

Medical Imaging Toolbox™

User's Guide



MATLAB®

R2023a



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

Medical Imaging Toolbox™ User's Guide

© COPYRIGHT 2022–2023 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

Trademarks

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

Patents

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

Revision History

September 2022	Online only	New for Version 1.0 (Release 2022b)
March 2023	Online Only	Revised for Version 1.1 (Release 2023a)

Getting Started

1

Medical Imaging Toolbox Product Description	1-2
Read, Process, and Write 3-D Medical Images	1-3
Get Started with Medical Image Labeler	1-9
Open Medical Image Labeler App	1-10
Create or Open Labeling Session	1-10
Load Image Data	1-10
Visually Explore Data	1-11
Create Label Definitions	1-11
Label Image	1-12
Export Labeling Results	1-14
How Medical Image Labeler Manages Ground Truth Labels	1-14
Medical Image Coordinate Systems	1-16
Patient Coordinate System	1-16
Intrinsic Coordinate System	1-17
Introduction to Medical Imaging	1-19
Common Medical Imaging Modalities	1-19
Typical Workflow for Medical Image Analysis	1-21

Import, Export, and Spatial Referencing

2

Read, Process, and View Ultrasound Data	2-2
--	-----

Display and Volume Rendering

3

Choose Approach for Medical Image Visualization	3-2
Display 2-D Medical Image Data	3-2
Display 3-D Medical Image Data	3-6
Visualize 3-D Medical Image Data Using Medical Image Labeler	3-11
Display Medical Image Volume in Patient Coordinate System	3-21

Display Labeled Medical Image Volume in Patient Coordinate System	3-24
Create STL Surface Model of Femur Bone for 3-D Printing	3-31
Medical Image-Based Finite Element Analysis of Spine	3-39

Image Preprocessing and Augmentation

4

Medical Image Preprocessing	4-2
Background Removal	4-2
Denoising	4-2
Resampling	4-2
Registration	4-3
Intensity Normalization	4-3
Preprocessing in Advanced Workflows	4-4
Medical Image Registration	4-5
Scenarios for Medical Image Registration	4-5
Functions for Medical Image Registration	4-6
Register Multimodal Medical Image Volumes with Spatial Referencing	4-9

Medical Image Labeling

5

Label 2-D Ultrasound Series Using Medical Image Labeler	5-2
Label 3-D Medical Image Using Medical Image Labeler	5-10
Automate Labeling in Medical Image Labeler	5-20
Collaborate on Multi-Labeler Medical Image Labeling Projects	5-28
Create Label Definitions and Assign Data to Labelers (Project Manager)	5-29
Label Data and Publish Labels for Review (Labeler)	5-30
Inspect Labeled Images (Reviewer)	5-32
Export Ground Truth Data and Send to Project Manager (Labeler)	5-32
Collect, Merge, and Create Training Data (Project Manager)	5-33

Medical Image Segmentation

6

Create Datastores for Medical Image Semantic Segmentation	6-2
Medical Image Ground Truth Data	6-2

Datastores for Semantic Segmentation	6-3
Convert Ultrasound Image Series into Training Data for 2-D Semantic Segmentation Network	6-5
Create Training Data for 3-D Medical Image Semantic Segmentation ...	6-9
Segment Lungs from CT Scan Using Pretrained Neural Network	6-14
Brain MRI Segmentation Using Pretrained 3-D U-Net Network	6-24
Breast Tumor Segmentation from Ultrasound Using Deep Learning ...	6-32
Cardiac Left Ventricle Segmentation from Cine-MRI Images Using U-Net Network	6-40

Getting Started

- “Medical Imaging Toolbox Product Description” on page 1-2
- “Read, Process, and Write 3-D Medical Images” on page 1-3
- “Get Started with Medical Image Labeler” on page 1-9
- “Medical Image Coordinate Systems” on page 1-16
- “Introduction to Medical Imaging” on page 1-19

Medical Imaging Toolbox Product Description

Visualize, register, segment, and label 2D and 3D medical images

Medical Imaging Toolbox provides apps, functions, and workflows for designing and testing diagnostic imaging applications. You can perform 3D rendering and visualization, multimodal registration, and segmentation and labeling of radiology images. The toolbox also lets you train predefined deep learning networks (with Deep Learning Toolbox™).

You can import, preprocess, and analyze radiology images from various imaging modalities, including projected X-ray imaging, computed tomography (CT), magnetic resonance imaging (MRI), ultrasound (US), and nuclear medicine (PET, SPECT). The Medical Image Labeler app lets you semi-automate 2D and 3D labeling for use in AI workflows. You can perform multimodal registration of medical images, including 2D images, 3D surfaces, and 3D volumes. The toolbox provides an integrated environment for end-to-end computer-aided diagnosis and medical image analysis.

Read, Process, and Write 3-D Medical Images

This example shows how to import and display volumetric medical image data, apply a filter to the image, and write the processed image to a new file. You can use the `medicalVolume` object to import image data and spatial information about a 3-D medical image in one object.

Download Image Volume Data

This example uses one chest CT volume saved as a directory of DICOM files. The volume is part of a data set containing three CT scans. The size of the entire data set is approximately 81 MB.

Run this code to download the data set from the MathWorks® website and unzip the folder.

```
zipFile = matlab.internal.examples.downloadSupportFile("medical", "MedicalVolumeDICOMData.zip");
filepath = fileparts(zipFile);
unzip(zipFile, filepath)
```

The `dataFolder` folder contains the downloaded and unzipped data.

```
dataFolder = fullfile(filepath, "MedicalVolumeDICOMData", "LungCT01");
```

Read Image File

The `medicalVolume` object imports data from the DICOM, NIFTI, and NRRD medical image file formats. DICOM volumes can be stored as a single file or as a directory containing individual files for each 2-D slice. The `medicalVolume` object automatically detects the file format and extracts the image data, spatial information, and modality from the file metadata. For this example, specify the data source as the download directory of the chest CT scan.

```
medVol = medicalVolume(dataFolder)
```

```
medVol =
  medicalVolume with properties:
      Voxels: [512x512x88 int16]
  VolumeGeometry: [1x1 medicalref3d]
    SpatialUnits: "mm"
      Orientation: "transverse"
    VoxelSpacing: [0.7285 0.7285 2.5000]
    NormalVector: [0 0 1]
  NumCoronalSlices: 512
  NumSagittalSlices: 512
  NumTransverseSlices: 88
    PlaneMapping: ["sagittal" "coronal" "transverse"]
      Modality: "CT"
  WindowCenters: [88x1 double]
  WindowWidths: [88x1 double]
```

The `Voxels` property contains the image intensity values. If the file metadata specifies the rescale intercept and slope, `medicalVolume` automatically rescales the voxel intensities to the specified units. In this example, the CT intensity values are rescaled to Hounsfield units.

```
intensities = medVol.Voxels;
```

The `VolumeGeometry` property contains a `medicalref3d` object that defines the spatial referencing for the image volume, including the mapping between the intrinsic and patient coordinate systems.

The intrinsic coordinate system is defined by the rows, columns, and slices of the `Voxels` array, with coordinates in voxel units. The patient coordinate system is defined relative to the anatomical axes of the patient, with real-world units such as millimeters.

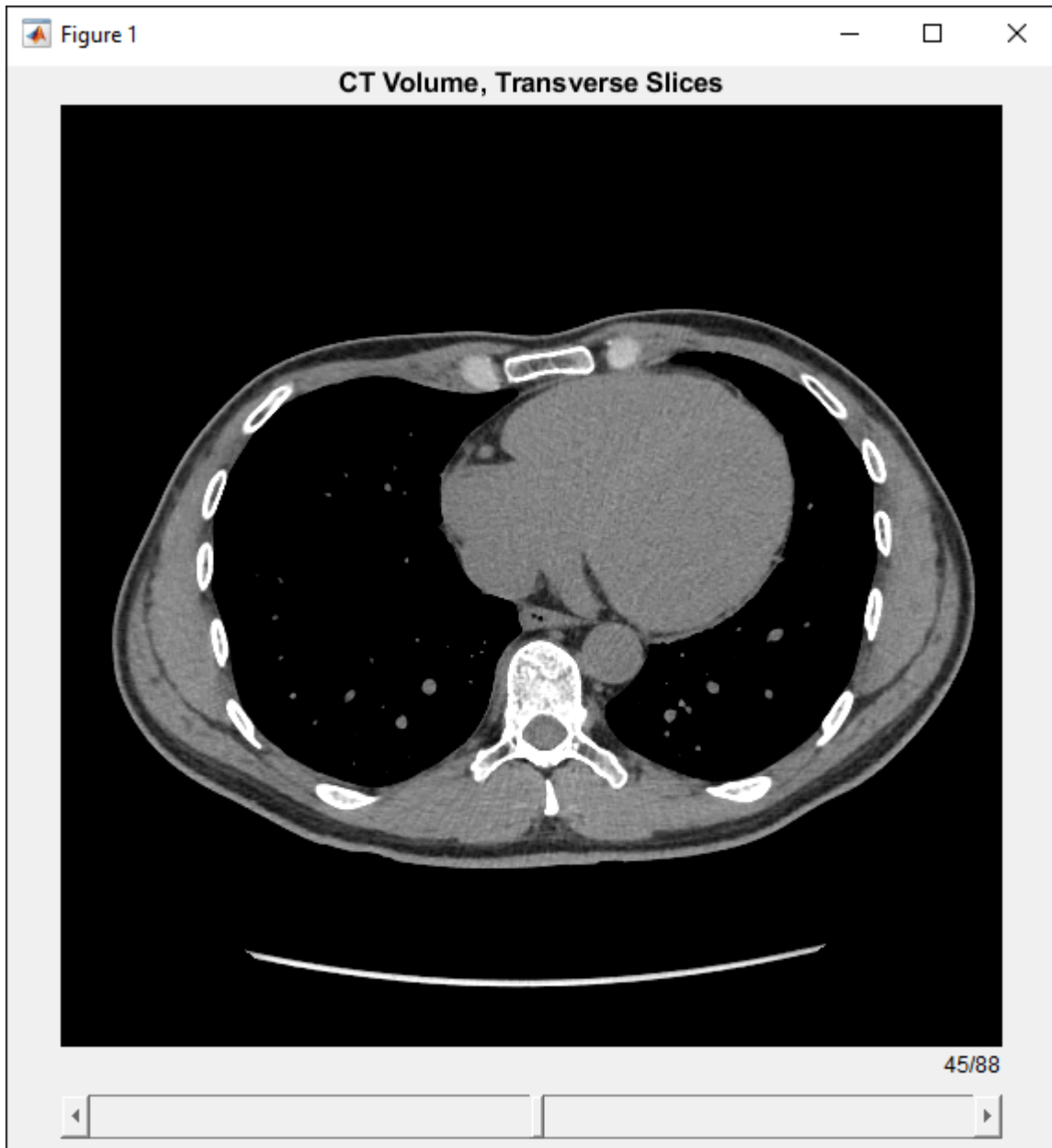
```
R = medVol.VolumeGeometry
```

```
R =  
  medicalref3d with properties:  
  
      VolumeSize: [512 512 88]  
      Position: [88x3 double]  
      VoxelDistances: {[88x3 double] [88x3 double] [88x3 double]}  
      PatientCoordinateSystem: "LPS+"  
      PixelSpacing: [88x2 double]  
      IsAffine: 1  
      IsAxesAligned: 1  
      IsMixed: 0
```

Display Volume as Slices

Medical Imaging Toolbox™ provides several options for visualizing medical image data. For details, see “Choose Approach for Medical Image Visualization” on page 3-2. For this example, display the transverse slices of the CT volume by creating a `sliceViewer` object. By default, the viewer opens to display the center slice. Use the slider to navigate through the volume.

```
sliceViewer(medVol)  
title("CT Volume, Transverse Slices")
```



Display the Volume in 3-D

Display the CT volume as a 3-D object by using the `volshow` function. The `volshow` function uses the spatial details in `medVol` to set the `Transformation` property of the output `Volume` object and display the volume in the patient coordinate system.

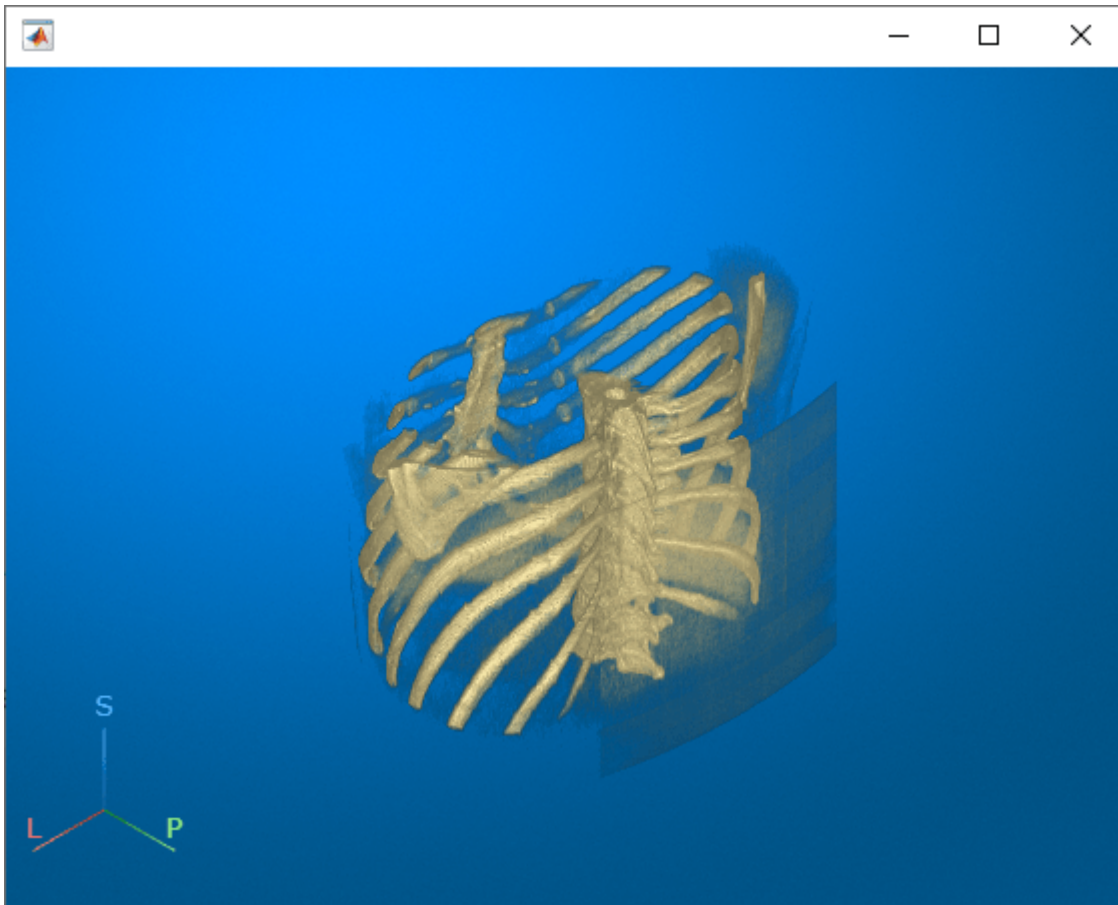
You can customize the volume display by setting properties of the `Volume` object. Specify a custom transparency map and colormap that highlight the rib cage. The `alpha` and `color` values are based on the CT-bone rendering style from the Medical Image Labeler app. The intensity values have been tuned for this volume using trial and error.

```
alpha = [0 0 0.72 0.72];  
color = [0 0 0; 231 208 141; 231 208 141; 255 255 255]/255;
```

```
intensity = [-3024 50 1400 1499];  
queryPoints = linspace(min(intensity),max(intensity),256);  
alphamap = interp1(intensity,alpha,queryPoints)';  
colormap = interp1(intensity,color,queryPoints);
```

Display the volume with the custom display settings.

```
vol = volshow(medVol,Colormap=colormap,Alphamap=alphamap);
```



Smooth Voxel Intensity Data

Smooth the image with a 3-D Gaussian filter. Applying a Gaussian filter is one approach for reducing noise in medical images.

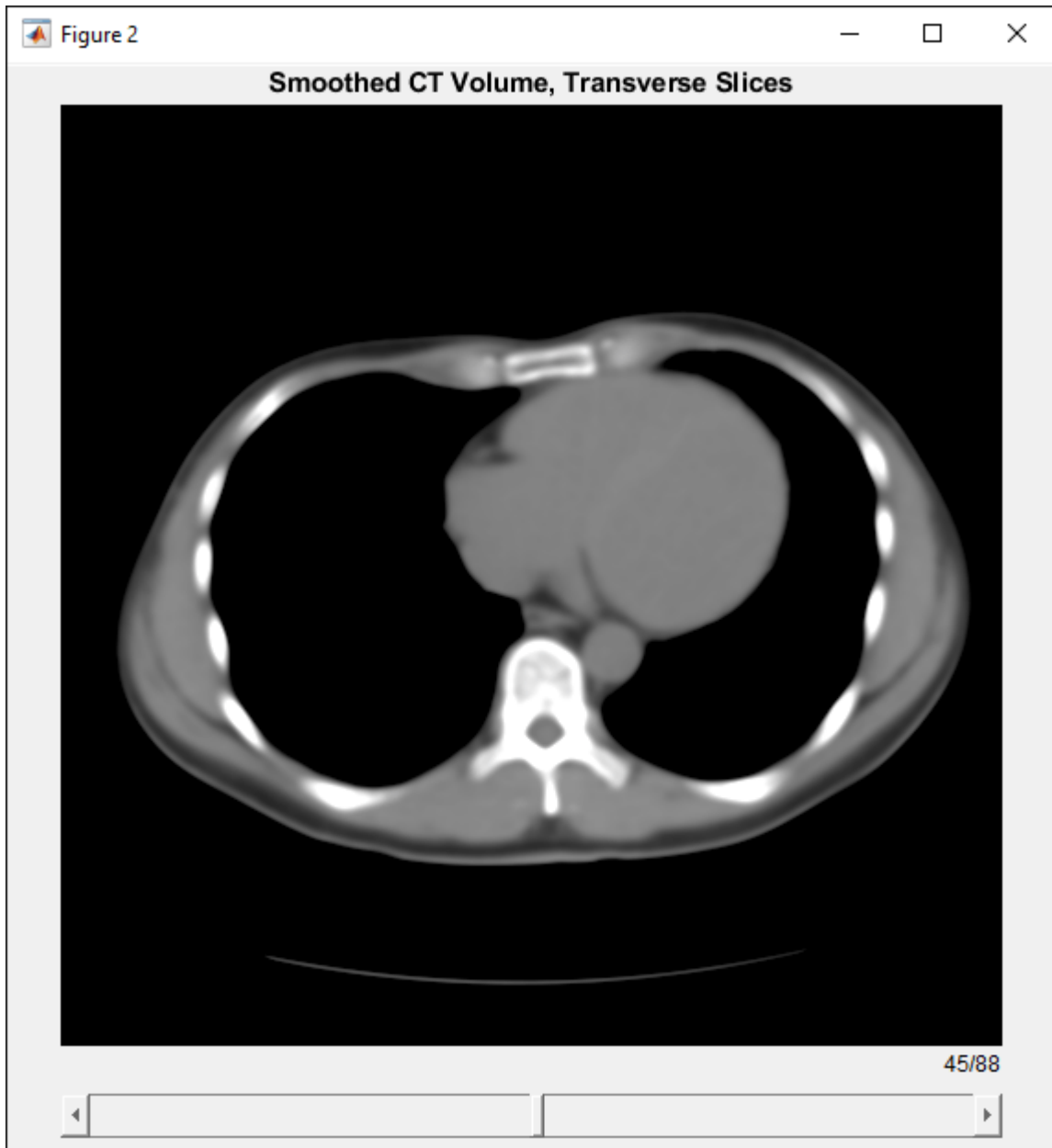
```
sigma = 2;  
intensitiesSmooth = imgaussfilt3(intensities,sigma);
```

Create a new `medicalVolume` object that contains the smoothed voxel intensities and preserves the spatial referencing of the original file. Create a copy of the original object `medVol` and set the `Voxels` property of the new object, `medVolSmooth`, to the smoothed image data.

```
medVolSmooth = medVol;  
medVolSmooth.Voxels = intensitiesSmooth;
```

Display the smoothed voxel data.

```
figure
sliceViewer(medVolSmooth)
title("Smoothed CT Volume, Transverse Slices")
```



Write Processed Data to New NIFTI File

Write the smoothed image data to a new NIFTI file by using the `write` object function. The function supports writing medical volume data in only the NIFTI file format.

```
niftiFilename = "LungCT01_smoothed.nii";
write(medVolSmooth,niftiFilename)
```

Read the new file using `medicalVolume`. Because the NIfTI file format does not contain metadata related to modality or display windows, the `Modality` property value is "unknown" and the `WindowCenters` and `WindowWidths` properties are empty.

```
medVolNIfTI = medicalVolume(niftiFilename);
```

See Also

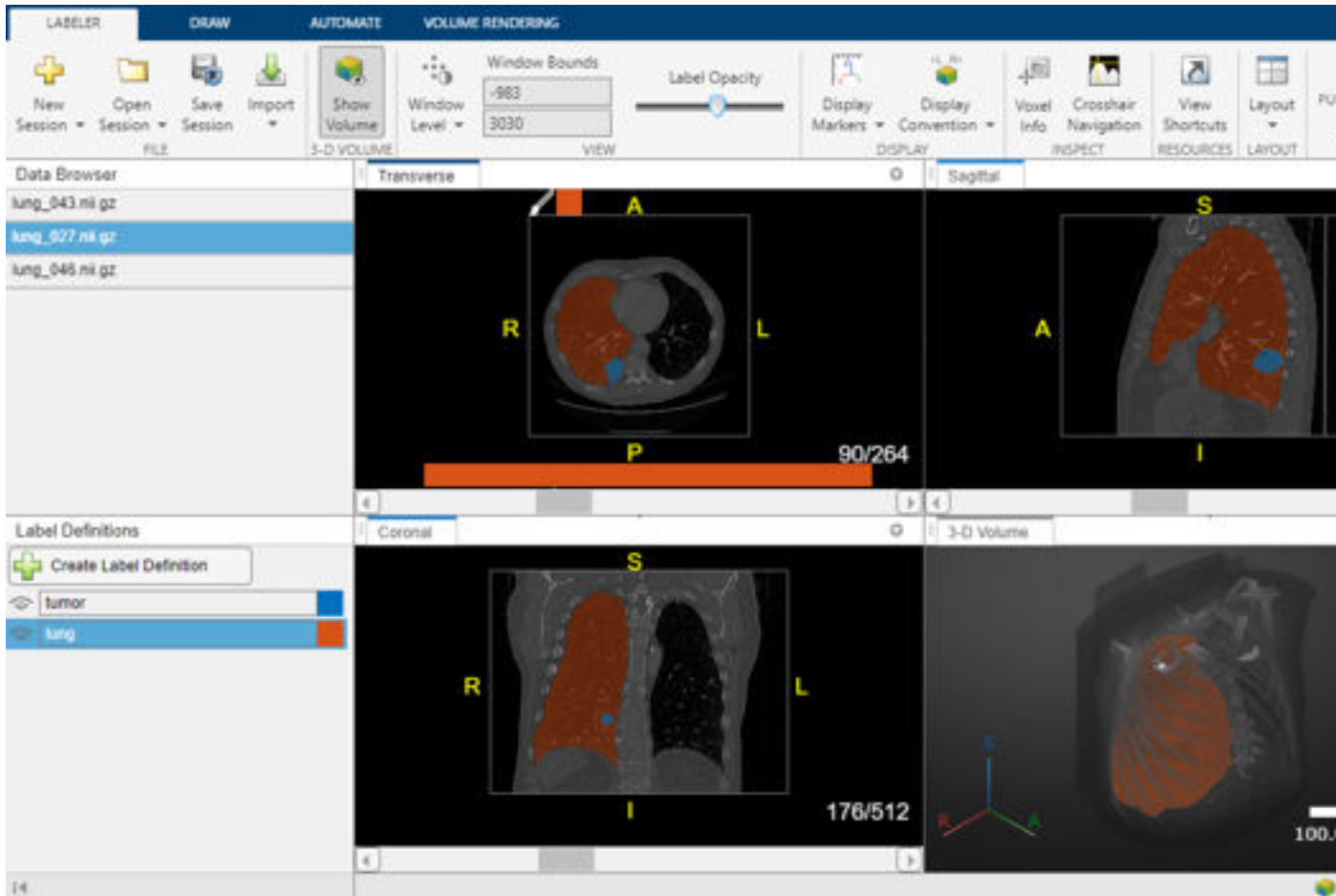
`medicalVolume` | `medicalref3d` | `extractSlice` | `sliceLimits`

More About

- "Medical Image Coordinate Systems" on page 1-16

Get Started with Medical Image Labeler

The **Medical Image Labeler** app enables you to explore and interactively label pixels in 2-D and 3-D medical images. You can export labeled data as a `groundTruthMedical` object to train semantic segmentation algorithms. You can publish snapshot images and animations with or without labels.



This tutorial provides an overview of the capabilities of the **Medical Image Labeler** and compares 2-D and 3-D image labeling. The typical app workflow includes these steps:

- 1 "Open Medical Image Labeler App" on page 1-10
- 2 "Create or Open Labeling Session" on page 1-10
- 3 "Load Image Data" on page 1-10
- 4 "Visually Explore Data" on page 1-11
- 5 "Create Label Definitions" on page 1-11
- 6 "Label Image" on page 1-12
- 7 "Export Labeling Results" on page 1-14

Open Medical Image Labeler App

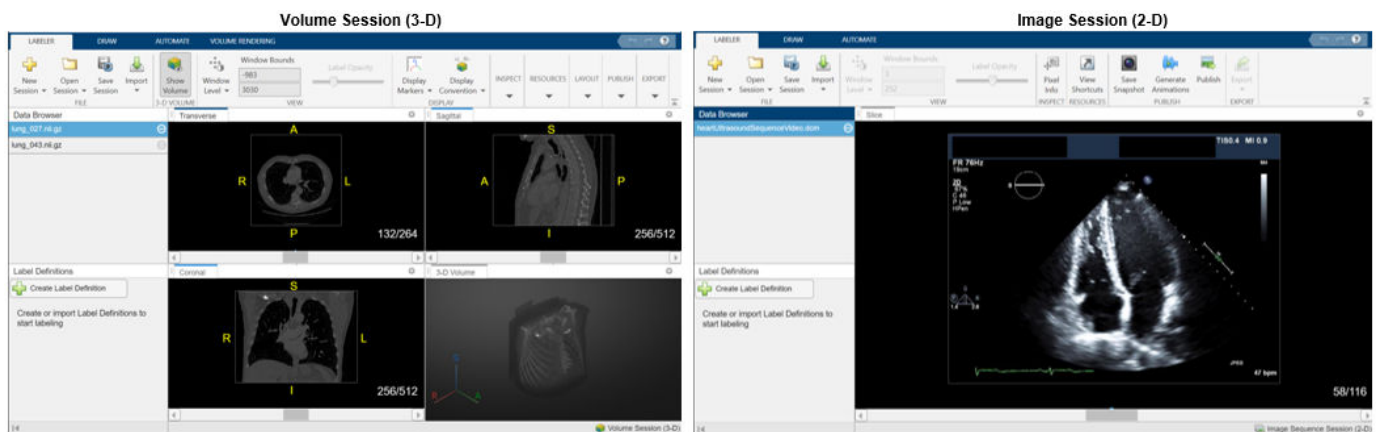
Open the **Medical Image Labeler** app from the **Apps** tab on the MATLAB® toolstrip, under **Image Processing and Computer Vision**. You can also load the app by using the `medicalImageLabeler` command.

Create or Open Labeling Session

Manage labeling in the **Medical Image Labeler** using app sessions. Within one app session, you can import and label multiple image files. These might be repeat scans from one patient or scans from multiple patients with the same set of tissues, organs, or other regions of interest to label. The app enables you to create either a volume session or an image session. Use a volume session to label 3-D medical image data. Use an image session to label 2-D images or an image series, such as an ultrasound video.

You can either create a new labeling session or reopen a previous session:

- **New session** — On the app toolstrip, click **New Session** and select **New Volume session (3-D)** or **New Image session (2-D)**. In the dialog box that opens, specify a session folder. As you draw the label images, the app automatically saves them to the session folder. Therefore, the session folder must have access to enough memory to save all label images for the session. For more details, see the “How Medical Image Labeler Manages Ground Truth Labels” on page 1-14 section.
- **Open session** — On the app toolstrip, click **Open Session** and select one of the listed recent sessions, or click **Open Session** and navigate to a previous session folder.



Load Image Data

Load images to label from a file or from a `groundTruthMedical` object. Load image files when starting a new labeling project. Use `groundTruthMedical` objects when working with labeled or partially labeled data from outside of MATLAB or that have been shared from a different workstation. For more details about working on a multi-person labeling team, see “Collaborate on Multi-Labeler Medical Image Labeling Projects” on page 5-28.

To load an image from a file, click **Import** and, under **Data**, select **From File**. The app supports loading these file formats:

- **Volume session** — Single NIFTI, NRRD, or DICOM file, or a directory containing multiple DICOM files corresponding to one image volume.

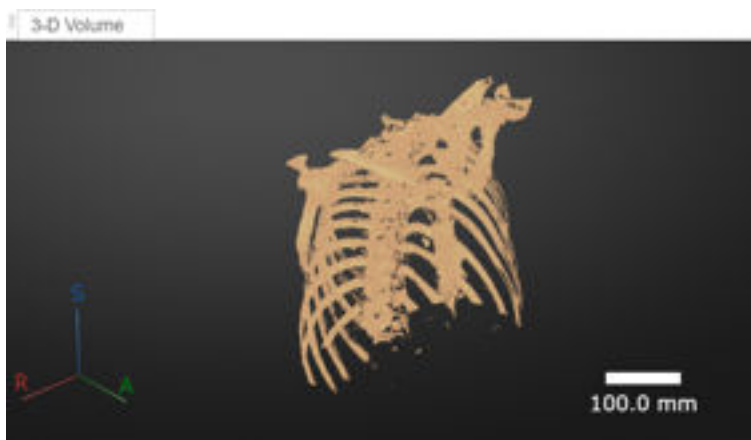
- Image session — Single NIFTI or DICOM file.

To load a `groundTruthMedical` object, click **Import**, and under **Ground Truth**, select **From File** to load the object from a MAT file or **From Workspace** to load the object from the MATLAB workspace. Importing a `groundTruthMedical` object loads the image data, label data, and label definitions stored in the object into the app.

The **Data Browser** pane lists all of the image files currently loaded in the app. Click a filename in the **Data Browser** to change the file to display and label.

Visually Explore Data

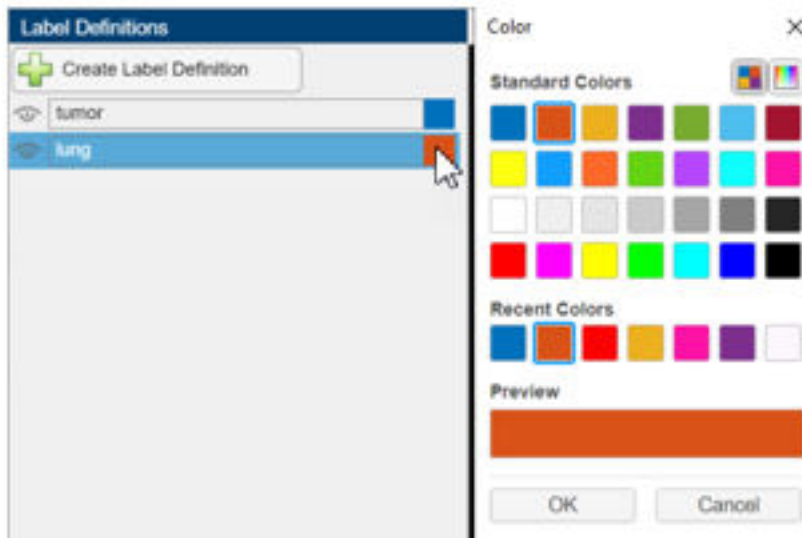
The app displays 3-D image data using individual panes for the **Transverse**, **Sagittal**, and **Coronal** slice planes and a **3-D Volume** pane. For an example of how to customize the display of 3-D images, and publish images and animations, see “Visualize 3-D Medical Image Data Using Medical Image Labeler” on page 3-11. The app displays 2-D image data in the **Slice** pane. For details about navigating frames or adjusting the brightness and contrast of a 2-D image series, see “Label 2-D Ultrasound Series Using Medical Image Labeler” on page 5-2.



Create Label Definitions

A label definition specifies the name, color, and order of each label assigned to the image. You must use the same label definitions across all images within an app session. The **Medical Image Labeler** app supports pixel labeling.

You can create a label definition interactively in the app by clicking **Create Label Definition** in the **Label Definitions** pane. Optionally, you can assign a name for the label by clicking it, or change the color of the label by clicking the color square next to the label name.



Alternatively, import label definitions from a label definitions file or as part of a `groundTruthMedical` object. You can create a label definitions file programmatically, or load one exported from a previous app session. For more details about creating a label definitions file programmatically, see the `LabelDefinitions` property of the `groundTruthMedical` object.

Label Image

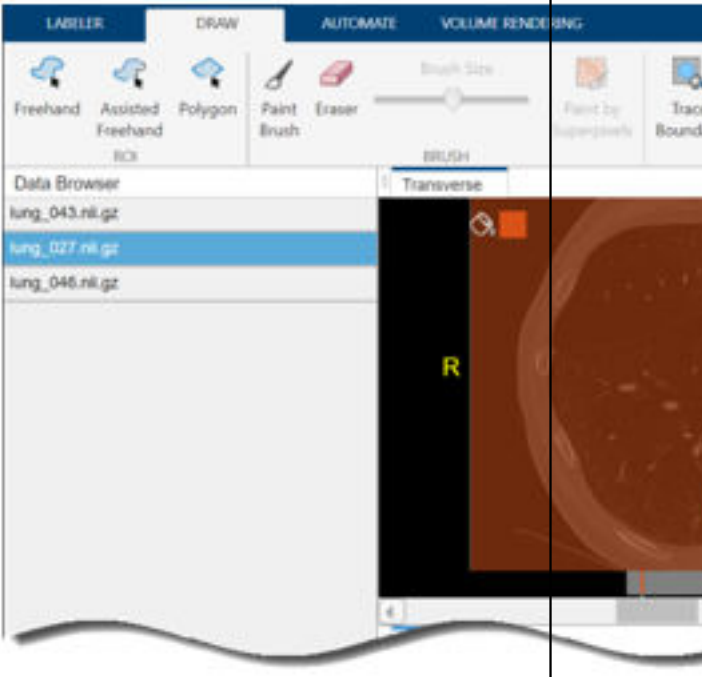
Label pixels using the tools in the **Draw** and **Automate** tabs of the app toolbar. The app provides manual, semi-automated, and automated labeling tools.

Manual Labeling

Manually draw labels using the **Freehand**, **Assisted Freehand**, **Polygon**, and **Paintbrush** tools.

Semi-Automated Labeling

Add labels using semi-automated tools including **Fill Region**, **Paint by Superpixels**, and **Trace Boundary**. You can interpolate labeled regions between image frames or volume slices using the **Auto Interpolate** and **Manually Interpolate** tools.

Tool	Description	Image
Fill Region	Flood fill label image or fill holes in label region.	
Paint by Superpixels	Manually paint within an adjustable-sized grid of pixels. To use this tool, first select Paint Brush and then click Paint by Superpixels . Each superpixel contains a cluster of similar intensity values. Adjust the Superpixels Size to change the size of the superpixel grid.	
Trace Boundary	Label connected regions that have similar intensity values. Select Trace Boundary in the app toolbar, and then pause on a seed pixel or voxel in the region you want to label. The tool predicts the boundary of the region by including pixels or voxels with intensities that are similar to the current seed. Use the Threshold slider to adjust the similarity threshold used to predict the region boundary. Increasing the threshold includes a wider range of intensities above and below the seed value in the predicted region. Move your cursor to change the seed.	

Automated Labeling

The **Automate** tab contains built-in automation algorithms to refine existing labels or fully automate labeling. The app provides slice-based algorithms including **Active Contours**, **Adaptive Threshold**, **Dilate**, and **Erode**. In a volume session, the app additionally provides the **Filter and Threshold**, **Smooth Edges**, and **Otsu's Threshold** algorithms, which you can apply to all slices or to a specified slice range.

You can add a custom automation algorithm to use in the app. On the **Automate** tab, click **Add Algorithm**. Import an existing algorithm by selecting **From File**, or create a new algorithm using the provided function or class template. See “Automate Labeling in Medical Image Labeler” on page 5-20 for an example that applies a custom automation algorithm for 2-D labeling.

Labeling Examples

- For an example of labeling 2-D image data, see “Label 2-D Ultrasound Series Using Medical Image Labeler” on page 5-2.
- For an example of labeling 3-D image data, see “Label 3-D Medical Image Using Medical Image Labeler” on page 5-10.

Export Labeling Results

You can export the `groundTruthMedical` object and label definitions as files to share with a colleague.

- To export the `groundTruthMedical` object as a MAT file, on the **Labeler** tab, click **Export** and, under **Ground Truth**, select **To File**.
- To export the label definitions as a MAT file, on the **Labeler** tab, click **Export** and, under **Label Definitions**, select **To File**.

Note The app stores the actual pixel label data in the `LabelData` subfolder of the session folder. See “How Medical Image Labeler Manages Ground Truth Labels” on page 1-14 for details.

How Medical Image Labeler Manages Ground Truth Labels

As you label images in the **Medical Image Labeler** app, the app automatically saves three sets of data in the session folder associated with the current app session.

- A `groundTruthMedical` object stored as a MAT file. The `groundTruthMedical` object specifies the file locations of the unlabeled images and corresponding label images, as well as the name and color associated with each label.
- A subfolder named `LabelData`, which contains the label images.

The app saves pixel label images created in an image session in the `LabelData` folder as MAT files. A MAT file stores the pixel labels as a `uint8` array. You can read the label image into the MATLAB workspace by using the `load` function.

The app saves voxel label images created in a volume session in the `LabelData` folder as NIFTI files. A NIFTI file stores the voxel labels as a `uint8` array. You can read the label images into the MATLAB workspace by using the `niftiread` function.

- A subfolder named `AppData`, which contains data about the app session stored as a MAT file.

See Also

Medical Image Labeler | `groundTruthMedical`

Related Examples

- “Visualize 3-D Medical Image Data Using Medical Image Labeler” on page 3-11
- “Label 2-D Ultrasound Series Using Medical Image Labeler” on page 5-2
- “Label 3-D Medical Image Using Medical Image Labeler” on page 5-10
- “Automate Labeling in Medical Image Labeler” on page 5-20
- “Collaborate on Multi-Labeler Medical Image Labeling Projects” on page 5-28

Medical Image Coordinate Systems

In medical imaging, there are two distinct coordinate systems: the *intrinsic* coordinate system and the *world*, or *patient*, coordinate system. You can access locations in medical images using the intrinsic coordinate system and the patient coordinate system.

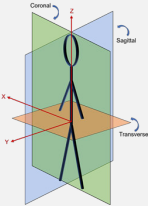



The intrinsic coordinate system defines the voxel space, and the patient coordinate system defines the anatomical space. To understand the position and orientation of intrinsic coordinates with respect to the patient, you must transform the data into the patient coordinate system. You can use the `intrinsicToWorldMapping` function to obtain the transformation between the intrinsic coordinates and patient coordinates of a 3-D medical image volume stored as a `medicalVolume` object.

Patient Coordinate System

The patient coordinate system is made up of three orthogonal axes:

- Left (L)/Right (R) — *x*-axis
- Anterior (A)/Posterior (P) — *y*-axis
- Inferior (I)/Superior (S) — *z*-axis

The patient xyz-axes define the coronal, sagittal, and transverse anatomical planes. This table shows the relationship between the anatomical planes and patient axes.

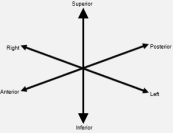
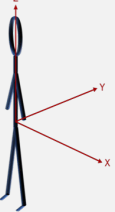
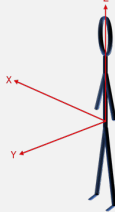
Anatomical Planes and Patient Axes	Sagittal Plane	Coronal Plane	Transverse Plane
	<ul style="list-style-type: none"> • Defined by the <i>y</i>- and <i>z</i>-axes. • Divides the body into right and left segments. 	<ul style="list-style-type: none"> • Defined by the <i>x</i>- and <i>z</i>-axes. • Divides the body into anterior and posterior segments. 	<ul style="list-style-type: none"> • Defined by the <i>x</i>- and <i>y</i>-axes. • Divides the body into inferior and superior segments. 

Note The patient coordinate system rotates together with the physical orientation of the patient.

Mapping Patient Coordinate Axes to Anatomical Planes

Medical image files store a transformation matrix to map intrinsic coordinates (i, j, k) to patient coordinates (x, y, z), and each file format has a different convention that defines the positive direction of each axis.

For example, a DICOM file uses an LPS+ coordinate system and a NIFTI file uses an RAS+ coordinate system.

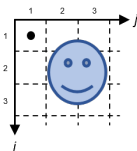
Patient Axes	DICOM	NIFTI
	<ul style="list-style-type: none"> • x-axis values increase from right to left • y-axis values increase from anterior to posterior • z-axis values increase from inferior to superior 	<ul style="list-style-type: none"> • x-axis values increase from left to right • y-axis values increase from posterior to anterior • z-axis values increase from inferior to superior
		

Intrinsic Coordinate System

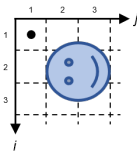
The intrinsic coordinate system describes the spatial dimensions of the patient coordinate system. Intrinsic coordinates are in units of voxels, while patient coordinates have real-world dimensions and are usually in units of millimeters. You can obtain pixel dimensions from the `PixelSpacing` property of a `medicalImage` object for 2-D data, and from the `VoxelSpacing` property of a `medicalVolume` object for 3-D data.

The origin of the intrinsic coordinate system is located at the center of the first pixel (2-D image) or voxel (3-D volume), represented by the black circle. The *i*-axis corresponds to the first dimension (rows), the *j*-axis corresponds to the second dimension (columns), and the *k*-axis corresponds to the third dimension of a medical image or volume.

This image grid shows the *i*-axis corresponding to the anatomical *z*-axis and the *j*-axis corresponding to the anatomical *x*-axis,



whereas this image grid shows the *i*-axis corresponding to the anatomical *x*-axis and the *j*-axis corresponding to the anatomical *z*-axis.



Both image grids represent a valid way a medical device might store voxels in a file. When you create a `medicalVolume` object for an image volume, the spatial dimensions of the patient coordinate system correspond to the values in the `Voxels` property of the object.

Tip Use the `intrinsicToWorldMapping` function to compute the geometric transformation between the intrinsic and patient coordinate systems for a medical image volume.

