell array of function handles specifying the functions for calculating validation metrics of quantized network.

Example: `options = dlquantizationOptions('MetricFcn', {@(x)hComputeModelAccuracy(x, net, groundTruth)});`

Data Types: `cell`

FPGA Execution Environment Options

`Bitstream` — Bitstream name

'zcu102_int8' | 'zc706_int8' | 'arria10soc_int8'

This property affects FPGA targeting only.

Name of the FPGA bitstream specified as a character vector.

Example: `'Bitstream', 'zcu102_int8'`

`Target` — Name of the `dlhdl.Target` object
hT

This property affects FPGA targeting only.

Name of the `dlhdl.Target` object that has the board name and board interface information.

Example: `'Target', hT`

Examples

Quantize a Neural Network

This example shows how to quantize learnable parameters in the convolution layers of a neural network, 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
```

```
net =

  DAGNetwork with properties:

         Layers: [68x1 nnet.cnn.layer.Layer]
    Connections: [75x2 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. Save this function in a local file.

```
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 =

  95x5 table

                Optimized Layer Name                    Network Layer Name       Learnables / Activations     MinValue      MaxVal
    ____________________________________________    ____________________    ____________________________    _________    ______

    {'conv1_relu_conv1_Weights'                  }    {'relu_conv1'          }        "Weights"               -0.91985       0.8
    {'conv1_relu_conv1_Bias'                     }    {'relu_conv1'          }        "Bias"                  -0.07925       0.2
    {'fire2-squeeze1x1_fire2-relu_squeeze1x1_Weights'}    {'fire2-relu_squeeze1x1'}        "Weights"                  -1.38         1.
    {'fire2-squeeze1x1_fire2-relu_squeeze1x1_Bias'   }    {'fire2-relu_squeeze1x1'}        "Bias"                  -0.11641       0.2
    {'fire2-expand1x1_fire2-relu_expand1x1_Weights'  }    {'fire2-relu_expand1x1' }        "Weights"                -0.7406       0.9
    {'fire2-expand1x1_fire2-relu_expand1x1_Bias'     }    {'fire2-relu_expand1x1' }        "Bias"                  -0.060056      0.1
    {'fire2-expand3x3_fire2-relu_expand3x3_Weights'  }    {'fire2-relu_expand3x3' }        "Weights"                -0.74397      0.6
    {'fire2-expand3x3_fire2-relu_expand3x3_Bias'     }    {'fire2-relu_expand3x3' }        "Bias"                  -0.051778      0.07
    {'fire3-squeeze1x1_fire3-relu_squeeze1x1_Weights'}    {'fire3-relu_squeeze1x1'}        "Weights"                -0.77263      0.6
    {'fire3-squeeze1x1_fire3-relu_squeeze1x1_Bias'   }    {'fire3-relu_squeeze1x1'}        "Bias"                  -0.10141       0.3
    {'fire3-expand1x1_fire3-relu_expand1x1_Weights'  }    {'fire3-relu_expand1x1' }        "Weights"                -0.72131      0.9
    {'fire3-expand1x1_fire3-relu_expand1x1_Bias'     }    {'fire3-relu_expand1x1' }        "Bias"                  -0.067043      0.3
    {'fire3-expand3x3_fire3-relu_expand3x3_Weights'  }    {'fire3-relu_expand3x3' }        "Weights"                -0.61196      0.7
    {'fire3-expand3x3_fire3-relu_expand3x3_Bias'     }    {'fire3-relu_expand3x3' }        "Bias"                  -0.053612      0.1
    {'fire4-squeeze1x1_fire4-relu_squeeze1x1_Weights'}    {'fire4-relu_squeeze1x1'}        "Weights"                -0.74145         1.
    {'fire4-squeeze1x1_fire4-relu_squeeze1x1_Bias'   }    {'fire4-relu_squeeze1x1'}        "Bias"                  -0.10886       0.1
...
```

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: [1x1 struct]
```

Examine the `MetricResults.Result` field of the validation output to see the performance of the quantized network.

```
valResults.MetricResults.Result
```

```
ans =

  2x3 table

    NetworkImplementation    MetricOutput    LearnableParameterMemory(bytes)
    _____________________    ____________    _______________________________

     {'Floating-Point'}           1                   2.9003e+06
     {'Quantized'     }           1                   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 FPGA Execution Environment

This example shows how to quantize learnable parameters in the convolution layers of a neural network, and explore the behavior of the quantized network. In this example, you quantize the `LogoNet` neural network. Quantization helps reduce the memory requirement of a deep neural network by quantizing weights, biases and activations of network layers to 8-bit scaled integer data types. Use MATLAB® to retrieve the prediction results from the target device.

To run this example, you need the products listed under FPGA in “Quantization Workflow Prerequisites”.

For additional requirements, see “Quantization Workflow Prerequisites”.

Create a file in your current working directory called `getLogoNetwork.m`. Enter these lines into the file:

```
function net = getLogoNetwork()
    data = getLogoData();
    net  = data.convnet;
end

function data = getLogoData()
    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');
end
```

Load the pretrained network.

```
snet = getLogoNetwork();
```

```
snet =
```

```
  SeriesNetwork with properties:

        Layers: [22×1 nnet.cnn.layer.Layer]
     InputNames: {'imageinput'}
    OutputNames: {'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.

This example uses the images in the `logos_dataset` data set. Define an `augmentedImageDatastore` object to resize the data for the network. Then, split the data into calibration and validation data sets.

```
curDir = pwd;
newDir = fullfile(matlabroot,'examples','deeplearning_shared','data','logos_dataset.zip');
copyfile(newDir,curDir);
unzip('logos_dataset.zip');
imageData = imageDatastore(fullfile(curDir,'logos_dataset'),...
 'IncludeSubfolders',true,'FileExtensions','.JPG','LabelSource','foldernames');
[calibrationData, validationData] = splitEachLabel(imageData, 0.5,'randomized');
```

Create a `dlquantizer` object and specify the network to quantize.

```
dlQuantObj = dlquantizer(snet,'ExecutionEnvironment','FPGA');
```

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.