See Also

Objects

`medicalref3d` | `medicalVolume`

Functions

`intrinsicToWorldMapping`

Related Examples

- “Read, Process, and Write 3-D Medical Images” on page 1-3
- “Display Medical Image Volume in Patient Coordinate System” on page 3-21

Introduction to Medical Imaging

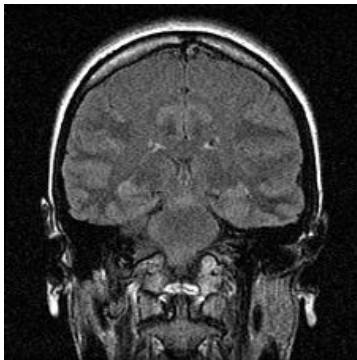
Medical imaging is the acquisition and processing of images of the human body for clinical applications. You can use medical image processing to improve the quality of medical images, for diagnosis of medical conditions, for surgical planning, or for research. Medical imaging enables the detailed, yet non-invasive, study of human anatomy. The success of medical imaging requires collaboration between medical professionals such as radiologists, pathologists, or clinicians, and technology professionals skilled in image processing. The major types of medical images you can process for clinical applications are radiology images, such as MRI scans, CT scans, X-ray scans, ultrasound scans, or PET/SPECT scans, or pathological microscopy images, such as biopsies and blood smears.

You can analyze radiology images for a variety of applications.

- **Diagnostic Systems:** For example, to detect tumors from brain MRI scans, to detect COVID-19 from CT scans, to detect pneumonia from chest X-ray scans, or to detect tumors from breast ultrasound.
- **Biomedical Engineering:** For example, to model bones or to design prostheses.
- **Functional Analysis:** For example, to analyze brain function from functional MRI.
- **Pharmaceutical Research:** For example, to measure drug efficacy and clearance time.
- **Device Design:** For example, to build new MRI, CT, ultrasound devices.

You can use Medical Imaging Toolbox to analyze radiology images for such applications. Although the functions in Medical Imaging Toolbox are modality-agnostic, the major medical imaging modalities that you can use the toolbox for include MRI, CT, X-ray, ultrasound, and PET/SPECT.

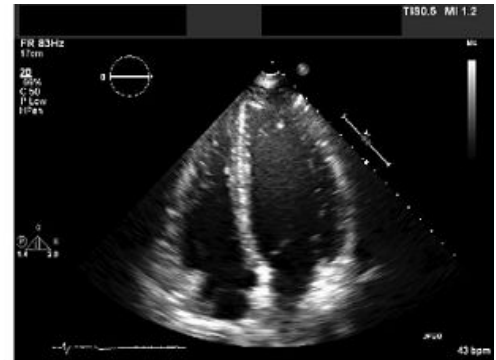
Common Medical Imaging Modalities



MRI



CT



Ultrasound

Magnetic Resonance Imaging (MRI)

The human body consists mostly of water, and therefore contains many hydrogen nuclei. MRI scans acquire an image by disturbing the magnetic equilibrium of the hydrogen nuclei in the body. The scanner then measures the time taken by the hydrogen nuclei to regain equilibrium, which varies based on the composition of the organ being imaged. MRI scans are particularly useful for imaging soft tissues such as the brain, spinal cord, nerves, muscles, ligaments, and tendons, as soft tissues have more water content than bone. Unlike other radiology modalities, MRI scans do not use any ionizing radiation. However, medical professionals must ensure that the patient undergoing the scan

does not have any metal on or in their body that might be attracted to the magnetic field. There are several forms of MRI, based on the nature of particles and the type of magnetization property measured, including T1-weighted MRI, T2-weighted MRI, diffusion MRI, and functional MRI. The different types of MRI provide different insights into the human body.

Mathematically, an MRI scanner generates an image in the Fourier domain also known as k-space. Each scan typically consists of a collection of 2-D slices imaged in k-space. The scanner transforms the k-space image to the spatial domain, enabling you to observe the imaged anatomy. The final output of the MRI scanner is a 3-D volume in the spatial domain with spatial localization details. You can use a `medicalVolume` object to store the voxel data and spatial referencing information for the MRI volume. MRI images are prone to degradations in the form of acquisition noise, undersampling artifacts, and patient motion artifacts.

Computed Tomography (CT)

CT scans use X-ray radiation to image human anatomy. The magnitude of attenuation of the radiation depends on the organ being imaged. Because bones effectively block X-rays, CT scans image them particularly well. You can image tissues in the human body, using contrast agents, which help attenuate the X-rays. As a result, CT scans are versatile and can be used for imaging of the head and neck, as well as organs such as heart, lungs, abdomen, and pelvis. Additionally, CT scans are fast and cost-effective for patients.

Mathematically, a CT scanner reconstructs the image from a series of projections obtained using the Radon transform, typically represented as a sinogram. The Radon transform produces a collection of projections of radiation through the body along different angles. There are a variety of techniques to reconstruct an image from the projection data, including the inverse Radon transformation and other iterative methods, enabling you to observe the imaged anatomy. The final output of the CT scanner is a 3-D volume in the spatial domain with spatial localization details. You can use a `medicalVolume` object to store the voxel data and spatial referencing information for the CT volume. CT images are prone to degradations in the form of low contrast and artifacts due to miscalibration of the X-ray detectors.

X-Ray Imaging

X-ray imaging is a direct digitized recording of the attenuation of the X-ray radiation on a 2-D sensor array or a radiographic film. It is fast and cost-effective compared to other scans. Thus, it is a good option for preliminary diagnosis. Mathematically, an X-ray image is a simple 2-D image captured by an X-ray detector. You can use a `medicalImage` object to store the pixel data and metadata for an X-ray image.

Ultrasound (US)

Ultrasound imaging involves emitting of ultrasound waves and measuring the strength of the reflected echo waves. The strength of the reflected echo waves depends on the organ being imaged. Because air can block ultrasound waves, it is not suitable for imaging bones, or tissues that contain air such as lungs. You can use ultrasound imaging to, for example, monitor the development of a fetus during pregnancy, or for imaging the heart, breast, and abdomen. You can use Doppler ultrasound for functional imaging of the blood flow in the blood vessels.

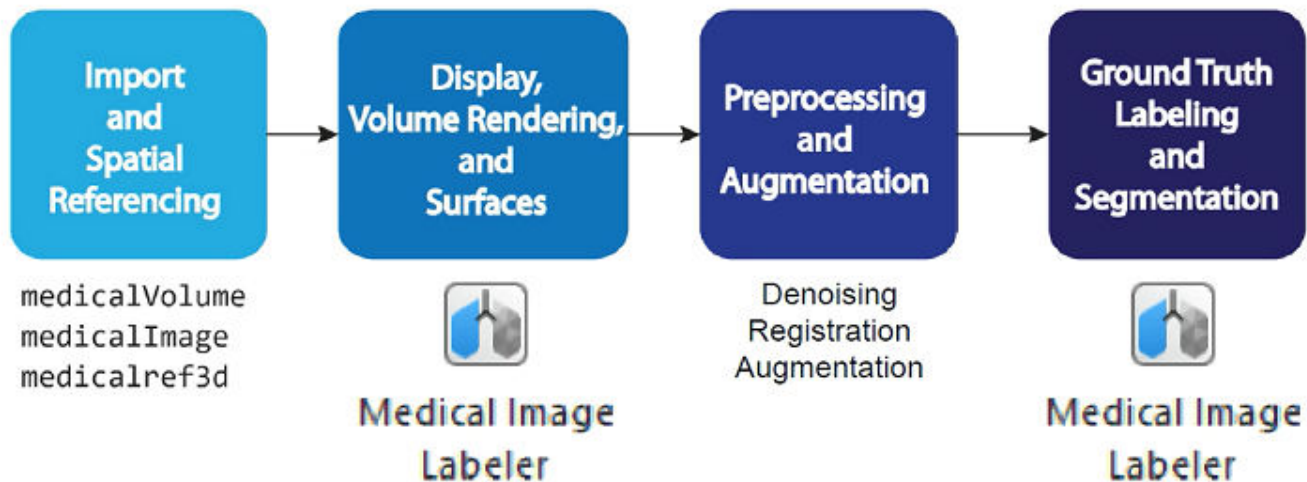
Mathematically, an ultrasound image is derived from the reflected ultrasound waves. The final output of the ultrasound scan is a sequence of 2-D images in the spatial domain. You can use a `medicalImage` object to store the pixel data and metadata for an ultrasound image sequence. Note that the emitted and reflected ultrasound waves can cause interference, which can show up as speckle in the ultrasound image.

Nuclear Medicine Imaging

Nuclear medicine imaging involves introducing radioactive tracers, also known as radiotracers or radiopharmaceuticals, that contain radioactive isotopes into the body of the patient. The movement of the radiotracers in the body of the patient provides insights about the organs being imaged. Different types of nuclear medicine imaging employ different radiotracers and are used for different purposes. Mathematically, nuclear medicine imaging is performed as a tomography. The decay of the radiotracers emits radiation, and the scanner measures the attenuation of the radiation in the tomography. The final output of the scanner is a 3-D volume in the spatial domain with spatial localization details. You can use a `medicalVolume` object to store the voxel data and spatial referencing information for a 3-D volume.

Positron emission tomography (PET) and single-photon emission computed tomography (SPECT) use different types of radiotracers for imaging. The decay of the radiotracers used in PET emits positrons. PET is used primarily for diagnosis and tracking of cancer. The decay of the radiotracers used in SPECT emits gamma rays. SPECT is used primarily for diagnosis and tracking of heart disease.

Typical Workflow for Medical Image Analysis



Import and Spatial Referencing

A typical medical imaging workflow begins with importing the medical images into the workspace. Medical images are available in file formats such as NIFTI, DICOM, NRRD, Analyze7.5, and Interfile. Medical Imaging Toolbox provides several functions you can use to import medical images into the workspace and export them back to medical image formats after processing. For more information, see “Read, Process, and Write 3-D Medical Images” on page 1-3.

Display, Volume Rendering, and Surfaces

Once you have imported a medical image into the workspace, you can view and inspect the image to plan your workflow. Medical Imaging Toolbox provides tools for detailed viewing of the medical images in different orientations, for volume rendering to visualize intensity volumes in 3-D, and for generating surfaces for efficient display or 3-D printing or modeling applications. For more information, see “Visualize 3-D Medical Image Data Using Medical Image Labeler” on page 3-11.

Preprocessing and Augmentation

An imported medical image can contain noise that you must reduce. Multiple medical images that you must process together can be misaligned and require registration. Also, if the medical imaging data set is too small for your intended application, you might need to augment it. Medical Imaging Toolbox provides functions for preprocessing and augmentation. For more information, see “Medical Image Preprocessing” on page 4-2.

Ground Truth Labeling and Segmentation

For object detection in deep learning applications, you might need to perform segmentation and labeling of the preprocessed medical image training data set. Medical Imaging Toolbox provides the **Medical Image Labeler** app for segmentation and ground truth labeling. For more information, see “Get Started with Medical Image Labeler” on page 1-9.

Import, Export, and Spatial Referencing

Read, Process, and View Ultrasound Data

This example shows how to import and display a 2-D multiframe ultrasound series, and how to apply a denoising filter to each frame of the series.

You can use the `medicalImage` object to import image data and metadata from 2-D medical images and series of images related by time. In this example, you use the properties and the `extractFrame` object function of a `medicalImage` object to work with a multiframe echocardiogram ultrasound series.

Read Ultrasound Image Series

Specify the name of an echocardiogram ultrasound series contained in a DICOM file.

```
filename = "heartUltrasoundSequenceVideo.dcm";
```

Read the metadata and image data from the file by creating a `medicalImage` object. The image data is stored in the `Pixels` property. Each frame of the image series is a 600-by-800-by-3 pixel RGB image. The `FrameTime` property indicates that each frame has a duration of 33.333 milliseconds. The `NumFrames` property indicates that the series has a total of 116 image frames.

```
medImg = medicalImage(filename)
```

```
medImg =  
  medicalImage with properties:  
    Pixels: [600x800x116x3 uint8]  
    Colormap: []  
    SpatialUnits: "unknown"  
    FrameTime: 33.3330  
    NumFrames: 116  
    PixelSpacing: [1 1]  
    Modality: 'US'  
    WindowCenter: []  
    WindowWidth: []
```

Display Ultrasound Image Frames as Montage

Display the frames in the `Pixels` property of `medImg` as a montage.

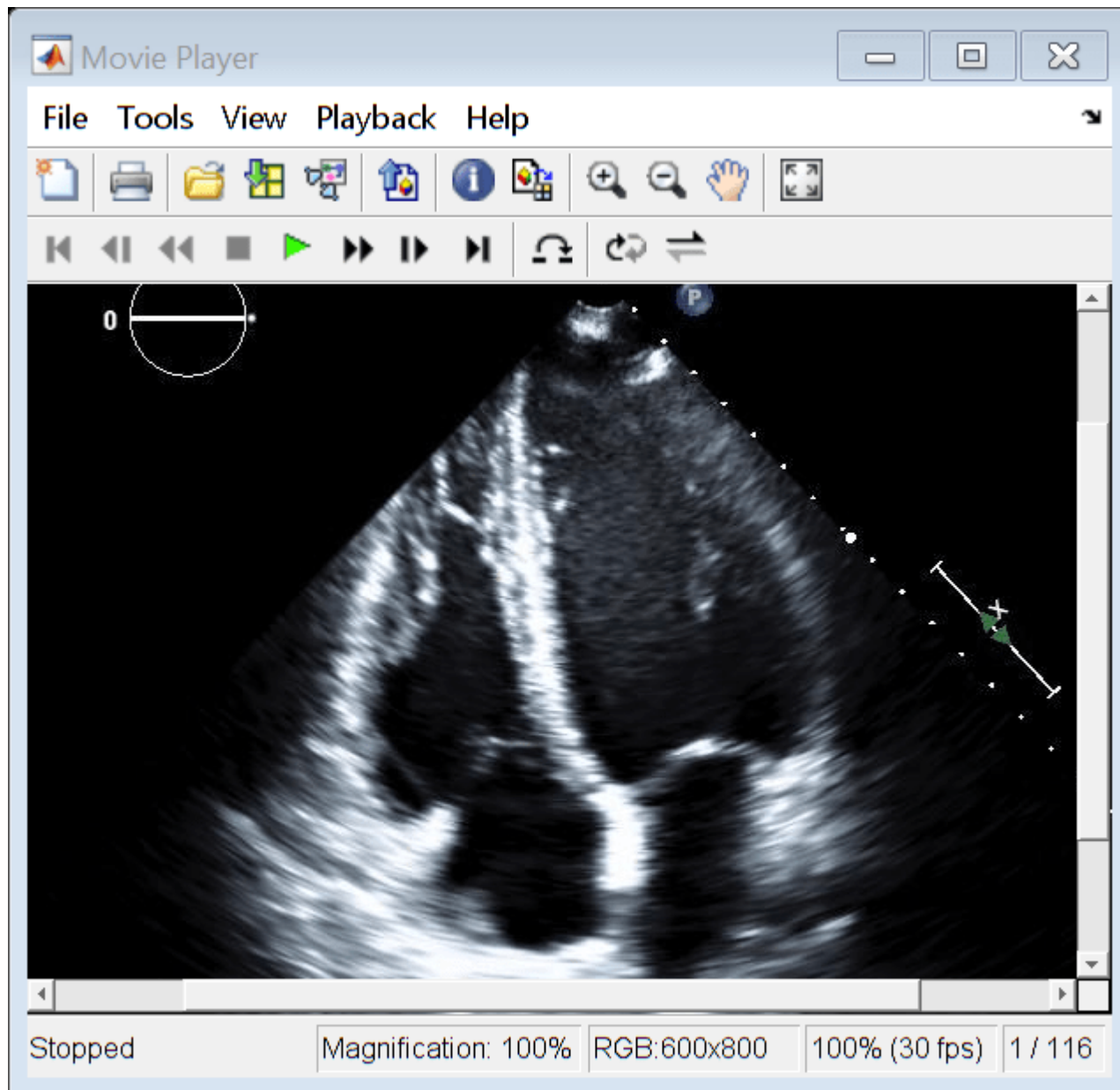
```
montage(medImg)
```



Display Ultrasound Series as Video

Open the Video Viewer app to view the ultrasound series as a video by using the `implay` function. The `implay` function automatically sets the frame rate using the `FrameTime` property of `medImg`.

```
implay(medImg)
```



Reduce Speckle Noise

Reduce the noise in the ultrasound image series by applying a speckle-reducing anisotropic diffusion filter to each frame. Extract each frame from the `medicalImage` object by using the `extractFrame` object function, and convert the frame from RGB to grayscale. Apply the filter by using the `specklefilt` function. Specify the `DegreeOfSmoothing` and `NumIterations` name-value arguments to control the smoothing parameters.

```
[h,w,d,c] = size(medImg.Pixels);
pixelsSmoothed = zeros(h,w,d);
for i = 1:d
    Ii = extractFrame(medImg,i);
    Ii = im2double(im2gray(Ii));
```



```
pixelsSmoothed(:,:,i) = specklefilt(Ii,DegreeOfSmoothing=0.6,NumIterations=50);  
end
```

Store Smoothed Data as Medical Image Object

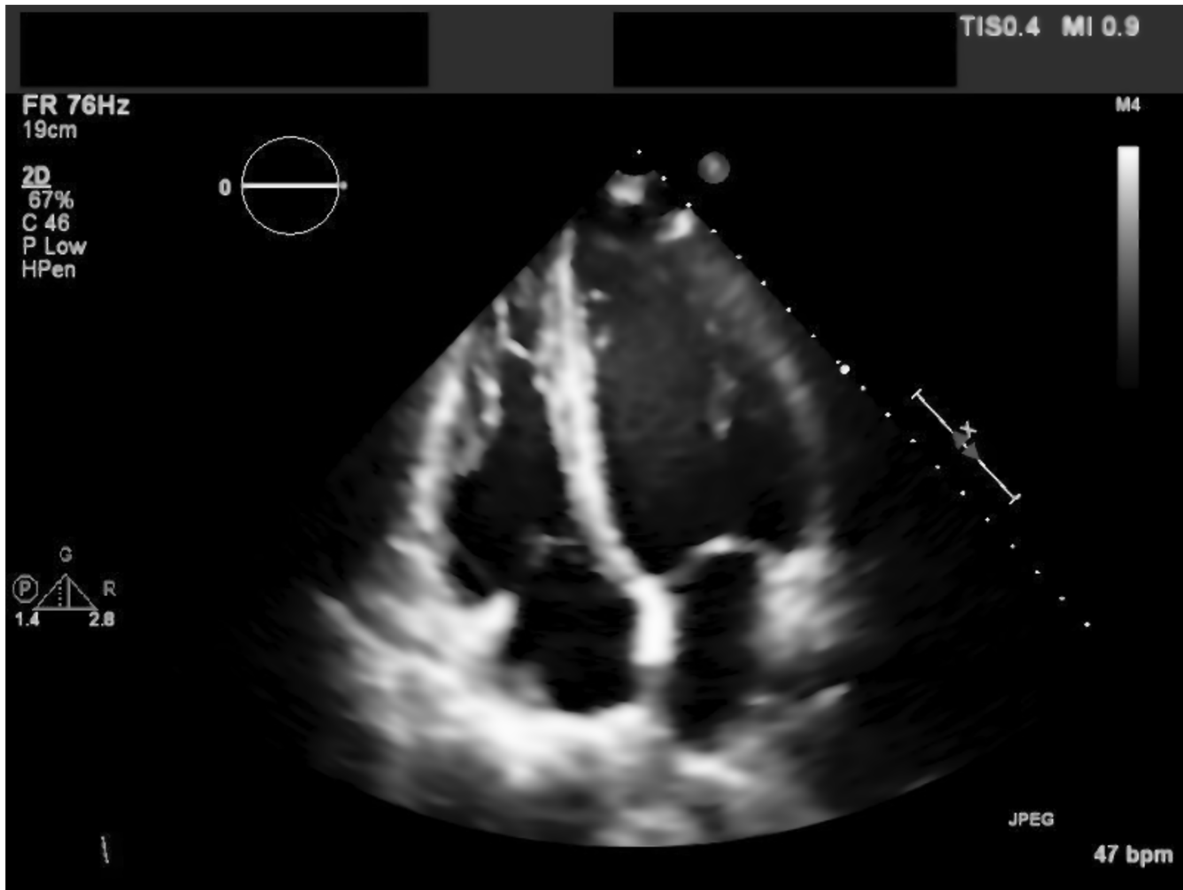
Create a new `medicalImage` object that contains the smoothed image pixels. Use the property values from the original file to maintain the correct frame information.

```
medImgSmoothed = medImg;  
medImgSmoothed.Pixels = pixelsSmoothed;  
medImgSmoothed
```

```
medImgSmoothed =  
  medicalImage with properties:  
  
    Pixels: [600x800x116 double]  
    Colormap: []  
    SpatialUnits: "unknown"  
    FrameTime: 33.3330  
    NumFrames: 116  
    PixelSpacing: [1 1]  
    Modality: 'US'  
    WindowCenter: []  
    WindowWidth: []
```

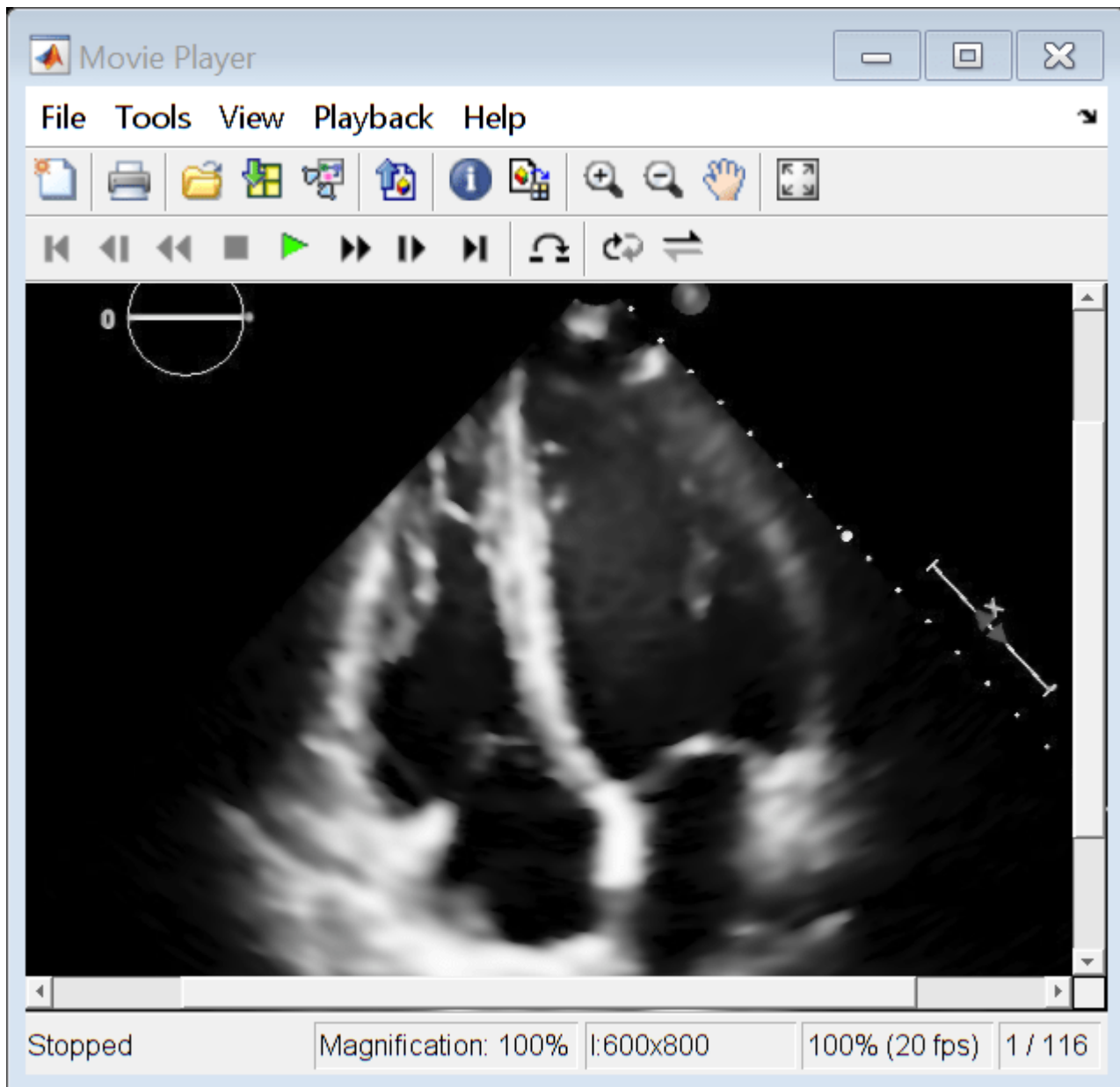
View the first frame of the smoothed image series by using the `montage` function with the `Indices` name-value argument set to 1.

```
figure  
montage(medImgSmoothed,Indices=1)
```



Display the smoothed image data as a video.

```
imshow(pixelSmoothed)
```



See Also

`medicalImage` | `extractFrame` | `specklefilt` | `montage` | **Video Viewer**

Related Examples

- “Read, Process, and Write 3-D Medical Images” on page 1-3

Display and Volume Rendering

- “Choose Approach for Medical Image Visualization” on page 3-2
- “Visualize 3-D Medical Image Data Using Medical Image Labeler” on page 3-11
- “Display Medical Image Volume in Patient Coordinate System” on page 3-21
- “Display Labeled Medical Image Volume in Patient Coordinate System” on page 3-24
- “Create STL Surface Model of Femur Bone for 3-D Printing” on page 3-31
- “Medical Image-Based Finite Element Analysis of Spine” on page 3-39

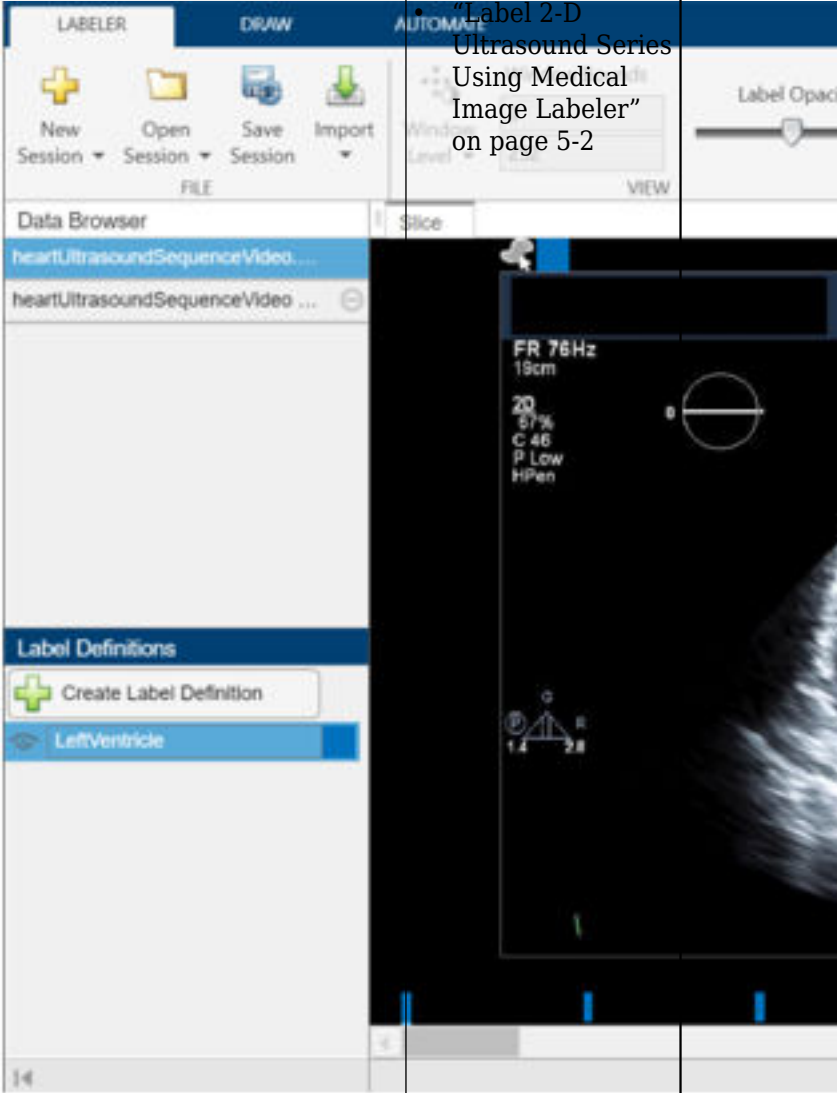
Choose Approach for Medical Image Visualization

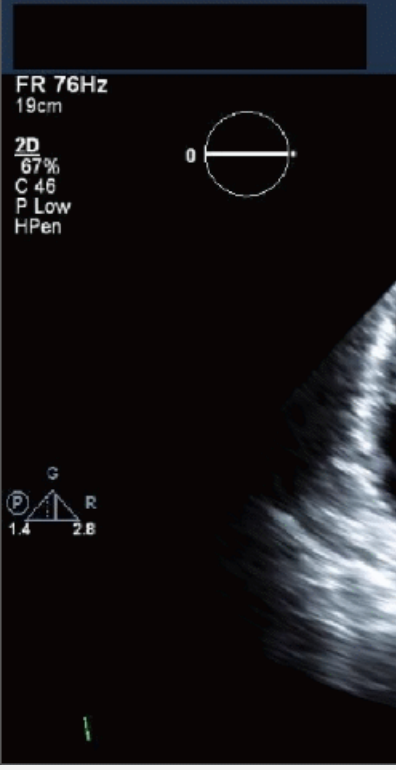
Medical Imaging Toolbox provides tools to display, explore, and publish 2-D and 3-D medical image data. Visualization is important for clinical diagnosis, treatment planning, and image analysis. Sharing snapshots and animations helps convey useful clinical information to patients, colleagues, and in publications.

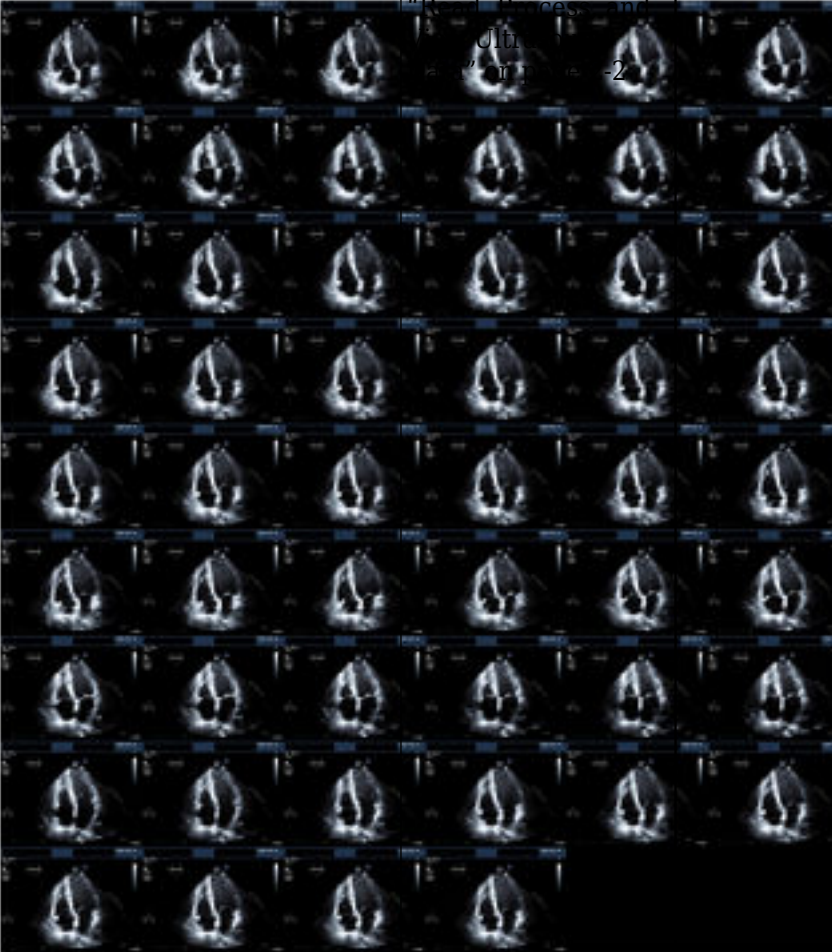
Display 2-D Medical Image Data

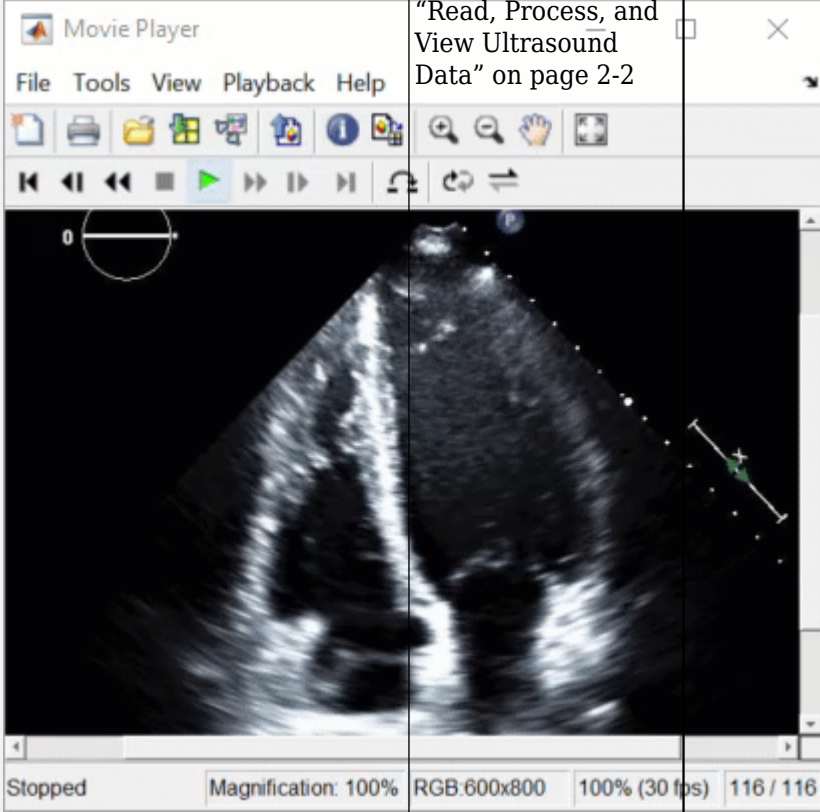
2-D medical image data includes single images such as X-rays as well as multiframe image series such as ultrasound videos. The **Medical Image Labeler** app is useful for interactive display, especially when viewing multiple image files or labeled data. To view 2-D images without needing to launch the app, use the `montage` and `imshow` object functions, which accept a `medicalImage` object as input.

This table describes the options for displaying 2-D medical image data using Medical Imaging Toolbox. If you do not have Medical Imaging Toolbox installed, see “Display and Exploration” (Image Processing Toolbox™).

Goal	Approach	Image	See Also
<p>Interactively explore images:</p> <ul style="list-style-type: none"> • View multiple related image files in one app session. • Toggle visibility of labels and adjust label opacity. • Adjust display contrast by modifying the intensity window and level. 	<p>Medical Image Labeler app</p> <ul style="list-style-type: none"> • Open the app from the MATLAB Toolstrip, on the Apps tab, under Image Processing and Computer Vision. • Open the app from the MATLAB command prompt using <code>medicalImageLabeler</code>. 		<ul style="list-style-type: none"> • “Label 2-D Ultrasound Series Using Medical Image Labeler” on page 5-2

Goal	Approach	Image	See Also
<p>Publish snapshots and animations to view outside of MATLAB.</p>	<p>Medical Image Labeler app</p> <ul style="list-style-type: none"> • Open the app from the MATLAB Toolstrip, on the Apps tab, under Image Processing and Computer Vision. • Open the app from the MATLAB command prompt using <code>medicalImageLabeler</code>. 		<ul style="list-style-type: none"> • Export Snapshot Image from App on page 3-9 • Export Animation from App on page 3-9

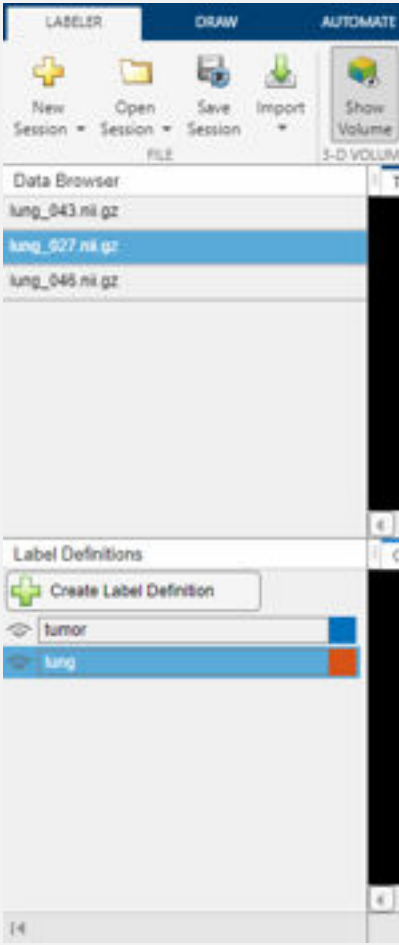
Goal	Approach	Image	See Also
Programmatically display all frames in an image series.	Display the data in the <code>medicalImage</code> object <code>medImage</code> by using the command <code>montage (medImage)</code> .		"Read, Process, and Display Medical Images" on page 2

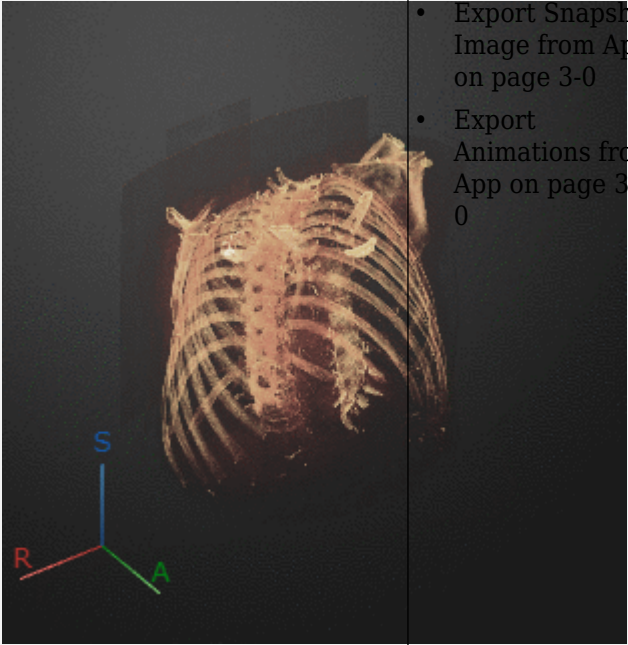
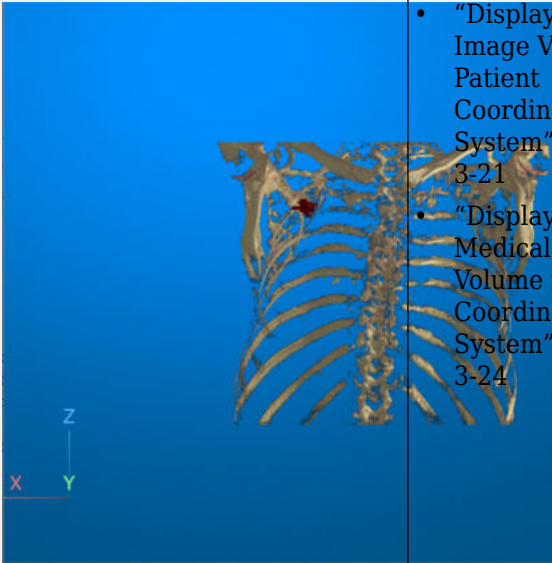
Goal	Approach	Image	See Also
<p>Programmatically play image series as a video.</p>	<p>Display the data in the <code>medicalImage</code> object by using the command <code>implay(medImage)</code>.</p>		<p>“Read, Process, and View Ultrasound Data” on page 2-2</p>

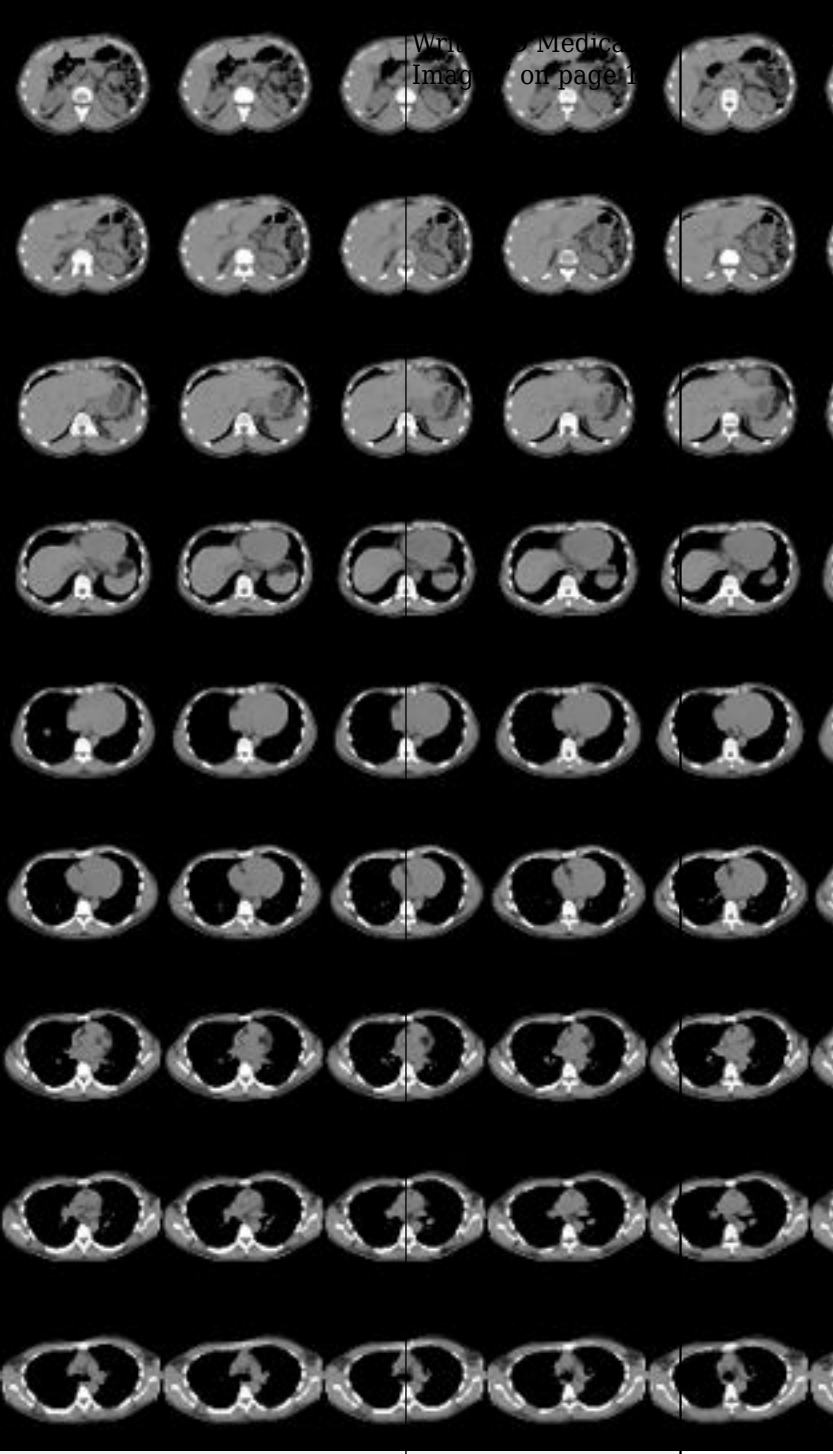
Display 3-D Medical Image Data

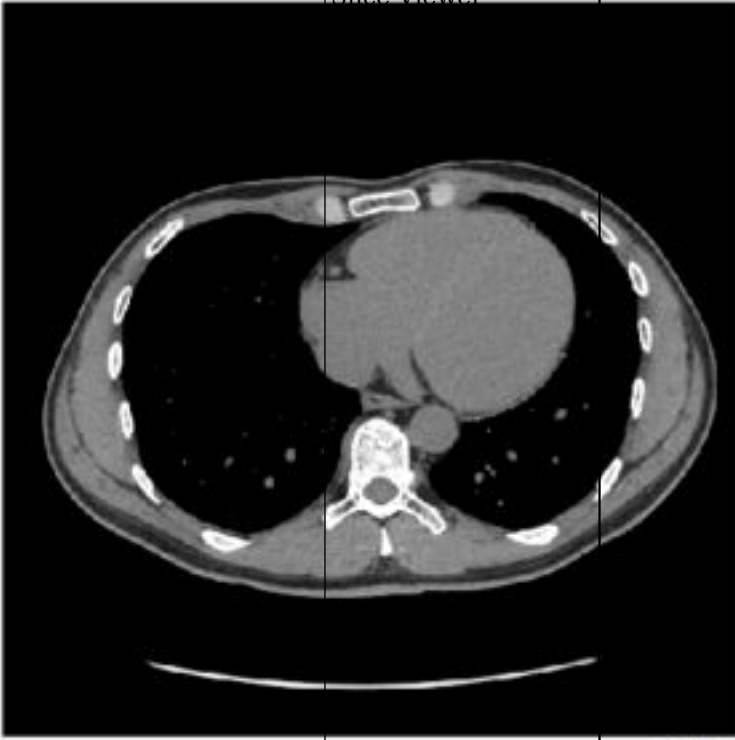
3-D medical image data includes volumes from modalities such as computed tomography (CT), magnetic resonance imaging (MRI), and positron emission tomography (PET). The **Medical Image Labeler** app is useful for interactive display, especially when viewing multiple image files or labeled data. To view 3-D images without needing to launch the app, use the `volshow` and `montage` object functions or create a `sliceViewer` object, which all accept a `medicalVolume` object as input.