```
 dlQuantObj.calibrate(calibrationData)
```

```
ans =
      Optimized Layer Name        Network Layer Name    Learnables / Activations     MinValue       MaxValue
    ________________________    ____________________    ________________________    ___________    __________

    {'conv_1_Weights'          }    {'conv_1'    }          "Weights"                -0.048978      0.039352
    {'conv_1_Bias'             }    {'conv_1'    }          "Bias"                     0.99996        1.0028
    {'conv_2_Weights'          }    {'conv_2'    }          "Weights"                -0.055518      0.061901
    {'conv_2_Bias'             }    {'conv_2'    }          "Bias"                  -0.00061171       0.00227
    {'conv_3_Weights'          }    {'conv_3'    }          "Weights"                -0.045942      0.046927
    {'conv_3_Bias'             }    {'conv_3'    }          "Bias"                   -0.0013998     0.0015218
    {'conv_4_Weights'          }    {'conv_4'    }          "Weights"                -0.045967          0.051
    {'conv_4_Bias'             }    {'conv_4'    }          "Bias"                    -0.00164     0.0037892
    {'fc_1_Weights'            }    {'fc_1'      }          "Weights"                -0.051394      0.054344
    {'fc_1_Bias'               }    {'fc_1'      }          "Bias"                  -0.00052319    0.00084454
    {'fc_2_Weights'            }    {'fc_2'      }          "Weights"                 -0.05016      0.051557
    {'fc_2_Bias'               }    {'fc_2'      }          "Bias"                   -0.0017564     0.0018502
    {'fc_3_Weights'            }    {'fc_3'      }          "Weights"                -0.050706       0.04678
    {'fc_3_Bias'               }    {'fc_3'      }          "Bias"                    -0.02951      0.024855
    {'imageinput'              }    {'imageinput'}          "Activations"                    0           255
    {'imageinput_normalization'}    {'imageinput'}          "Activations"             -139.34        198.72
```

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 create the target object, enter:

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

Define a metric function to use to compare the behavior of the network before and after quantization. Save this function in a local file.

```
function accuracy = hComputeAccuracy(predictionScores, net, dataStore)
%% hComputeAccuracy test helper function computes model level accuracy statistics

% Copyright 2020 The MathWorks, Inc.

    % Load ground truth
    groundTruth = dataStore.Labels;

    % 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.

```
options = dlquantizationOptions('MetricFcn', ...
    {@(x)hComputeModelAccuracy(x, snet, validationData)},'Bitstream','arria10soc_int8',...
'Target',hTarget);
```

To compile and deploy the quantized network, run the `validate` function of the `dlquantizer` object. Use the `validate` function to quantize the learnable parameters in the convolution layers of the network and exercise the network. 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 checks for the Intel Quartus tool and the supported tool version. It then starts programming the FPGA device by using the sof file, displays progress messages, and the time it takes to deploy the network. The function uses the metric function defined in the `dlquantizationOptions` object to compare the results of the network before and after quantization.

```
prediction = dlQuantObj.validate(validationData,options);
```

```
          offset_name          offset_address     allocated_space
    _______________________    ______________    _________________

    "InputDataOffset"           "0x00000000"     "48.0 MB"
    "OutputResultOffset"        "0x03000000"     "4.0 MB"
    "SystemBufferOffset"        "0x03400000"     "60.0 MB"
    "InstructionDataOffset"     "0x07000000"     "8.0 MB"
    "ConvWeightDataOffset"      "0x07800000"     "8.0 MB"
    "FCWeightDataOffset"        "0x08000000"     "12.0 MB"
    "EndOffset"                 "0x08c00000"     "Total: 140.0 MB"

### Programming FPGA Bitstream using JTAG...
### Programming the FPGA bitstream has been completed successfully.
### Loading weights to Conv Processor.
### Conv Weights loaded. Current time is 16-Jul-2020 12:45:10
### Loading weights to FC Processor.
### FC Weights loaded. Current time is 16-Jul-2020 12:45:26
### Finished writing input activations.
### Running single input activations.


             Deep Learning Processor Profiler Performance Results

                    LastLayerLatency(cycles)   LastLayerLatency(seconds)       FramesNum      Total Latency     Frames/s
                    ------------               -------------                   ---------        ---------       ---------
Network                   13570959                    0.09047                      30          380609145            11.8
    conv_module           12667786                    0.08445
        conv_1             3938907                    0.02626
        maxpool_1          1544560                    0.01030
```

```
        conv_2             2910954                 0.01941
        maxpool_2           577524                 0.00385
        conv_3             2552707                 0.01702
        maxpool_3           676542                 0.00451
        conv_4              455434                 0.00304
        maxpool_4            11251                 0.00008
    fc_module               903173                 0.00602
        fc_1                536164                 0.00357
        fc_2                342643                 0.00228
        fc_3                 24364                 0.00016
 * The clock frequency of the DL processor is: 150MHz


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


             Deep Learning Processor Profiler Performance Results

                  LastLayerLatency(cycles)   LastLayerLatency(seconds)       FramesNum      Total Latency     Frames/s
                  ------------------------   -------------------------       ---------      -------------     ---------
Network                   13570364                  0.09047                     30          380612682            11.8
    conv_module           12667103                  0.08445
        conv_1             3939296                  0.02626
        maxpool_1          1544371                  0.01030
        conv_2             2910747                  0.01940
        maxpool_2           577654                  0.00385
        conv_3             2551829                  0.01701
        maxpool_3           676548                  0.00451
        conv_4              455396                  0.00304
        maxpool_4            11355                  0.00008
    fc_module               903261                  0.00602
        fc_1                536206                  0.00357
        fc_2                342688                  0.00228
        fc_3                 24365                  0.00016
 * The clock frequency of the DL processor is: 150MHz


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


             Deep Learning Processor Profiler Performance Results

                  LastLayerLatency(cycles)   LastLayerLatency(seconds)       FramesNum      Total Latency     Frames/s
                  ------------------------   -------------------------       ---------      -------------     ---------
Network                   13571561                  0.09048                     30          380608338            11.8
    conv_module           12668340                  0.08446
        conv_1             3939070                  0.02626
        maxpool_1          1545327                  0.01030
        conv_2             2911061                  0.01941
        maxpool_2           577557                  0.00385
        conv_3             2552082                  0.01701
        maxpool_3           676506                  0.00451
        conv_4              455582                  0.00304
        maxpool_4            11248                  0.00007
    fc_module               903221                  0.00602
        fc_1                536167                  0.00357
        fc_2                342643                  0.00228
        fc_3                 24409                  0.00016
 * The clock frequency of the DL processor is: 150MHz


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


             Deep Learning Processor Profiler Performance Results

                  LastLayerLatency(cycles)   LastLayerLatency(seconds)       FramesNum      Total Latency     Frames/s
                  ------------------------   -------------------------       ---------      -------------     ---------
Network                   13569862                  0.09047                     30          380613327            11.8
    conv_module           12666756                  0.08445
        conv_1             3939212                  0.02626
        maxpool_1          1543267                  0.01029
        conv_2             2911184                  0.01941
        maxpool_2           577275                  0.00385
        conv_3             2552868                  0.01702
        maxpool_3           676438                  0.00451
        conv_4              455353                  0.00304
        maxpool_4            11252                  0.00008
```

```
    fc_module               903106                 0.00602
        fc_1                536050                 0.00357
        fc_2                342645                 0.00228
        fc_3                 24409                 0.00016
 * The clock frequency of the DL processor is: 150MHz


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


             Deep Learning Processor Profiler Performance Results

                  LastLayerLatency(cycles)   LastLayerLatency(seconds)       FramesNum      Total Latency     Frames/s
                        -------------              -------------               ---------        ---------       ---------
Network                  13570823                 0.09047                     30          380619836            11.8
    conv_module          12667607                 0.08445
        conv_1            3939074                 0.02626
        maxpool_1         1544519                 0.01030
        conv_2            2910636                 0.01940
        maxpool_2          577769                 0.00385
        conv_3            2551800                 0.01701
        maxpool_3          676795                 0.00451
        conv_4             455859                 0.00304
        maxpool_4           11248                 0.00007
    fc_module              903216                 0.00602
        fc_1               536165                 0.00357
        fc_2               342643                 0.00228
        fc_3                24406                 0.00016
 * The clock frequency of the DL processor is: 150MHz


          offset_name          offset_address     allocated_space
    ___________________________    _____________    _________________

    "InputDataOffset"             "0x00000000"     "48.0 MB"
    "OutputResultOffset"          "0x03000000"     "4.0 MB"
    "SystemBufferOffset"          "0x03400000"     "60.0 MB"
    "InstructionDataOffset"       "0x07000000"     "8.0 MB"
    "ConvWeightDataOffset"        "0x07800000"     "8.0 MB"
    "FCWeightDataOffset"          "0x08000000"     "12.0 MB"
    "EndOffset"                   "0x08c00000"     "Total: 140.0 MB"

### FPGA bitstream programming has been skipped as the same bitstream is already loaded on the target FPGA.
### Deep learning network programming has been skipped as the same network is already loaded on the target FPGA.
### Finished writing input activations.
### Running single input activations.


             Deep Learning Processor Profiler Performance Results

                  LastLayerLatency(cycles)   LastLayerLatency(seconds)       FramesNum      Total Latency     Frames/s
                        -------------              -------------               ---------        ---------       ---------
Network                  13572329                 0.09048                     10          127265075            11.8
    conv_module          12669135                 0.08446
        conv_1            3939559                 0.02626
        maxpool_1         1545378                 0.01030
        conv_2            2911243                 0.01941
        maxpool_2          577422                 0.00385
        conv_3            2552064                 0.01701
        maxpool_3          676678                 0.00451
        conv_4             455657                 0.00304
        maxpool_4           11227                 0.00007
    fc_module              903194                 0.00602
        fc_1               536140                 0.00357
        fc_2               342688                 0.00228
        fc_3                24364                 0.00016
 * The clock frequency of the DL processor is: 150MHz


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


             Deep Learning Processor Profiler Performance Results

                  LastLayerLatency(cycles)   LastLayerLatency(seconds)       FramesNum      Total Latency     Frames/s
                        -------------              -------------               ---------        ---------       ---------
Network                  13572527                 0.09048                     10          127266427            11.8
    conv_module          12669266                 0.08446
        conv_1            3939776                 0.02627
```

```
       maxpool_1          1545632                 0.01030
       conv_2             2911169                 0.01941
       maxpool_2           577592                 0.00385
       conv_3             2551613                 0.01701
       maxpool_3           676811                 0.00451
       conv_4              455418                 0.00304
       maxpool_4            11348                 0.00008
   fc_module               903261                 0.00602
       fc_1                536205                 0.00357
       fc_2                342689                 0.00228
       fc_3                 24365                 0.00016
 * The clock frequency of the DL processor is: 150MHz
```

Examine the `MetricResults.Result` field of the validation output to see the performance of the quantized network.

```
validateOut = prediction.MetricResults.Result
```

```
ans =
    NetworkImplementation    MetricOutput
    _____________________    ____________

     {'Floating-Point'}         0.9875
     {'Quantized'     }         0.9875
```

Examine the `QuantizedNetworkFPS` field of the validation output to see the frames per second performance of the quantized network.

```
prediction.QuantizedNetworkFPS
```

```
ans = 11.8126
```

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.

See Also

Apps

Deep Network Quantizer

Functions

`calibrate` | `dlquantizer` | `validate`

Topics

“Quantization of Deep Neural Networks”

Introduced in R2020a

calibrate

Simulate and collect ranges of a deep neural network

Syntax

```
calibrationResults = calibrate(quantObj, calData)
calibrationResults = calibrate(quantObj, calData,Name,Value)
```

Description

`calibrationResults = 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`.

`calibrationResults = calibrate(quantObj, calData,Name,Value)` 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`, with additional arguments specified by one or more name-value pair arguments.

To learn about the products required to quantize a deep neural network, see “Quantization Workflow Prerequisites”

Examples

Quantize a Neural Network

This example shows how to quantize learnable parameters in the convolution layers of a neural network, 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
```

```
net =

  DAGNetwork with properties:

         Layers: [68x1 nnet.cnn.layer.Layer]
    Connections: [75x2 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. Save this function in a local file.

```
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 =

  95x5 table

                Optimized Layer Name                 Network Layer Name       Learnables / Activations     MinValue      MaxVal
```

```
    {'conv1_relu_conv1_Weights'                      }    {'relu_conv1'           }        "Weights"               -0.91985        0.8
    {'conv1_relu_conv1_Bias'                         }    {'relu_conv1'           }        "Bias"                  -0.07925        0.2
    {'fire2-squeeze1x1_fire2-relu_squeeze1x1_Weights'}    {'fire2-relu_squeeze1x1'}        "Weights"                  -1.38         1.
    {'fire2-squeeze1x1_fire2-relu_squeeze1x1_Bias'   }    {'fire2-relu_squeeze1x1'}        "Bias"                  -0.11641        0.2
    {'fire2-expand1x1_fire2-relu_expand1x1_Weights'  }    {'fire2-relu_expand1x1' }        "Weights"                -0.7406        0.9
    {'fire2-expand1x1_fire2-relu_expand1x1_Bias'     }    {'fire2-relu_expand1x1' }        "Bias"                  -0.060056       0.1
    {'fire2-expand3x3_fire2-relu_expand3x3_Weights'  }    {'fire2-relu_expand3x3' }        "Weights"                -0.74397        0.6
    {'fire2-expand3x3_fire2-relu_expand3x3_Bias'     }    {'fire2-relu_expand3x3' }        "Bias"                  -0.051778       0.07
    {'fire3-squeeze1x1_fire3-relu_squeeze1x1_Weights'}    {'fire3-relu_squeeze1x1'}        "Weights"                -0.77263        0.6
    {'fire3-squeeze1x1_fire3-relu_squeeze1x1_Bias'   }    {'fire3-relu_squeeze1x1'}        "Bias"                  -0.10141        0.3
    {'fire3-expand1x1_fire3-relu_expand1x1_Weights'  }    {'fire3-relu_expand1x1' }        "Weights"                -0.72131        0.9
    {'fire3-expand1x1_fire3-relu_expand1x1_Bias'     }    {'fire3-relu_expand1x1' }        "Bias"                  -0.067043       0.3
    {'fire3-expand3x3_fire3-relu_expand3x3_Weights'  }    {'fire3-relu_expand3x3' }        "Weights"                -0.61196        0.7
    {'fire3-expand3x3_fire3-relu_expand3x3_Bias'     }    {'fire3-relu_expand3x3' }        "Bias"                  -0.053612       0.1
    {'fire4-squeeze1x1_fire4-relu_squeeze1x1_Weights'}    {'fire4-relu_squeeze1x1'}        "Weights"                -0.74145         1.
    {'fire4-squeeze1x1_fire4-relu_squeeze1x1_Bias'   }    {'fire4-relu_squeeze1x1'}        "Bias"                  -0.10886        0.1
...
```

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: [1x1 struct]
```

Examine the `MetricResults.Result` field of the validation output to see the performance of the quantized network.