This table describes the options for displaying 3-D medical image data using Medical Imaging Toolbox. If you do not have Medical Imaging Toolbox installed, see “Display and Exploration” (Image Processing Toolbox).

Goal	Approach	Image	See Also
<p>Interactively explore image volumes:</p> <ul style="list-style-type: none"> • View multiple related image files in one app session. • Toggle visibility of labels and adjust label opacity. • Adjust display contrast by modifying the intensity window and level. • Publish snapshots and animations to view outside of MATLAB. 	<p>Medical Image Labeler app</p> <ul style="list-style-type: none"> • Open the app from the MATLAB Toolstrip, on the Apps tab, under Image Processing and Computer Vision. • Open the app from the MATLAB command prompt using <code>medicalImageLabeler</code>. 	 <p>The screenshot shows the MATLAB Medical Image Labeler app interface. At the top, there are three tabs: LABELER, DRAW, and AUTOMATE. Below the tabs is a menu bar with options: New Session, Open Session, Save Session, Import, and Show Volume. The main area is divided into several panels. On the left is the 'Data Browser' showing a list of image files: lung_043.nii.gz, lung_027.nii.gz, and lung_045.nii.gz. Below that is the 'Label Definitions' panel, which includes a 'Create Label Definition' button and a list of labels: 'tumor' and 'lung'. On the right is the '3-D VOLUME' panel, which displays two cross-sectional views of a lung scan. The top view is a 'Transverse' slice, and the bottom view is a 'Coronal' slice. Both views show a 3D visualization of the lung with a red region representing a tumor. The interface also includes a 'Label Opacity' slider on the far right.</p>	<p>“Visualize 3-D Medical Image Data Using Medical Image Labeler” on page 3-11</p>

Goal	Approach	Image	See Also
<p>Publish snapshots and animations to view outside of MATLAB.</p>	<p>Medical Image Labeler app</p> <ul style="list-style-type: none"> • Open the app from the MATLAB Toolstrip, on the Apps tab, under Image Processing and Computer Vision. • Open the app from the MATLAB command prompt using <code>medicalImageLabeler</code>. 		<ul style="list-style-type: none"> • Export Snapshot Image from App on page 3-0 • Export Animations from App on page 3-0
<p>Programmatically display volume in a figure window, with optional label overlays.</p>	<p>Display the data in the <code>medicalVolume</code> object <code>medVol</code> by using the command <code>volshow(medVol)</code>.</p>		<ul style="list-style-type: none"> • “Display Medical Image Volume in Patient Coordinate System” on page 3-21 • “Display Labeled Medical Image Volume in Patient Coordinate System” on page 3-24

Goal	Approach	Image	See Also
Programmatically display slices along one dimension.	Display the data in the <code>medicalVolume</code> object <code>medVol</code> by using the command <code>montage (medVol)</code> .		Write Medical Images on page 1

Goal	Approach	Image	See Also
<p>Programmatically display slices along one dimension in a scrollable figure window.</p>	<p>Display the data in the <code>medicalVolume</code> object <code>medVol</code> by using the command <code>sliceViewer(medVol)</code>.</p>		<p>"Display Medical Image Volume in Slice Viewer"</p>

See Also

[Medical Image Labeler](#) | [imshow](#) | [montage](#) | [volshow](#) | [sliceViewer](#)

Related Examples

- "Image Display and Exploration Overview"
- "Visualize 3-D Medical Image Data Using Medical Image Labeler" on page 3-11

Visualize 3-D Medical Image Data Using Medical Image Labeler

This example shows how to interactively explore medical image volumes and export snapshots and animations from the **Medical Image Labeler** app.

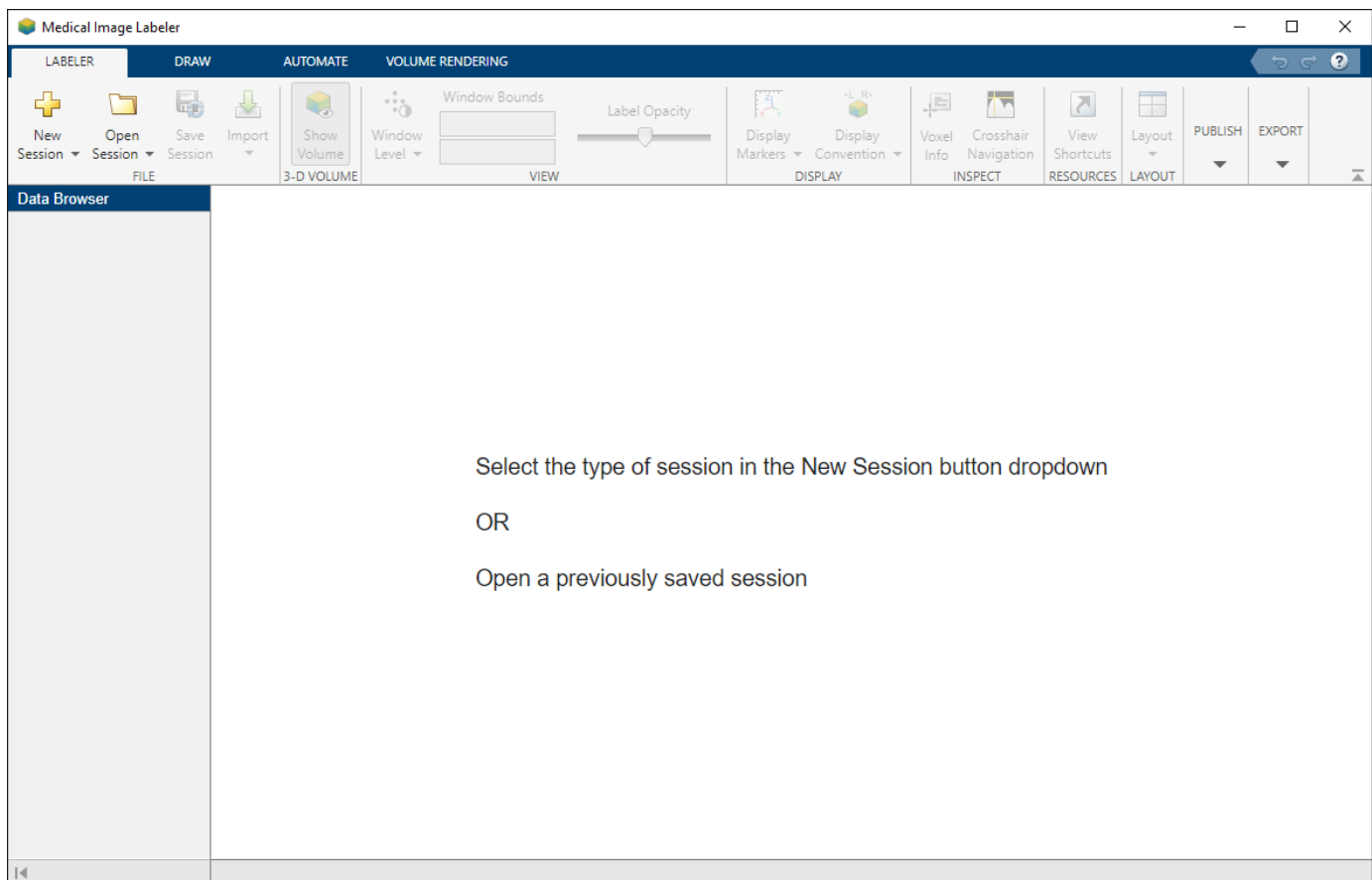
Download Data

This example labels chest CT data from a subset of the Medical Segmentation Decathlon data set [1 on page 3-20]. Download the `MedicalVolumeNIftIData.zip` file from the MathWorks® website, then unzip the file. The size of the subset of data is approximately 76 MB.

```
zipFile = matlab.internal.examples.downloadSupportFile("medical","MedicalVolumeNIftIData.zip");
filepath = fileparts(zipFile);
unzip(zipFile,filepath)
dataFolder = fullfile(filepath,"MedicalVolumeNIftIData");
```

Open Medical Image Labeler

Open the **Medical Image Labeler** app from the **Apps** tab on the MATLAB® toolstrip, under **Image Processing and Computer Vision**. You can also open the app by using the `medicalImageLabeler` command.

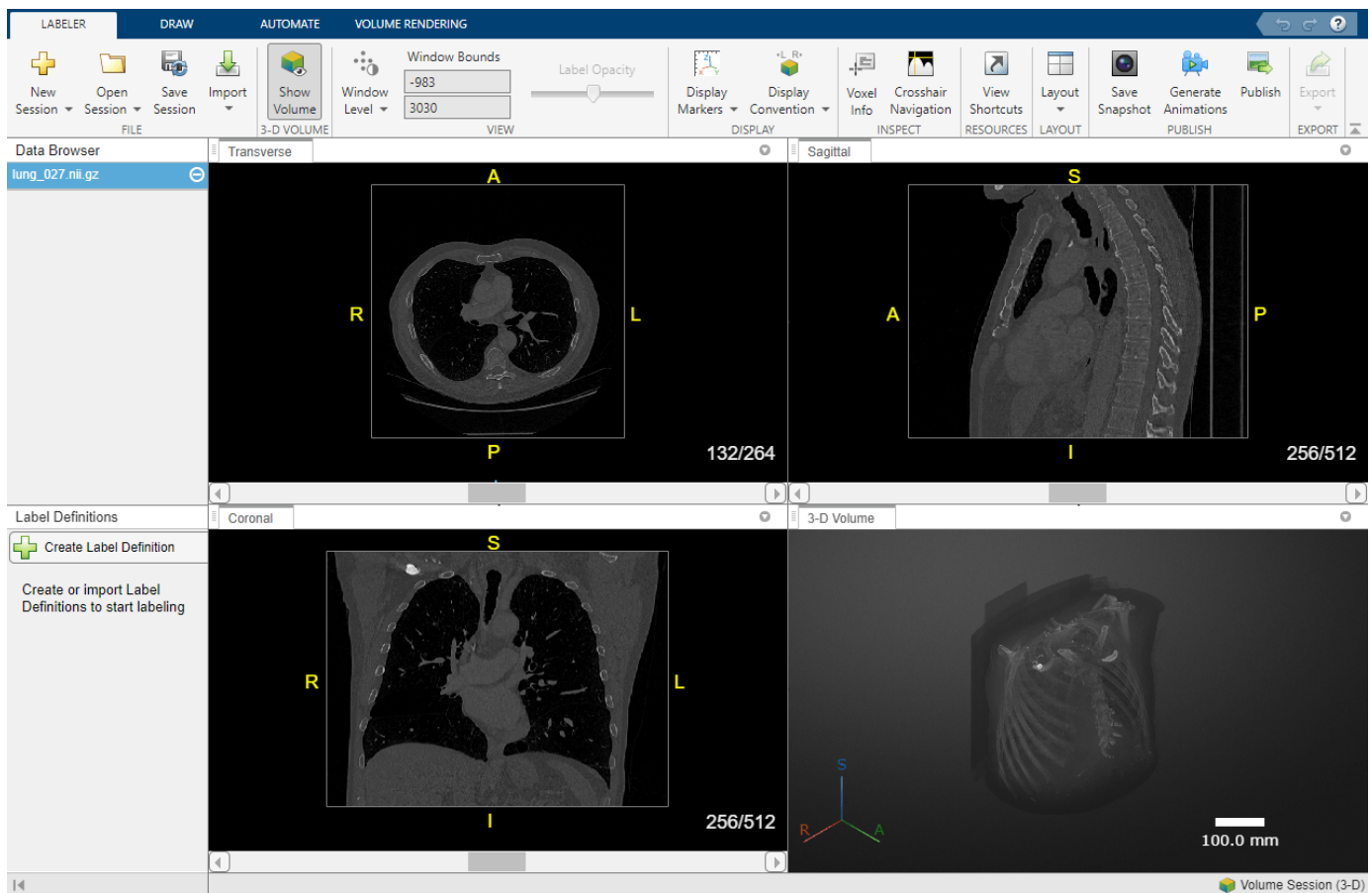


Create New Volume Labeling Session

To start a new 3-D session, on the app toolbar, click **New Session** and select **New Volume session (3-D)**. In the Create a new session folder dialog box, specify a location in which to save the new session folder by entering a path or selecting **Browse** and navigating to your desired location. In the **New Session Folder** box, specify a name for the folder for this app session. Select **Create Session**.

Load Image Data into Medical Image Labeler

To load an image into the **Medical Image Labeler** app, on the app toolbar, click **Import**. Then, under **Data**, select **From File**. Browse to the location containing the downloaded data, specified by the dataFolder workspace variable, and select the file lung_027.nii.gz. For a **Volume Session**, the imported data file can be a single DICOM or NIFTI file containing a 3-D image volume, or a directory containing multiple DICOM files corresponding to a single image volume.



View Data as Anatomical Slice Planes


The app displays the data as anatomical slice planes in the **Transverse**, **Sagittal**, and **Coronal** panes.


By default, the app displays the center slice in each slice plane. You can change the displayed slice by using the scroll bar at the bottom of a slice pane, or you can click the pane and then press the left and right arrow keys. The app displays the current slice number out of the total number of slices, such as 132/264, for each slice pane. The app also displays anatomical display markers indicating the

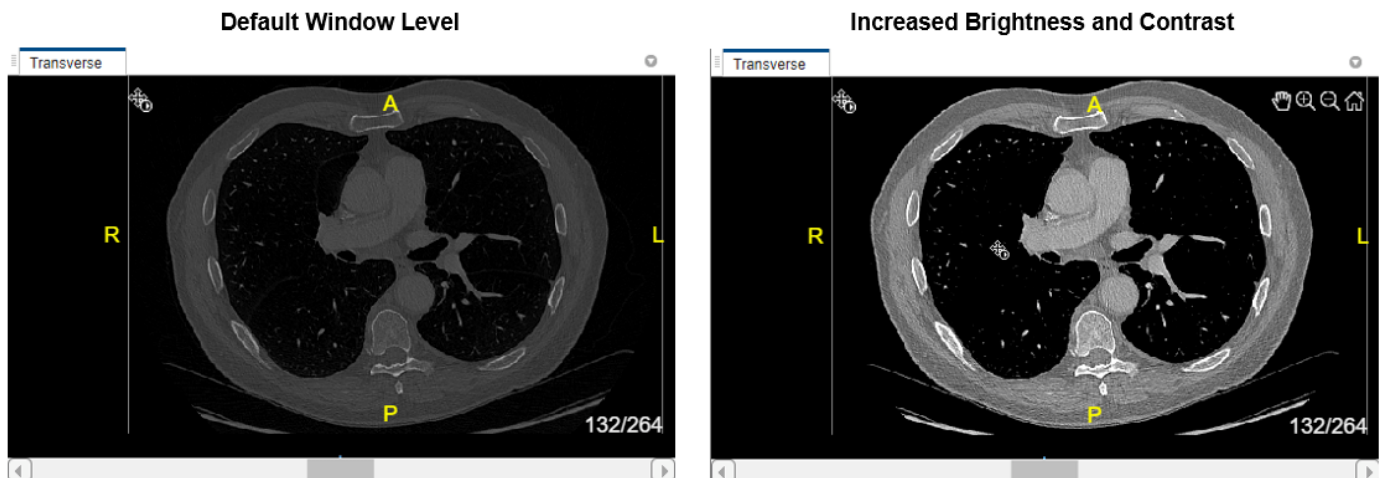
anterior (A), posterior (P), left (L), right (R), superior (S), and inferior (I) directions. To toggle the visibility of the display markers, on the app toolbar, click **Display Markers** and, under **2-D Slices**, select or clear **2D Orientation Markers**. You can zoom in on the current slice plane using the mouse scroll wheel or the zoom controls that appear when you pause on the slice plane.

By default, the app displays the scan using the **Radiological** display convention, with the left side of the patient on the right side of the image. To display the left side of the patient on the left side of the image, on the app toolbar, click **Display Convention** and select **Neurological**.

You can adjust the brightness and contrast used to display grayscale image data by using the **Window Level** tool on the **Labeler** tab of the app toolbar. First, on the app toolbar, select

. Then, click and hold in any of the slice panes, and drag up and down to increase and decrease the brightness, respectively, or left and right to increase and decrease the contrast. The

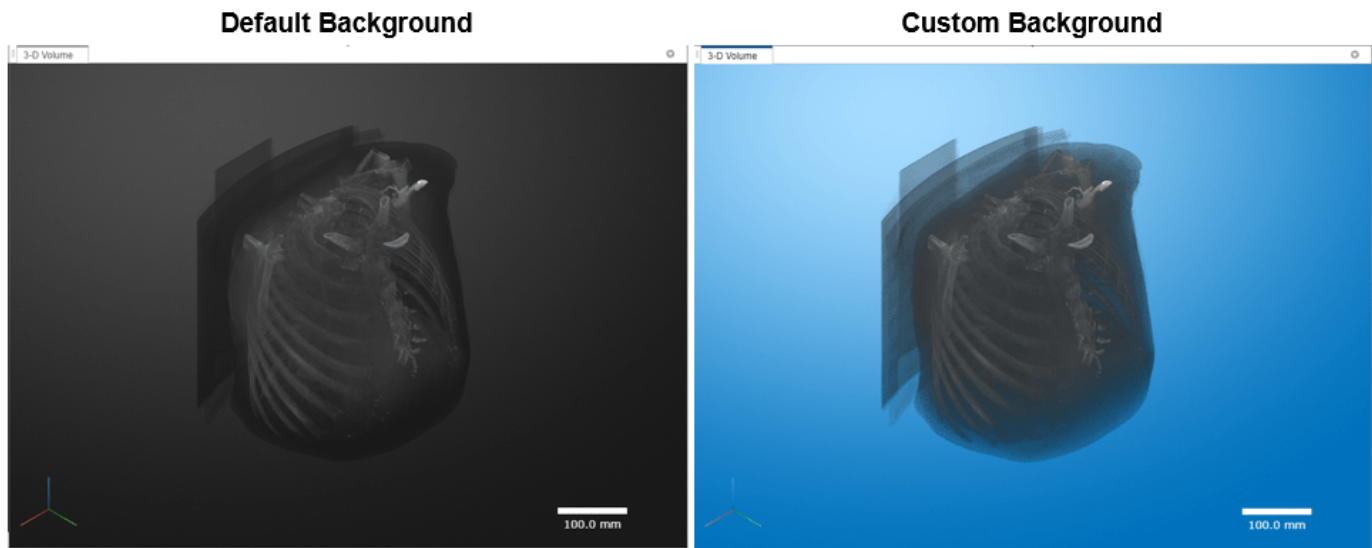
updated window bounds are displayed in the app toolbar under **Window Bounds**. Clear  to deactivate the tool. Changing the display window does not modify the image data. To reset the brightness and contrast, click **Window Level** and select **Reset**.



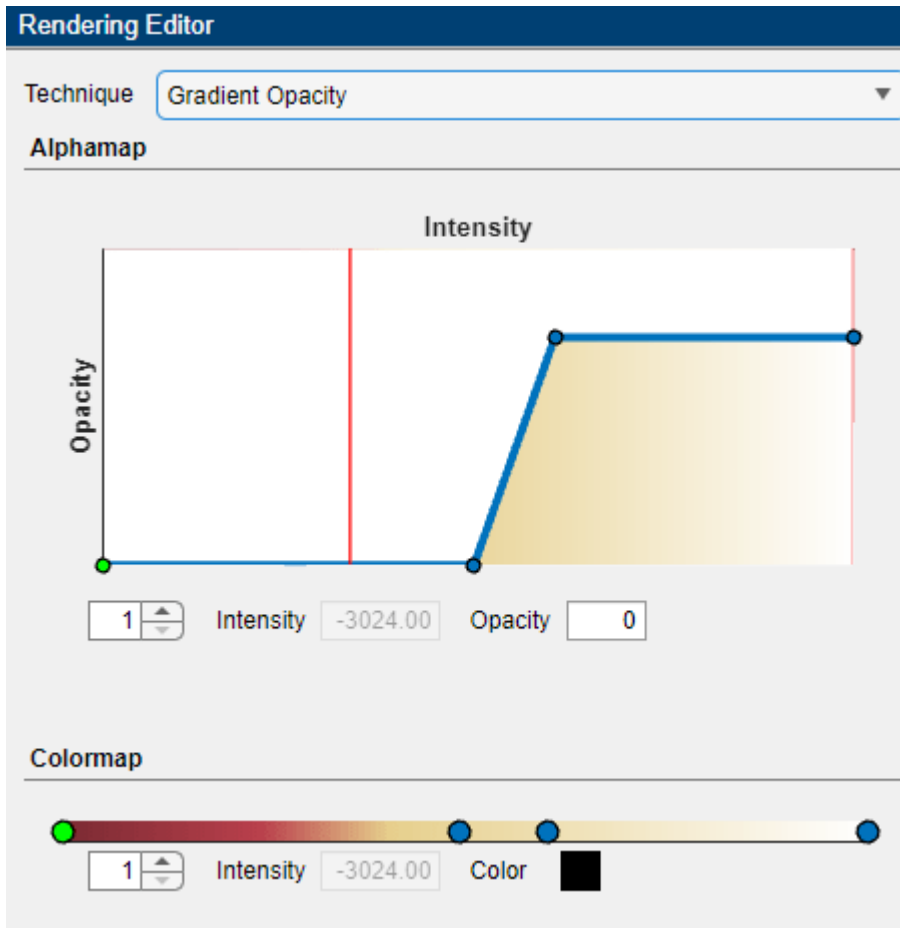
View Data as 3-D Volume

The app displays the data as a 3-D volume in the **3-D Volume** pane. You can toggle the visibility of the volume display using the **Show Volume** button on the **Labeler** tab of the app toolbar.

In the **3-D Volume** pane, click and drag to rotate the volume or use the scroll wheel to zoom. You can adjust the volume display using the tools in the **Volume Rendering** tab of the app toolbar. To change the background color, click **Background Color**. Toggle the use of a background gradient by clicking **Use Gradient**. To change the gradient color, with **Use Gradient** selected, click **Gradient Color**. To restore the default background settings, click **Restore Background**. You can also view orientation axes and scale bar display markers in the **3-D Volume** pane. To toggle the visibility of each display marker, on the **Labeler** tab of the app toolbar, click **Display Markers**, and under **3-D Volume**, select or clear **3D Orientation Axes** and **Scale Bars**.



You can change how the app displays voxel intensity data by selecting one of several built-in rendering styles. On the **Volume Rendering** tab, select **CT - Bone** from the **Rendering Presets** gallery to use an alphamap and colormap recommended for highlighting bone tissue in computed tomography (CT) scans. To refine the built-in displays for your data set, adjust the display parameters by using the **Rendering Editor**. To open the editor, on the **Volume Rendering** tab of the app toolbar, click **Rendering Editor**.

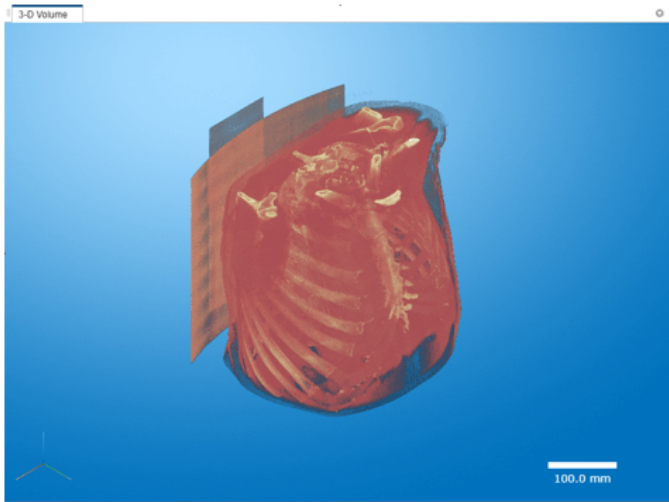


Using the **Rendering Editor**, you can manually adjust the rendering technique, the transparency alphamap, and the colormap of the 3-D volume rendering. The app supports these rendering techniques:

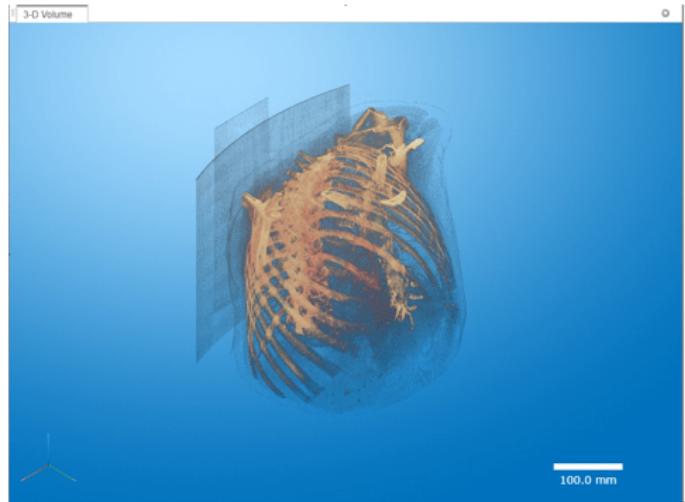
- **Volume Rendering** — View the volume based on the specified color and transparency for each voxel.
- **Maximum Intensity Projection** — View the voxel with the highest intensity value for each ray projected through the data.
- **Gradient Opacity** — View the volume based on the specified color and transparency, with an additional transparency applied if the voxel is similar in intensity to the previous voxel along the viewing ray.

When you render a volume with uniform intensity using **Gradient Opacity**, the internal portion of the volume appears more transparent than in the **Volume Rendering** rendering style, enabling better visualization of label data, if present, and intensity gradients in the volume. For this example, select **Gradient Opacity**.

CT – Bone Rendering in Volume Rendering



CT – Bone in Gradient Opacity



You can refine the alphasmap and colormap by manipulating the plots and colorbar in the **Rendering Editor** directly. To save your customized settings, in the **Volume Rendering** tab of the app toolbar, click **Save Rendering**. The app adds an icon for your custom settings to the gallery in the app toolbar, under **User-Defined**. If you have multiple files loaded in the app session, you can apply the same built-in or custom rendering settings to all volumes by clicking **Apply To All Volumes** in the app toolbar.

Export Snapshot Image from App

You can export images from the **Medical Image Labeler** app to use for figures. In the **Labeler** tab of the app toolbar, click **Save Snapshot**. In the Save Snapshot dialog box, select the views you want to export and click **Save**. By default, the app saves a snapshot for the transverse, coronal, and sagittal slice planes, as well as the 3-D volume window. In the second dialog box, specify the location where you want to save the images, and click **Save** again. Each snapshot is saved as a separate PNG file. The 2-D snapshots match the current brightness and contrast settings of the app, but show the full slice without any zooming or anatomical display markers. The 3-D snapshot matches the current rotation and zoom values in the **3-D Volume** pane, and contains any enabled display markers.

Export Animations from App

You can generate 2-D animations that scroll through transverse, sagittal, or coronal slices, or 3-D animations of the volume rotating about a specified axis. Access the animation generator tool in the **Labeler** tab of the app toolbar by clicking **Generate Animations**.

To export a 2-D animation, in the **Animation** editor, select **2-D slices**. In the **Direction** section, specify the **Slice Direction** along which you want to view slices. Use the settings in the **Animation** section to customize the **Slice Range** and **Step Size**. Select **Append animation in reverse** to make the animation scroll forward and then backward through the slides, rather than forward only. In the **Export** section, specify the **Loop Count** and **Animation Length**. Specify the loop count as a positive integer, for a fixed number of loops, or as `inf` to create a continuously looping animation. Specify the animation length in seconds. The editor displays the speed in frames per second based on your specified slice range and animation length. The settings in this image create an animation that captures every eighth slice along the transverse direction. Each loop animates forward and then in reverse through the slice range. The animation loops infinitely, with a length of 10 seconds per loop, and a frame rate of 6.475 FPS.

Animation

3-D Volume **2-D slices**

Direction

Slice Direction Transverse ▼

Animation

Slice Range 1 264

Step Size 8 frames

Append animation in reverse

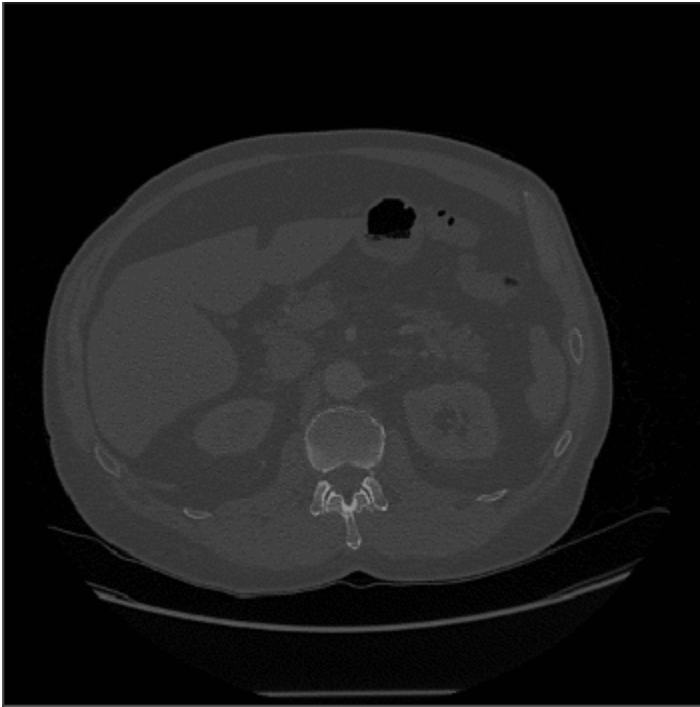
Export

Loop Count Inf

Animation Length 10.0 secs
6.475 FPS

Preview Export GIF

Click **Preview** to preview one loop of the animation with the specified **Direction** and **Animation** settings. The preview does not reflect the loop count or animation length of the exported GIF. The preview plays in the slice view of the selected slice direction. If you are satisfied with the preview, click **Export GIF**, and, in the dialog box, select the location to save the animation GIF file. You can open and view the GIF file from the saved location and share the animation outside of MATLAB.



To export a 3-D animation, in the **Animation** editor, select **3-D Volume**. Under **Camera**, select the **Rotation Axis** to rotate around as SI (superior-inferior), LR (left-right), AP (anterior-posterior), or the current up vector in the **3-D Volume** pane. Select **Snap Camera to Axis** to set the camera location to view the volume head-on versus maintaining the rotation in the **3-D Volume** pane. In the **Animation** section, use **Limits** to specify the orientations, in degrees, between which to rotate the volume. Specify the **Step Size** in degrees. The settings in this image create an animation of the volume rotating about the superior-inferior axis, with frames acquired every ten degrees of rotation. The animation loops infinitely, with a length of 3 seconds per loop.

Animation

3-D Volume 2-D slices

Camera

Rotation Axes

Snap Camera to Axis

Animation

Limits

Step Size

Append animation in reverse

Export

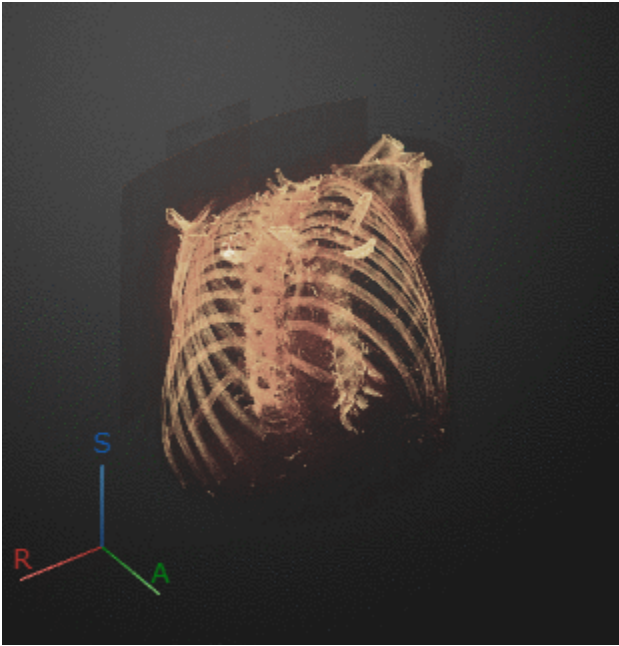
Loop Count

Animation Length

12 FPS

Preview Export GIF

Click **Preview** to preview one loop of the animation with the specified **Camera** and **Animation** settings. The preview does not reflect the loop count or animation length of the exported GIF. The preview plays in the **3-D Volume** pane. If you are satisfied with the preview, click **Export GIF**, and, in the dialog box, select the location to save the animation GIF file. You can open and view the GIF file from the saved location and share the animation outside of MATLAB.



References

[1] Medical Segmentation Decathlon. "Lung." Tasks. Accessed May 10, 2018. <http://medicaldecathlon.com>. The Lung data set is provided by the Medical Segmentation Decathlon under the CC-BY-SA 4.0 license. All warranties and representations are disclaimed. See the license for details. This example uses a subset of the original data set consisting of two CT volumes.

See Also

Medical Image Labeler | `sliceViewer` | `volshow` | `montage`

Related Examples

- "Choose Approach for Medical Image Visualization" on page 3-2
- "Display Medical Image Volume in Patient Coordinate System" on page 3-21
- "Display Labeled Medical Image Volume in Patient Coordinate System" on page 3-24

Display Medical Image Volume in Patient Coordinate System

This example shows how to display 3-D CT data in the patient coordinate system using `volshow`. The `volshow` function uses the spatial referencing information from a `medicalVolume` object to transform intrinsic image coordinates, in voxel units, to patient coordinates in real-world units such as millimeters. This is particularly useful for visualizing anisotropic image voxels, which have unequal spatial dimensions. Viewing images in the patient coordinate system accurately represents the aspect ratio of anisotropic voxels, which avoids distortions in the image. If you do not have Medical Imaging Toolbox™ installed, see `volshow` (Image Processing Toolbox™).

Download Image Volume Data

This example uses a chest CT volume saved as a directory of DICOM files. The volume is part of a data set containing three CT volumes. The size of the entire data set is approximately 81 MB. Download the data set from the MathWorks® website, then unzip the folder.

```
zipFile = matlab.internal.examples.downloadSupportFile("medical","MedicalVolumeDICOMData.zip");
filepath = fileparts(zipFile);
unzip(zipFile,filepath)
dataFolder = fullfile(filepath,"MedicalVolumeDICOMData","LungCT01");
```

Import Image Volume

Create a `medicalVolume` object that contains the image data and spatial referencing information for the CT volume. The `Voxels` property contains a numeric array of the voxel intensities. The `VoxelSpacing` property indicates that the voxels are anisotropic, with a size of 0.7285-by-0.7285-by-2.5 mm.

```
medVol = medicalVolume(dataFolder)
```

```
medVol =
    medicalVolume with properties:
```

```

        Voxels: [512x512x88 int16]
    VolumeGeometry: [1x1 medicalref3d]
      SpatialUnits: "mm"
      Orientation: "transverse"
    VoxelSpacing: [0.7285 0.7285 2.5000]
    NormalVector: [0 0 1]
    NumCoronalSlices: 512
    NumSagittalSlices: 512
    NumTransverseSlices: 88
      PlaneMapping: ["sagittal"    "coronal"    "transverse"]
      Modality: "CT"
    WindowCenters: [88x1 double]
    WindowWidths: [88x1 double]
```

Display Image Volume

Create a colormap and transparency map to display the rib cage. The `alpha` and `color` values are based on the CT-bone rendering style from the Medical Image Labeler app. The `intensity` values for this volume have been tuned in the app using trial and error.

```
alpha = [0 0 0.72 0.72];
color = [0 0 0; 231 208 141; 231 208 141; 255 255 255]/255;
```

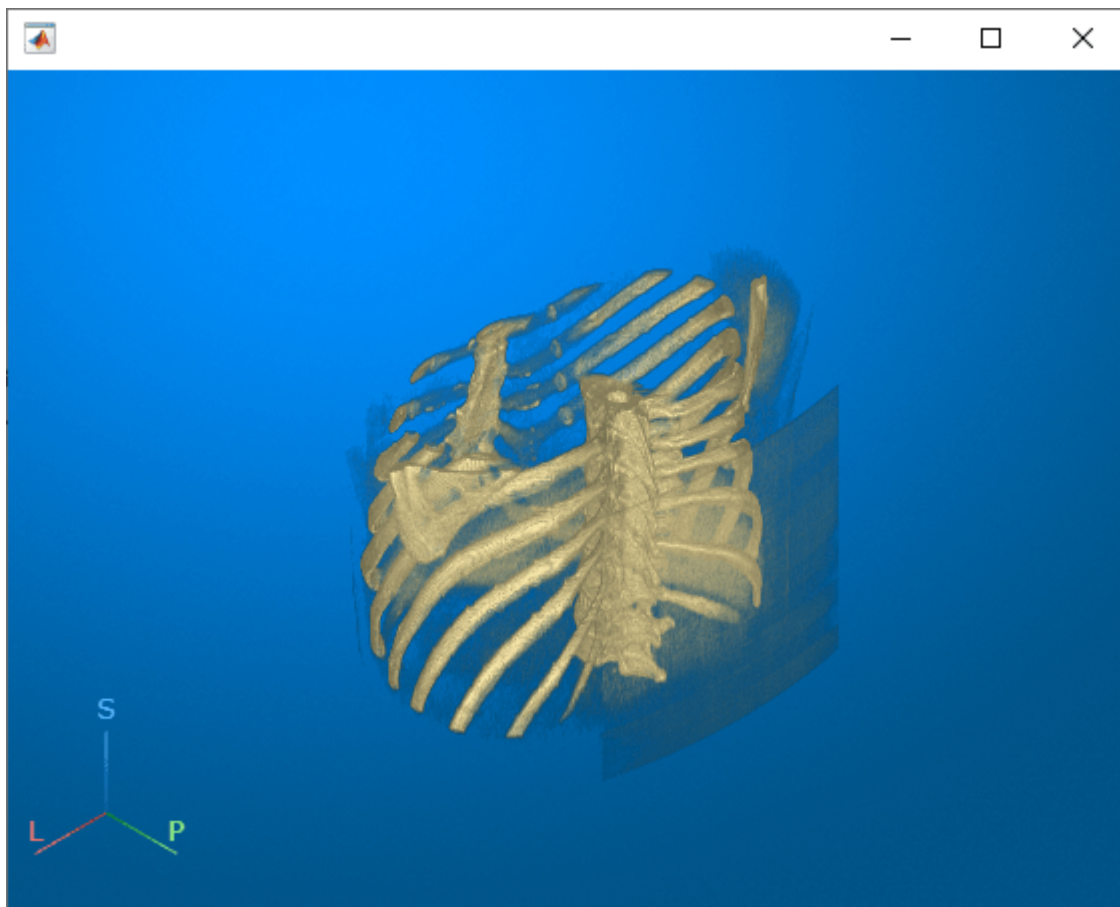
```
intensity = [-3024 50 1400 1499];  
queryPoints = linspace(min(intensity),max(intensity),256);  
alphamap = interp1(intensity,alpha,queryPoints)';  
colormap = interp1(intensity,color,queryPoints);
```

To display the volume in patient coordinates, pass the `medicalVolume` object as input to `volshow`. Specify the custom colormap and transparency map. The `volshow` function uses the spatial details in `medVol` to set the `Transformation` property of the output `Volume` object, `volPatient`. The voxels are scaled to the correct anisotropic dimensions. The axes display indicators label the inferior/superior (**S**), left/right (**L**), and anterior/posterior (**P**) anatomical axes.

```
volPatient = volshow(medVol,Colormap=colormap,Alphamap=alphamap);  
volPatient.Transformation.A
```

```
ans = 4x4
```

```
      0      0.7285      0 -186.5000  
0.7285      0      0 -186.5000  
      0      0      2.5000 -281.2500  
      0      0      0      1.0000
```



See Also

`medicalVolume` | `medicalref3d` | `intrinsicToWorldMapping` | `volshow`

Related Examples

- “Display Labeled Medical Image Volume in Patient Coordinate System” on page 3-24

Display Labeled Medical Image Volume in Patient Coordinate System

This example shows how to display labeled 3-D medical image volumes by using `volshow`. The `volshow` function uses the spatial referencing information from a `medicalVolume` object to transform intrinsic image coordinates, in voxel units, to patient coordinates in real-world units such as millimeters. You can visualize labels as an overlay by using the `OverlayData` property of the `Volume` object created by `volshow`. If you do not have Medical Imaging Toolbox™ installed, see `volshow` (Image Processing Toolbox™).

Download Image Volume Data

This example uses a subset of the Medical Segmentation Decathlon data set [1 on page 3-30]. The subset of data includes two CT chest volumes and corresponding label images, stored in the NIFTI file format.