```
valResults.MetricResults.Result
```

```
ans =

  2x3 table

    NetworkImplementation    MetricOutput    LearnableParameterMemory(bytes)
    _____________________    ____________    _______________________________

     {'Floating-Point'}           1                   2.9003e+06
     {'Quantized'     }           1                   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 FPGA Execution Environment

This example shows how to quantize learnable parameters in the convolution layers of a neural network, and explore the behavior of the quantized network. In this example, you quantize the `LogoNet` neural network. Quantization helps reduce the memory requirement of a deep neural network by quantizing weights, biases and activations of network layers to 8-bit scaled integer data types. Use MATLAB® to retrieve the prediction results from the target device.

To run this example, you need the products listed under FPGA in “Quantization Workflow Prerequisites”.

For additional requirements, see “Quantization Workflow Prerequisites”.

Create a file in your current working directory called `getLogoNetwork.m`. Enter these lines into the file:

```
function net = getLogoNetwork()
    data = getLogoData();
    net  = data.convnet;
end

function data = getLogoData()
    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');
end
```

Load the pretrained network.

```
snet = getLogoNetwork();
```

```
snet =

  SeriesNetwork with properties:

         Layers: [22×1 nnet.cnn.layer.Layer]
     InputNames: {'imageinput'}
    OutputNames: {'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.

This example uses the images in the `logos_dataset` data set. Define an `augmentedImageDatastore` object to resize the data for the network. Then, split the data into calibration and validation data sets.

```
curDir = pwd;
newDir = fullfile(matlabroot,'examples','deeplearning_shared','data','logos_dataset.zip');
copyfile(newDir,curDir);
unzip('logos_dataset.zip');
imageData = imageDatastore(fullfile(curDir,'logos_dataset'),...
 'IncludeSubfolders',true,'FileExtensions','.JPG','LabelSource','foldernames');
[calibrationData, validationData] = splitEachLabel(imageData, 0.5,'randomized');
```

Create a `dlquantizer` object and specify the network to quantize.

```
dlQuantObj = dlquantizer(snet,'ExecutionEnvironment','FPGA');
```

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.

```
 dlQuantObj.calibrate(calibrationData)
```

```
ans =
      Optimized Layer Name          Network Layer Name    Learnables / Activations     MinValue       MaxValue
    ________________________________    _______________    ________________________    ___________    __________

    {'conv_1_Weights'          }      {'conv_1'    }           "Weights"                -0.048978      0.039352
    {'conv_1_Bias'             }      {'conv_1'    }           "Bias"                     0.99996        1.0028
    {'conv_2_Weights'          }      {'conv_2'    }           "Weights"                -0.055518      0.061901
    {'conv_2_Bias'             }      {'conv_2'    }           "Bias"                  -0.00061171       0.00227
    {'conv_3_Weights'          }      {'conv_3'    }           "Weights"                -0.045942      0.046927
    {'conv_3_Bias'             }      {'conv_3'    }           "Bias"                   -0.0013998     0.0015218
    {'conv_4_Weights'          }      {'conv_4'    }           "Weights"                -0.045967          0.051
    {'conv_4_Bias'             }      {'conv_4'    }           "Bias"                    -0.00164     0.0037892
    {'fc_1_Weights'            }      {'fc_1'      }           "Weights"                -0.051394      0.054344
    {'fc_1_Bias'               }      {'fc_1'      }           "Bias"                  -0.00052319    0.00084454
    {'fc_2_Weights'            }      {'fc_2'      }           "Weights"                 -0.05016      0.051557
    {'fc_2_Bias'               }      {'fc_2'      }           "Bias"                   -0.0017564     0.0018502
    {'fc_3_Weights'            }      {'fc_3'      }           "Weights"                -0.050706       0.04678
    {'fc_3_Bias'               }      {'fc_3'      }           "Bias"                    -0.02951      0.024855
    {'imageinput'              }      {'imageinput'}           "Activations"                    0           255
    {'imageinput_normalization'}      {'imageinput'}           "Activations"             -139.34        198.72
```

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 create the target object, enter:

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

Define a metric function to use to compare the behavior of the network before and after quantization. Save this function in a local file.

```
function accuracy = hComputeAccuracy(predictionScores, net, dataStore)
%% hComputeAccuracy test helper function computes model level accuracy statistics

% Copyright 2020 The MathWorks, Inc.

    % Load ground truth
    groundTruth = dataStore.Labels;

    % 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.

```
options = dlquantizationOptions('MetricFcn', ...
    {@(x)hComputeModelAccuracy(x, snet, validationData)},'Bitstream','arria10soc_int8',...
'Target',hTarget);
```

To compile and deploy the quantized network, run the `validate` function of the `dlquantizer` object. Use the `validate` function to quantize the learnable parameters in the convolution layers of the network and exercise the network. 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 checks for the Intel Quartus tool and the supported tool version. It then starts programming the FPGA device by using the sof file, displays progress messages, and the time it takes to deploy the network. The function uses the metric function defined in the `dlquantizationOptions` object to compare the results of the network before and after quantization.

```
prediction = dlQuantObj.validate(validationData,options);
```

```
          offset_name          offset_address     allocated_space
    ___________________________    ______________    _______________
```

```
    "InputDataOffset"           "0x00000000"     "48.0 MB"
    "OutputResultOffset"        "0x03000000"     "4.0 MB"
    "SystemBufferOffset"        "0x03400000"     "60.0 MB"
    "InstructionDataOffset"     "0x07000000"     "8.0 MB"
    "ConvWeightDataOffset"      "0x07800000"     "8.0 MB"
    "FCWeightDataOffset"        "0x08000000"     "12.0 MB"
    "EndOffset"                 "0x08c00000"     "Total: 140.0 MB"

### Programming FPGA Bitstream using JTAG...
### Programming the FPGA bitstream has been completed successfully.
### Loading weights to Conv Processor.
### Conv Weights loaded. Current time is 16-Jul-2020 12:45:10
### Loading weights to FC Processor.
### FC Weights loaded. Current time is 16-Jul-2020 12:45:26
### Finished writing input activations.
### Running single input activations.


             Deep Learning Processor Profiler Performance Results

                    LastLayerLatency(cycles)   LastLayerLatency(seconds)       FramesNum      Total Latency     Frames/s
                          -------------              -------------               ---------        ---------      ---------
Network                    13570959                    0.09047                     30          380609145            11.8
    conv_module            12667786                    0.08445
        conv_1              3938907                    0.02626
        maxpool_1           1544560                    0.01030
        conv_2              2910954                    0.01941
        maxpool_2            577524                    0.00385
        conv_3              2552707                    0.01702
        maxpool_3            676542                    0.00451
        conv_4               455434                    0.00304
        maxpool_4             11251                    0.00008
    fc_module                903173                    0.00602
        fc_1                 536164                    0.00357
        fc_2                 342643                    0.00228
        fc_3                  24364                    0.00016
 * The clock frequency of the DL processor is: 150MHz


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


             Deep Learning Processor Profiler Performance Results

                    LastLayerLatency(cycles)   LastLayerLatency(seconds)       FramesNum      Total Latency     Frames/s
                          -------------              -------------               ---------        ---------      ---------
Network                    13570364                    0.09047                     30          380612682            11.8
    conv_module            12667103                    0.08445
        conv_1              3939296                    0.02626
        maxpool_1           1544371                    0.01030
        conv_2              2910747                    0.01940
        maxpool_2            577654                    0.00385
        conv_3              2551829                    0.01701
        maxpool_3            676548                    0.00451
        conv_4               455396                    0.00304
        maxpool_4             11355                    0.00008
    fc_module                903261                    0.00602
        fc_1                 536206                    0.00357
        fc_2                 342688                    0.00228
        fc_3                  24365                    0.00016
 * The clock frequency of the DL processor is: 150MHz


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


             Deep Learning Processor Profiler Performance Results

                    LastLayerLatency(cycles)   LastLayerLatency(seconds)       FramesNum      Total Latency     Frames/s
                          -------------              -------------               ---------        ---------      ---------
Network                    13571561                    0.09048                     30          380608338            11.8
    conv_module            12668340                    0.08446
        conv_1              3939070                    0.02626
        maxpool_1           1545327                    0.01030
        conv_2              2911061                    0.01941
        maxpool_2            577557                    0.00385
        conv_3              2552082                    0.01701
        maxpool_3            676506                    0.00451
```

```
        conv_4              455582                 0.00304
        maxpool_4            11248                 0.00007
    fc_module               903221                 0.00602
        fc_1                536167                 0.00357
        fc_2                342643                 0.00228
        fc_3                 24409                 0.00016
 * The clock frequency of the DL processor is: 150MHz


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


             Deep Learning Processor Profiler Performance Results

                    LastLayerLatency(cycles)   LastLayerLatency(seconds)       FramesNum      Total Latency     Frames/s
                    ------------               ------------                    ---------        ---------       ---------
Network                   13569862                 0.09047                      30          380613327            11.8
    conv_module           12666756                 0.08445
        conv_1             3939212                 0.02626
        maxpool_1          1543267                 0.01029
        conv_2             2911184                 0.01941
        maxpool_2           577275                 0.00385
        conv_3             2552868                 0.01702
        maxpool_3           676438                 0.00451
        conv_4              455353                 0.00304
        maxpool_4            11252                 0.00008
    fc_module               903106                 0.00602
        fc_1                536050                 0.00357
        fc_2                342645                 0.00228
        fc_3                 24409                 0.00016
 * The clock frequency of the DL processor is: 150MHz


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


             Deep Learning Processor Profiler Performance Results

                    LastLayerLatency(cycles)   LastLayerLatency(seconds)       FramesNum      Total Latency     Frames/s
                    ------------               ------------                    ---------        ---------       ---------
Network                   13570823                 0.09047                      30          380619836            11.8
    conv_module           12667607                 0.08445
        conv_1             3939074                 0.02626
        maxpool_1          1544519                 0.01030
        conv_2             2910636                 0.01940
        maxpool_2           577769                 0.00385
        conv_3             2551800                 0.01701
        maxpool_3           676795                 0.00451
        conv_4              455859                 0.00304
        maxpool_4            11248                 0.00007
    fc_module               903216                 0.00602
        fc_1                536165                 0.00357
        fc_2                342643                 0.00228
        fc_3                 24406                 0.00016
 * The clock frequency of the DL processor is: 150MHz


          offset_name          offset_address     allocated_space
    ___________________________    _____________    _______________

    "InputDataOffset"             "0x00000000"     "48.0 MB"
    "OutputResultOffset"          "0x03000000"     "4.0 MB"
    "SystemBufferOffset"          "0x03400000"     "60.0 MB"
    "InstructionDataOffset"       "0x07000000"     "8.0 MB"
    "ConvWeightDataOffset"        "0x07800000"     "8.0 MB"
    "FCWeightDataOffset"          "0x08000000"     "12.0 MB"
    "EndOffset"                   "0x08c00000"     "Total: 140.0 MB"

### FPGA bitstream programming has been skipped as the same bitstream is already loaded on the target FPGA.
### Deep learning network programming has been skipped as the same network is already loaded on the target FPGA.
### Finished writing input activations.
### Running single input activations.


             Deep Learning Processor Profiler Performance Results

                    LastLayerLatency(cycles)   LastLayerLatency(seconds)       FramesNum      Total Latency     Frames/s
                    ------------               ------------                    ---------        ---------       ---------
Network                   13572329                 0.09048                      10          127265075            11.8
```

```
    conv_module           12669135                 0.08446
        conv_1             3939559                 0.02626
        maxpool_1          1545378                 0.01030
        conv_2             2911243                 0.01941
        maxpool_2           577422                 0.00385
        conv_3             2552064                 0.01701
        maxpool_3           676678                 0.00451
        conv_4              455657                 0.00304
        maxpool_4            11227                 0.00007
    fc_module               903194                 0.00602
        fc_1                536140                 0.00357
        fc_2                342688                 0.00228
        fc_3                 24364                 0.00016
 * The clock frequency of the DL processor is: 150MHz


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


             Deep Learning Processor Profiler Performance Results

                    LastLayerLatency(cycles)   LastLayerLatency(seconds)       FramesNum      Total Latency     Frames/s
                    ------------------------   -------------------------       ---------      -------------     ---------
Network                     13572527                 0.09048                     10          127266427            11.8
    conv_module             12669266                 0.08446
        conv_1               3939776                 0.02627
        maxpool_1            1545632                 0.01030
        conv_2               2911169                 0.01941
        maxpool_2             577592                 0.00385
        conv_3               2551613                 0.01701
        maxpool_3             676811                 0.00451
        conv_4                455418                 0.00304
        maxpool_4              11348                 0.00008
    fc_module                 903261                 0.00602
        fc_1                  536205                 0.00357
        fc_2                  342689                 0.00228
        fc_3                   24365                 0.00016
 * The clock frequency of the DL processor is: 150MHz
```

Examine the `MetricResults.Result` field of the validation output to see the performance of the quantized network.

```
validateOut = prediction.MetricResults.Result
```

```
ans =
    NetworkImplementation    MetricOutput
    _____________________    ____________

     {'Floating-Point'}         0.9875
     {'Quantized'     }         0.9875
```

Examine the `QuantizedNetworkFPS` field of the validation output to see the frames per second performance of the quantized network.

```
prediction.QuantizedNetworkFPS
```

```
ans = 11.8126
```

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.