Run this code to download the `MedicalVolumeNIFTIData.zip` file from the MathWorks® website, then unzip the file. The size of the data file is approximately 76 MB.

```
zipFile = matlab.internal.examples.downloadSupportFile("medical", "MedicalVolumeNIFTIData.zip");
filepath = fileparts(zipFile);
unzip(zipFile, filepath)
```

The folder `dataFolder` contains the downloaded and unzipped data.

```
dataFolder = fullfile(filepath, "MedicalVolumeNIFTIData");
```

Specify the filenames of the CT volume and label image used in this example.

```
dataFile = fullfile(dataFolder, "lung_043.nii.gz");
labelDataFile = fullfile(dataFolder, "LabelData", "lung_043.nii.gz");
```

Import Image Volume

Create a medical volume object that contains the image data and spatial referencing information for the CT volume. The `Voxels` property contains a numeric array of the voxel intensities. The `VoxelSpacing` property indicates that the voxels are anisotropic, with a size of 0.7695-by-0.7695-by-2.5 mm.

```
medVolData = medicalVolume(dataFile)
```

```
medVolData =
    medicalVolume with properties:
```

```

        Voxels: [512×512×129 single]
    VolumeGeometry: [1×1 medicalref3d]
      SpatialUnits: "mm"
      Orientation: "transverse"
    VoxelSpacing: [0.7695 0.7695 2.5000]
    NormalVector: [0 0 -1]
  NumCoronalSlices: 512
  NumSagittalSlices: 512
  NumTransverseSlices: 129
    PlaneMapping: ["sagittal"    "coronal"    "transverse"]
      Modality: "unknown"
```

```
WindowCenters: 0
WindowWidths: 0
```

Import Label Data

Create a medical volume object that contains the label image data. The label image has the same spatial information as the intensity CT volume.

```
medvolLabels = medicalVolume(labelDataFile)

medvolLabels =
    medicalVolume with properties:
        Voxels: [512x512x129 uint8]
        VolumeGeometry: [1x1 medicalref3d]
        SpatialUnits: "mm"
        Orientation: "transverse"
        VoxelSpacing: [0.7695 0.7695 2.5000]
        NormalVector: [0 0 -1]
        NumCoronalSlices: 512
        NumSagittalSlices: 512
        NumTransverseSlices: 129
        PlaneMapping: ["sagittal" "coronal" "transverse"]
        Modality: "unknown"
        WindowCenters: 0
        WindowWidths: 0
```

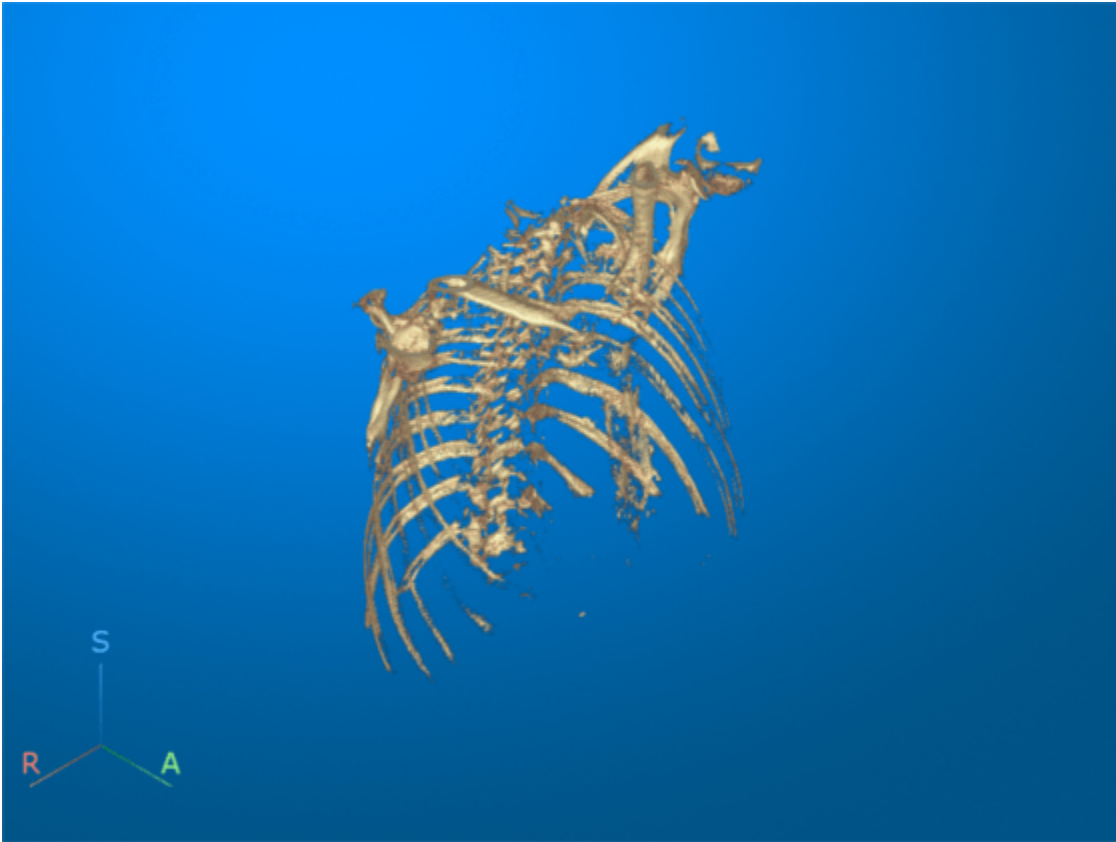
Display CT Volume with Tumor Overlay

Create a colormap and transparency map to display the rib cage. The alpha and color values are based on the CT-bone rendering style from the Medical Image Labeler app. The intensity values have been tuned for this volume using trial and error.

```
alpha = [0 0 0.72 0.72];
color = ([0 0 0; 186 65 77; 231 208 141; 255 255 255]) ./ 255;
intensity = [-3024 -200 0 3071];
queryPoints = linspace(min(intensity),max(intensity),256);
alphamap = interp1(intensity,alpha,queryPoints)';
colormap = interp1(intensity,color,queryPoints);
```

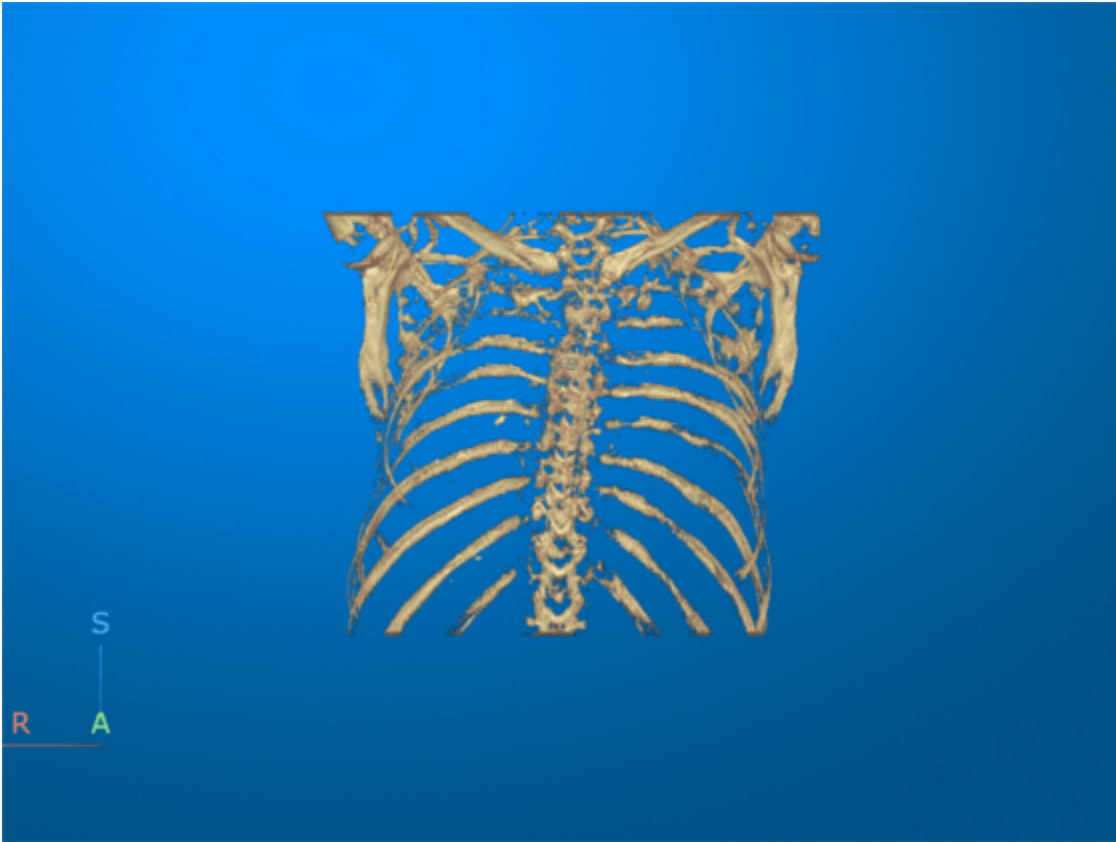
To display the volume in the patient coordinate system, pass the `medicalVolume` object as input to `volshow`. Specify the custom colormap and transparency map. The `volshow` function uses the spatial details in `medVol` to set the `Transformation` property of the output `Volume` object, `vol`. The voxels are scaled to the correct anisotropic dimensions. The axes display indicators label the inferior/superior (**S**), left/right (**L**), and anterior/posterior (**P**) anatomical axes.

```
vol = volshow(medVolData, ...
    Colormap=colormap, ...
    Alphamap=alphamap);
```



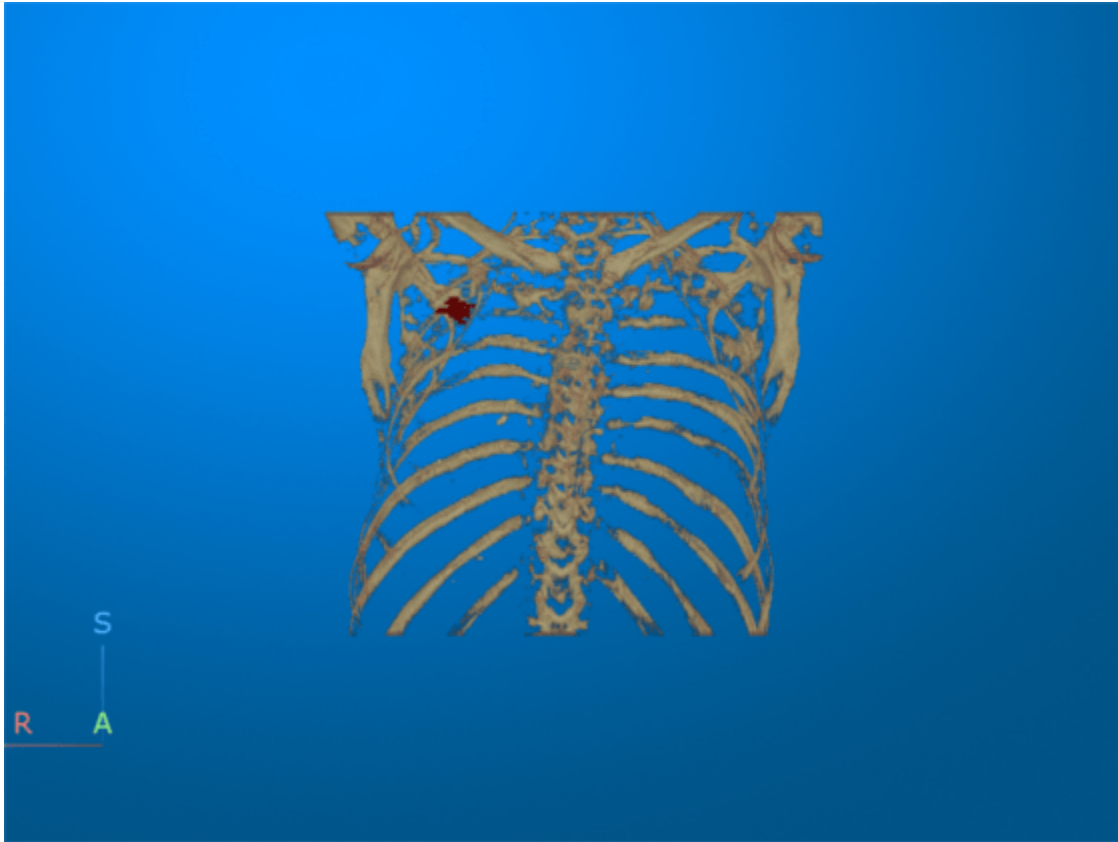
Programatically set the camera position and camera target of the scene to view the volume in the coronal anatomical plane. You can also set the view interactively by clicking the **A** orientation marker in the figure window.

```
scene = vol.Parent;  
scene.CameraPosition = [-4.0999 314.9362 -143.0000];  
scene.CameraTarget = [-4.0999 -6.9636 -143.0000];
```



View the tumor label image as an overlay on the CT volume. You can set the `OverlayData` and `OverlayAlphamap` properties of an existing `Volume` object, or specify them during creation using `volshow`. Note that you must set the `OverlayData` property to the numeric array in the `Voxels` property of `medVolLabels`, rather than the `medicalVolume` object itself.

```
vol.OverlayData = medvolLabels.Voxels;  
vol.OverlayAlphamap = 1;
```

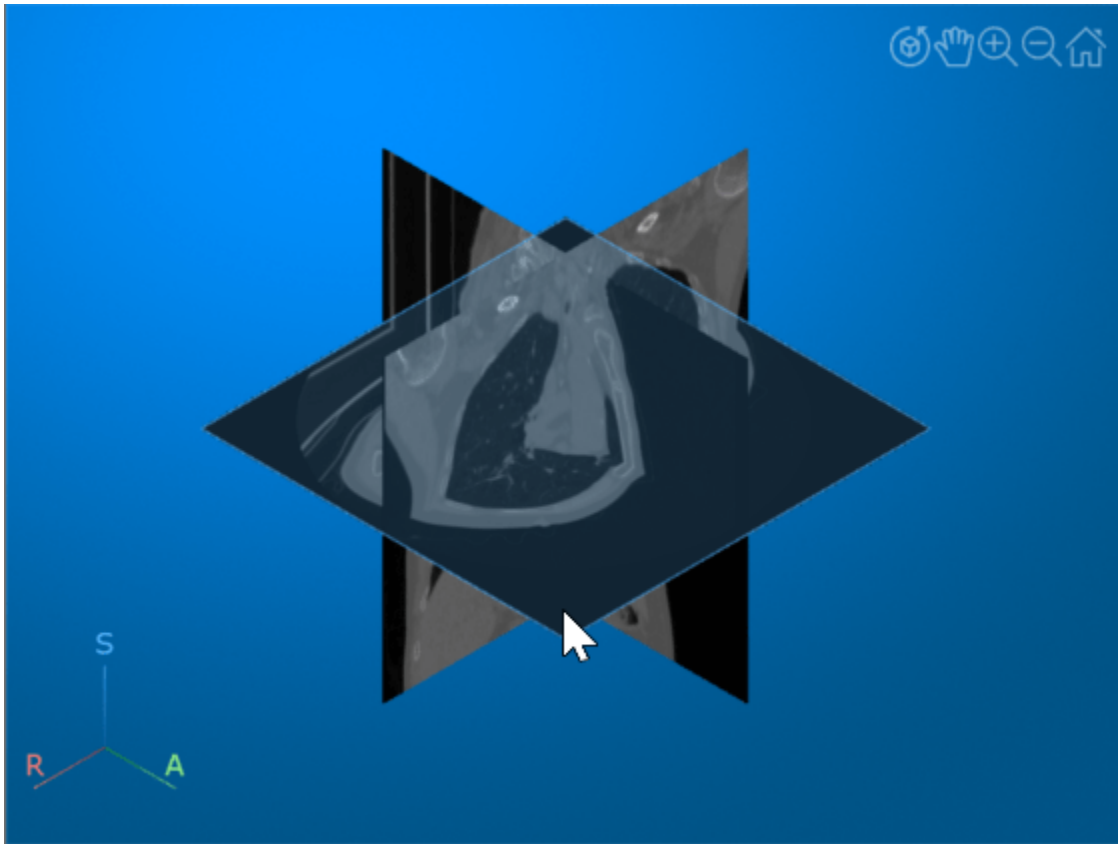


Display CT Volume as Slice Planes

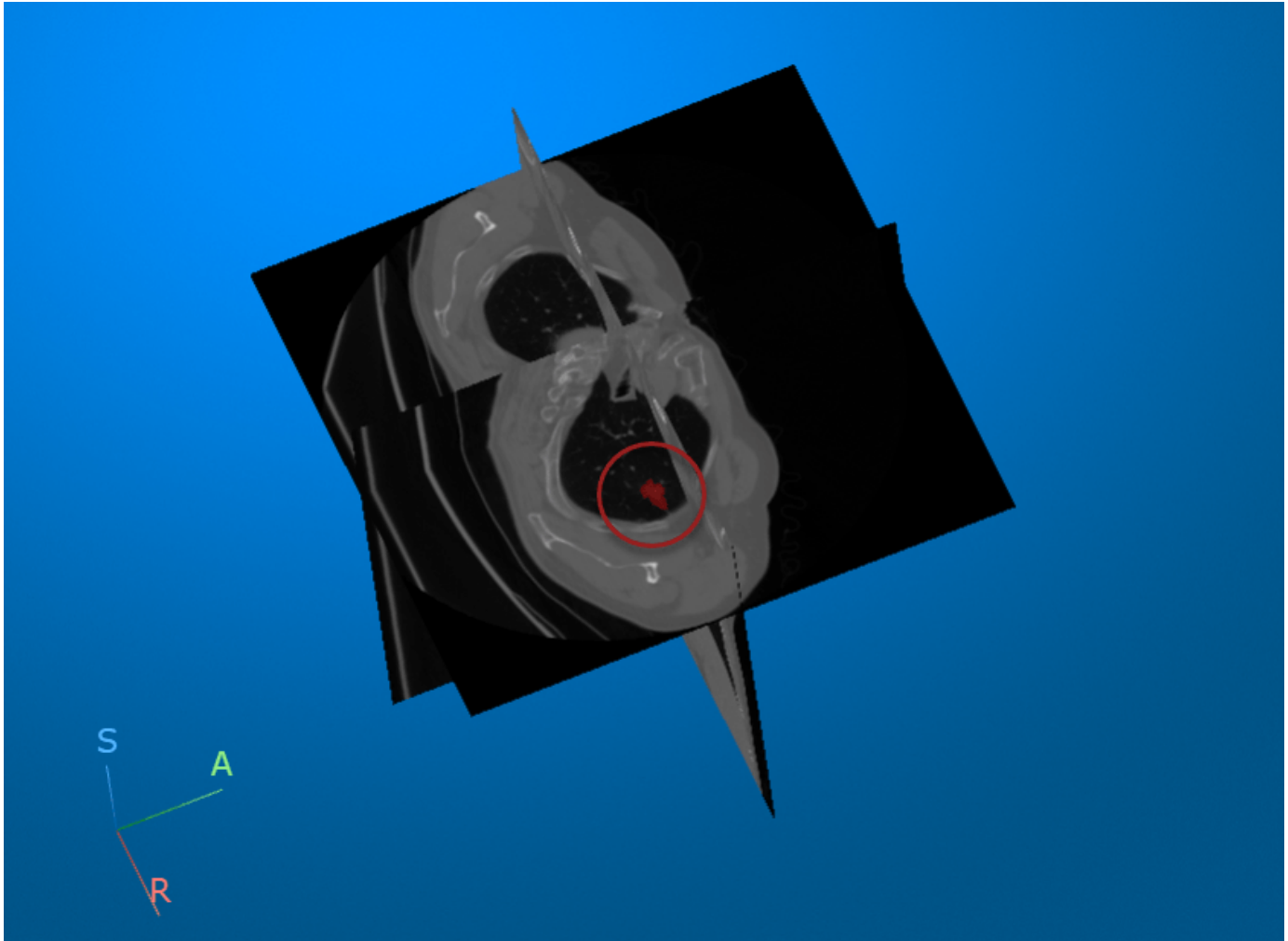
Visualize the CT volume and label overlay as slice planes. Use the `RenderingStyle` name-value argument to specify the rendering style as "SlicePlanes". Specify the tumor label overlay using the `OverlayData` name-value argument. Note that you must set the `OverlayData` property to the numeric array in the `Voxels` property of `medVolLabels`, rather than the `medicalVolume` object itself.

```
volSlice = volshow(medVolData, ...  
    OverlayData=medvolLabels.Voxels, ...  
    RenderingStyle="SlicePlanes", ...  
    Alphamap=linspace(0.01,0.2,256), ...  
    OverlayAlphamap=0.75);
```

To scroll through the transverse slices, pause the cursor on the transverse slice until it highlights in blue, then drag the cursor along the inferior/superior axis.



Drag the cursor to rotate the volume. The tumor overlay is visible in the slices for which the overlay is defined.



References

[1] Medical Segmentation Decathlon. "Lung." Tasks. Accessed May 10, 2018. <http://medicaldecathlon.com/>. The Medical Segmentation Decathlon data set is provided under the CC-BY-SA 4.0 license. All warranties and representations are disclaimed. See the license for details.

See Also

`volshow` | `Volume Properties` | `medicalref3d` | `medicalVolume` | `intrinsicToWorldMapping`

Related Examples

- "Display Medical Image Volume in Patient Coordinate System" on page 3-21

Create STL Surface Model of Femur Bone for 3-D Printing

This example shows how to convert a segmentation mask from a CT image into an STL surface model suitable for 3-D printing.

There are several clinical applications and research areas involving 3-D printing of medical image data:

- Surgical treatment planning using patient-specific anatomical models.
- Fabrication of custom prosthetics, implants, or surgical tools.
- Bioprinting of tissues and organs.

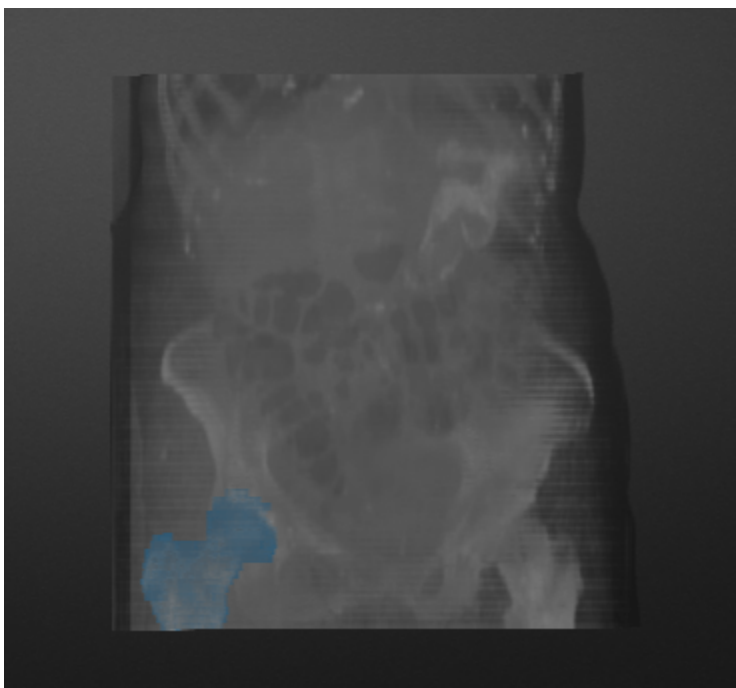
3-D printing creates physical models of computer-generated surface models. The typical workflow for 3-D printing anatomical structures from medical image volumes includes these steps:

- 1 Segment the region of interest, such as a bone, organ, or implant.
- 2 Convert the segmentation mask to a triangulated surface, defined by faces and vertices.
- 3 Write the triangulation to an STL file, which most commercial 3-D printers accept.
- 4 Import the STL file into the 3-D printing software, and print the model.

This example converts a mask of a femur bone into a triangulated surface, and writes an STL file suitable for 3-D printing.

Download Image Volume Data

This example uses a binary segmentation mask of a femur. The mask was created by segmenting a CT scan from the Medical Segmentation Decathlon liver data set [1 on page 3-37] using the Medical Image Labeler app. For an example of how to segment medical image volumes, see “Label 3-D Medical Image Using Medical Image Labeler” on page 5-10.



When you label an image in the **Medical Image Labeler**, the app saves the segmentation mask as a NIFTI file. You can find the file path of the mask generated in an app session by checking the `LabelData` property of the `groundTruthMedical` object exported from that session.

This example downloads the femur mask as part of a data set containing the original abdominal CT volume as well as masks of the femur and liver. Download the data set from the MathWorks® website, then unzip the folder.

```
zipFile = matlab.internal.examples.downloadSupportFile("medical", ...  
    "MedicalVolumeLiverNIFTIData.zip");  
filepath = fileparts(zipFile);  
unzip(zipFile,filepath)  
dataFolder = fullfile(filepath,"MedicalVolumeLiverNIFTIData");
```

Specify the filename of the femur mask.

```
filename = fullfile(dataFolder,"Femur.nii");
```

Load CT Segmentation Mask

Import the femur mask by creating a `medicalVolume` object for the NIFTI file. The image data is stored in the `Voxels` property. The `VoxelSpacing` property indicates that the voxels are anisotropic, with a size of 0.7-by-0.7-by-5.0 mm.

```
medVol = medicalVolume(filename)
```

```
medVol =  
    medicalVolume with properties:  
  
        Voxels: [512x512x75 uint8]  
    VolumeGeometry: [1x1 medicalref3d]  
      SpatialUnits: "mm"  
      Orientation: "transverse"  
    VoxelSpacing: [0.7031 0.7031 5]  
    NormalVector: [0 0 1]  
    NumCoronalSlices: 512  
    NumSagittalSlices: 512  
    NumTransverseSlices: 75  
      PlaneMapping: ["sagittal"    "coronal"    "transverse"]  
      Modality: "unknown"  
    WindowCenters: 0  
    WindowWidths: 0
```

The `VolumeGeometry` property of a medical volume object contains a `medicalref3d` object that specifies the spatial referencing information. Extract the `medicalref3d` object for the CT scan into a new variable, `R`.

```
R = medVol.VolumeGeometry;
```

Extract Isosurface of Femur

Specify the isovalue for isosurface extraction.

```
isovalue = 0.05;
```

Extract the vertices and faces that define an isosurface for the image volume at the specified isovalue.

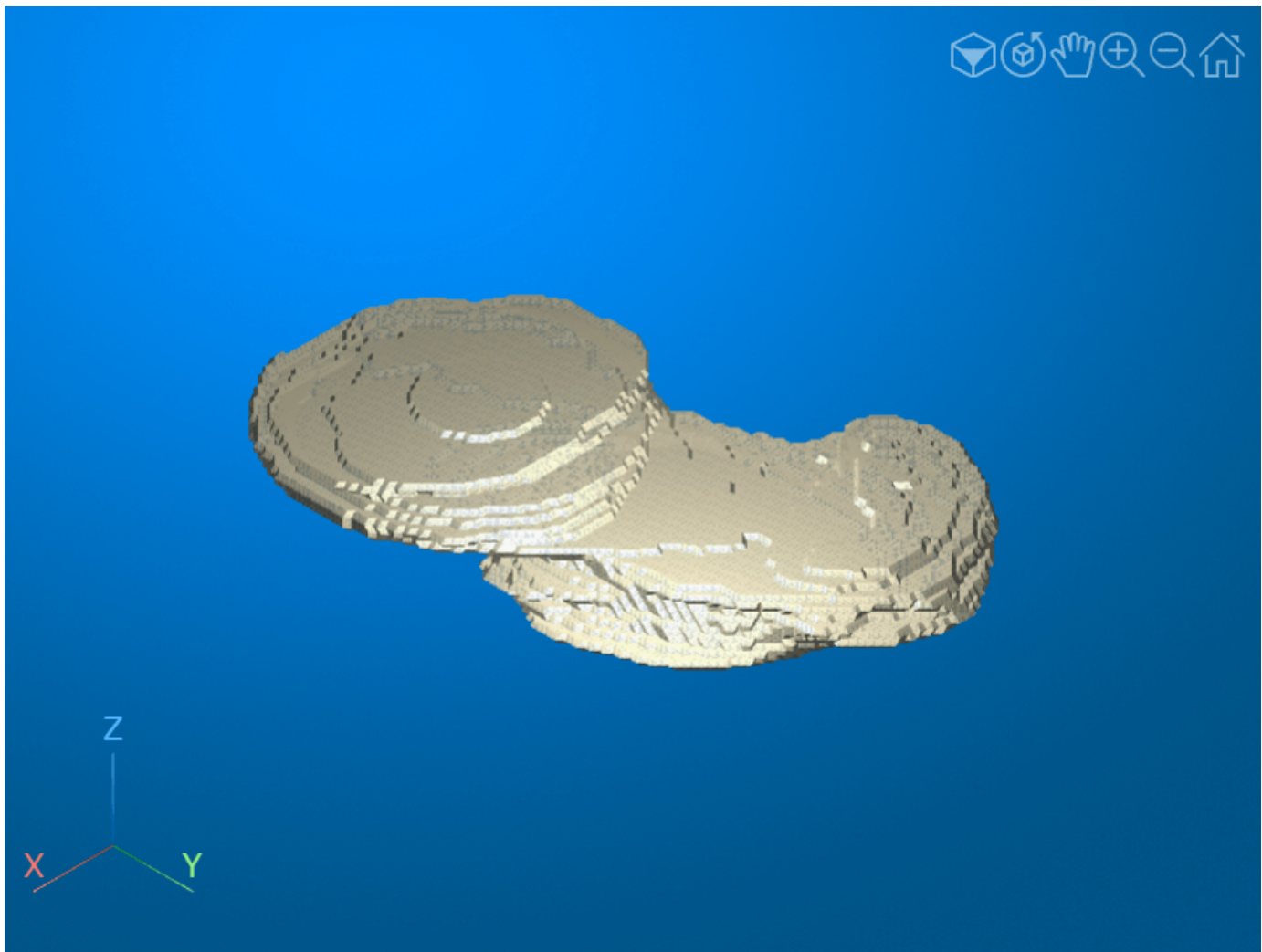
```
[faces,vertices] = extractIsosurface(medVol.Voxels,isovalue);
```

The `vertices` output provides the intrinsic *ijk*-coordinates of the surface points, in voxels. Create a triangulation of the vertices in intrinsic coordinates.

```
triIntrinsic = triangulation(double(faces),double(vertices));
```

If you display the surface defined by `vertices` in intrinsic coordinates, the femur surface appears squished. Intrinsic coordinates do not account for the real-world voxel dimensions, and the voxels in this example are larger in the third dimension than in the first two dimensions.

```
viewerIntrinsic = viewer3d;  
obj = images.ui.graphics3d.Surface(viewerIntrinsic, ...  
    Data=triIntrinsic, ...  
    Color=[0.88 0.84 0.71], ...  
    Alpha=0.9);
```



Transform Vertices into Patient Coordinates

To accurately represent the femur surface points, transform the *ijk*-coordinates in `vertices` to the patient coordinate system defined by the `medicalref3d` object, `R`. The `X`, `Y`, and `Z` outputs define the *xyz*-coordinates of the surface points, in millimeters.

```
I = vertices(:,1);  
J = vertices(:,2);  
K = vertices(:,3);
```

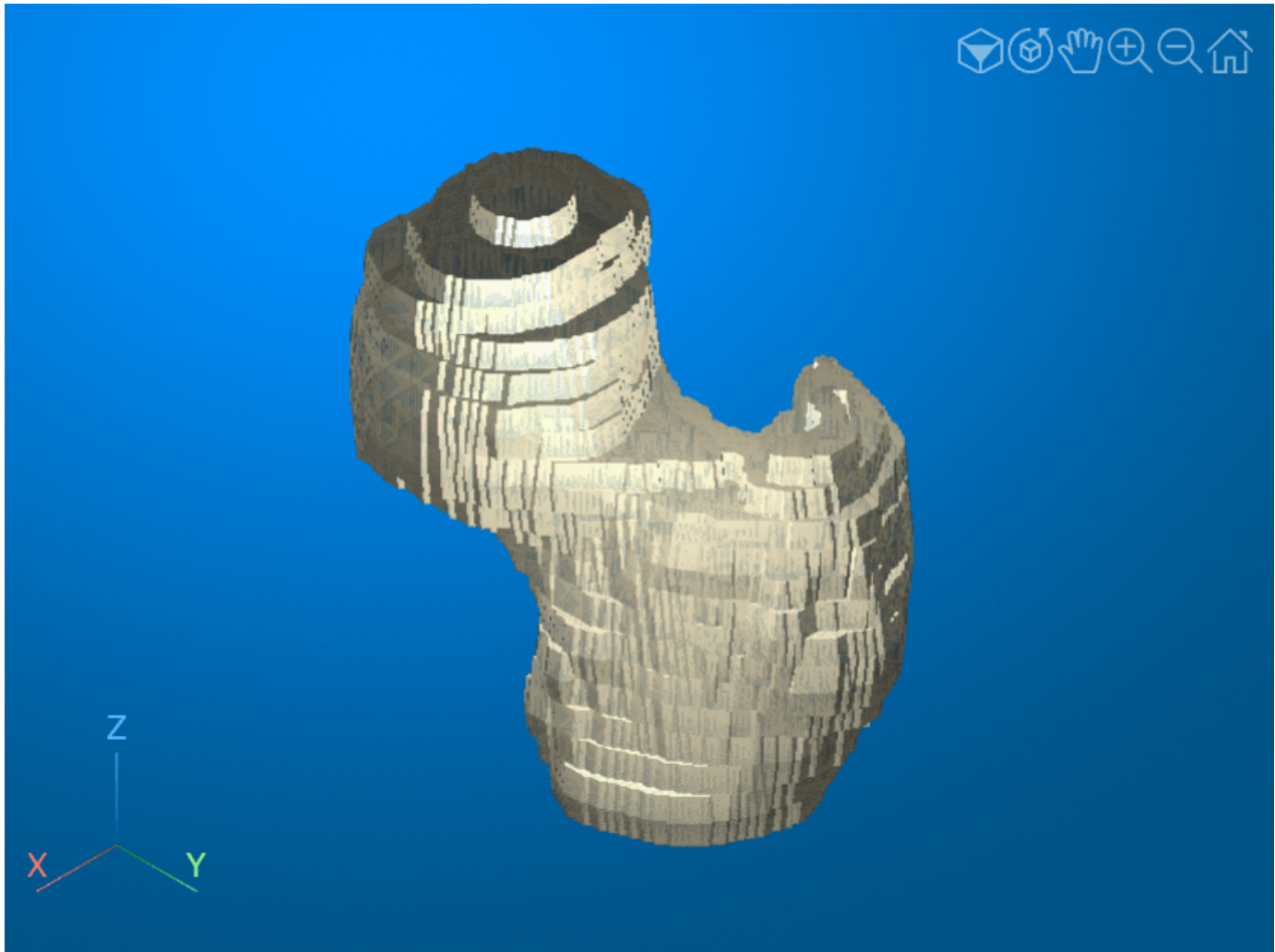
```
[X,Y,Z] = intrinsicToWorld(R,I,J,K);  
verticesPatientCoords = [X Y Z];
```

Create a triangulation of the transformed vertices, `verticesPatientCoords`. Maintain the original connectivity defined by `faces`.

```
triPatient = triangulation(double(faces),double(verticesPatientCoords));
```

Display the surface defined by the transformed patient coordinates. The femur surface no longer appears squished.

```
viewerPatient = viewer3d;  
obj = images.ui.graphics3d.Surface(viewerPatient, ...  
    Data=triPatient, ...  
    Color=[0.88 0.84 0.71], ...  
    Alpha=0.9);
```



Create STL File

Check the number of vertices in the surface, which corresponds to the length of the `Points` property.

```
triPatient
```

```
triPatient =
  triangulation with properties:
      Points: [21950x3 double]
  ConnectivityList: [43896x3 double]
```

You can reduce the number of triangles required to represent a surface, while preserving the shape of the associated object, by using the `reducepatch` function. Reduce the number of triangles in the femur surface by 50%. The `reducepatch` function returns a structure with fields `faces` and `vertices`.

```
pReduced = reducepatch(faces,verticesPatientCoords,0.5);
```

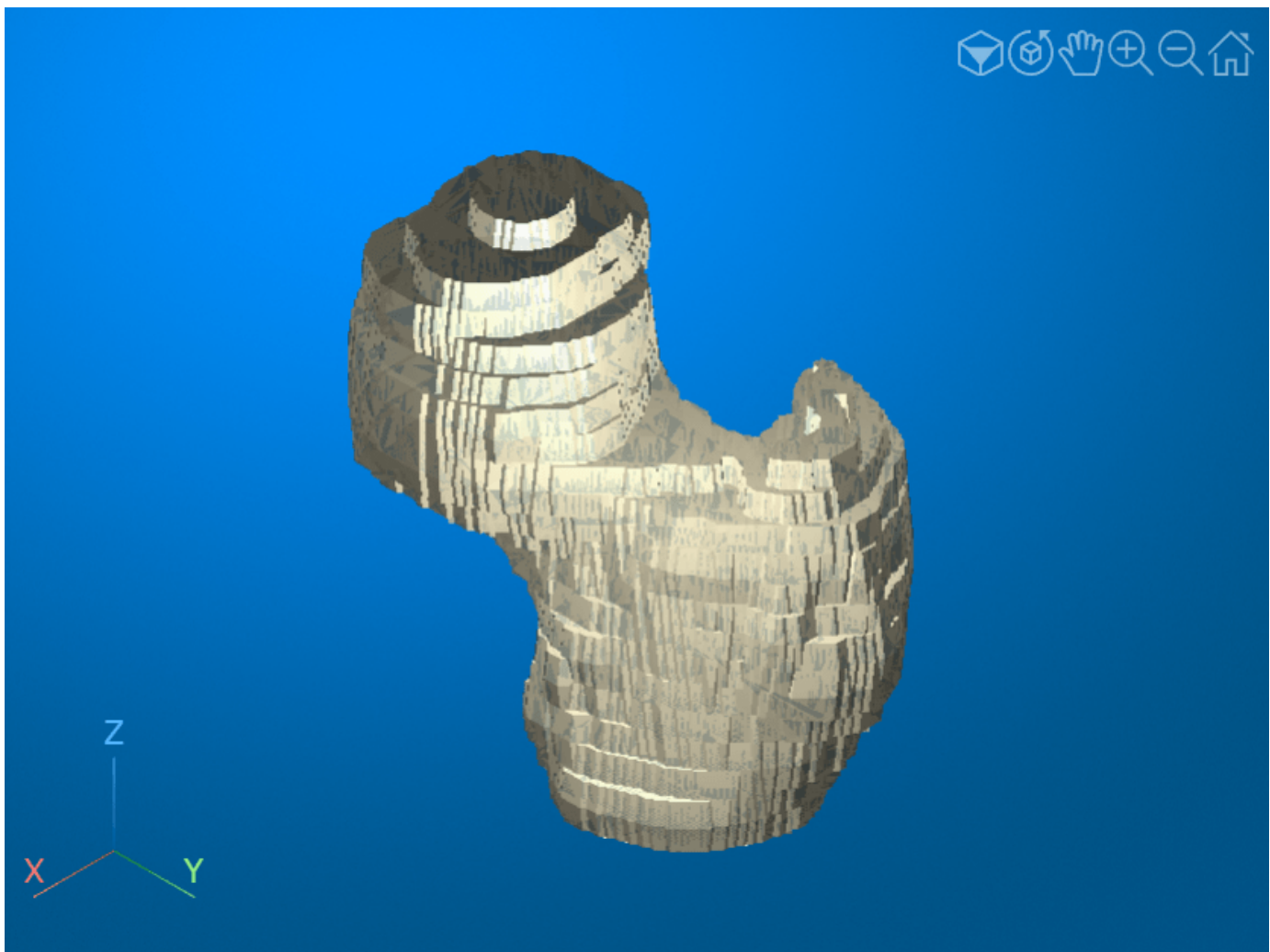
Create a triangulation of the reduced surface. The number of vertices, specified by the `Points` property, is approximately one-half of the number of vertices in `triPatient`.

```
triReduced = triangulation(double(pReduced.faces),double(pReduced.vertices))

triReduced =
    triangulation with properties:
        Points: [10976x3 double]
        ConnectivityList: [21948x3 double]
```

Display the reduced surface to verify that the shape of the femur is preserved.

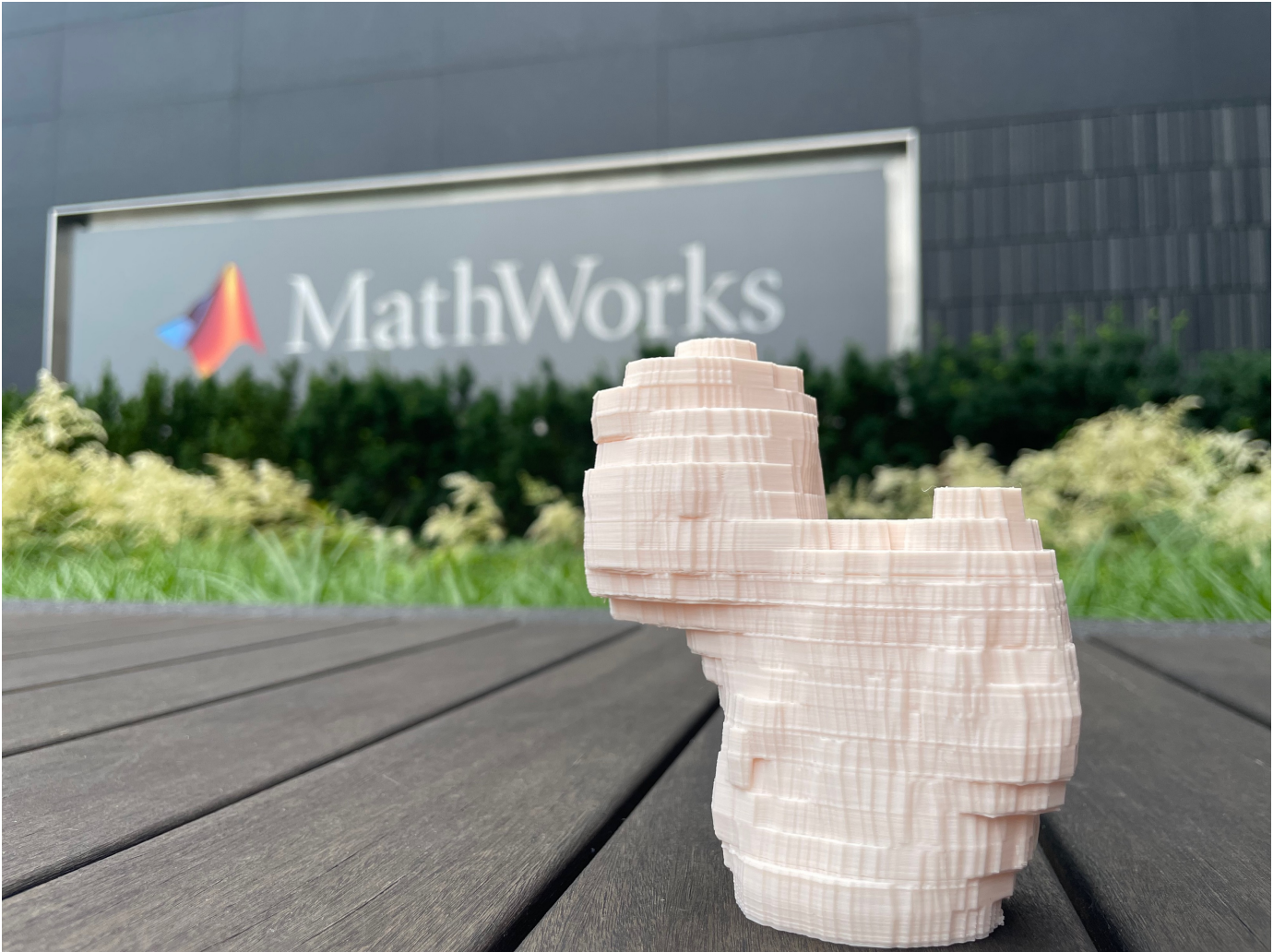
```
viewerPatient = viewer3d;
obj = images.ui.graphics3d.Surface(viewerPatient, ...
    Data=triReduced, ...
    Color=[0.88 0.84 0.71], ...
    Alpha=0.9);
```



Write the surface data to an STL format file by using the `stlwrite` function.


```
Name = "Femur.stl";  
stlwrite(triReduced,Name)
```

You can use the STL model file as input to most commercial 3-D printers to generate a physical 3-D model of the femur. This image shows a 3-D print of the STL file.



References

[1] Medical Segmentation Decathlon. "Liver." Tasks. Accessed May 10, 2018. <http://medicaldecathlon.com/>.

The Medical Segmentation Decathlon data set is provided under the CC-BY-SA 4.0 license. All warranties and representations are disclaimed. See the license for details.

See Also

Apps
Medical Image Labeler

Functions

`extractIsosurface` | `stlwrite` | `triangulation` | `patch` | `reducepatch`

Objects

`medicalref3d` | `medicalVolume`

Related Examples

- “Display Medical Image Volume in Patient Coordinate System” on page 3-21

Medical Image-Based Finite Element Analysis of Spine

This example shows how to estimate bone stress and strain in a vertebra bone under axial compression using finite element (FE) analysis.

Bones experience mechanical loading due to gravity, motion, and muscle contraction. Estimating the stresses and strains within bone tissue can help predict bone strength and fracture risk. This example uses image-based FE analysis to predict bone stress and strain within a vertebra under axial compression. Image-based FE uses medical images, such as computed tomography (CT) scans, to generate the geometry and material properties of the FE model.

In this example, you create and analyze a 3-D model of a single vertebra under axial loading using this workflow:

- 1 Segment vertebra from CT scan on page 3-39.
- 2 Extract vertebra geometry and generate FE mesh on page 3-40.
- 3 Assign material properties on page 3-43.
- 4 Apply loading and solve model on page 3-44.
- 5 Analyze results on page 3-45.

Download Data

This example uses a CT scan saved as a directory of DICOM files. The scan is part of a data set that contains three CT volumes. Run this code to download the data set from the MathWorks website as a zip file and unzip the file. The total size of the data set is approximately 81 MB.

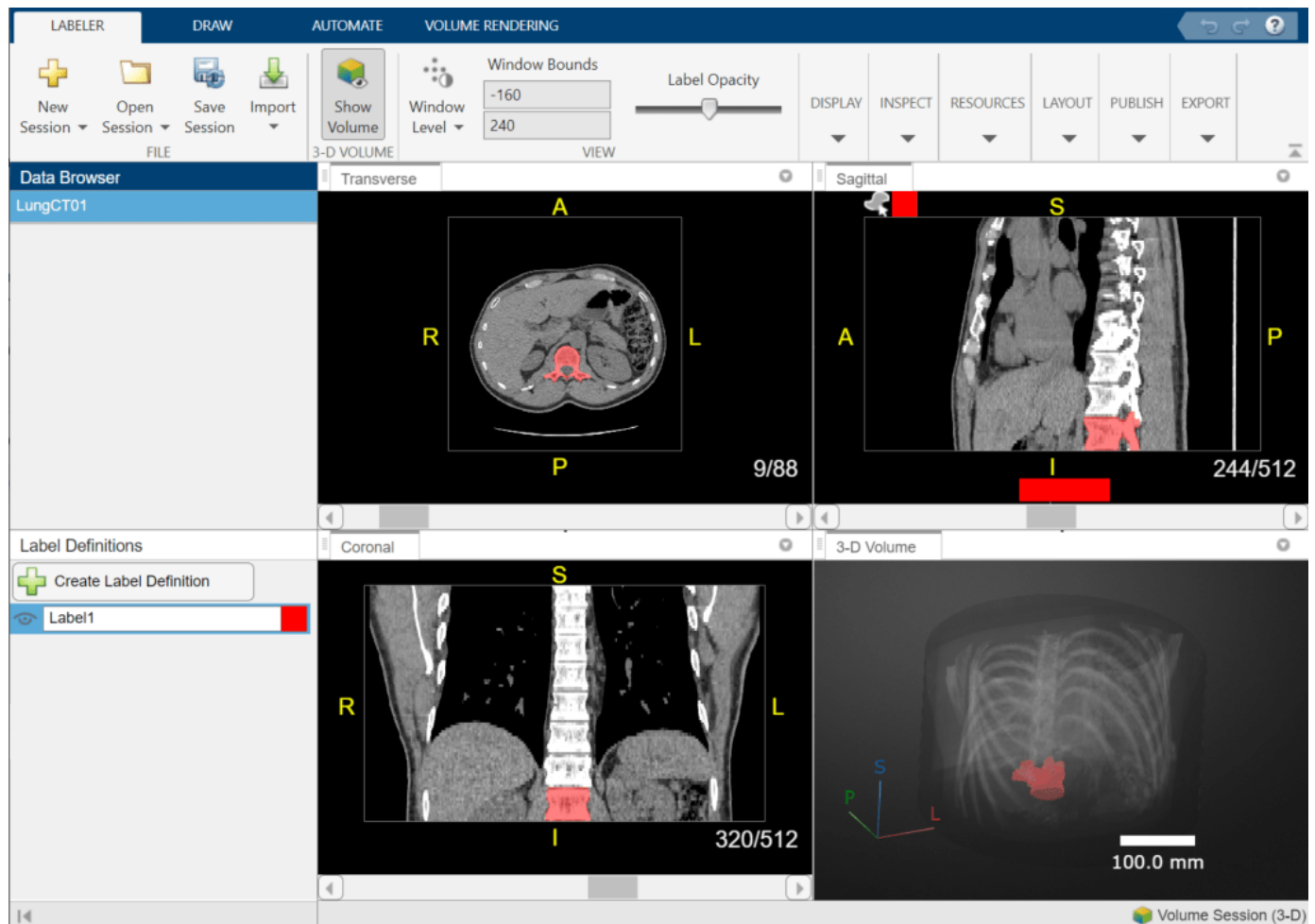
```
zipFile = matlab.internal.examples.downloadSupportFile("medical", "MedicalVolumeDICOMData.zip");  
filepath = fileparts(zipFile);  
unzip(zipFile, filepath)
```

After you download and unzip the data set, the scan used in this example is available in the `dataFolder` directory.

```
dataFolder = fullfile(filepath, "MedicalVolumeDICOMData", "LungCT01");
```

Segment Vertebra from CT Scan

This example uses a binary segmentation mask of one vertebra in the spine. The mask was created by segmenting the spine from a chest CT scan using the **Medical Image Labeler** app. For an example of how to segment medical image volumes in the app, see “Label 3-D Medical Image Using Medical Image Labeler” on page 5-10.



The segmentation mask is attached to this example as a supporting file. Load the mask as a `medicalVolume` object.

```
labelVol = medicalVolume("LungCT01.nii.gz");
```

The `VolumeGeometry` property of a `medicalVolume` object contains a `medicalref3d` object that defines the patient coordinate system for the scan. Extract the `medicalref3d` object for the mask into a new variable, `R`.

```
R = labelVol.VolumeGeometry;
```

Extract Vertebra Geometry and Generate FE Mesh

Extract Isosurface of Vertebra Mask

Calculate the isosurface faces and vertices for the vertebra mask by using the `extractIsosurface` function. The vertices coordinates are in the intrinsic coordinate system defined by rows, columns, and slices.

```
isovalue = 1;
[faces,vertices] = extractIsosurface(labelVol.Voxels,isovalue);
```

Transform vertices to the patient coordinate system defined by R by using the `intrinsicToWorld` object function. The X, Y, and Z outputs define the xyz-coordinates of the surface points, in millimeters.

```
I = vertices(:,1);
J = vertices(:,2);
K = vertices(:,3);
```

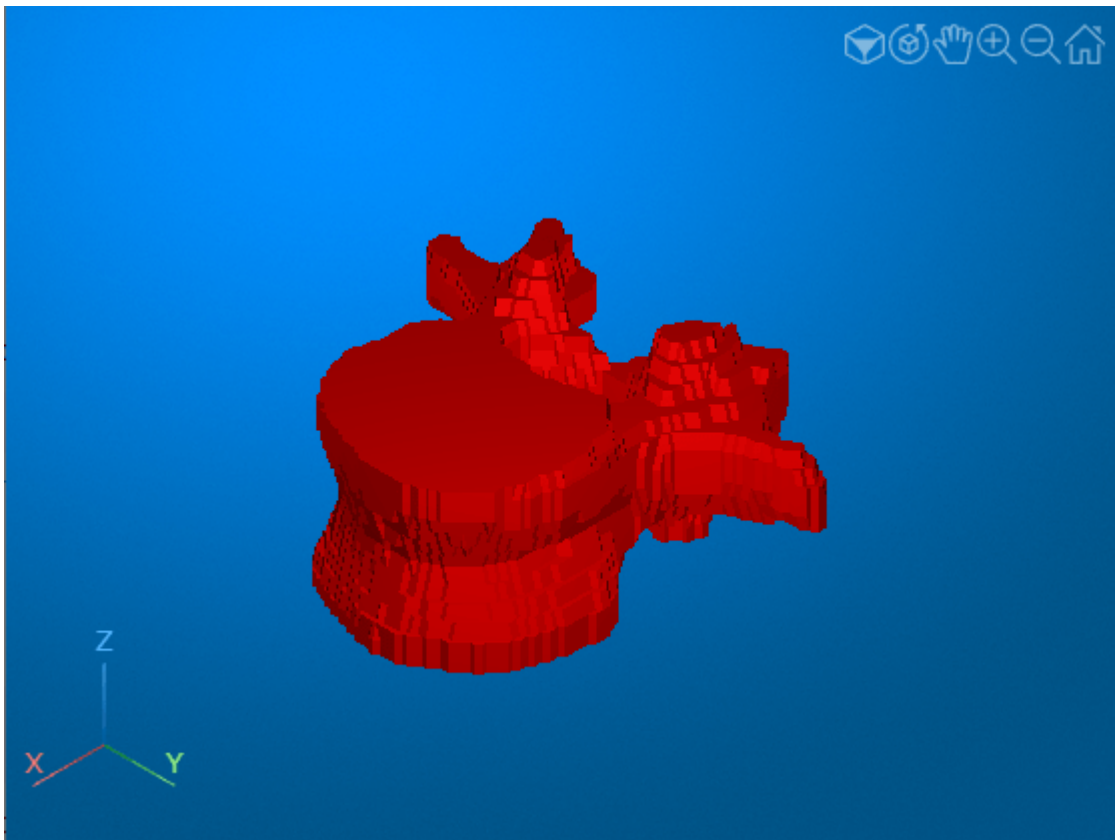
```
[X,Y,Z] = intrinsicToWorld(R,I,J,K);
verticesPatient = [X Y Z];
```

Convert the vertex coordinates to meters.

```
verticesPatientMeters = verticesPatient.*10^-3;
```

Display the vertebral isosurface as a triangulated Surface object.

```
triangul = triangulation(double(faces),double(verticesPatientMeters));
viewer = viewer3d;
surface = images.ui.graphics3d.Surface(viewer,data=triangul,Color=[1 0 0],Alpha=1);
```



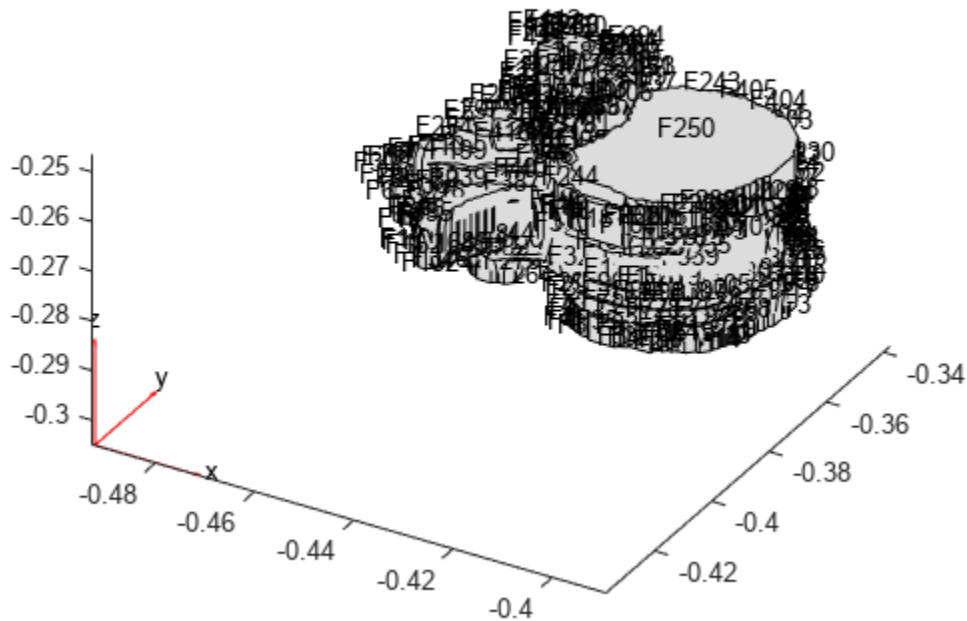
Create PDE Model Geometry

Create a general `PDEModel` model container for a system of three equations. A general model allows you to assign spatially varying material properties based on bone density.

```
model = createpde(3);
```

Specify the geometry for the model by using the `geometryFromMesh` (Partial Differential Equation Toolbox) object function, with the isosurface in patient coordinates as input.

```
geometryFromMesh(model,verticesPatientMeters',faces');  
pdegplot(model,FaceLabels="on")
```



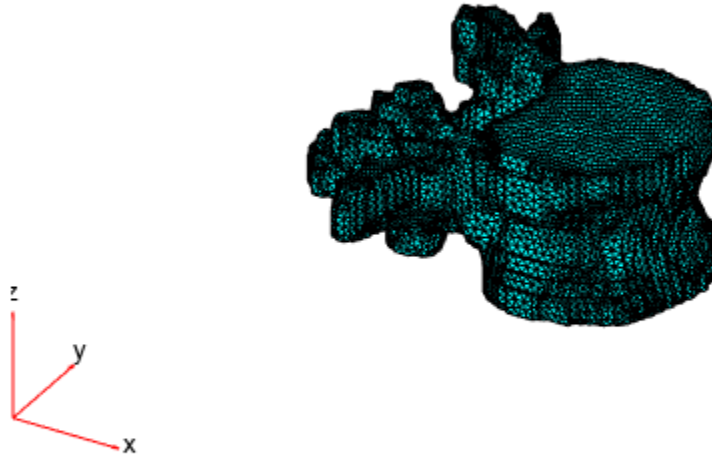
Generate Mesh from Model Geometry

Specify the maximum edge length for the FE mesh elements. This example uses a max edge length of 1.6 mm, which was determined using a mesh convergence analysis.

```
hMax = 0.0016;
```

Generate the mesh from the model geometry. Specify the maximum edge length, and specify that the minimum edge length is half of the maximum edge length. The elements are quadratic tetrahedra.

```
msh = generateMesh(model,Hmax=hMax,Hmin=hMax/2);  
pdemesh(model);
```



Assign Material Properties

Bone material properties vary based on spatial location and direction:

- Spatial location — Local regions with greater bone density are stiffer than low density regions.
- Direction — Bone stiffness depends on the loading direction. This example models bone using *transverse isotropy*. A transversely isotropic material is stiffer in the axial direction versus in the transverse plane, and is uniform within the transverse plane.

In this example, you assign spatially varying, transversely isotropic linear elastic material properties.

Extract HU Values Within Vertebra Mask

To assign spatially varying material properties, you need to map CT intensity values to FE mesh coordinates.

Load the original, unsegmented CT scan using a `medicalVolume` object. The `medicalVolume` object automatically converts the intensity to Hounsfield units (HU). The HU value is a standard for measuring radiodensity.

```
DCMData = medicalVolume(dataFolder);
```

Extract the indices and intensities of CT voxels inside the vertebra mask.

```
trueIdx = find(labelVol.Voxels==1);
HUVertebra = double(DCMData.Voxels(trueIdx));
```

Convert the linear indices in `trueIdx` into subscript indices, in units of voxels.

```
[row,col,slice] = ind2sub(size(labelVol.Voxels),trueIdx);
```

Transform the subscript indices into patient coordinates and convert the coordinates from millimeters to meters.

```
[X2,Y2,Z2] = intrinsicToWorld(R,col,row,slice);  
HUVertebraMeters = [X2 Y2 Z2].*10^-3;
```

Map HU Values from CT Voxels to Mesh Nodes

The FE node coordinates are scattered and not aligned with the CT voxel grid. Create a `scatteredInterpolant` object to define the 3-D interpolation between the voxel grid and FE nodes.

```
F = scatteredInterpolant(HUVertebraMeters,HUVertebra);
```

Specify Coefficients

Specify the PDE coefficients for the model. In this structural analysis, the *c* coefficient defines the material stiffness of the model, which is inversely related to compliance. To define a spatially varying *c* coefficient in Partial Differential Equation Toolbox™, represent *c* in function form. In this example, the `HU2TransverseIsotropy` on page 3-49 helper function defines transversely isotropic material properties based on bone density. Bone density is calculated for a given location using the scattered interpolant *F*. The helper function is wrapped in an anonymous function, `ccoefffunc`, which passes the location and state structures to `HU2TransverseIsotropy`. The FE solver automatically computes location and state structure arrays and passes them to your function during the simulation.

```
ccoefffunc = @(location,state) HU2TransverseIsotropy(location,state,F);  
specifyCoefficients(model,'m',0, ...  
    'd',0, ...  
    'c',ccoefffunc, ...  
    'a',0, ...  
    'f',[0;0;0]);
```

Apply Loading and Solve Model

To simulate axial loading of the vertebra, fix the bottom surface and apply a downward load to the top surface. To simulate distributed loading, the load is applied as a pressure.

Identify the faces in the model geometry to apply the boundary conditions. In this example, the faces were identified using visual inspection of the plot created above using `pdegplot`.

```
bottomSurfaceFace = 1;  
topSurfaceFace = 250;
```

Specify the total force to apply, in newtons.

```
forceInput = -3000;
```

Estimate the area of the top surface.

```
nf2=findNodes(model.Mesh,"region",Face=topSurfaceFace);  
positions = model.Mesh.Nodes(:,nf2)';  
surfaceShape = alphaShape(positions(:,1:2));  
faceArea = area(surfaceShape);
```


Calculate the pressure magnitude as a function of force and area, in pascals.

```
inputPressure_Pa = forceInput/faceArea;
```

Apply the boundary conditions.

```
applyBoundaryCondition(model, "dirichlet", Face=bottomSurfaceFace, u=[0,0,0]);
applyBoundaryCondition(model, "neumann", Face=topSurfaceFace, g=[0;0;inputPressure_Pa]);
```

Solve the model. The output is a `StationaryResults` (Partial Differential Equation Toolbox) object that contains nodal displacements and their gradients.

```
Rs = solvepde(model);
```

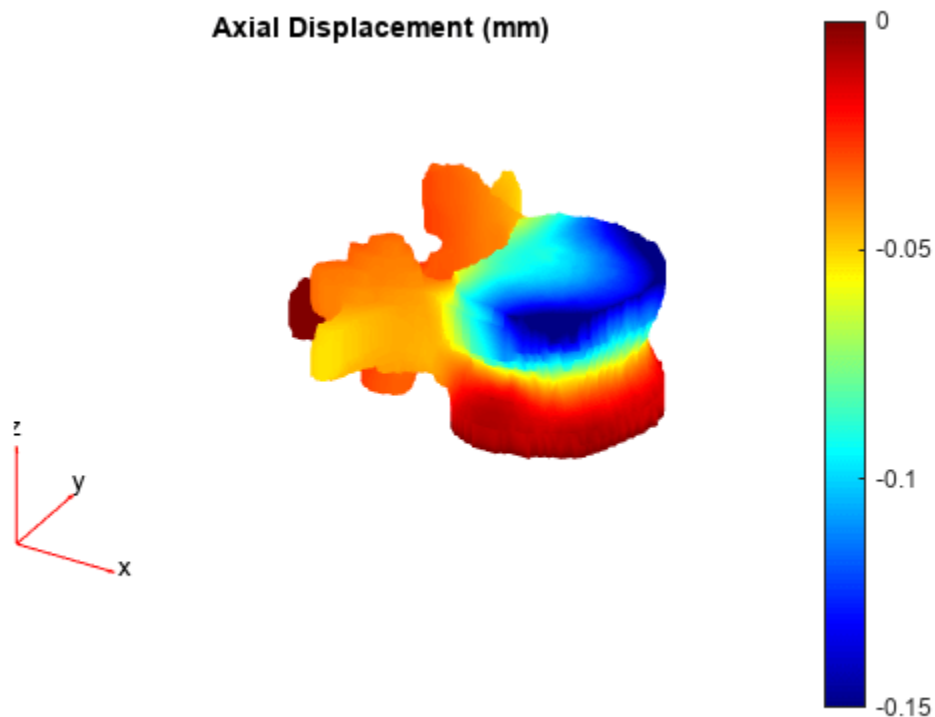
Analyze Results

View the results of the simulation by plotting the axial displacement, stress, and strain within the model.

Displacement

Plot axial displacement, in millimeters, for the full model by using the `pdeplot3D` function.

```
Uz = Rs.NodalSolution(:,3)*10^3;
pdeplot3D(model, "ColorMapData", Uz)
clim([-0.15 0])
title("Axial Displacement (mm)")
```



Plot displacement in a transverse slice at the midpoint of vertebral height.

Create a rectangular grid that covers the geometry of the transverse (xy) slice with spacing of 1 mm in each direction. The x and y vectors define the spatial limits and spacing in the transverse plane, and z is a scalar that provides the midpoint of vertebral height. The Xg , Yg , and Zg variables define the grid coordinates.

```
x = min(msh.Nodes(1,:)):0.001:max(msh.Nodes(1,:));  
y = min(msh.Nodes(2,:)):0.001:max(msh.Nodes(2,:));  
z = min(msh.Nodes(3,:))+0.5*(max(msh.Nodes(3,:))-min(msh.Nodes(3,:)));
```

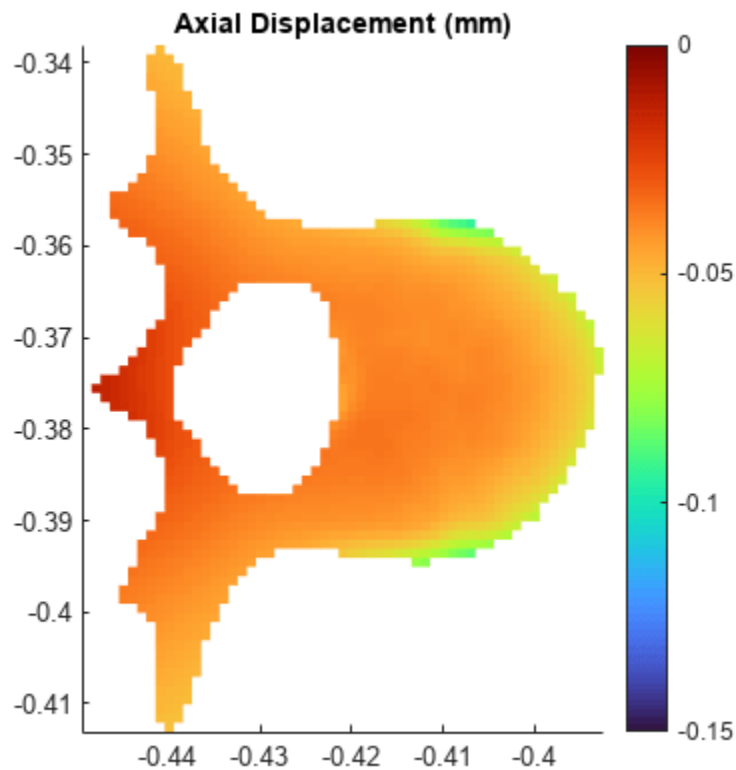
```
[Xg,Yg] = meshgrid(x,y);  
Zg = z*ones(size(Xg));
```

Interpolate normal axial displacement onto the grid coordinates by using the `interpolateSolution` (Partial Differential Equation Toolbox) object function. Convert displacement from meters to millimeters, and reshape the displacement vector to the grid size.

```
U = interpolateSolution(Rs,Xg,Yg,Zg,3);  
U = U*10^3;  
Ug = reshape(U,size(Xg));
```

Plot axial displacement in the transverse slice.

```
surf(Xg,Yg,Ug,LineStyle="none")  
axis equal  
grid off  
view(0,90)  
colormap("turbo")  
colorbar  
clim([-0.15 0])  
title("Axial Displacement (mm)")
```



Stress

In a structural analysis, stress is the product of the c coefficient and the gradients of displacement. Calculate normal stresses at the transverse slice grid coordinates by using the `evaluateCGradient` (Partial Differential Equation Toolbox) object function.

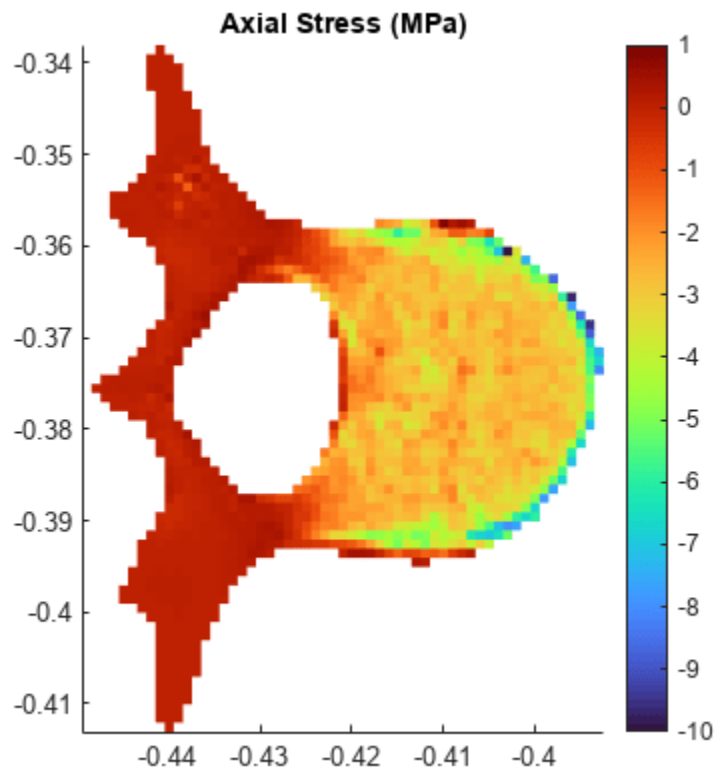
```
[cgradx,cgrady,cgradz] = evaluateCGradient(Rs,Xg,Yg,Zg,3);
```

Convert normal axial stress to megapascals, and reshape the stress vector to the grid size.

```
cgradz = cgradz*10^-6;
cgradzg = reshape(cgradz,size(Xg));
```

Plot the normal axial stress in the transverse slice.

```
surf(Xg,Yg,cgradzg,LineStyle="none");
axis equal
grid off
view(0,90)
colormap("turbo")
colorbar
clim([-10 1])
title("Axial Stress (MPa)")
```



Strain

In a structural analysis, strain is the gradient of the displacement result. Extract the normal strain at the transverse slice grid coordinates by using the `evaluateGradient` (Partial Differential Equation Toolbox) object function.

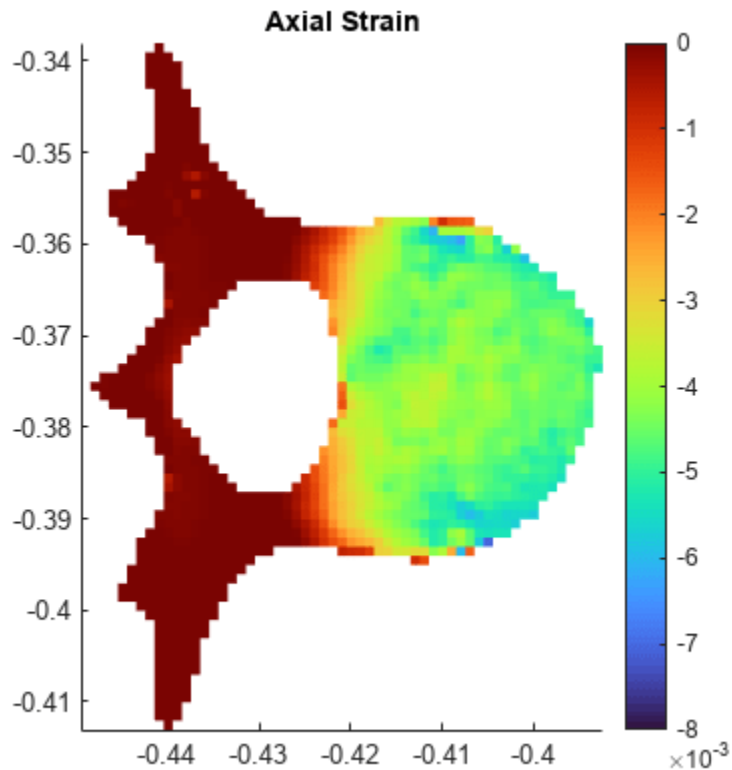
```
[gradx,grady,gradz] = evaluateGradient(Rs,Xg,Yg,Zg,3);
```

Reshape the axial strain vector to the grid size.

```
gradzg = reshape(gradz,size(Xg));
```

Plot the axial strain in the transverse slice.

```
surf(Xg,Yg,gradzg,LineStyle="none");  
axis equal  
grid off  
view(0,90)  
colormap("turbo")  
colorbar  
clim([-0.008 0])  
title("Axial Strain")
```



Helper Functions

The `HU2TransverseIsotropy` helper function specifies transverse isotropic material properties using these steps:

- Map voxel intensity to Hounsfield units (HU) at nodal locations using the `scatteredInterpolant` object, `F`.
- Convert HU to CT density using a linear calibration equation from [1 on page 3-51].
- Convert CT density to elastic modulus in the axial direction, `E3`, and in the transverse plane, `E12`, as well as the bulk modulus, `G`, using density-elasticity relationships from [2 on page 3-51]. The Poisson's ratio in the axial direction, `v3`, and transverse plane, `v12`, are also assigned based on [2].
- Define the 6-by-6 compliance matrix for transverse isotropy by using the `elasticityTransOrthoC3D` on page 3-50 helper function.
- Convert the 6-by-6 compliance, or *c*-matrix, to the vector form required by Partial Differential Equation Toolbox by using the `SixMat2NineMat` on page 3-50 and `SquareMat2CCoeffVec` on page 3-51 helper functions.

```
function ccoeff = HU2TransverseIsotropy(location,state,F)
    HU = F(location.x,location.y,location.z);
    rho = 5.2+0.8*HU;
    E3 = -34.7 + 3.230.*rho;
    E12 = 0.333*E3;
```

```
v12 = 0.104;
v3 = 0.381;
G = 0.121*E3;

ccoeff = zeros(45,length(location.x));
for i = 1:length(location.x)
    cMatrix = elasticityTransOrthoC3D(E12(i),E3(i),v12,v3,G(i));
    nineMat = SixMat2NineMat(cMatrix);
    ccoeff(:,i) = SquareMat2CCoeffVec(nineMat);
end
end
```

The `elasticityTransOrthoC3D` helper function defines the 6-by-6 compliance matrix for a transversely isotropic linear elastic material based on the elastic moduli, $E12$ and $E3$, bulk modulus, G , and Poisson's ratios, $v12$ and $v3$. The helper function converts all modulus values from megapascals to pascals before computing the compliance matrix.

```
function C = elasticityTransOrthoC3D(E12,E3,v12,v3,G)
    E12 = E12*10^6;
    E3 = E3*10^6;
    G = G*10^6;
    v_zp = (E3*v3)/E12;

    C = zeros(6,6);
    C(1,1) = 1/E12;
    C(1,2) = -v12/E12;
    C(1,3) = -v_zp/E3;
    C(2,1) = C(1,2);
    C(2,2) = 1/E12;
    C(2,3) = -v_zp/E3;
    C(3,1) = -v3/E12;
    C(3,2) = C(2,3);
    C(3,3) = 1/E3;
    C(4,4) = 1/(2*G);
    C(5,5) = 1/(2*G);
    C(6,6) = (1+v12)/E12;

    C=inv(C);
end
```

The `SixMat2NineMat` helper function converts a 6-by-6 c coefficient matrix in Voigt notation to a 9-by-9 matrix corresponding to expanded form.

```
function nineMat = SixMat2NineMat(sixMat)

    for i = 1:6
        nineVecs(i,:) = sixMat(i,[1 6 5 6 2 4 5 4 3]);
    end

    nineMat = [ nineVecs(1,:); ...
               nineVecs(6,:); ...
               nineVecs(5,:); ...
```

```

nineVecs(6,:); ...
nineVecs(2,:); ...
nineVecs(4,:); ...

nineVecs(5,:); ...
nineVecs(4,:); ...
nineVecs(3,:)]];

```

```
end
```

The `SquareMat2CCoeffVec` converts a 9-by-9 c coefficient matrix into vector form as required by Partial Differential Equation Toolbox. This helper function creates a 45-element vector, corresponding to the "3N(3N+1)/2-Element Column Vector c , 3-D Systems" case in "c Coefficient for specifyCoefficients" (Partial Differential Equation Toolbox).

```

function c45Vec = SquareMat2CCoeffVec(nineMat)

C11 = [nineMat(1,1) nineMat(1,2) nineMat(2,2) nineMat(1,3) nineMat(2,3) nineMat(3,3)];
C12 = [nineMat(1,4) nineMat(2,4) nineMat(3,4) nineMat(1,5) nineMat(2,5) nineMat(3,5) ...
      nineMat(1,6) nineMat(2,6) nineMat(3,6)];
C13 = [nineMat(1,7) nineMat(2,7) nineMat(3,7) nineMat(1,8) nineMat(2,8) nineMat(3,8) ...
      nineMat(1,9) nineMat(2,9) nineMat(3,9)];
C22 = [nineMat(4,4) nineMat(4,5) nineMat(5,5) nineMat(4,6) nineMat(5,6) nineMat(6,6)];
C23 = [nineMat(4,7) nineMat(5,7) nineMat(6,7) nineMat(4,8) nineMat(5,8) nineMat(6,8) ...
      nineMat(4,9) nineMat(5,9) nineMat(6,9)];
C33 = [nineMat(7,7) nineMat(7,8) nineMat(8,8) nineMat(7,9) nineMat(8,9) nineMat(9,9)];
c45Vec = [C11 C12 C22 C13 C23 C33]';

```

```
end
```

References

[1] Turbucz, Mate, Agoston Jakab Pokorni, György Szóke, Zoltan Hoffer, Rita Maria Kiss, Aron Lazary, and Peter Endre Eltes. "Development and Validation of Two Intact Lumbar Spine Finite Element Models for In Silico Investigations: Comparison of the Bone Modelling Approaches." *Applied Sciences* 12, no. 20 (January 2022): 10256. <https://doi.org/10.3390/app122010256>.

[2] Unnikrishnan, Ginu U., and Elise F. Morgan. "A New Material Mapping Procedure for Quantitative Computed Tomography-Based, Continuum Finite Element Analyses of the Vertebra." *Journal of Biomechanical Engineering* 133, no. 7 (July 1, 2011): 071001. <https://doi.org/10.1115/1.4004190>.

See Also

Medical Image Labeler | `medicalVolume` | `intrinsicToWorld` | `scatteredInterpolant` | `createpde` | `geometryFromMesh` | `solvepde` | `StationaryResults`

Related Examples

- "Medical Image Coordinate Systems" on page 1-16
- "Label 3-D Medical Image Using Medical Image Labeler" on page 5-10

- “Create STL Surface Model of Femur Bone for 3-D Printing” on page 3-31
- “c Coefficient for specifyCoefficients” (Partial Differential Equation Toolbox)

Image Preprocessing and Augmentation

- “Medical Image Preprocessing” on page 4-2
- “Medical Image Registration” on page 4-5
- “Register Multimodal Medical Image Volumes with Spatial Referencing” on page 4-9

Medical Image Preprocessing

In this section...

“Background Removal” on page 4-2

“Denoising” on page 4-2

“Resampling” on page 4-2

“Registration” on page 4-3

“Intensity Normalization” on page 4-3

“Preprocessing in Advanced Workflows” on page 4-4

Image preprocessing prepares data for a target workflow. The main goals of medical image preprocessing are to reduce image acquisition artifacts and to standardize images across a data set. Your exact preprocessing requirements depend on the modality and procedure used to acquire data, as well as your target workflow. Some common preprocessing steps include background removal, denoising, resampling, registration, and intensity normalization.

Background Removal

Background removal involves segmenting the region of interest from the image background. By limiting the image to the region of interest, you can improve the efficiency and accuracy of your target workflow. One example of background removal is skull stripping, which removes the skull and other background regions from MRI images of the brain. Background removal typically consists of applying a mask of the region of interest that you create using morphological operations or other segmentation techniques. For more information about morphological operations, see “Types of Morphological Operations”.

To perform background removal, multiply the mask image and the original image. For example, consider a grayscale image, `im`, and a mask image, `mask`, that is the same size as `im` and has a value of 1 for every element in the region of interest and 0 for each element of the background. This code returns a new image, `imROI`, in which the elements in the region of interest have the same values as in `im`, and the background elements all have values of 0.

```
imROI = im.*mask;
```

Denoising

Medical imaging modalities are susceptible to noise, which introduces random intensity fluctuations in an image. To reduce noise, you can filter images in the spatial and frequency domains. Medical Imaging Toolbox provides the `specklefilt` function, which reduces the speckle noise common in ultrasound images. For additional image filtering tools, see “Image Filtering” in Image Processing Toolbox. You can also denoise medical image data using deep learning. For details, see “Train and Apply Denoising Neural Networks”.

Resampling

Use resampling to change the pixel or voxel size of an image without changing its spatial limits in the patient coordinate system. Resampling is useful for standardizing image resolution across a data set that contains images from multiple scanners.

To resample 3-D image data in a `medicalVolume` object, use the `resample` object function. Using the `resample` object function maintains the spatial referencing of the volume.

To resample a 2-D medical image, you can use the `imresize` function with a target number of columns and rows. For example, this code shows how to resample the pixel data in the `medicalImage` object `medImg` to a target pixel size, specified by `targetPixelSpacing`.

```
ratios = medImg.PixelSpacing./targetPixelSpacing;
targetSize = ceil(ratios.*size(medImg.Pixels)); % in pixels
resamplePixels = imresize(medImg.Pixels,targetSize);
```

Define the spatial referencing of the original and resampled images by using the `imref2d` object.

```
originalR = imref2d(size(medImg.Pixels),medImg.PixelSpacing(1),medImg.PixelSpacing(2));
resampledR = imref2d(size(resamplePixels),targetPixelSpacing(1),targetPixelSpacing(2));
```

Registration

You can use image registration to standardize the spatial alignment of 2-D or 3-D medical images in a data set. Registration can be useful for aligning images of different patients, or images of the same patient acquired at different times, on different scanners, or using different imaging modalities. For more details, see “Medical Image Registration” on page 4-5.

Intensity Normalization

Intensity normalization standardizes the range of image intensity values across a data set. Typically, you perform this process in two steps. First, clip intensities to a smaller range. Second, normalize the clipped intensity range to the range of the image data type, such as `[0, 1]` for `double` or `[0, 255]` for `uint8`. Whereas visualizing image data using a display window changes how you view the data, intensity normalization actually updates the image values.

One normalization strategy is to rescale the intensity values within each image relative to the minimum and maximum values in the image. For example, this code uses the `rescale` function to limit the intensities in the image `im` to the 0.5 and 99.5 percentile values, and to rescale the values to the range `[0, 1]`:

```
imMax = prctile(im(:),99.5);
imMin = prctile(im(:),0.5);
imNormalized = rescale(im,0,1,InputMin=imMin,InputMax=imMax);
```

Another strategy is to rescale values to the same intensity range across images. In CT imaging, intensity windowing limits intensity values, in Hounsfield units (HU), to a suitable range for the tissue of interest. HU is a standard CT density scale defined by the density of air (-1000 HU) and water (0 HU). The `medicalVolume` object automatically rescales the data returned in the `Voxels` property for DICOM files that define the `RescaleIntercept` and `RescaleSlope` metadata attributes. You can find suggested intensity windows for your application in clinical guidelines or from your scanner manufacturer. For example, to segment lungs from a chest CT, you might use the range `[-1200, 600]` HU, and to segment bone, which is more dense, you might use the range `[-1400, 2400]`. This figure shows a transverse slice of a chest CT with no intensity windowing, a “lung” intensity window, and a “bone” intensity window.

This code applies intensity windowing to the image stored in the DICOM file `dicomFile`, assuming the values of `medVol.Voxels` are in units of HU:

```
medVol = medicalVolume(dicomFile);  
lungWindow = [-1200 600];  
imWindowed = rescale(medVol.Voxels,0,1,InputMin=lungwin(1),InputMax=lungwin(2));
```

Preprocessing in Advanced Workflows

For an example that preprocesses image data as part of a deep learning workflow, see “Brain MRI Segmentation Using Pretrained 3-D U-Net Network” on page 6-24.

See Also

`specklefilt` | `resample` | `rescale` | `medicalVolume`

Related Examples

- “Register Multimodal Medical Image Volumes with Spatial Referencing” on page 4-9
- “Brain MRI Segmentation Using Pretrained 3-D U-Net Network” on page 6-24
- “Segment Lungs from CT Scan Using Pretrained Neural Network” on page 6-14

More About

- “Types of Morphological Operations”
- “Train and Apply Denoising Neural Networks”
- “Medical Image Registration” on page 4-5

Medical Image Registration

Medical image registration is the process of aligning multiple medical images, volumes, or surfaces to a common coordinate system. In medical imaging, you may need to compare scans of multiple patients or scans of the same patient taken in different sessions under different conditions. Use medical image registration as a preprocessing step to align the medical images to a common coordinate system before you analyze them.

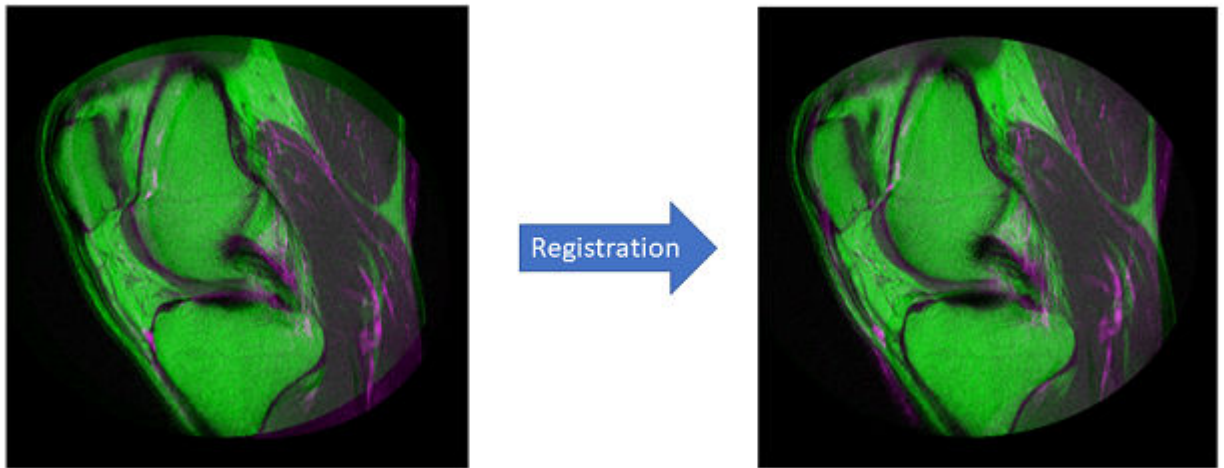
Scenarios for Medical Image Registration

Classification by Type of Misalignment

- *Translation registration* - Required when the two images, volumes, or surfaces differ by a global shift common to all pixels, voxels, or points.