Input Arguments

`quantObj` — Network to quantize

`dlquantizer` object

`dlquantizer` object containing the network to quantize.

`calData` — Data to use for calibration of quantized network

`imageDatastore` object | `augmentedImageDatastore` object | `pixelLabelImageDatastore` object

Data to use for calibration of quantized network, specified as an `imageDatastore` object, an `augmentedImageDatastore` object, or a `pixelLabelImageDatastore` object.

Name-Value Pair Arguments

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`.

Example: `calResults = calibrate(quantObj, calData, 'UseGPU', 'on')`

FPGA Execution Environment Options

`UseGPU` — Logical flag to use GPU for calibration

'off' (default) | 'on'

This property affects FPGA targeting only.

Logical flag to use a GPU for calibration when the `dlquantizer` object `ExecutionEnvironment` is set to FPGA.

Example: `'UseGPU', 'on'`

Output Arguments

`calibrationResults` — 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.

See Also

Apps
Deep Network Quantizer

Functions
`dlquantizationOptions` | `dlquantizer` | `validate`

Topics
“Quantization of Deep Neural Networks”

Introduced in R2020a

validate

Quantize and validate a deep neural network

Syntax

```
validationResults = validate(quantObj, valData)
validationResults = validate(quantObj, valData, quantOpts)
```

Description

`validationResults = 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` and using the data specified by `valData`.

`validationResults = validate(quantObj, valData, quantOpts)` 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`, and the optional argument `quantOpts` that specifies a metric function to evaluate the performance of the quantized network.

To learn about the products required to quantize a deep neural network, see “Quantization Workflow Prerequisites”.

Examples

Quantize a Neural Network

This example shows how to quantize learnable parameters in the convolution layers of a neural network, 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
```

```
net =

  DAGNetwork with properties:

         Layers: [68x1 nnet.cnn.layer.Layer]
    Connections: [75x2 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. Save this function in a local file.

```
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 =

  95x5 table

                Optimized Layer Name                    Network Layer Name       Learnables / Activations    MinValue      MaxVal
    ____________________________________________    ____________________________    ____________________    _________    ______

    {'conv1_relu_conv1_Weights'                }    {'relu_conv1'           }         "Weights"              -0.91985       0.8
    {'conv1_relu_conv1_Bias'                   }    {'relu_conv1'           }         "Bias"                 -0.07925       0.2
```

```
    {'fire2-squeeze1x1_fire2-relu_squeeze1x1_Weights'}    {'fire2-relu_squeeze1x1'}         "Weights"                  -1.38         1.
    {'fire2-squeeze1x1_fire2-relu_squeeze1x1_Bias'   }    {'fire2-relu_squeeze1x1'}         "Bias"                   -0.11641        0.2
    {'fire2-expand1x1_fire2-relu_expand1x1_Weights'  }    {'fire2-relu_expand1x1' }         "Weights"                  -0.7406       0.9
    {'fire2-expand1x1_fire2-relu_expand1x1_Bias'     }    {'fire2-relu_expand1x1' }         "Bias"                   -0.060056       0.1
    {'fire2-expand3x3_fire2-relu_expand3x3_Weights'  }    {'fire2-relu_expand3x3' }         "Weights"                  -0.74397      0.6
    {'fire2-expand3x3_fire2-relu_expand3x3_Bias'     }    {'fire2-relu_expand3x3' }         "Bias"                   -0.051778       0.07
    {'fire3-squeeze1x1_fire3-relu_squeeze1x1_Weights'}    {'fire3-relu_squeeze1x1'}         "Weights"                  -0.77263      0.6
    {'fire3-squeeze1x1_fire3-relu_squeeze1x1_Bias'   }    {'fire3-relu_squeeze1x1'}         "Bias"                     -0.10141      0.3
    {'fire3-expand1x1_fire3-relu_expand1x1_Weights'  }    {'fire3-relu_expand1x1' }         "Weights"                  -0.72131      0.9
    {'fire3-expand1x1_fire3-relu_expand1x1_Bias'     }    {'fire3-relu_expand1x1' }         "Bias"                   -0.067043       0.3
    {'fire3-expand3x3_fire3-relu_expand3x3_Weights'  }    {'fire3-relu_expand3x3' }         "Weights"                  -0.61196      0.7
    {'fire3-expand3x3_fire3-relu_expand3x3_Bias'     }    {'fire3-relu_expand3x3' }         "Bias"                   -0.053612       0.1
    {'fire4-squeeze1x1_fire4-relu_squeeze1x1_Weights'}    {'fire4-relu_squeeze1x1'}         "Weights"                  -0.74145        1.
    {'fire4-squeeze1x1_fire4-relu_squeeze1x1_Bias'   }    {'fire4-relu_squeeze1x1'}         "Bias"                     -0.10886      0.1
...
```

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: [1x1 struct]
```

Examine the `MetricResults.Result` field of the validation output to see the performance of the quantized network.

```
valResults.MetricResults.Result
```

```
ans =

  2x3 table

    NetworkImplementation    MetricOutput    LearnableParameterMemory(bytes)
    _____________________    ____________    _______________________________

     {'Floating-Point'}           1                    2.9003e+06
     {'Quantized'     }           1                    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 FPGA Execution Environment

This example shows how to quantize learnable parameters in the convolution layers of a neural network, and explore the behavior of the quantized network. In this example, you quantize the `LogoNet` neural network. Quantization helps reduce the memory requirement of a deep neural network by quantizing weights, biases and activations of network layers to 8-bit scaled integer data types. Use MATLAB® to retrieve the prediction results from the target device.

To run this example, you need the products listed under FPGA in “Quantization Workflow Prerequisites”.

For additional requirements, see “Quantization Workflow Prerequisites”.

Create a file in your current working directory called `getLogoNetwork.m`. Enter these lines into the file:

```
function net = getLogoNetwork()
    data = getLogoData();
    net  = data.convnet;
end

function data = getLogoData()
    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');
end
```

Load the pretrained network.

```
snet = getLogoNetwork();
```

```
snet =

  SeriesNetwork with properties:

         Layers: [22×1 nnet.cnn.layer.Layer]
     InputNames: {'imageinput'}
    OutputNames: {'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.

This example uses the images in the `logos_dataset` data set. Define an `augmentedImageDatastore` object to resize the data for the network. Then, split the data into calibration and validation data sets.