- *Rigid registration* - Required when the two images, volumes, or surfaces differ by a global shift and a global rotation common to all pixels, voxels, or points.
- *Similarity registration* - Required when the two images, volumes, or surfaces differ by a global shift, a global rotation, and a global scale factor common to all pixels, voxels, or points.
- *Affine registration* - Required when the two images, volumes, or surfaces differ by a global shift, a global rotation, a global scale factor, and a global shear factor common to all pixels, voxels, or points.



- *Deformable registration* (also known as non-rigid registration) - Required when the images, volumes, or surfaces differ by local transformations specific to certain pixels, voxels, or points.

Classification by Type of Input

- *Image registration* - Aligns 2-D grayscale images.
- *Volume registration* - Aligns 3-D intensity volumes.
- *Surface registration* - Aligns surfaces extracted from 3-D intensity volumes.
- *Groupwise registration* - Aligns slices in a series of 2-D medical images, such as a timeseries, to reduce sliding motion between the slices.

Functions for Medical Image Registration

Medical Imaging Toolbox provides various functions for medical image registration.

Function	Type of Misalignment	Type of Input	Function Details	Method
<code>imregister</code>	Translation Rigid Similarity Affine	2-D grayscale images 3-D intensity volumes	Specify configuration using <code>imregconfig</code> . Transform returned by <code>imregtform</code> .	Optimization-based technique. Regular step gradient descent optimizer with mean squares metric for monomodal configuration. One-plus-one evolutionary optimizer with Mattes mutual information metric for multimodal configuration.

Function	Type of Misalignment	Type of Input	Function Details	Method
imregicp	Translation Rigid	Surfaces	Transform returned by function itself.	Optimization-based technique. Uses the iterative closest point (ICP) algorithm.
imregmtb	Translation	2-D grayscale images 3-D RGB images	Transform returned by function itself.	Fast registration technique. Uses median threshold bitmap (MTB) method. Suitable for monomodal and multimodal images.
imregmoment	Translation Rigid Similarity	2-D grayscale images 3-D intensity volumes	Transform returned by function itself.	Fast registration technique. Uses moment of mass method, with the option to also use the median threshold bitmap (MTB) method. Suitable for monomodal and multimodal images and volumes.
imregdemons	Deformable	2-D grayscale images 3-D intensity volumes	Displacement field returned by function itself.	Deformable registration technique. Uses a diffusion-based Demons algorithm.
imregdeform	Deformable	2-D grayscale images 3-D intensity volumes	Displacement field returned by function itself.	Deformable registration technique. Uses the isotropic total variation regularization method.
imreggroupwise	Deformable	3-D image series consisting of 2-D slices	Does not require a reference image. Displacement field returned by function itself.	Groupwise registration technique. Uses the isotropic total variation regularization method.

Function	Type of Misalignment	Type of Input	Function Details	Method
imregcorr	Translation Rigid Similarity	2-D grayscale images 3-D RGB images	Function returns only the transform. Use <code>imwarp</code> to apply the transformation and get the registered image.	FFT-based technique. Uses the phase correlation method.

See Also

Functions

`imregister` | `imregtform` | `imregconfig` | `imregicp` | `imregmtb` | `imregmoment` | `imregdemons` | `imregdeform` | `imreggroupwise` | `imregcorr` | `imwarp`

Related Examples

- “Register Multimodal Medical Image Volumes with Spatial Referencing” on page 4-9

Register Multimodal Medical Image Volumes with Spatial Referencing

This example shows how to align two medical image volumes using moment-of-mass-based registration.

Multimodal image registration aligns images acquired using different imaging modalities, such as MRI and CT. Even when acquired for the same patient and region of interest, multimodal images can be misaligned due to differences in patient positioning (translation or rotation) and voxel size (scaling). Different imaging modalities often have different voxel sizes due to scanner variability and concerns about scan time and radiation dose. The `imregmoment` function enables fast registration of 3-D image volumes, including multimodal images.

Medical images have an intrinsic coordinate system defined by rows, columns, and slices and a patient coordinate system that describes the real-world position and voxel spacing. You can use the `medicalVolume` object to import and manage the spatial referencing of a medical image volume. To register images with different voxel sizes using `imregmoment`, you must use spatial referencing to register the images in the patient coordinate system. During registration, `imregmoment` aligns a moving image with a fixed image, and then resamples the aligned image in the intrinsic coordinates of the fixed image. The registered volumes share one patient coordinate system and have the same number of rows, columns, and slices.

In this example, you use `medicalVolume` and `imregmoment` to register multimodal MRI and CT images of the head.

Load Images

The data used in this example is a modified version of the 3-D CT and MRI data sets from The Retrospective Image Registration Evaluation (RIRE) Dataset, provided by Dr. Michael Fitzpatrick. For more information, see the RIRE Project homepage. The modified data set contains one CT scan and one MRI scan stored in the NRRD file format. The size of the entire data set is approximately 35 MB. Download the data set from the MathWorks® website, then unzip the folder.

```
zipFile = matlab.internal.examples.downloadSupportFile("medical", ...
    "MedicalRegistrationNRRDdata.zip");
filepath = fileparts(zipFile);
unzip(zipFile,filepath)
```

In image registration, consider one image to be the fixed image and the other image to be the moving image. The goal of registration is to align the moving image with the fixed image. In this example, the fixed image is a T1 weighted MRI image. The moving image is a CT image from the same patient. The images are stored in the NRRD file format.

Use `medicalVolume` to read the MRI image. By looking at the `Voxels` and `VoxelSpacing` properties, you can determine that the MRI volume is a 256-by-256-by-26 voxel array, where each voxel is 1.25-by-1.25-by-4.0 mm.

```
filenameMRI = fullfile(filepath,"supportfilesNRRD","Patient007MRT1.nrrd");
fixedMRIVolume = medicalVolume(filenameMRI)

fixedMRIVolume =
    medicalVolume with properties:
```

```

        Voxels: [256×256×26 single]
VolumeGeometry: [1×1 medicalref3d]
  SpatialUnits: "unknown"
  Orientation: "unknown"
  VoxelSpacing: [1.2500 1.2500 4]
  NormalVector: [0 0 1]
NumCoronalSlices: []
NumSagittalSlices: []
NumTransverseSlices: []
  PlaneMapping: ["unknown" "unknown" "unknown"]
  Modality: "unknown"
WindowCenters: []
WindowWidths: []

```

Import the CT image. The size of each voxel in the CT image is 0.65-by-0.65-by-4 mm, which is smaller than in the MRI image.

```

filenameCT = fullfile(filepath,"supportfilesNRRD","Patient007CT.nrrd");
movingCTVolume = medicalVolume(filenameCT)

```

```

movingCTVolume =
  medicalVolume with properties:

```

```

        Voxels: [512×512×28 single]
VolumeGeometry: [1×1 medicalref3d]
  SpatialUnits: "unknown"
  Orientation: "unknown"
  VoxelSpacing: [0.6536 0.6536 4]
  NormalVector: [0 0 1]
NumCoronalSlices: []
NumSagittalSlices: []
NumTransverseSlices: []
  PlaneMapping: ["unknown" "unknown" "unknown"]
  Modality: "unknown"
WindowCenters: []
WindowWidths: []

```

Extract the image voxel data from the `Voxels` property of the `medicalVolume` objects.

```

fixedMRIVoxels = fixedMRIVolume.Voxels;
movingCTVoxels = movingCTVolume.Voxels;

```

Display Unregistered Images

Use the `imshowpair` function to judge image alignment. First, calculate the location of the middle slice of each volume. The `VolumeGeometry` property of the `medicalVolume` object contains a `medicalref3d` object, which specifies spatial details about the volume. Use the `VolumeSize` property of each `medicalref3d` object to calculate the location of the middle slice of the corresponding volume, in voxels.

```

fixedVolumeSize = fixedMRIVolume.VolumeGeometry.VolumeSize;
movingVolumeSize = movingCTVolume.VolumeGeometry.VolumeSize;

```

```

centerFixed = fixedVolumeSize/2;
centerMoving = movingVolumeSize/2;

```

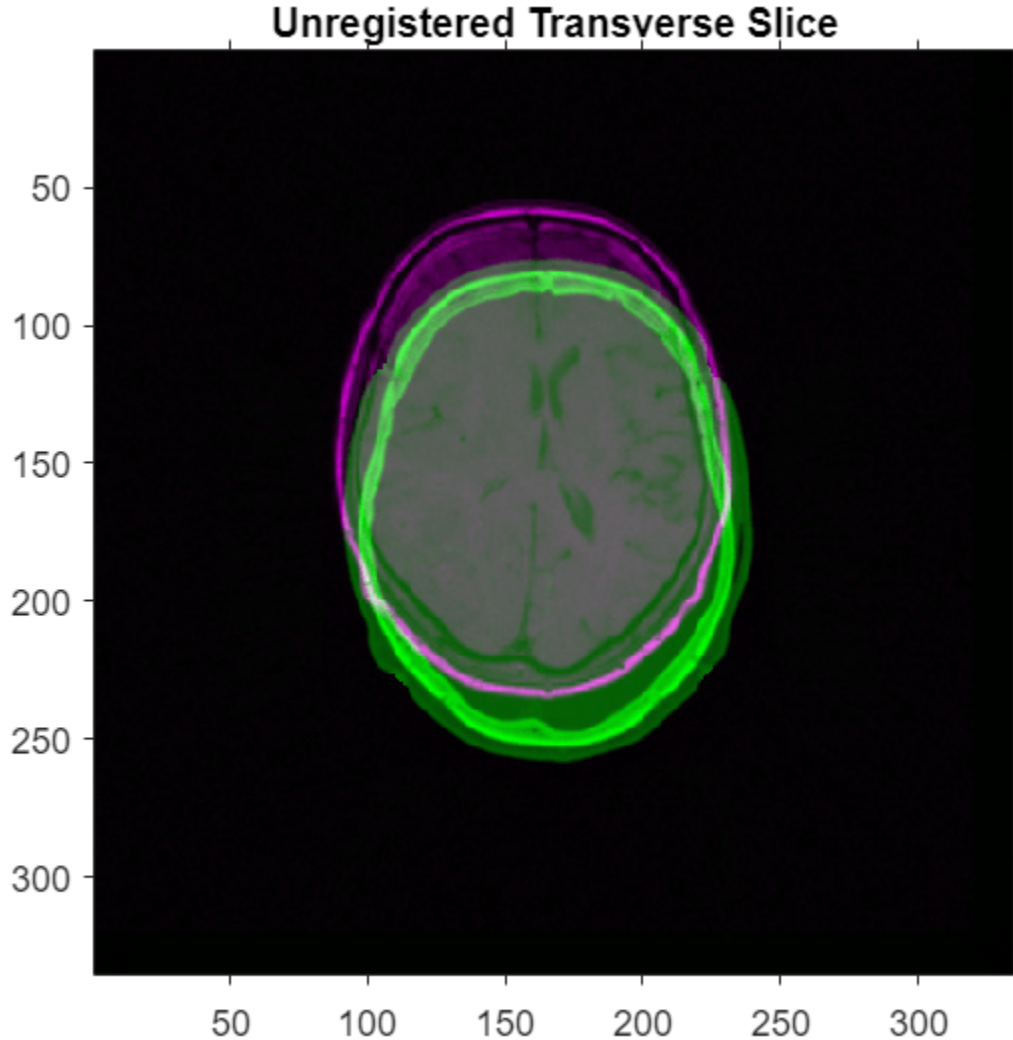
Define spatial referencing for the 2-D slices using `imref2d`. Specify the size, in voxels, and voxel spacing, in millimeters, of the transverse slices.

```
fixedVoxelSpacing = fixedMRIVolume.VoxelSpacing;  
movingVoxelSpacing = movingCTVolume.VoxelSpacing;
```

```
Rfixed2D = imref2d(fixedVolumeSize(1:2),fixedVoxelSpacing(2),fixedVoxelSpacing(1));  
Rmoving2D = imref2d(movingVolumeSize(1:2),movingVoxelSpacing(2),movingVoxelSpacing(1));
```

Display the image slices in the patient coordinate system using `imshowpair`. Plot the slice in the third spatial dimension, which corresponds to the transverse anatomical plane. The MRI slice is magenta, and the CT slice is green.

```
figure  
imshowpair(movingCTVoxels(:,:,centerMoving(3)), ...  
           Rmoving2D, ...  
           fixedMRIVoxels(:,:,centerFixed(3)), ...  
           Rfixed2D)  
title("Unregistered Transverse Slice")
```

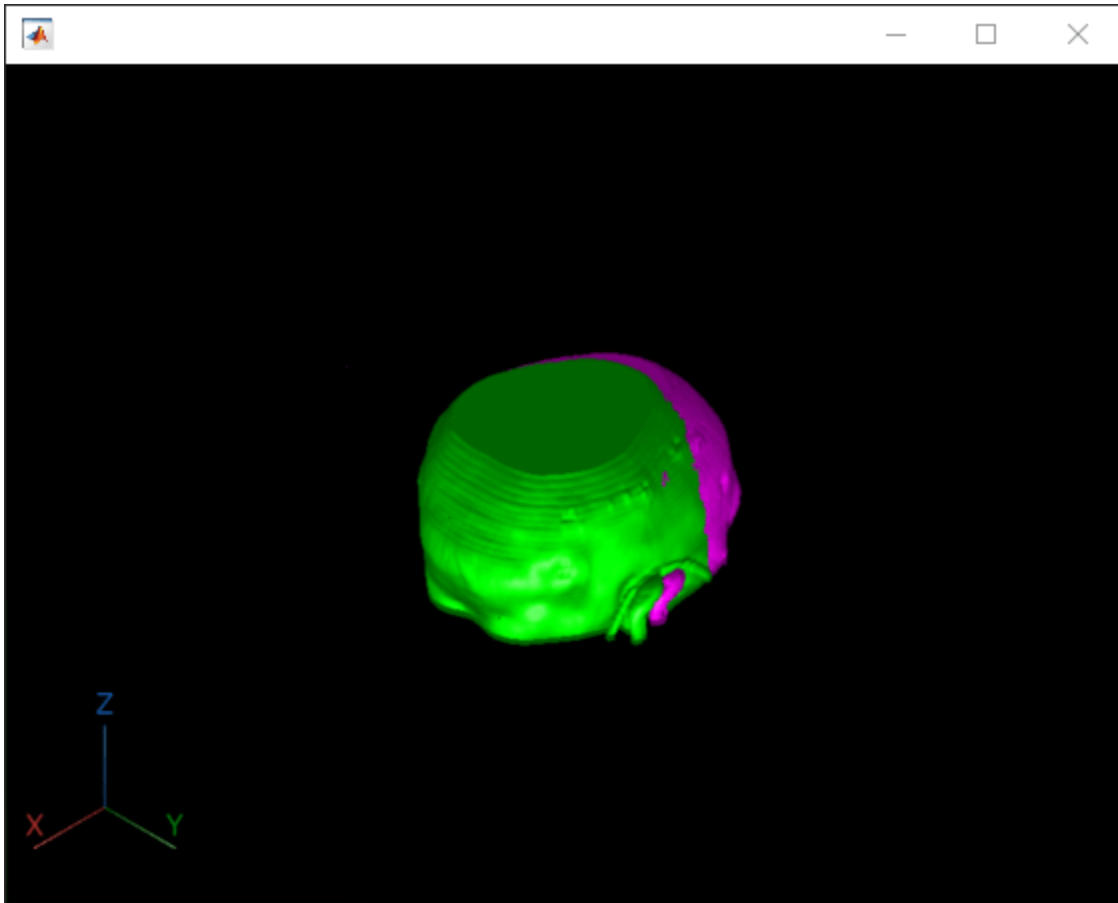


You can also display the volumes as 3-D objects. Create a `viewer3d` object, in which you can display multiple volumes.

```
viewerUnregistered = viewer3d(BackgroundColor="black",BackgroundGradient="off");
```

Display the `medicalVolume` objects as 3-D isosurfaces using `volshow`. The `volshow` function uses `medicalVolume` properties to display each volume in its respective patient coordinate system.

```
volshow(fixedMRIVolume,Parent=viewerUnregistered,RenderingStyle="Isosurface",IsosurfaceValue=0.05,
        Colormap=[1 0 1],Alphamap=1);
volshow(movingCTVolume,Parent=viewerUnregistered,RenderingStyle="Isosurface",IsosurfaceValue=0.05,
        Colormap=[0 1 0],Alphamap=1);
```



Register Images

To accurately register volumes with different voxel dimensions, specify the spatial referencing for each volume using `imref3d` objects. Create each `imref3d` object using the volume size in voxels and the voxel dimensions in millimeters.

```
Rfixed3d = imref3d(fixedVolumeSize, fixedVoxelSpacing(2), ...
    fixedVoxelSpacing(1), fixedVoxelSpacing(3));
Rmoving3d = imref3d(movingVolumeSize, movingVoxelSpacing(2), ...
    movingVoxelSpacing(1), movingVoxelSpacing(3));
```

The `imregmoment` function sets fill pixels added to the registered volume to 0. To improve the display of registration results, scale the CT intensities to the range `[0, 1]`, so that the fill value is equal to the minimum of the image data range.

```
rescaledMovingCTVoxels = rescale(movingCTVoxels);
```

Register the volumes using `imregmoment`, including the `imref3d` objects as inputs. Specify the `MedianThresholdBitmap` name-value argument as `true`, which is appropriate for multimodal images.

```
[geomtform, movingCTRegisteredVoxels] = imregmoment(rescaledMovingCTVoxels, Rmoving3d, ...
    fixedMRIVoxels, Rfixed3d, MedianThresholdBitmap=true);
```

The `geomtform` output is an `affinetform3d` geometric transformation object. The `T` property of `geomtform` contains the 3-D affine transformation matrix that maps the moving CT volume in its patient coordinate system to the fixed MRI volume in its patient coordinate system.

```
geomtform.T
```

```
ans = 4x4
```

```

    1.0000    -0.0039    0.0013    0
    0.0039    1.0000    0.0063    0
   -0.0014   -0.0063    1.0000    0
   -4.8094  -16.0063   -1.3481    1.0000

```

The `movingRegisteredVoxels` output contains the registered CT volume. The `imregmont` function resamples the aligned CT volume in the MRI intrinsic coordinates, so the registered volumes have the same number of rows, columns, and slices.

```
whos movingCTRegisteredVoxels fixedMRIVoxels
```

Name	Size	Bytes	Class	Attributes
fixedMRIVoxels	256x256x26	6815744	single	
movingCTRegisteredVoxels	256x256x26	6815744	single	

Create medicalVolume Object for Registered Image

Create a new `medicalVolume` object that contains the registered voxel data and its spatial referencing information. You can create a `medicalVolume` object by specifying a voxel array and a `medicalref3d` object. The registered CT volume has the same spatial referencing as the original MRI volume, so use the `medicalref3d` object stored in the `VolumeGeometry` property of `fixedMRIVolume`.

```
R = fixedMRIVolume.VolumeGeometry;
movingRegisteredVolume = medicalVolume(movingCTRegisteredVoxels,R)
```

```
movingRegisteredVolume =
```

```
    medicalVolume with properties:
```

```

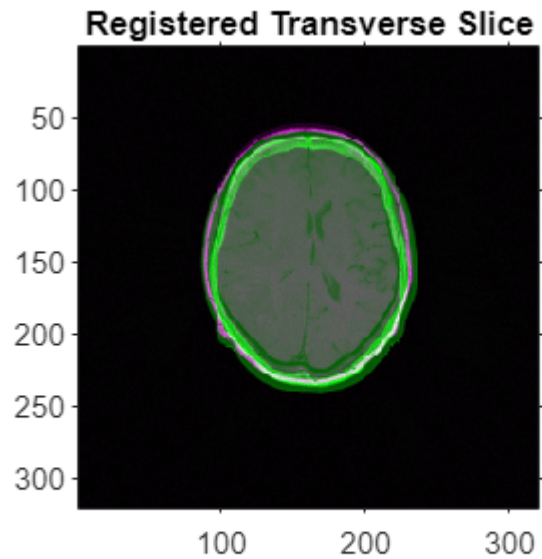
        Voxels: [256x256x26 single]
    VolumeGeometry: [1x1 medicalref3d]
      SpatialUnits: "unknown"
      Orientation: "unknown"
     VoxelSpacing: [1.2500 1.2500 4]
    NormalVector: [0 0 1]
    NumCoronalSlices: []
    NumSagittalSlices: []
    NumTransverseSlices: []
      PlaneMapping: ["unknown" "unknown" "unknown"]
        Modality: "unknown"
    WindowCenters: []
    WindowWidths: []

```

Display Registered Images

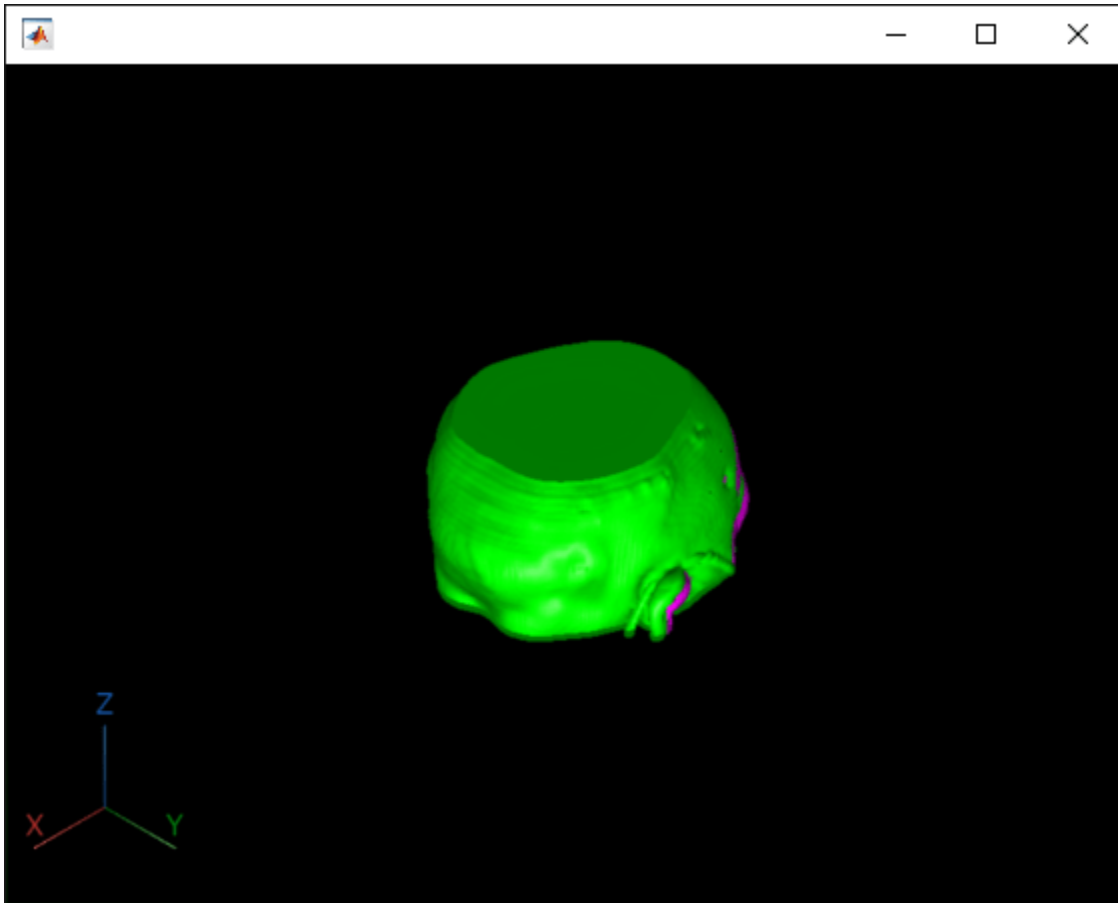
To check the registration results, use `imshowpair` to view the middle transverse slices of the fixed and registered volumes. Use the spatial referencing object for the fixed volume.

```
imshowpair(movingRegisteredVolume.Voxels(:,:,centerFixed(3)), ...
    Rfixed2D, fixedMRIVoxels(:,:,centerFixed(3)), Rfixed2D)
title("Registered Transverse Slice")
```



Display the registered volumes as 3-D isosurfaces using `volshow`. You can zoom and rotate the display to assess the registration results.

```
viewerRegistered = viewer3d(BackgroundColor="black",BackgroundGradient="off");
volshow(fixedMRIVolume,Parent=viewerRegistered,RenderingStyle="Isosurface",IsosurfaceValue=0.05,
    Colormap=[1 0 1],Alphamap=1);
volshow(movingRegisteredVolume,Parent=viewerRegistered,RenderingStyle="Isosurface",IsosurfaceValue=0.05,
    Colormap=[0 1 0],Alphamap=1);
```



See Also

`imregmoment` | `medicalref3d` | `medicalVolume`

Related Examples

- “Medical Image Registration” on page 4-5

Medical Image Labeling

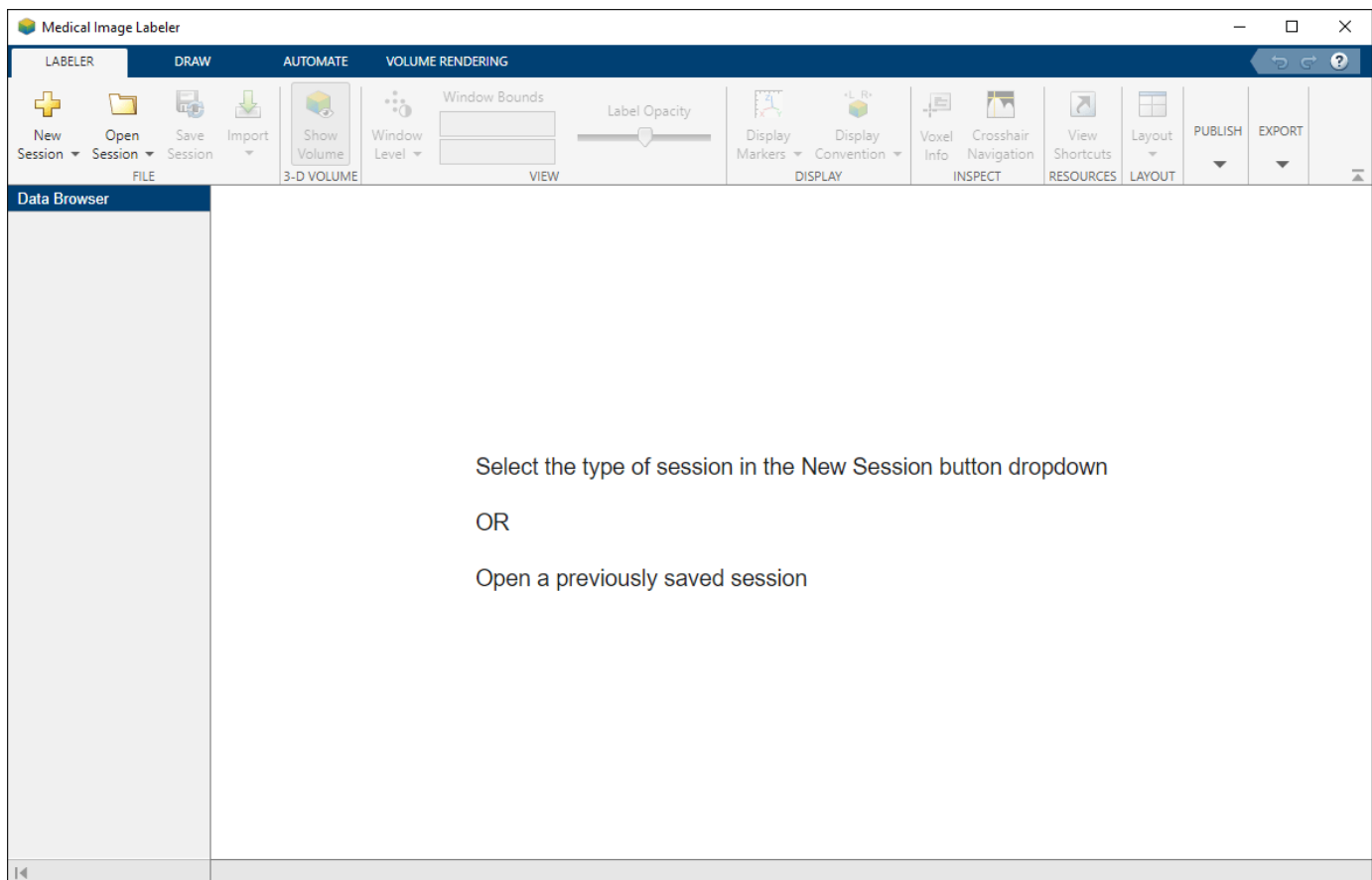
- “Label 2-D Ultrasound Series Using Medical Image Labeler” on page 5-2
- “Label 3-D Medical Image Using Medical Image Labeler” on page 5-10
- “Automate Labeling in Medical Image Labeler” on page 5-20
- “Collaborate on Multi-Labeler Medical Image Labeling Projects” on page 5-28

Label 2-D Ultrasound Series Using Medical Image Labeler

This example shows how to label 2-D image data using the **Medical Image Labeler** app. The **Medical Image Labeler** app provides manual, semi-automated, and automated tools for labeling 2-D medical image data. This example labels the left ventricle in an echocardiogram ultrasound image series.

Open the Medical Image Labeler

Open the **Medical Image Labeler** app from the **Apps** tab on the MATLAB® toolstrip, under **Image Processing and Computer Vision**. You can also load the app by using the `medicalImageLabeler` command.




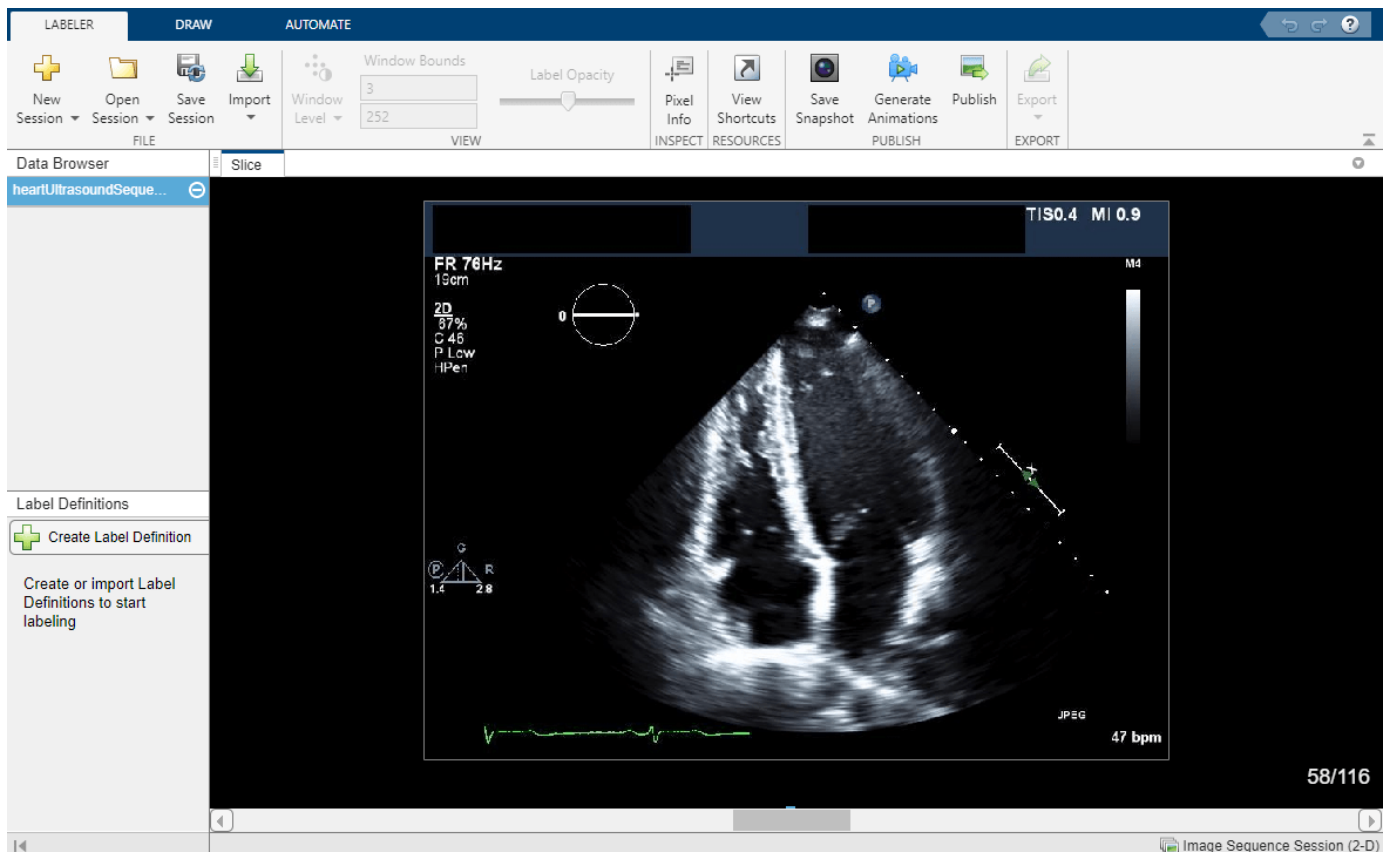
Create New Image Labeling Session

To start a new 2-D labeling session, on the app toolstrip, click **New Session** and select **New Image session (2-D)**. In the Create a new session folder dialog box, specify a location in which to save the new session folder by entering a path or selecting **Browse** and navigating to your desired location. In the New Session Folder dialog box, specify a name for the folder for this labeling session. Then, select **Create Session**.

Load Image Data into Medical Image Labeler

To load an image into the **Medical Image Labeler** app, on the app toolbar, click **Import Data**. Then, under **Data**, select **From File**. Browse to the location of `heartUltrasoundSequenceVideo.dcm` in the same directory as this example file. For an image session, the imported data file must be a single DICOM or NIfTI file containing a 2-D image or a series of 2-D images related by time.

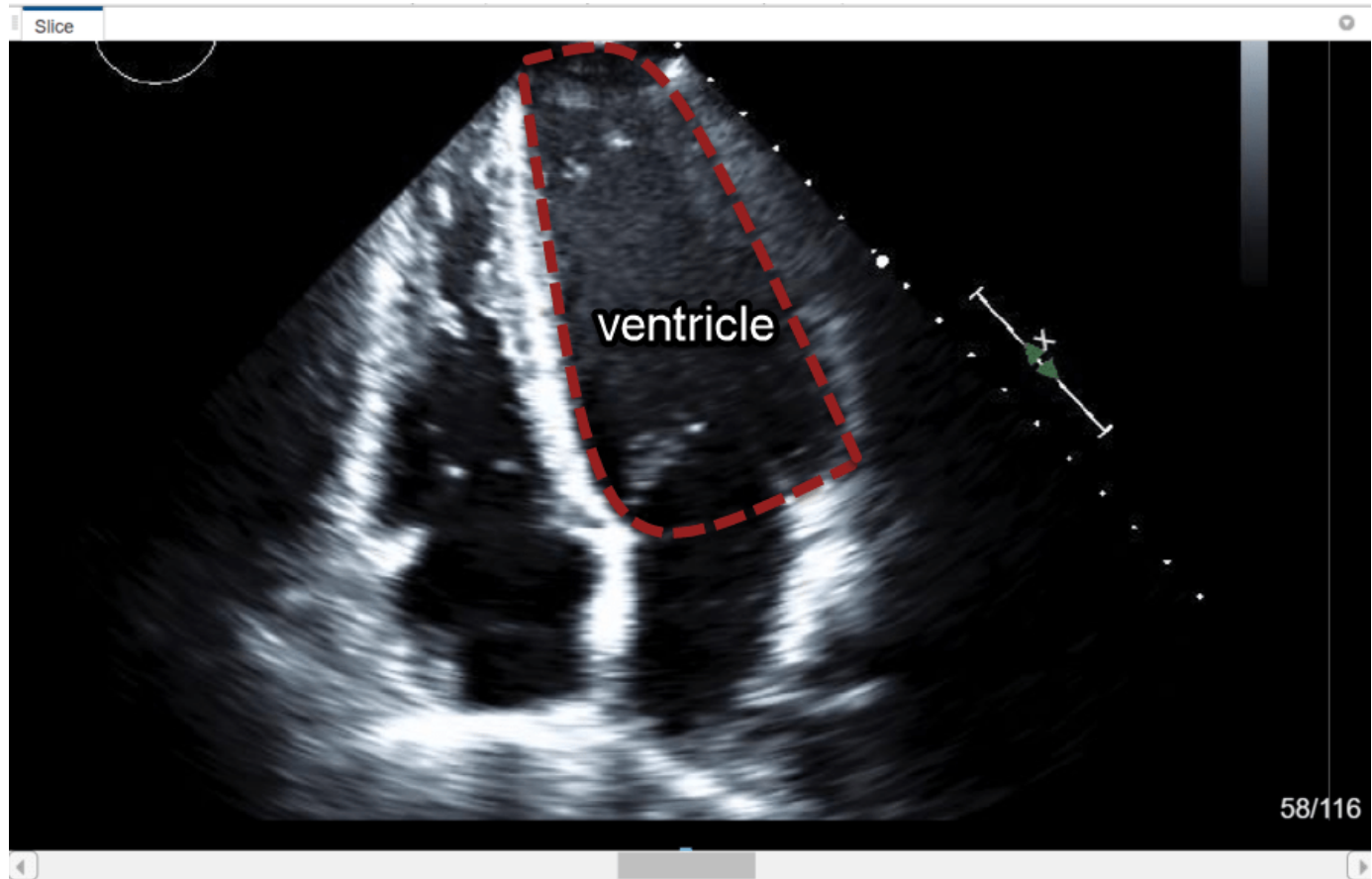
The name of the imported image is visible in the **Data Browser** pane. The no labels symbol  next to the file name indicates that the image file does not contain any pixel labels. You can import multiple 2-D image files into an image session. All files imported into a single app session must label the same regions of interest, such as tumor or ventricle, and are exported together as one `groundTruthMedical` object.



Explore the Image Series

The **Medical Image Labeler** app displays the imported image in the **Slice** pane. By default, the app displays the middle frame in the ultrasound series. You can change the displayed frame by using the scroll bar at the bottom of the **Slice** pane, or you can click the pane and then press the left and right arrow keys. The app displays the current frame number out of the total number of frames, such as 58/116. You can zoom in on the current frame using the mouse scroll wheel or the zoom controls that appear when you pause on the **Slice** pane.

Explore the echocardiogram image series to identify the left ventricle. The annotated image shows the approximate outline of the ventricle to label. Note that all labels in this example have been created for illustrative purposes only and have not been verified by a clinical professional.



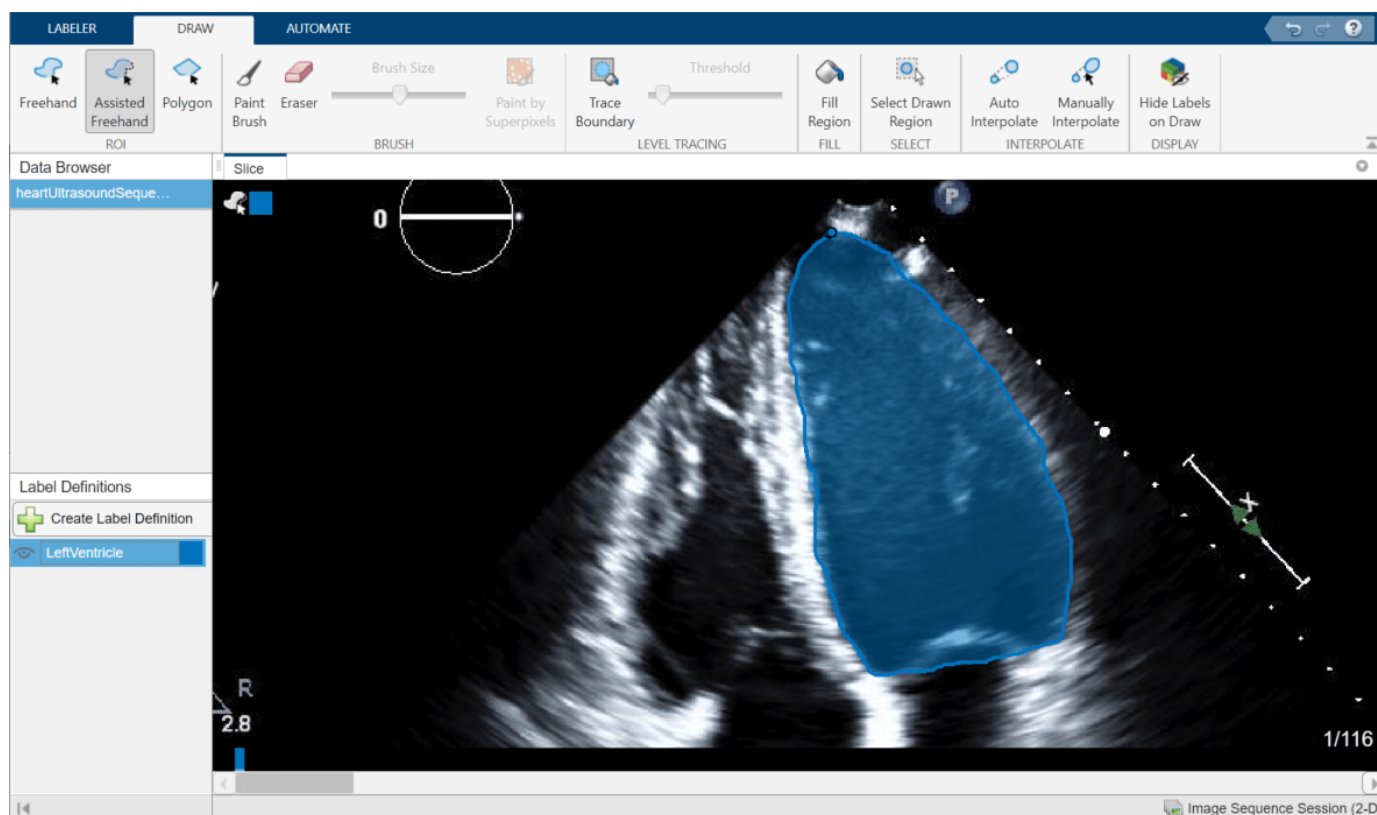
Create Label Definitions

A *label definition* specifies the name, color, and numeric index of a label. In the **Label Definitions** pane, select **Create Label Definition** to create a label with the default name `Label1`. To change the name of the label, click the label and enter a new name. The label name must be a valid MATLAB variable name with no spaces. For this example, specify the name of the label as `LeftVentricle`. To change the default color associated with the label, click the colored square in the label identifier and select a color from the Color dialog box.



Use Drawing Tools to Label Regions in Image

To assign pixels to the `LeftVentricle` label, click the `LeftVentricle` label in the **Label Definitions** pane. You can use the tools on the **Draw** tab in the app toolstrip to define the region. You can choose from the **Freehand**, **Assisted Freehand**, **Polygon**, **Paint Brush**, and **Trace Boundary** tools. Label frame 1. When you add the label, the app adds a bar above the slider, using the color associated with the label, to indicate which frames you have labeled.

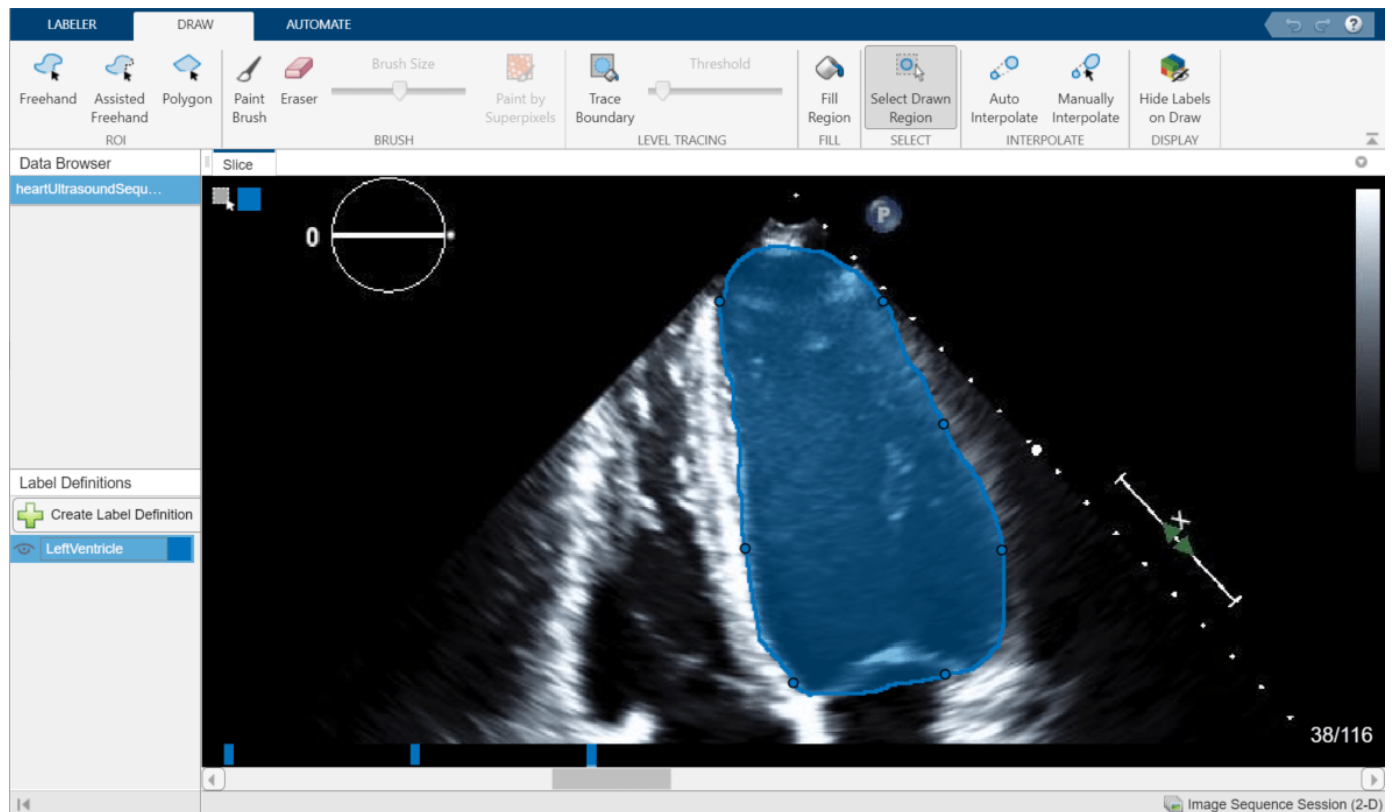


In addition to the drawing tools, you can add or refine label regions using the tools in the **Automate** tab of the app toolstrip. The app provides automation algorithms including **Active Contours**, **Adaptive Threshold**, **Dilate**, and **Erode**. To apply an algorithm, click **Algorithm Parameters** to adjust settings, if applicable, and click **Run**. You can also specify a custom range of frames to process by specifying a **Start** frame and an **End** frame.

Use Interpolation to Speed Up Labeling

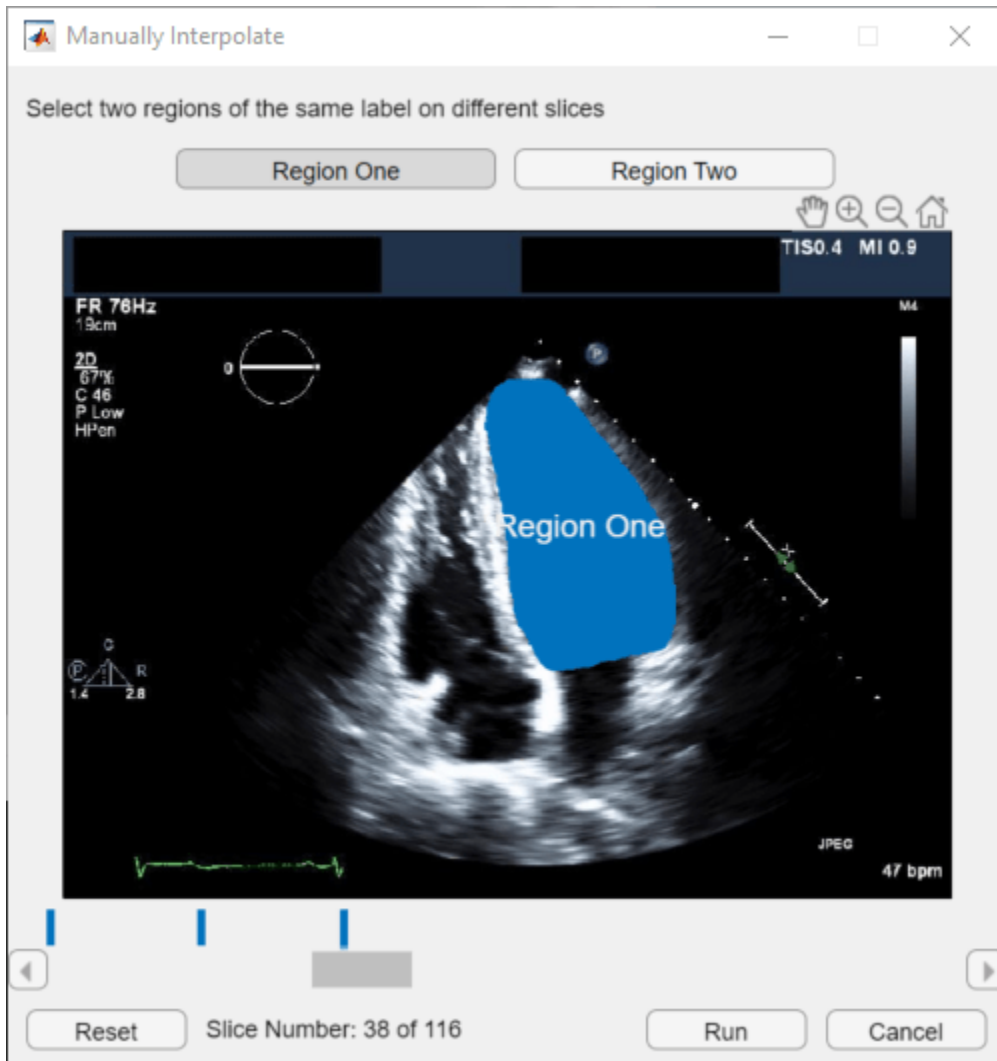
You can move through the image series and draw labels frame-by-frame, but the **Medical Image Labeler** app also provides interpolation tools in the **Draw** tab that can help you label an object between frames. Interpolation is most suitable between frames where the region of interest has a similar shape and size. In an echocardiogram, the ventricle experiences cycles of contraction, during which the ventricle rapidly changes in shape, and relaxation, during which the ventricle is relatively still. Therefore, you can most effectively use interpolation between the start and end of a relaxation period, such as between frames 20 and 38.

To use interpolation, you must first manually label a region in two frames. Label the ventricle in frame 20 and frame 38. If the **Auto Interpolate** button is not active, make sure the labeled region is selected by clicking **Select Drawn Region** in the app toolstrip and selecting the labeled region.



Click **Auto Interpolate**. The app automatically labels the ventricle in the intermediate frames. The app adds bars above the slider to indicate all the frames that have labeled pixels, which now appears as a solid bar from frame 20 to frame 38.

Alternatively, after labeling an ROI on two frames, you can click **Manually Interpolate**. With this option, the app opens the Manually Interpolate dialog box. Select the two regions between which you want to interpolate, **Region One** and **Region Two**. To select the first region, use the slider at the bottom of the dialog box to navigate to frame 20, and then click inside the labeled ventricle. To select the second region, click **Region Two**, navigate to frame 38, and click inside the labeled ventricle. After selecting both regions, click **Run** to interpolate the label in the intermediate frames.



After using interpolation, check the individual frames to see if the interpolation created satisfactory labels. You can manually correct labels using the **Paint Brush** and **Eraser** tools. Alternatively, you can refine the labels using one of the tools in the **Automate** tab. For example, you can use **Active Contours** to grow an ROI in a frame where it does not fill the full area of the ventricle. You can also undo the interpolation and try interpolating across fewer frames to improve results.

Modify Labels

To refine drawn labels, you can remove label data from individual pixels, from individual frames within an image series, from an entire image, or from the whole labeling session.

- Remove labels from individual pixels — In the **Draw** tab, use the **Eraser** tool.
- Remove labels from one connected region in a frame — Click **Select Drawn Region** and, in the **Slice** pane, select the region from which you want to remove labels. Press **Delete**, or right-click and select **Delete**, to remove labels from the region.
- Remove all labels from a frame — Right-click anywhere on the frame and select **Select All**. Press **Delete**, or right-click and select **Delete**, to remove all labels from the frame.

- Remove all labels from an image series — Right-click the file name of the image series in the **Data Browser** and select **Remove Labels**. Removing labels from an image removes all pixel labels for all label definitions within the file.
- Remove a label from all images within the app session — In the **Label Definitions** pane, right-click the label name and select **Delete**. This deletes the label from the `groundTruthMedical` object in the session folder, and removes all pixel labels for the deleted label name in all images in the session.

Apply Custom Automation Algorithm

You can add a custom automation algorithm to use in the app. On the **Automate** tab, click **Add Algorithm**. Import an existing algorithm by selecting **From File**, or create a new algorithm using the provided function or class template. See “Automate Labeling in Medical Image Labeler” on page 5-20 for an example that applies a custom automation algorithm for 2-D labeling.

Under **Slice-Based**, select the **New** option and click **Function Template** to create a new function that operates on each 2-D image frame. The app opens the template in the MATLAB editor. Replace the sample code in the template with the code for your algorithm. Your function must accept two arguments: each frame as a separate image, and a mask. Your function must also return a mask image.

When you are done editing the template, save the file. The **Medical Image Labeler** app automatically creates a button in the **Automate** tab of the toolbar for your function. To test your function, select it and click **Run**. By default, the app applies the function to only the current frame.



After testing your function on a single frame, you can run it on all of the frames, or a subset of the frames. You can run it from the current frame to the end (the highest numbered frame) or from the current frame back to the beginning (frame 1). You can also specify a range of frames by specifying the starting frame and the ending frame.

Export Ground Truth Data

For this example, the labels for `heartUltrasoundSequenceVideo.dcm` are complete when you have labeled the ventricle in each frame. The app automatically assigns a value of 0 to unlabeled pixels in the label images saved to the session folder, so you do not need to label the background manually.

The **Medical Image Labeler** app automatically saves a `groundTruthMedical` object as a MAT file in the session folder. You can also export a `groundTruthMedical` object, saved as a MAT file, to an alternate file location from the app. On the **Labeler** tab, click **Export** and, under **Ground Truth**, select **To File**.

You can load the exported MAT file into the MATLAB workspace using the `load` function. The properties of the `groundTruthMedical` object, `gTruthMed`, contain information about the image data source, label definitions, and location of the saved label images. Display information about the object and each of its properties using these commands.

- `gTruthMed` — Display the properties of the `groundTruthMedical` object.
- `gTruthMed.DataSource` — Location of the source of the unlabeled medical images and image series.
- `gTruthMed.LabelDefinitions` — Table of information about label definitions.
- `gTruthMed.LabelData` — Locations of the saved, labeled images.

See Also

Medical Image Labeler | `groundTruthMedical`

Related Examples

- “Get Started with Medical Image Labeler” on page 1-9
- “Label 3-D Medical Image Using Medical Image Labeler” on page 5-10
- “Automate Labeling in Medical Image Labeler” on page 5-20

Label 3-D Medical Image Using Medical Image Labeler

This example shows how to label 3-D image data using the Medical Image Labeler app. The **Medical Image Labeler** app provides manual, semi-automated, and automated tools for labeling 3-D medical image data. This example segments a chest CT volume to label a lung tumor region.

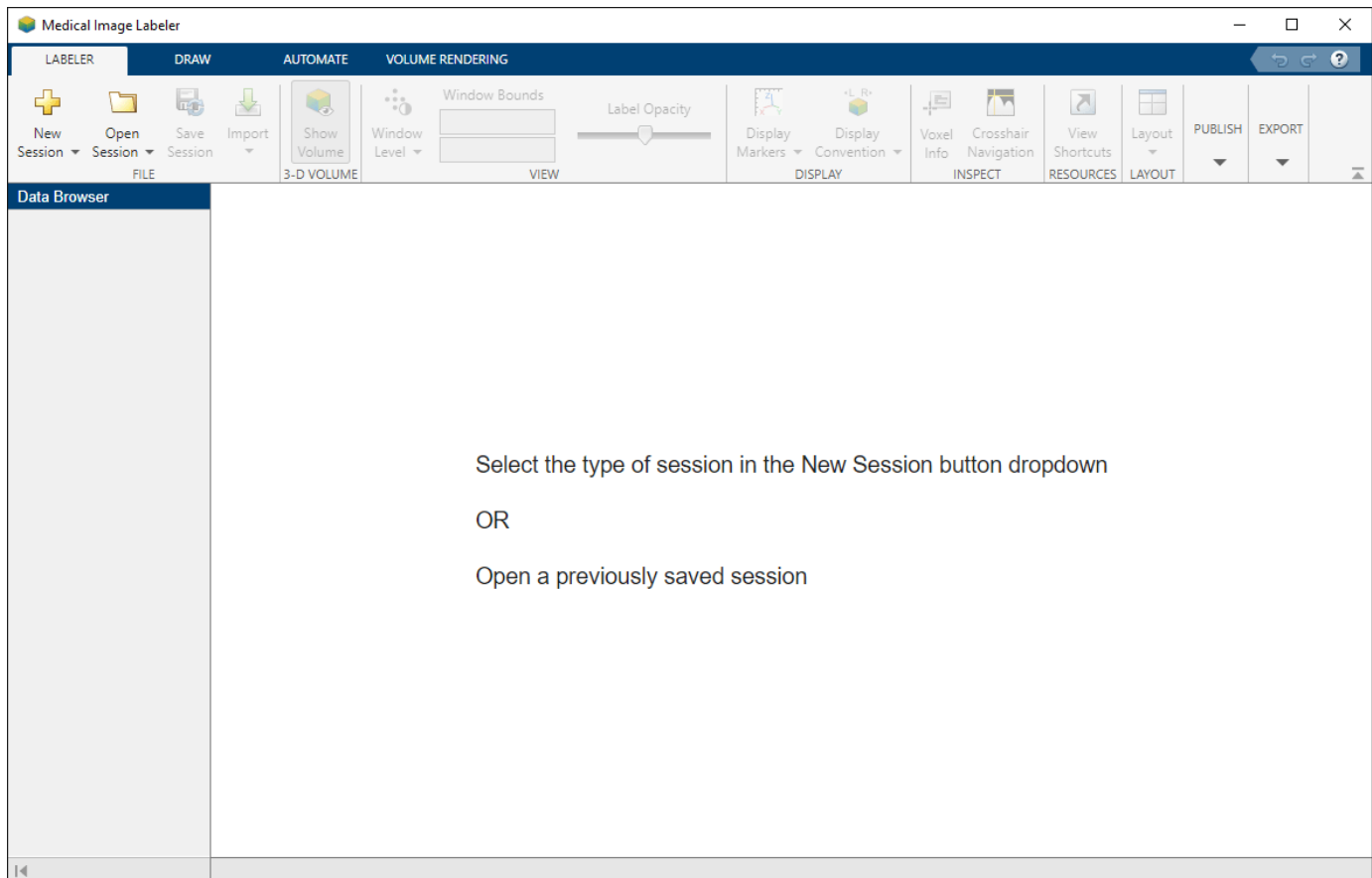
Download Data to Label

This example labels chest CT data from a subset of the Medical Segmentation Decathlon data set [1 on page 5-19]. The size of the subset of data is approximately 76 MB. Download the `MedicalVolumeNIftIData.zip` file from the MathWorks® website, then unzip the file.

```
zipFile = matlab.internal.examples.downloadSupportFile("medical", "MedicalVolumeNIftIData.zip");
filepath = fileparts(zipFile);
unzip(zipFile, filepath)
dataFolder = fullfile(filepath, "MedicalVolumeNIftIData");
```

Open the Medical Image Labeler

Open the **Medical Image Labeler** app from the **Apps** tab on the MATLAB® toolstrip, under **Image Processing and Computer Vision**. You can also load the app by using the `medicalImageLabeler` command.




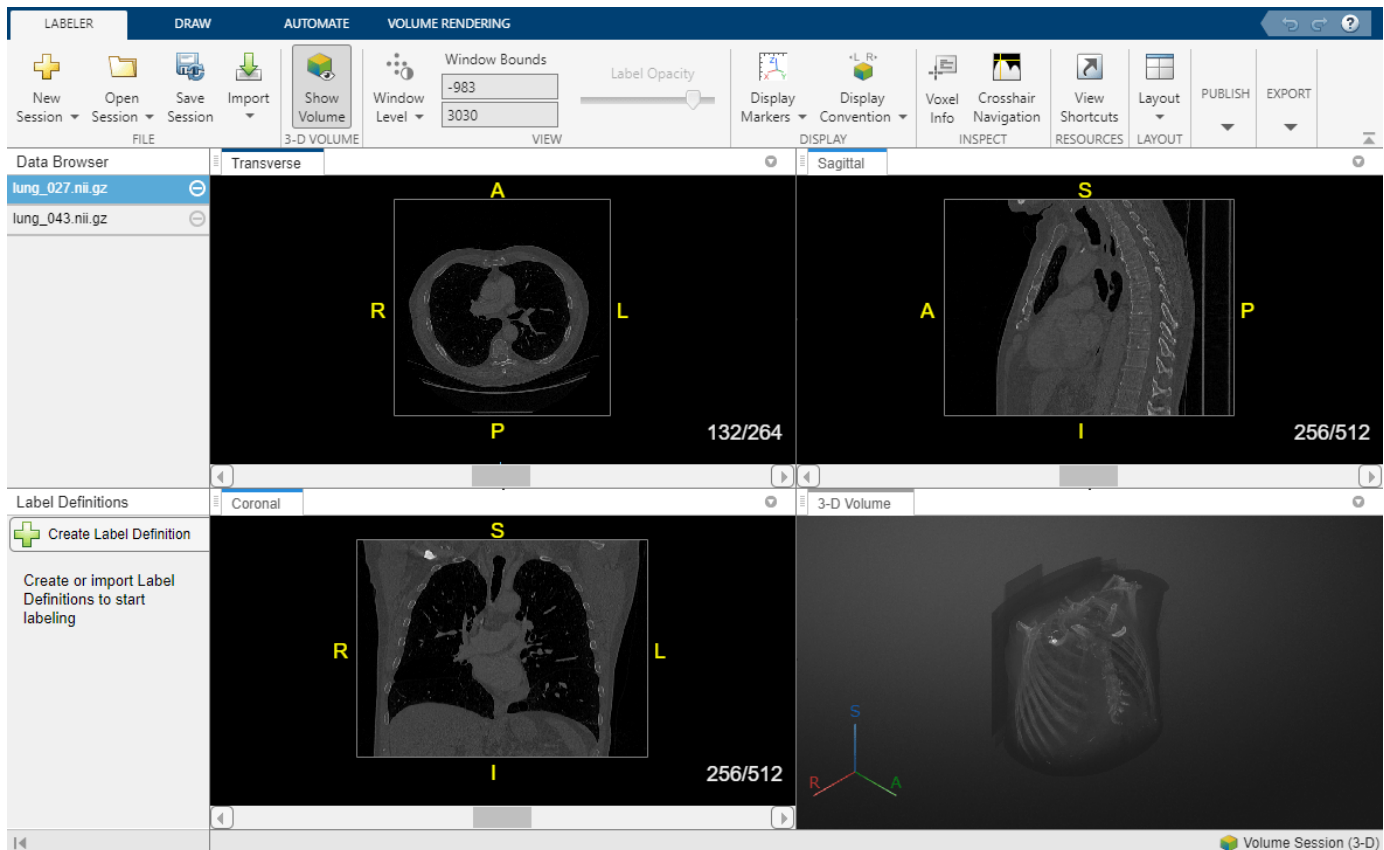
Create New Volume Labeling Session

To start a new 3-D labeling session, on the app toolbar, click **New Session** and select **New Volume session (3-D)**. In the **Create a new session folder** dialog box, specify a location in which to save the new session folder by entering a path or selecting **Browse** and navigating to your desired location. In the **New Session Folder** dialog box, specify a name for the folder for this labeling session. Then, select **Create Session**.

Load Image Data into Medical Image Labeler

To load an image into the **Medical Image Labeler** app, on the app toolbar, click **Import Data**. Then, under **Data**, select **From File**. Browse to the location where you downloaded the data, specified by the `dataFolder` workspace variable, and select the file `lung_027.nii.gz`. For a **Volume Session**, the imported data file can be a single DICOM or NIfTI file containing a 3-D image volume, or a directory containing multiple DICOM files corresponding to a single image volume.

The name of the imported image is visible in the **Data Browser** pane. The no labels symbol  next to the file name indicates that the volume does not contain any pixel labels. You can import multiple 3-D image files into a **Volume Session**. All files imported into a single app session must label the same regions of interest, such as tumor or lung, and are exported together as one `groundTruthMedical` object. Import the other file in the download location, `lung_043.nii.gz`.




Explore the Image Volume

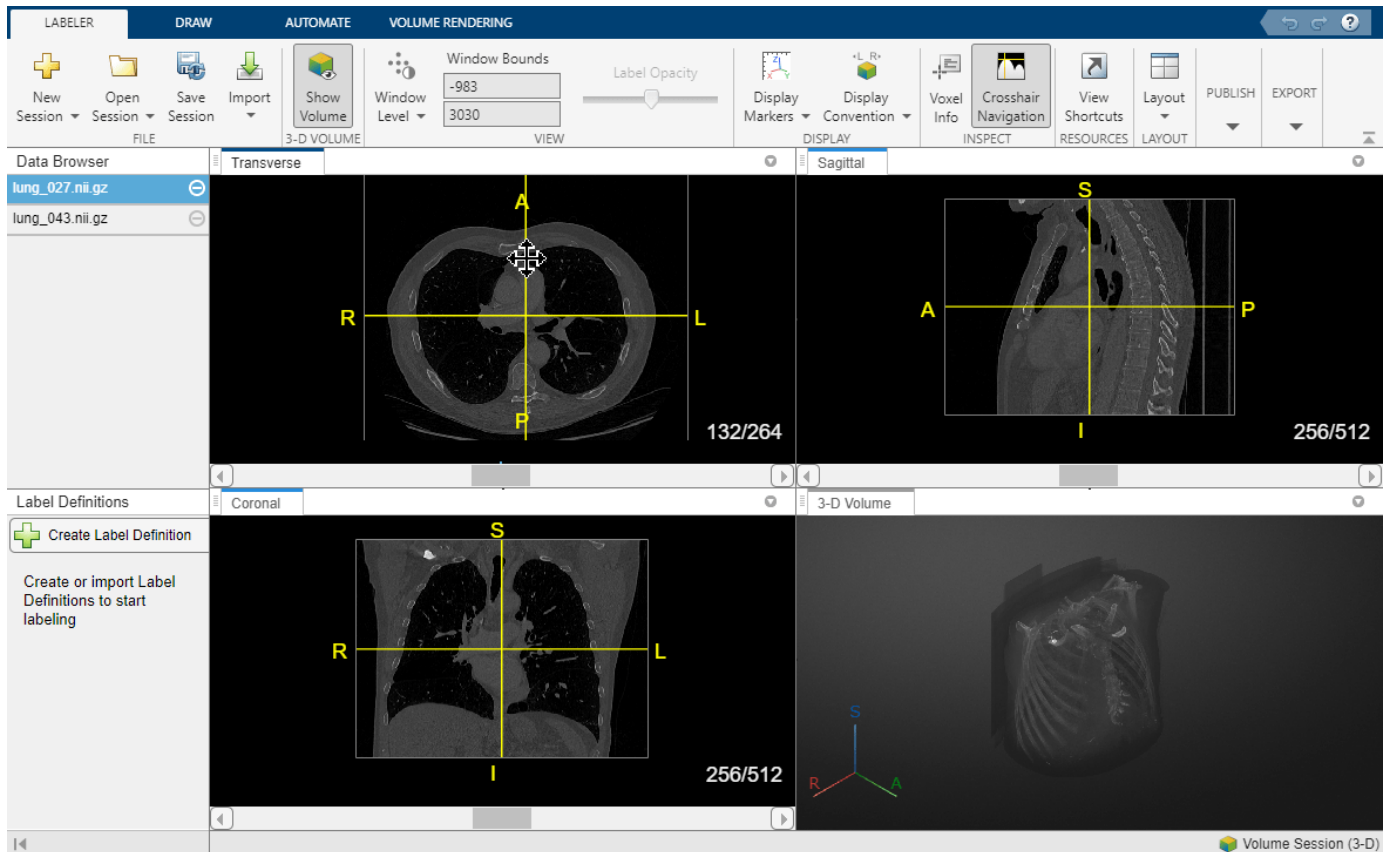
The **Medical Image Labeler** app displays a 3-D rendering of the scan in the **3-D Volume** pane, and displays anatomical slice planes in the **Transverse**, **Sagittal**, and **Coronal** panes. You can toggle the

visibility of the volume display using the **Show Volume** button on the **Labeler** tab of the app toolbar.

By default, the app displays the center slice in each slice plane. You can change which slice is displayed by using the scroll bar at the bottom of a slice pane, or you can click the pane and then press the left and right arrow keys. The app displays the current slice number out of the total number of slices, such as 132/264, for each slice pane. The app also displays anatomical display markers indicating the anterior (A), posterior (P), left (L), right (R), superior (S), and inferior (I) directions. You can zoom in on the current slice pane using the mouse scroll wheel or the zoom controls that appear when you pause on the slice pane.


You can also navigate between slices in the three planes using crosshair navigation. To turn on the crosshair indicators, in the **Labeler** tab of the app toolbar, select **Crosshair Navigation**. The indicators show the relative positions of the slices in the other 2-D views. To navigate slices, pause on


one of the crosshair axes until the cursor changes to the fleur shape, , and then click and drag to a new position. The other slice views update automatically. To hide the crosshairs, in the app toolbar, clear **Crosshair Navigation**.



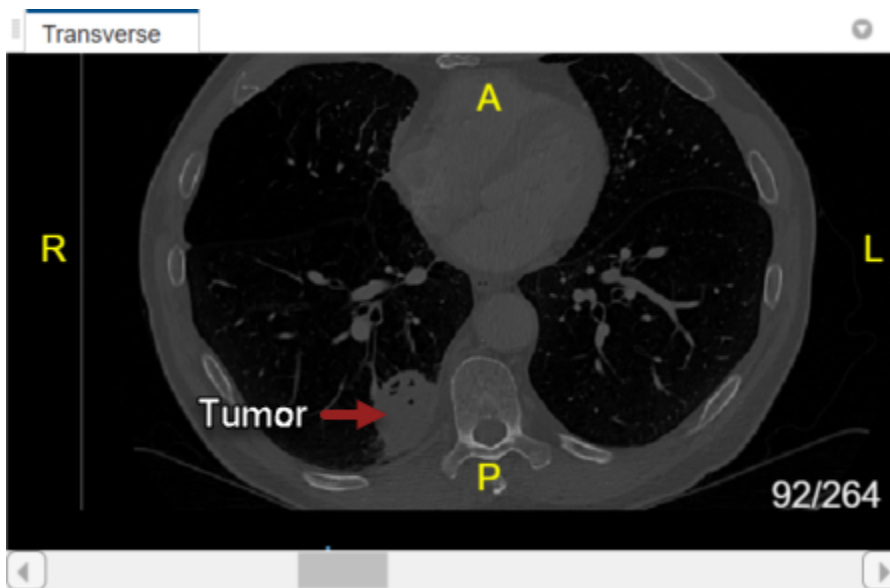
By default, the app displays the scan using the **Radiological** display convention, with the left side of the patient on the right side of the image. To display the left side of the patient on the left side of the image, on the app toolbar, click **Display Convention** and select **Neurological**.

You can adjust the brightness and contrast used to display grayscale image data by using the

Window Level tool in the **Labeler** tab of the app toolbar. First, on the app toolbar, select . Then, click and hold in any of the slice panes, and drag up and down to increase and decrease the brightness, respectively, or left and right to increase and decrease the contrast. The updated window

bounds are displayed in the app toolbar under **Window Bounds**. Click  to deactivate the tool. Changing the display window does not modify the image data.

With `lung_027.nii.gz` selected in the **Data Browser** pane, explore the chest volume to identify the tumor region you want to label. The tumor is visible between slices 79 and 105 in the **Transverse** slice pane.



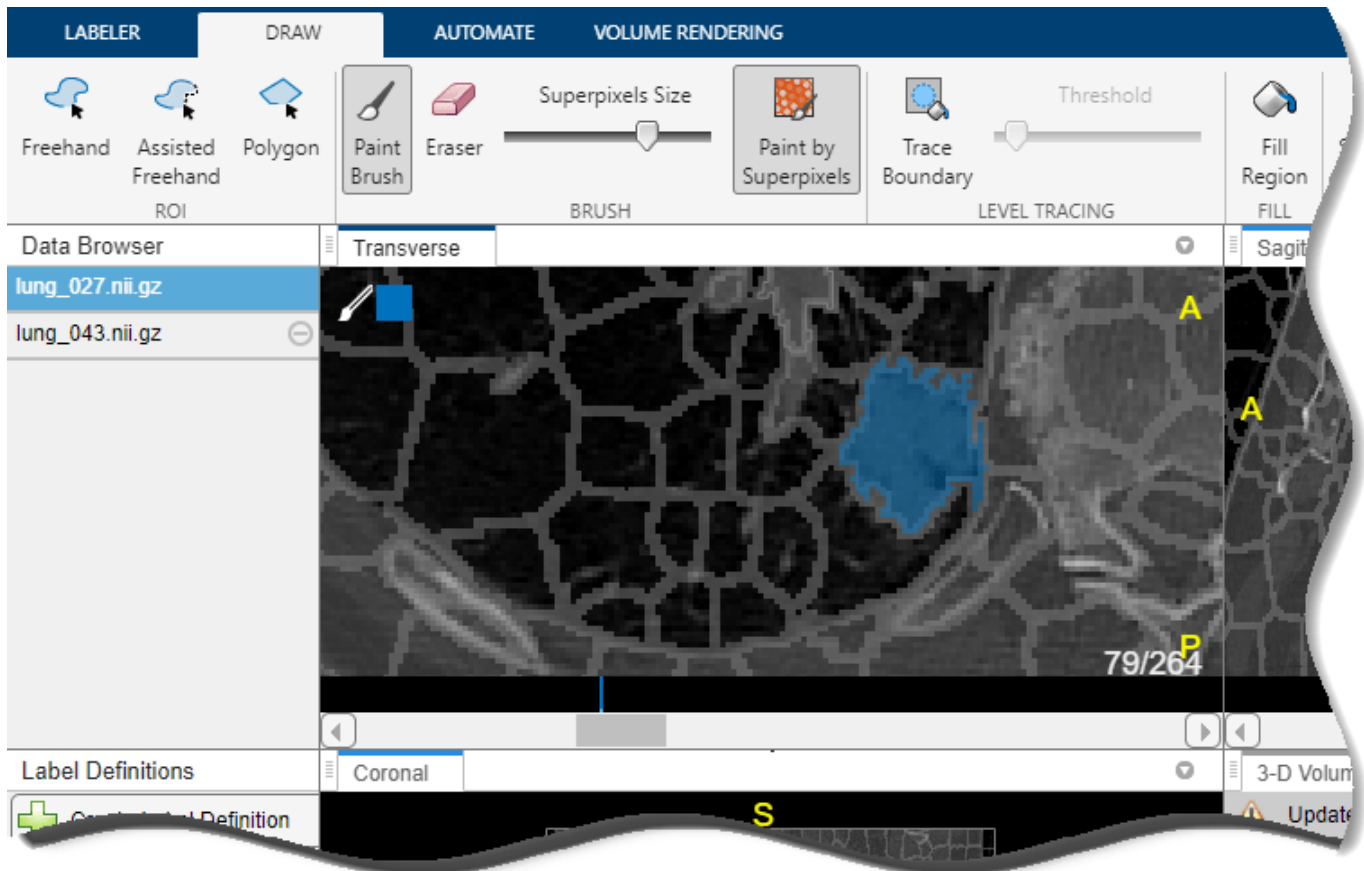
Create Label Definitions

A *label definition* specifies the name, color, and numeric index of a label. In the **Label Definitions** pane, select **Create Label Definition** to create a label with the default name `Label1`. To change the name of the label, double-click on the label and type in a new name. The label name must be a valid MATLAB® variable name with no spaces. For this example, specify the name of the label as `tumor`. To change the default color associated with the label, double-click on the colored square in the label identifier and select a color from the Color dialog box.



Use Drawing Tools to Label Regions in Image

Click on the tumor label in the **Label Definitions** pane to assign pixels to the tumor label. You can use the tools on the **Draw** tab in the app toolbar to define the region. You can choose from the **Freehand**, **Assisted Freehand**, **Polygon**, **Paint Brush**, and **Trace Boundary** tools. This example uses the **Paint Brush** tool, with **Paint by Superpixels** enabled, to label the tumor, but you can use any of the drawing tools. You can draw labels in the **Transverse**, **Sagittal**, or **Coronal** slice panes. Label slice 79. When you add the label, the app adds a bar above the slider, using the color associated with the label, to indicate which slices you have labeled.

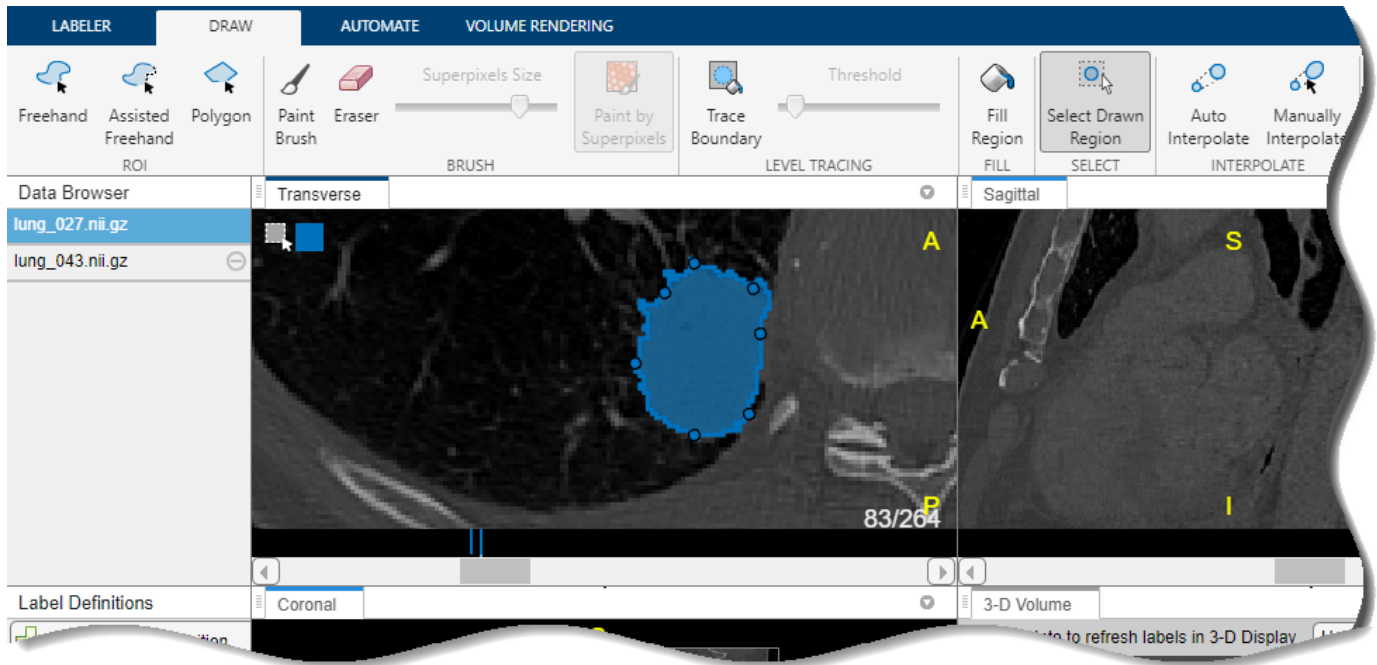


In addition to the drawing tools, you can add or refine label regions using the tools in the **Automate** tab of the app toolbar. The app provides slice-based automation algorithms including **Active Contours**, **Adaptive Threshold**, **Dilate**, and **Erode**, as well as volume-based algorithms including **Filter and Threshold**, **Smooth Edges**, and **Otsu's Threshold**. To apply a slice-based algorithm, click **Algorithm Parameters** to adjust settings if applicable, select an option from the **Slice Direction** list, and click **Run**. You can also specify a custom range of slices to process by specifying a **Start** slice and an **End** slice.

Use Interpolation to Speed Up Labeling

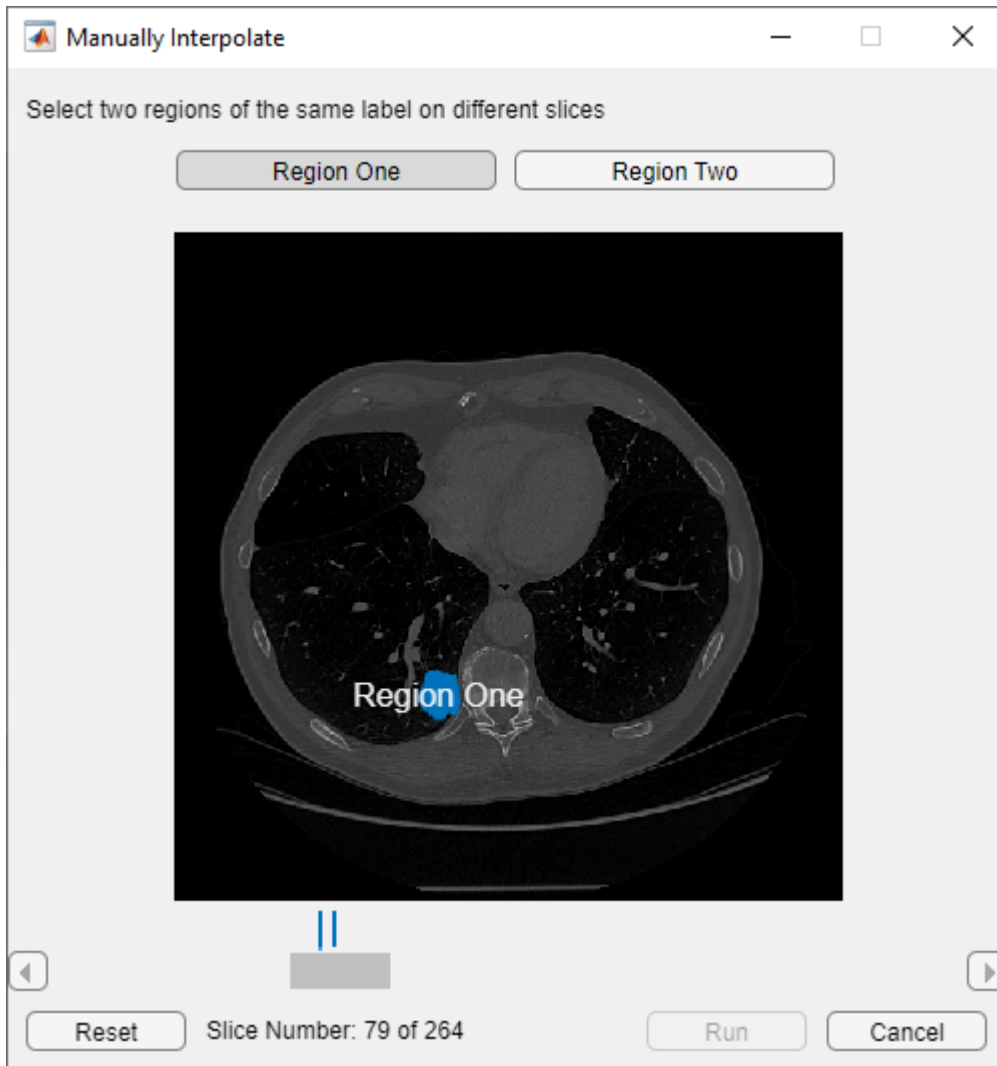
You can move through the image volume and draw labels slice-by-slice. The **Medical Image Labeler** app also provides interpolation tools in the **Draw** tab that can help with labeling an object between slices.

To use interpolation, you must first manually label a region on two slices. You have already labeled the tumor in slice 79. Use the same process to label the tumor in slice 83. If the **Auto Interpolate** button is not active, make sure the labeled region is selected by clicking **Select Drawn Region** in the app toolbar and selecting the labeled region.




Click **Auto Interpolate**. The app automatically labels the tumor in the intermediate slices. The app adds bars above the slider to indicate all the slices that have labeled pixels, which now appears as a solid bar from slice 79 to slice 83.

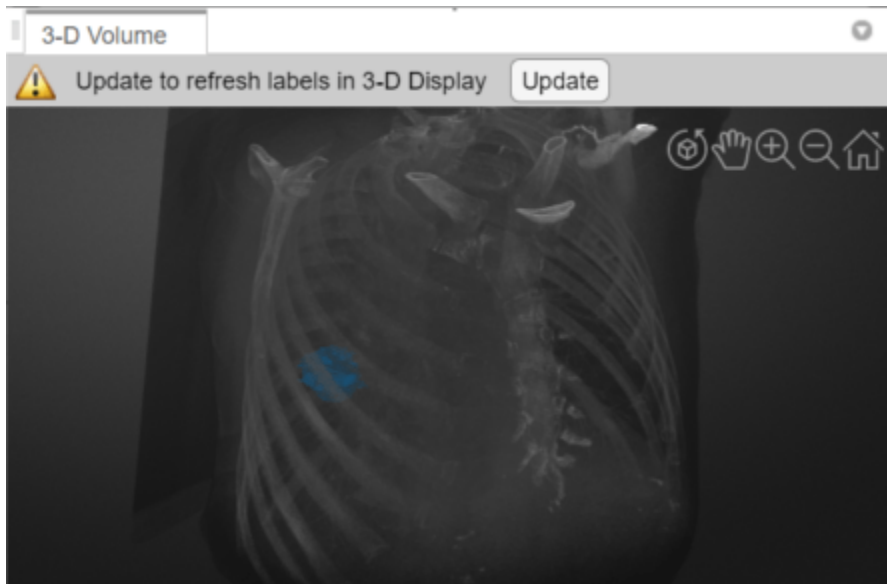
Alternatively, after labeling an ROI on two slices, you can click **Manually Interpolate**. With this option, the app opens the Manually Interpolate dialog box. Select the two regions between which you want to interpolate, **Region One** and **Region Two**. To select the first region, use the slider at the bottom of the dialog box to navigate to slice 79, and then click inside the labeled tumor. To select the second region, click **Region Two**, navigate to slice 83, and click inside the labeled tumor. After selecting both regions, click **Run** to interpolate the label in the intermediate slices.