```
curDir = pwd;
newDir = fullfile(matlabroot,'examples','deeplearning_shared','data','logos_dataset.zip');
copyfile(newDir,curDir);
unzip('logos_dataset.zip');
imageData = imageDatastore(fullfile(curDir,'logos_dataset'),...
 'IncludeSubfolders',true,'FileExtensions','.JPG','LabelSource','foldernames');
[calibrationData, validationData] = splitEachLabel(imageData, 0.5,'randomized');
```

Create a `dlquantizer` object and specify the network to quantize.

```
dlQuantObj = dlquantizer(snet,'ExecutionEnvironment','FPGA');
```

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.

```
 dlQuantObj.calibrate(calibrationData)
```

```
ans =
        Optimized Layer Name        Network Layer Name    Learnables / Activations     MinValue       MaxValue
```

```
    _____________________________    __________________    ________________________    ___________    __________

    {'conv_1_Weights'           }    {'conv_1'    }        "Weights"                   -0.048978     0.039352
    {'conv_1_Bias'              }    {'conv_1'    }        "Bias"                       0.99996        1.0028
    {'conv_2_Weights'           }    {'conv_2'    }        "Weights"                   -0.055518     0.061901
    {'conv_2_Bias'              }    {'conv_2'    }        "Bias"                   -0.00061171       0.00227
    {'conv_3_Weights'           }    {'conv_3'    }        "Weights"                   -0.045942     0.046927
    {'conv_3_Bias'              }    {'conv_3'    }        "Bias"                    -0.0013998     0.0015218
    {'conv_4_Weights'           }    {'conv_4'    }        "Weights"                   -0.045967         0.051
    {'conv_4_Bias'              }    {'conv_4'    }        "Bias"                      -0.00164     0.0037892
    {'fc_1_Weights'             }    {'fc_1'      }        "Weights"                   -0.051394     0.054344
    {'fc_1_Bias'                }    {'fc_1'      }        "Bias"                   -0.00052319    0.00084454
    {'fc_2_Weights'             }    {'fc_2'      }        "Weights"                    -0.05016     0.051557
    {'fc_2_Bias'                }    {'fc_2'      }        "Bias"                    -0.0017564     0.0018502
    {'fc_3_Weights'             }    {'fc_3'      }        "Weights"                   -0.050706       0.04678
    {'fc_3_Bias'                }    {'fc_3'      }        "Bias"                      -0.02951      0.024855
    {'imageinput'               }    {'imageinput'}        "Activations"                      0           255
    {'imageinput_normalization'}     {'imageinput'}        "Activations"                -139.34        198.72
```

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 create the target object, enter:

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

Define a metric function to use to compare the behavior of the network before and after quantization. Save this function in a local file.

```
function accuracy = hComputeAccuracy(predictionScores, net, dataStore)
%% hComputeAccuracy test helper function computes model level accuracy statistics

% Copyright 2020 The MathWorks, Inc.

    % Load ground truth
    groundTruth = dataStore.Labels;

    % 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.

```
options = dlquantizationOptions('MetricFcn', ...
    {@(x)hComputeModelAccuracy(x, snet, validationData)},'Bitstream','arria10soc_int8',...
'Target',hTarget);
```

To compile and deploy the quantized network, run the `validate` function of the `dlquantizer` object. Use the `validate` function to quantize the learnable parameters in the convolution layers of the network and exercise the network. 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 checks for the Intel Quartus tool and the supported tool version. It then starts programming the FPGA device by using the sof file, displays progress messages, and the time it takes to deploy the network. The function uses the metric function defined in the `dlquantizationOptions` object to compare the results of the network before and after quantization.

```
prediction = dlQuantObj.validate(validationData,options);
```

```
          offset_name          offset_address     allocated_space
    _______________________    ______________    _________________

    "InputDataOffset"           "0x00000000"     "48.0 MB"
```

```
    "OutputResultOffset"        "0x03000000"     "4.0 MB"
    "SystemBufferOffset"        "0x03400000"     "60.0 MB"
    "InstructionDataOffset"     "0x07000000"     "8.0 MB"
    "ConvWeightDataOffset"      "0x07800000"     "8.0 MB"
    "FCWeightDataOffset"        "0x08000000"     "12.0 MB"
    "EndOffset"                 "0x08c00000"     "Total: 140.0 MB"

### Programming FPGA Bitstream using JTAG...
### Programming the FPGA bitstream has been completed successfully.
### Loading weights to Conv Processor.
### Conv Weights loaded. Current time is 16-Jul-2020 12:45:10
### Loading weights to FC Processor.
### FC Weights loaded. Current time is 16-Jul-2020 12:45:26
### Finished writing input activations.
### Running single input activations.


             Deep Learning Processor Profiler Performance Results

                  LastLayerLatency(cycles)   LastLayerLatency(seconds)       FramesNum      Total Latency     Frames/s
                  ------------               ------------                    ---------        ---------       ---------
Network                  13570959                    0.09047                    30          380609145            11.8
    conv_module          12667786                    0.08445
        conv_1            3938907                    0.02626
        maxpool_1         1544560                    0.01030
        conv_2            2910954                    0.01941
        maxpool_2          577524                    0.00385
        conv_3            2552707                    0.01702
        maxpool_3          676542                    0.00451
        conv_4             455434                    0.00304
        maxpool_4           11251                    0.00008
    fc_module               903173                    0.00602
        fc_1                536164                    0.00357
        fc_2                342643                    0.00228
        fc_3                 24364                    0.00016
 * The clock frequency of the DL processor is: 150MHz


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


             Deep Learning Processor Profiler Performance Results

                  LastLayerLatency(cycles)   LastLayerLatency(seconds)       FramesNum      Total Latency     Frames/s
                  ------------               ------------                    ---------        ---------       ---------
Network                  13570364                    0.09047                    30          380612682            11.8
    conv_module          12667103                    0.08445
        conv_1            3939296                    0.02626
        maxpool_1         1544371                    0.01030
        conv_2            2910747                    0.01940
        maxpool_2          577654                    0.00385
        conv_3            2551829                    0.01701
        maxpool_3          676548                    0.00451
        conv_4             455396                    0.00304
        maxpool_4           11355                    0.00008
    fc_module               903261                    0.00602
        fc_1                536206                    0.00357
        fc_2                342688                    0.00228
        fc_3                 24365                    0.00016
 * The clock frequency of the DL processor is: 150MHz


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


             Deep Learning Processor Profiler Performance Results

                  LastLayerLatency(cycles)   LastLayerLatency(seconds)       FramesNum      Total Latency     Frames/s
                  ------------               ------------                    ---------        ---------       ---------
Network                  13571561                    0.09048                    30          380608338            11.8
    conv_module          12668340                    0.08446
        conv_1            3939070                    0.02626
        maxpool_1         1545327                    0.01030
        conv_2            2911061                    0.01941
        maxpool_2          577557                    0.00385
        conv_3            2552082                    0.01701
        maxpool_3          676506                    0.00451
        conv_4             455582                    0.00304
        maxpool_4           11248                    0.00007
```

```
    fc_module               903221                 0.00602
        fc_1                536167                 0.00357
        fc_2                342643                 0.00228
        fc_3                 24409                 0.00016
 * The clock frequency of the DL processor is: 150MHz


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


             Deep Learning Processor Profiler Performance Results

                  LastLayerLatency(cycles)   LastLayerLatency(seconds)       FramesNum      Total Latency     Frames/s
                        -------------              -------------               ---------        ---------       ---------
Network                  13569862                 0.09047                     30          380613327            11.8
    conv_module          12666756                 0.08445
        conv_1            3939212                 0.02626
        maxpool_1         1543267                 0.01029
        conv_2            2911184                 0.01941
        maxpool_2          577275                 0.00385
        conv_3            2552868                 0.01702
        maxpool_3          676438                 0.00451
        conv_4             455353                 0.00304
        maxpool_4           11252                 0.00008
    fc_module             903106                 0.00602
        fc_1              536050                 0.00357
        fc_2              342645                 0.00228
        fc_3               24409                 0.00016
 * The clock frequency of the DL processor is: 150MHz


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


             Deep Learning Processor Profiler Performance Results

                  LastLayerLatency(cycles)   LastLayerLatency(seconds)       FramesNum      Total Latency     Frames/s
                        -------------              -------------               ---------        ---------       ---------
Network                  13570823                 0.09047                     30          380619836            11.8
    conv_module          12667607                 0.08445
        conv_1            3939074                 0.02626
        maxpool_1         1544519                 0.01030
        conv_2            2910636                 0.01940
        maxpool_2          577769                 0.00385
        conv_3            2551800                 0.01701
        maxpool_3          676795                 0.00451
        conv_4             455859                 0.00304
        maxpool_4           11248                 0.00007
    fc_module             903216                 0.00602
        fc_1              536165                 0.00357
        fc_2              342643                 0.00228
        fc_3               24406                 0.00016
 * The clock frequency of the DL processor is: 150MHz


          offset_name          offset_address     allocated_space
    _______________________    ______________    _________________

    "InputDataOffset"           "0x00000000"     "48.0 MB"
    "OutputResultOffset"        "0x03000000"     "4.0 MB"
    "SystemBufferOffset"        "0x03400000"     "60.0 MB"
    "InstructionDataOffset"     "0x07000000"     "8.0 MB"
    "ConvWeightDataOffset"      "0x07800000"     "8.0 MB"
    "FCWeightDataOffset"        "0x08000000"     "12.0 MB"
    "EndOffset"                 "0x08c00000"     "Total: 140.0 MB"

### FPGA bitstream programming has been skipped as the same bitstream is already loaded on the target FPGA.
### Deep learning network programming has been skipped as the same network is already loaded on the target FPGA.
### Finished writing input activations.
### Running single input activations.


             Deep Learning Processor Profiler Performance Results

                  LastLayerLatency(cycles)   LastLayerLatency(seconds)       FramesNum      Total Latency     Frames/s
                        -------------              -------------               ---------        ---------       ---------
Network                  13572329                 0.09048                     10          127265075            11.8
    conv_module          12669135                 0.08446
        conv_1            3939559                 0.02626
```

```
        maxpool_1          1545378                0.01030
        conv_2             2911243                0.01941
        maxpool_2           577422                0.00385
        conv_3             2552064                0.01701
        maxpool_3           676678                0.00451
        conv_4              455657                0.00304
        maxpool_4            11227                0.00007
    fc_module               903194                0.00602
        fc_1                536140                0.00357
        fc_2                342688                0.00228
        fc_3                 24364                0.00016
 * The clock frequency of the DL processor is: 150MHz


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


             Deep Learning Processor Profiler Performance Results

                   LastLayerLatency(cycles)   LastLayerLatency(seconds)       FramesNum      Total Latency     Frames/s
                        -------------              -------------               ---------        ---------       ---------
Network                  13572527                0.09048                        10          127266427            11.8
    conv_module          12669266                0.08446
        conv_1            3939776                0.02627
        maxpool_1         1545632                0.01030
        conv_2            2911169                0.01941
        maxpool_2          577592                0.00385
        conv_3            2551613                0.01701
        maxpool_3          676811                0.00451
        conv_4             455418                0.00304
        maxpool_4           11348                0.00008
    fc_module              903261                0.00602
        fc_1               536205                0.00357
        fc_2               342689                0.00228
        fc_3                24365                0.00016
 * The clock frequency of the DL processor is: 150MHz
```

Examine the `MetricResults.Result` field of the validation output to see the performance of the quantized network.

```
validateOut = prediction.MetricResults.Result
```

```
ans =
    NetworkImplementation    MetricOutput
    _____________________    ____________

     {'Floating-Point'}         0.9875
     {'Quantized'     }         0.9875
```

Examine the `QuantizedNetworkFPS` field of the validation output to see the frames per second performance of the quantized network.

```
prediction.QuantizedNetworkFPS
```

```
ans = 11.8126
```

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.

Input Arguments

`quantObj` — Network to quantize

`dlquantizer` object

`dlquantizer` object specifying the network to quantize.

valData — Data to use for validation of quantized network
`imageDataStore` object | `augmentedImageDataStore` object | `pixelLabelImageDataStore` object

Data to use for validation of quantized network, specified as an `imageDataStore` object, an `augmentedImageDataStore` object, or a `pixelLabelImageDataStore` object.

quantOpts — Options for quantizing network
`dlQuantizationOptions` object

Options for quantizing the network, specified as a `dlquantizationOptions` object.

Output Arguments

validationResults — Results of quantization of network
struct

Results of quantization of the network, returned as a struct. The struct contains the following fields.

- `NumSamples` – The number of sample inputs used to validate the network.
- `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 the following fields.

Field	Description
`MetricFunction`	Function used to determine the performance of the quantized network. This function is specified in the `dlquantizationOptions` object.
`Result`	Table indicating the results of the metric function before and after quantization. The first row in the table contains the information for the original, floating-point implementation. The second row contains the information for the quantized implementation. The output of the metric function is displayed in the `MetricOutput` column, and the size of the network is displayed in the `LearnableParameterMemory (bytes)` column.

See Also

Apps
Deep Network Quantizer

Functions
`calibrate` | `dlquantizationOptions` | `dlquantizer`

Topics
“Quantization of Deep Neural Networks”

Introduced in R2020a