After using interpolation, check the individual slices to see if the interpolation created satisfactory labels. You can manually correct labels using the **Paint Brush** and **Eraser** tools. Alternatively, you can refine the labels using one of the tools in the **Automate** tab. For example, you can use **Active Contours** to grow an ROI in a slice where it does not fill the full area of the tumor. You can also undo the interpolation and try interpolating across fewer slices to improve results.

Visualize and Modify Labels

As you draw labels, you can view them as an overlay in the **3-D Volume** pane. In the warning message at the top of the **3-D Volume** pane, click **Update**. Note that **Update** is only available after you update the label data in a slice pane. To adjust the opacity of all labels in the session, in the **Labeler** tab of the app toolstrip, adjust the **Label Opacity** slider. To toggle the visibility of an individual label, in the **Label Definitions** pane, click the eye symbol  next to the label name.



To refine drawn labels, you can remove label data from individual pixels, from individual slices within an image, from an entire image, or from the whole labeling session.

- Remove labels from individual pixels — In the **Draw** tab, use the **Eraser** tool.
- Remove labels from one connected region in a slice — Click **Select Drawn Region** and, in the slice pane, select the region from which you want to remove labels. Press **Delete**, or right-click and select **Delete**, to remove labels from the region.
- Remove all labels from a slice — Right-click anywhere on the slice and select **Select All**. Press **Delete**, or right-click and select **Delete**, to remove all labels from the slice.
- Remove all labels from an image volume — Right-click the file name of the image volume in the **Data Browser** and select **Remove Labels**. Removing labels from an image volume removes all pixel labels for all label definitions within the file.
- Remove a label from all images within the app session — In the **Label Definitions** pane, right-click the label name and select **Delete**. This deletes the label from the `groundTruthMedical` object in the session folder, and removes all pixel labels for the deleted label name in all images in the session.

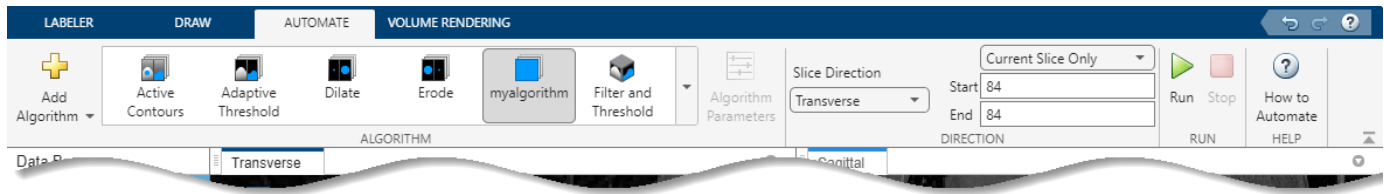
Apply Custom Automation Algorithm

You can add a custom automation algorithm to use in the app. On the **Automate** tab, click **Add Algorithm**. Import an existing algorithm by selecting **From File**, or create a new algorithm using the provided function or class template. See “Automate Labeling in Medical Image Labeler” on page 5-20 for an example that applies a custom automation algorithm for 2-D labeling.

For this example, under **Slice-Based**, select the **New** option and click **Function Template** to create a new function that operates on each 2-D image slice. The app opens the template in the MATLAB editor. Replace the sample code in the template with code that you want to use. Your function must accept two arguments: each slice as a separate image, and a mask. Your function must also return a mask image.

When you are done editing the template, save the file. The **Medical Image Labeler** app automatically creates a button in the **Automate** tab toolbar for your function. To test your function,

select an option from the **Slice Direction** list and click **Run**. By default, the app applies the function to only the current slice.

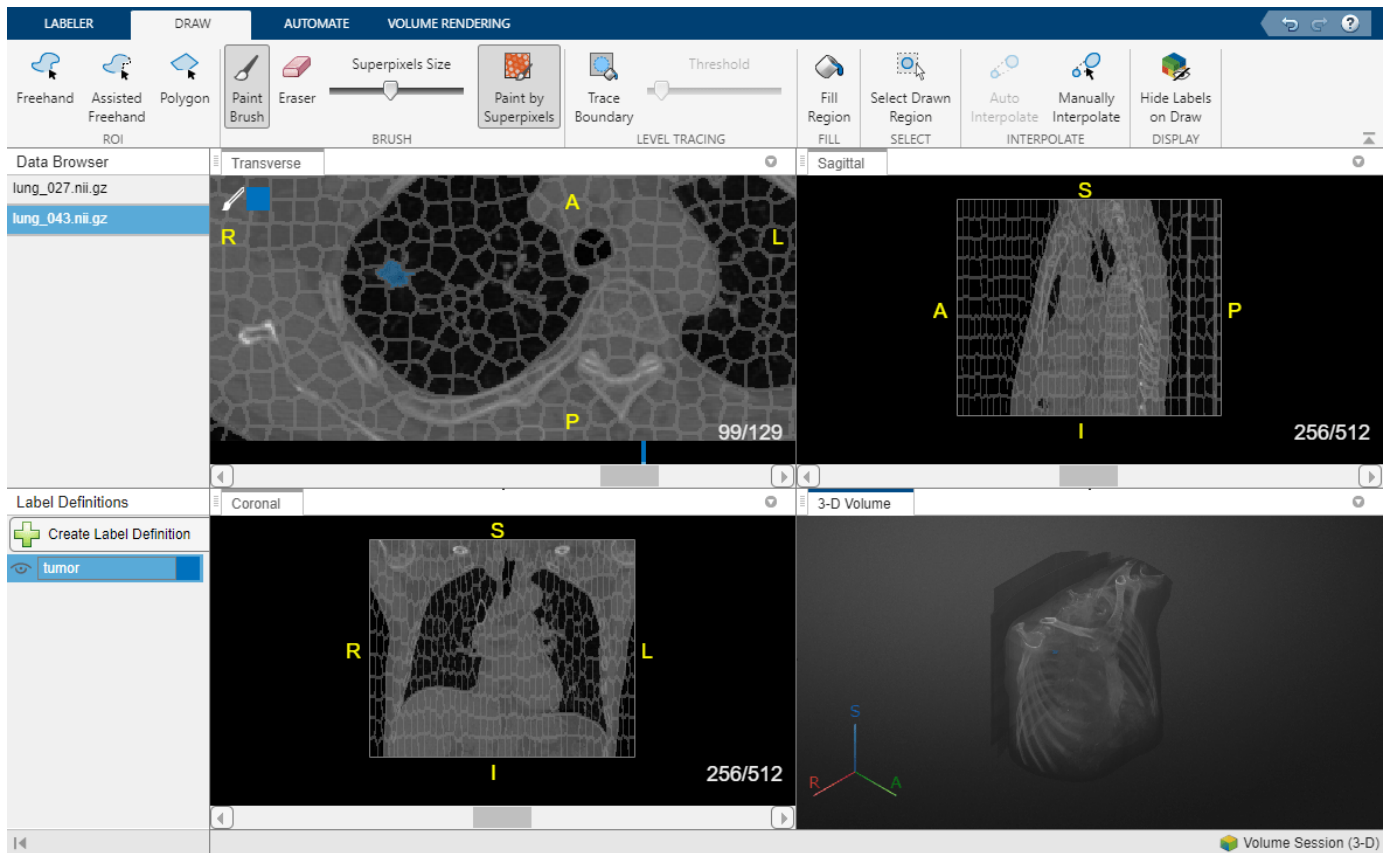


After testing your function on a single slice, you can run it on all of the slices, or a subset of the slices, in the selected direction. You can run it from the current slice to the end (the highest numbered slice) or from the current slice back to the beginning (slice 1). You can also specify a range of slices by specifying the starting slice and the ending slice.

Label Next Image Volume

For this example, the labels for `lung_027.nii.gz` are complete when you have labeled each slice of the tumor. **Medical Image Labeler** automatically assigns a value of 0 to unlabeled pixels in the label images saved to the session folder, so you do not need to label the background manually.

To move to the next volume, select `lung_043.nii.gz` in the **Data Browser** pane. Repeat the labeling process to draw labels on the tumor region in this volume.



Export Ground Truth Data

The **Medical Image Labeler** app automatically saves a `groundTruthMedical` object as a MAT file in the session folder. You can also export a `groundTruthMedical` object, saved as a MAT file, to an alternate file location from the app. On the **Labeler** tab, click **Export** and, under **Ground Truth**, select **To File**.

You can load the exported MAT file into the MATLAB workspace using the `load` function. The properties of the `groundTruthMedical` object, `gTruthMed`, contain information about the image data source, label definitions, and location of the saved label images. Display information about the object and each of its properties using these commands.

- `gTruthMed` — Display the properties of the `groundTruthMedical` object.
- `gTruthMed.DataSource` — Location of the source of the unlabeled medical volumes.
- `gTruthMed.LabelDefinitions` — Table of information about label definitions.
- `gTruthMed.LabelData` — Locations of the saved, labeled images.

References

[1] Medical Segmentation Decathlon. "Lung." Tasks. Accessed May 10, 2018. <http://medicaldecathlon.com/>.

The Lung data set is provided by the Medical Segmentation Decathlon under the CC-BY-SA 4.0 license. All warranties and representations are disclaimed. See the license for details. This example uses a subset of the original data set consisting of two CT volumes. The labels shown in this example were created for illustration purposes and have not been verified by a clinician.

See Also

Medical Image Labeler | `groundTruthMedical`

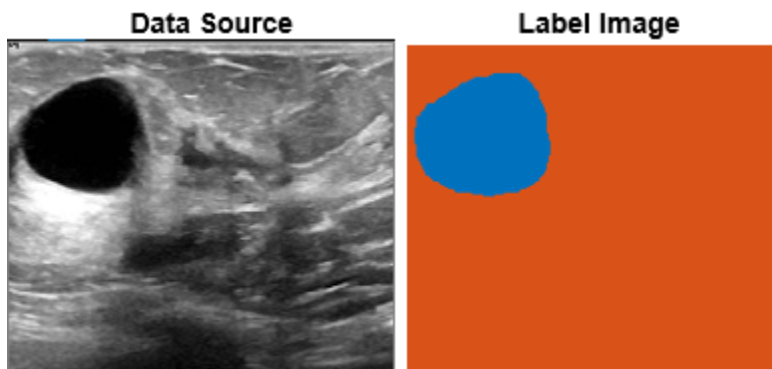
Related Examples

- "Get Started with Medical Image Labeler" on page 1-9
- "Label 2-D Ultrasound Series Using Medical Image Labeler" on page 5-2
- "Automate Labeling in Medical Image Labeler" on page 5-20

Automate Labeling in Medical Image Labeler

This example shows how to implement a deep learning automation algorithm for labeling tumors in breast ultrasound images by using the Medical Image Labeler app.

Semantic segmentation assigns a class label to every pixel in an image. The Medical Image Labeler app provides manual, semiautomated, and automated tools to label medical images for semantic segmentation. Using automation, you can create and apply custom segmentation functions that use image processing or deep learning. For an example that shows how to train the network used in this example, see “Breast Tumor Segmentation from Ultrasound Using Deep Learning” on page 6-32.



Download Pretrained Network

Create a folder to store the pretrained network and image data set.

```
dataDir = fullfile(tempdir,"BreastSegmentation");
if ~exist(dataDir,"dir")
    mkdir(dataDir)
end
```

Download the pretrained DeepLab v3+ network and test image by using the `downloadTrainedNetwork` helper function. The helper function is attached to this example as a supporting file.

```
pretrainedNetwork_url = "https://www.mathworks.com/supportfiles/" + ...
    "image/data/breastTumorDeepLabV3.tar.gz";
downloadTrainedNetwork(pretrainedNetwork_url,dataDir);
```

Unzip the TAR GZ file completely.

```
gunzip(fullfile(dataDir,"breastTumorDeepLabV3.tar.gz"),dataDir);
untar(fullfile(dataDir,"breastTumorDeepLabV3.tar"),dataDir);
exampleDir = fullfile(dataDir,"breastTumorDeepLabV3");
```

Download Image Data

This example uses a subset of the Breast Ultrasound Images (BUSI) data set [1 on page 5-27]. The BUSI data set contains 2-D ultrasound images stored in the PNG file format. The total size of the data set is 197 MB. The data set contains 133 normal scans, 487 scans with benign tumors, and 210 scans with malignant tumors. This example uses images from the tumor groups only. Each ultrasound image has a corresponding tumor mask image. The tumor mask labels have been reviewed by clinical radiologists [1].

Run this code to download the data set from the MathWorks® website and unzip the downloaded folder.

```
zipFile = matlab.internal.examples.downloadSupportFile("image", ...
    "data/Dataset_BUSI.zip");
filepath = fileparts(zipFile);
unzip(zipFile,filepath)
```

The `imageDir` folder contains the downloaded and unzipped data set.

```
imageDir = fullfile(filepath,"Dataset_BUSI_with_GT");
```

Extract a subset of 20 benign tumor images to a new folder, `dicomDir`. Write the copied images in the DICOM file format, which the Medical Image Labeler supports.

```
imds = imageDatastore(fullfile(imageDir,"benign","*").png));

dicomDir = fullfile(exampleDir,"images");
if ~exist(dicomDir,"dir")
    mkdir(dicomDir)
    for i = 1:20
        I = imread(imds.Files{i});
        [~,filename,~] = fileparts(imds.Files{i});
        dicomwrite(I,fullfile(exampleDir,"images",filename+".dcm"));
    end
end
```

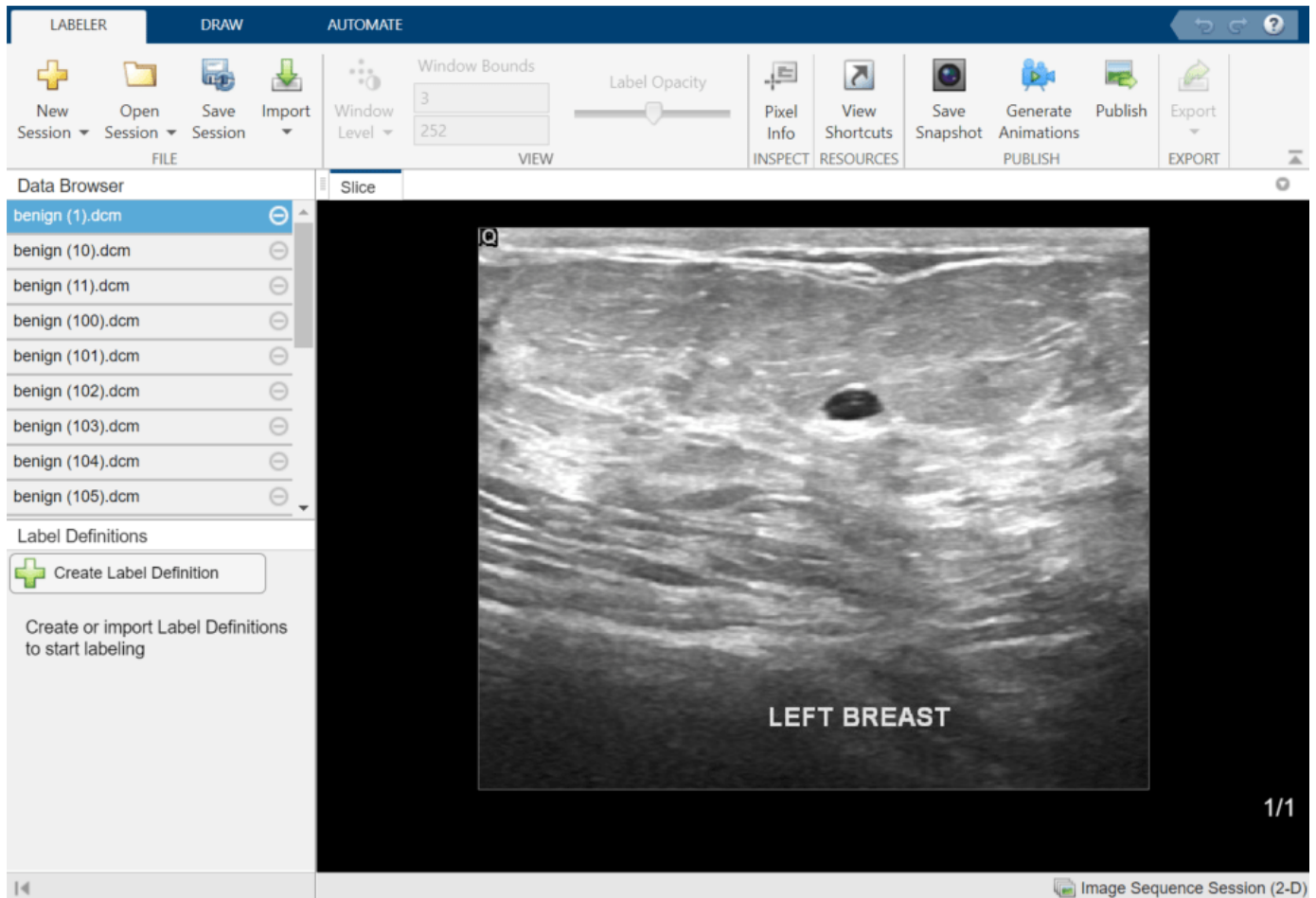
Load Data Source Images into Medical Image Labeler

Open the Medical Image Labeler app and create a new 2-D image session.

```
medicalImageLabeler("Image")
```

The app launches and opens the Create a new session folder dialog box. In the dialog box, specify a location to save the new session folder by entering a path or selecting **Browse** and navigating to your desired location. Then, select **Create Session**.

To load the ultrasound images into the app, on the app toolbar, click **Import**. Then, under **Data**, select **From File**. Browse to the location of `dicomDir`. Select all of the files by clicking the first filename, pressing **Shift**, and clicking the last filename. Click **Open**. The app loads the files and lists the names in the **Data Browser** pane.



Create Label Definition

A *label definition* specifies the name, color, and numeric index of a label. In the **Label Definitions** pane, select **Create Label Definition** to create a label with the default name `Label1`. To change the name of the label, click the label and enter a new name. The label name must be a valid MATLAB variable name with no spaces. For this example, specify the name of the label as `tumor`.



Create Automation Algorithm

In the **Automate** tab of the app toolstrip, click **Add Algorithm** and select **New > Function Template**. In the function template, enter the below code for the automation algorithm and click **Save**. This example uses the default name for the algorithm, `myalgorithm`, but you can specify any name. The automation function must have two inputs, `I` and `MASK`.

- **I** is the input data source image, which in this example is the ultrasound image. When you run the algorithm, the app automatically passes the data source image selected in the **Data Browser** to the algorithm as **I**.
- **MASK** is the initial label image. When you run the algorithm, the app automatically passes the currently selected label in the **Label Definitions** pane to the algorithm as **MASK**. The label image can be empty or contain some labeled pixels.

The automation function must return one output, **MASK**, which is the final label image. The **MASK** image must be a logical array of the same size as **I**, with pixels valued as `true` where the label is present.

The algorithm function used in this example, `myAlgorithm`, performs these steps:

- Resize the input image **I** to the input size of the network.
- Load the pretrained network.
- Apply the network to the input image **I** by using the `semanticseg` function. The network returns the segmented image, `segmentedImg`, as a categorical array with labels "tumor" and "background".
- Convert the categorical label matrix to a binary image that is `true` within the tumor label region.
- Resize the logical label mask, **MASK**, back to the original size of the input image.

```
function MASK = myAlgorithm(I,MASK)
%Medical Image Processing Function
%
% I      - RGB or grayscale image I that corresponds to the image data of
%         the slice during automation.
% MASK   - A logical array where the first two dimensions match the first
%         two dimensions of input image I. If the user has already created
%         a labeled region, MASK may have pixels labeled as true when
%         passed to this function.
%
%-----
% Auto-generated by the Medical Image Labeler App.
%
% When used by the App, this function will be called for every slice of the
% volume as specified by the user.
%
%-----

% Replace the sample below with your code-----
imshow = size(I);
networkSize = [256 256];
I = imresize(I,networkSize);

persistent trainedNet

if isempty(trainedNet)
pretrainedFolder = fullfile(tempdir,"BreastSegmentation","breastTumorDeepLabV3");
savedData = load(fullfile(pretrainedFolder,"breast_seg_deepLabV3.mat"));
trainedNet = savedData.trainedNet;
end

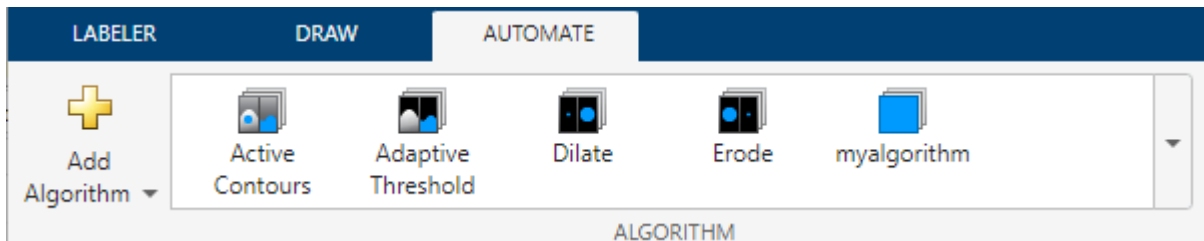
segmentedImg = semanticseg(I,trainedNet);
```

```
MASK = segmentedImg=="tumor";
MASK = imresize(MASK,imsize(1:2));
```

```
%-----
```

```
end
```

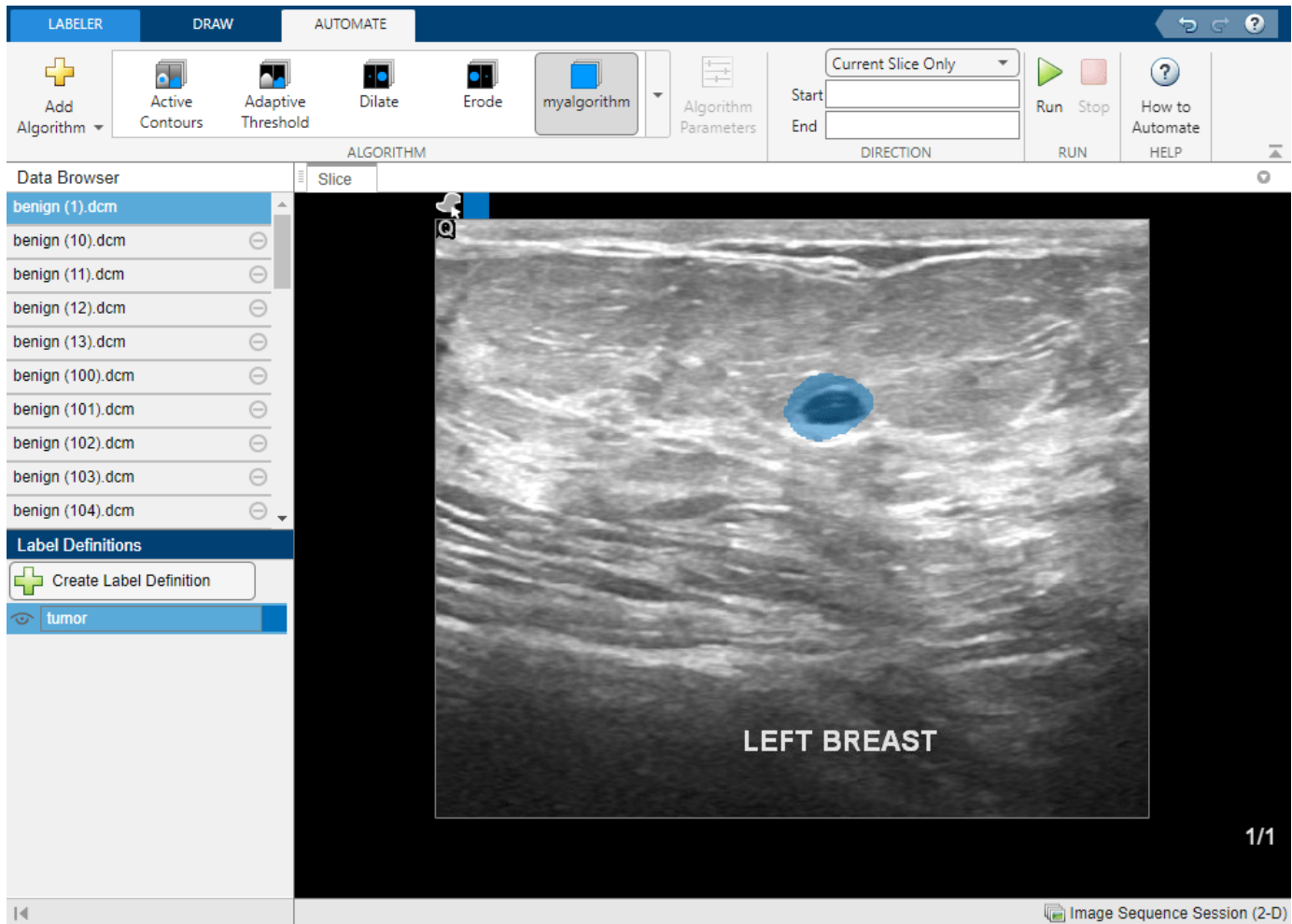
Close the function template. In the **Automate** tab of the app toolstrip, your new algorithm appears in the **Algorithm** gallery.



Run Automation Algorithm

Select the image to run the algorithm on in the **Data Browser** pane. Make sure that the **tumor** label definition is selected in the **Label Definitions** pane. In the **Automation** tab of the app toolstrip, in the **Algorithm** gallery, select **myAlgorithm** and select **Run**. If your data is a multiframe image series, you can adjust the **Direction** settings to specify whether the algorithm is applied to the current frame, from the current frame back to the first frame, or from the current frame to the last frame. Alternatively, you can directly enter a custom frame range using the **Start** and **End** text boxes.

When the algorithm finishes running, the label image is appears in the **Slice** panel. View the label image to visually check the network performance.



If you need to update the algorithm, reopen the function template in the location you specified when you saved the function template. Before rerunning an updated algorithm, you can remove labels from the first iteration by right-clicking on the filename in the **Data Browser** and selecting **Remove Labels**. Alternatively, you can create new label definitions for each new iteration to visually compare results in the **Slice** pane.

Apply Algorithm to Batch of Images

Once you are satisfied with your automation algorithm, you can apply it to each image loaded in the app one by one by selecting the next file in the **Data Browser** and selecting **Run**.

Alternatively, if you have many images to label, applying the algorithm outside the app can be faster. To apply the algorithm outside the app, first export the app data as a `groundTruthMedical` object. On the **Labeler** tab of the app toolbar, select **Export**. Under **Ground Truth**, select **To File**. In the Export Ground Truth dialog box, navigate to the folder specified by `exampleDir` and click **Save**. Then, load the `groundTruthMedical` object into the MATLAB workspace.

```
load(fullfile(exampleDir, "groundTruthMed.mat"))
```

Extract the list of label images in the `groundTruthMedical` object.

```
gTruthLabelData = gTruthMed.LabelData;
```

Find the data source images that contain no labels.

```
idxPixel = gTruthLabelData == "";  
imagesNotProcessedIdx = find(idxPixel);  
imageProcessedIdx = find(~idxPixel,1);
```

Find the name of the directory that contains the label images.

```
labelDataLocation = fileparts(gTruthLabelData(imageProcessedIdx));
```

Find the numeric pixel label ID for the tumor label definition.

```
idxPixel = strfind(gTruthMed.LabelDefinitions.Name,"tumor");  
pixID = gTruthMed.LabelDefinitions.PixelLabelID(idxPixel);
```

Loop through the data source files with no labels, applying the `myAlgorithm` function to each image. Write each output label image as a MAT file to the directory specified by `labelDataLocation`. Add the filenames of the new label images to the list `gTruthLabelData`.

```
for i = imagesNotProcessedIdx'  
  
    imageFile = gTruthMed.DataSource.Source{i};  
  
    medImage = medicalImage(imageFile);  
    labels = zeros(size(medImage.Pixels,1:3),"uint8");  
  
    for j = 1:medImage.NumFrames  
        tumorMask = myAlgorithm(extractFrame(medImage,j),[]);  
        temp = labels(:,:,j);  
        temp(tumorMask) = pixID;  
        labels(:,:,j) = temp;  
    end  
  
    [~,filename] = fileparts(imageFile);  
    filename = strcat(filename,".mat");  
    labelFile = fullfile(labelDataLocation,filename);  
    save(labelFile,"labels")  
  
    gTruthLabelData(i) = string(labelFile);  
  
end
```

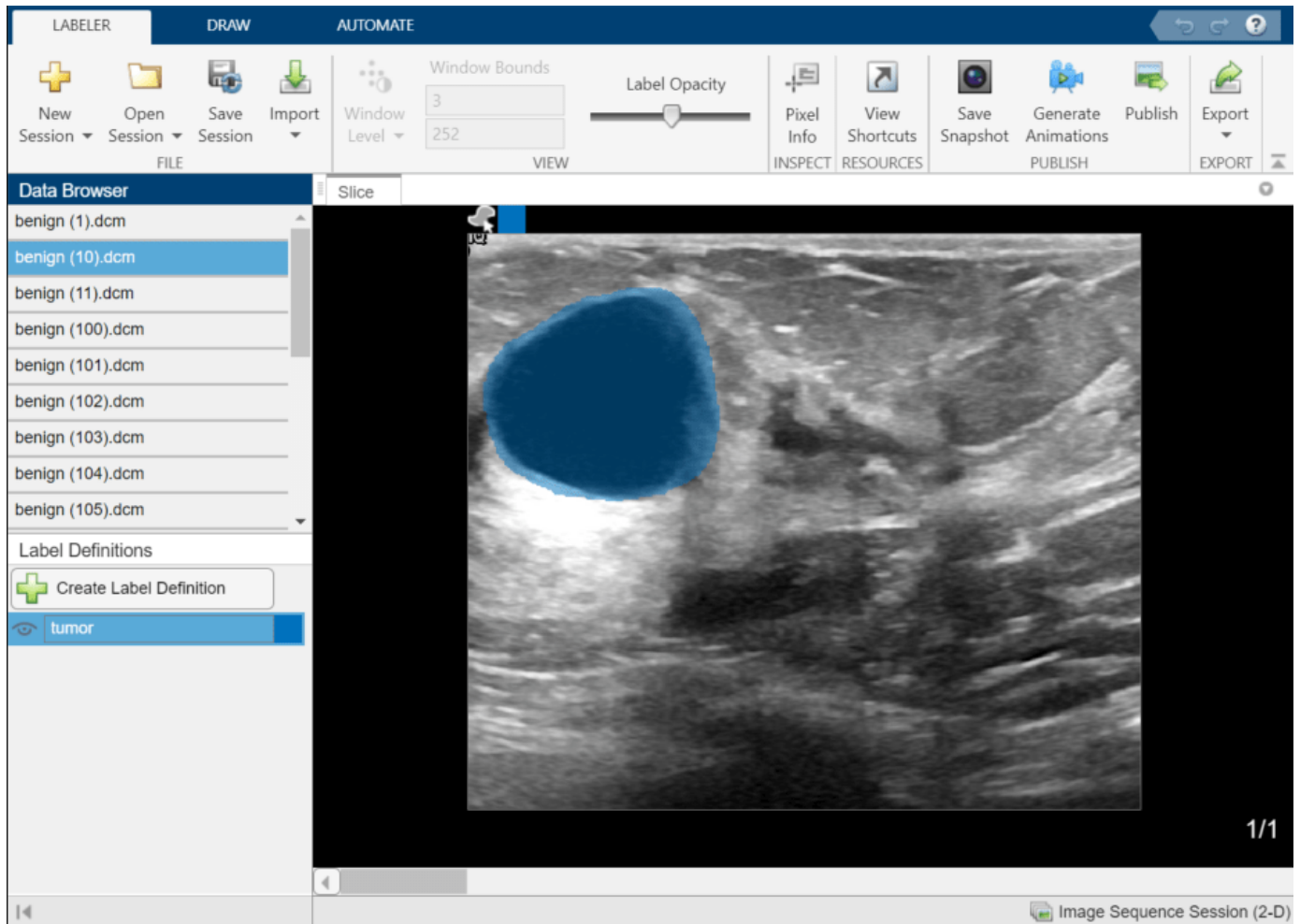
Create a new `groundTruthMedical` object that contains the original ultrasound images and label definitions plus the new label image names.

```
newGroundTruthMed = groundTruthMedical(gTruthMed.DataSource, ...  
    gTruthMed.LabelDefinitions,gTruthLabelData);
```

View the fully labeled data set by loading the new `groundTruthMedical` object in the Medical Image Labeler app. Upon launching, the app opens the Create a new session folder dialog box. You must use the dialog box to create a new app session to view the update `groundTruthMedical` object data.

```
medicalImageLabeler(newGroundTruthMed);
```

View different images by clicking filenames in the **Data Browser**. Running the algorithm on the batch of images is complete, so all of the images have tumor labels.



References

[1] Al-Dhabyani, Walid, Mohammed Gomaa, Hussien Khaled, and Aly Fahmy. "Dataset of Breast Ultrasound Images." *Data in Brief* 28 (February 2020): 104863. <https://doi.org/10.1016/j.dib.2019.104863>.

See Also

Medical Image Labeler | groundTruthMedical

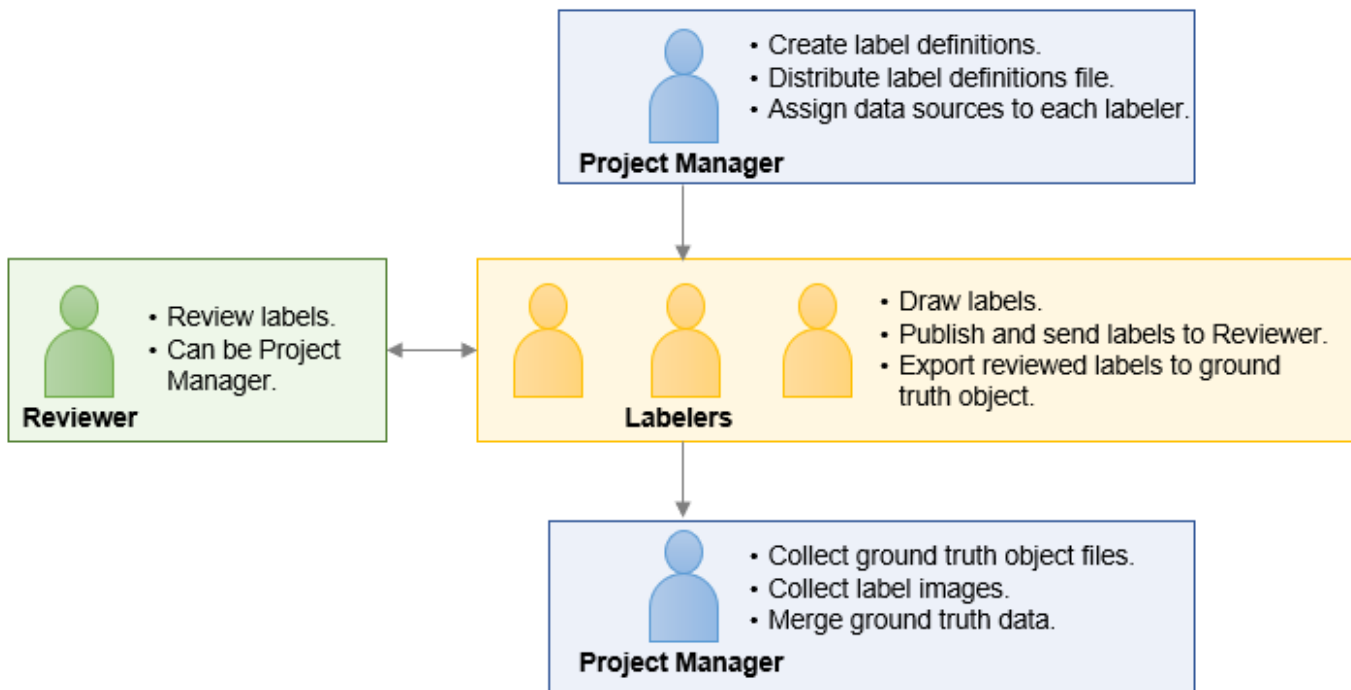
Related Examples

- "Get Started with Medical Image Labeler" on page 1-9
- "Breast Tumor Segmentation from Ultrasound Using Deep Learning" on page 6-32
- "Label 2-D Ultrasound Series Using Medical Image Labeler" on page 5-2
- "Label 3-D Medical Image Using Medical Image Labeler" on page 5-10

Collaborate on Multi-Labeler Medical Image Labeling Projects

This page shows how to work with a multi-person team to label large medical image data sets using the **Medical Image Labeler** app. Use this workflow to label a data set that consists of all 2-D images or all 3-D images and has the same set of target labels (for example tumor, lung, and chest). The original intensity images to be labeled are data source images.

A labeling team consists of a project manager, individual labelers, and one or more reviewers. The project manager creates the label definitions, assigns the images to be labeled by each labeler, and collects and compiles the labeled data. The labelers label the data source images using the label definitions provided by the project manager, request feedback from the reviewer, and send approved labels to the project manager. The reviewer, who can also be the project manager, checks labeled images and provides feedback to the labelers. This figure illustrates the overall workflow and responsibilities of each role.



The multi-person labeling process consists of these steps:

- 1 “Create Label Definitions and Assign Data to Labelers (Project Manager)” on page 5-29
- 2 “Label Data and Publish Labels for Review (Labeler)” on page 5-30
- 3 “Export Ground Truth Data and Send to Project Manager (Labeler)” on page 5-32
- 4 “Inspect Labeled Images (Reviewer)” on page 5-32
- 5 “Collect, Merge, and Create Training Data (Project Manager)” on page 5-33

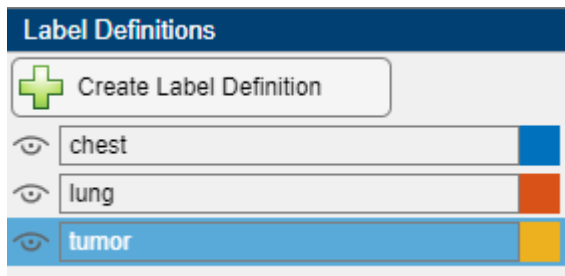
Create Label Definitions and Assign Data to Labelers (Project Manager)

Create Medical Image Labeler Session

- 1 Open the **Medical Image Labeler** app from the **Apps** tab on the MATLAB toolstrip, or by using the `medicalImageLabeler` command.
- 2 In the app toolstrip, click **New Session**. If the data set is 2-D, select **New Image session (2-D)**. If the data set is 3-D, select **New Volume session (3-D)**.

Create Label Definitions

- 1 Click **Create Label Definition** in the **Label Definitions** pane.
- 2 Click on the label to change the default name. The label name must be a valid MATLAB® variable name with no spaces. For details about valid variable names, see “Variable Names”.
- 3 To change the color associated with the label, click the colored square in the label identifier and select a color from the Color dialog box.
- 4 Click **Create Label Definition** to create additional labels. Create all of the labels required for the labeling project. When labels are nested within one another, such as tumors within an organ within the chest cavity, create the labels in order from outermost to innermost (`chest`, then `lung`, then `tumor`).

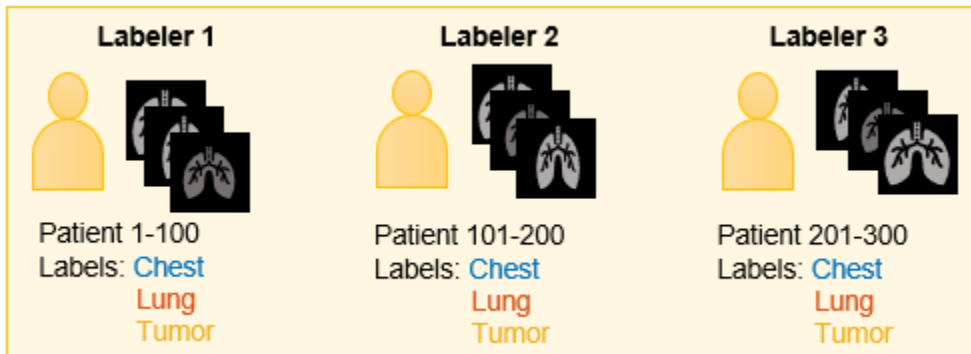


- 5 Click **Export** and, under **Label Definitions**, select **To File** to export the label definitions to a MAT file.

Distribute Labeling Assignments

Assign each labeler a subset of images in the data set to label. For example, for a data set of 300 chest CT scans, labeler 1 might label the `tumor`, `lung`, and `chest` cavity in scans 1-100, and labeler 2 might label the `tumor`, `lung`, and `chest` cavity in scans 101-200. To ensure you can merge the ground truth data after labeling, you must assign each labeler a unique set of data source images to label, and use the same label definitions for each set. Send this information to each labeler:

- Label definitions, saved as a MAT file.
- List of data source files to label. Each labeler must have access to their assigned images, either in a shared network location or on their local machine.



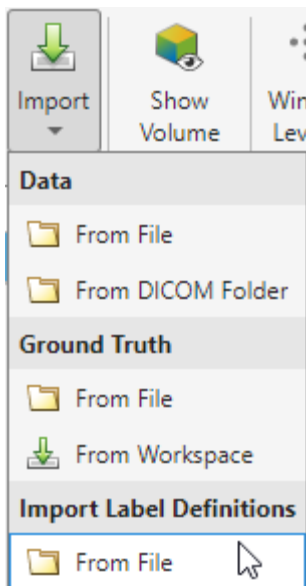
Label Data and Publish Labels for Review (Labeler)

Create Medical Image Labeler session

- 1 Open the **Medical Image Labeler** app from the **Apps** tab on the MATLAB toolstrip, or by using the `medicalImageLabeler` command.
- 2 In the app toolstrip, click **New Session**. If the data set is 2-D, select **New Image session (2-D)**. If the data set is 3-D, select **New Volume session (3-D)**.

Import Label Definitions File

- 1 On the app toolstrip, click **Import**.
- 2 Under **Import Label Definitions**, select **From File**.



- 3 Select the label definitions MAT file provided by the project manager. The imported labels appear in the **Label Definitions** pane.


Import Data to Label

Load the data source images assigned to you by the project manager into the app. To load an image, click **Import** on the app toolbar and select an option under **Data**. Select one or more files to import in the Import Image dialog box.

- For a volume session, you can import an image from a file or from a directory of DICOM files corresponding to one volume.
- For an image session, you can import an image or image series from a file.

You can check that the app successfully imported the files by reading the list of filenames in the **Data Browser** pane.


Label Images

Select an image in the **Data Browser** to begin labeling. The no labels symbol  next file names in the **Data Browser** indicates which image files have not been labeled.

Draw pixel labels using the labels in the **Label Definitions** pane. For an example showing how to label 3-D medical images, see “Label 3-D Medical Image Using Medical Image Labeler” on page 5-10. For an example showing how to label a 2-D image series, see “Label 2-D Ultrasound Series Using Medical Image Labeler” on page 5-2. The labels are saved as label images, which contain a mask of the drawn labels for a data source image.

Publish Label Overlay Images for Review

To get feedback from the reviewer, you can publish and share label overlay images, which show the data source image with its corresponding label image as an overlay. You can publish label overlay images as individual PNG files or as a single PDF, which the reviewer can inspect without opening MATLAB. To publish label overlay images, follow these steps:

- 1 The published images match the current settings in the app. Configure the app display settings as desired by using these steps:
 - a You can change the visibility of an individual label by clicking the  icon next to the label name in the **Label Definitions** pane.
 - b On the **Labeler** tab of the app toolbar, use the **Label Opacity** slider to set the transparency of the labels.
 - c On the **Labeler** tab of the app toolbar, use the **Window Level** tool to set the contrast of 2-D slice images.
 - d To include a 3-D snapshot in the published images, on the **Labeler** tab of the app toolbar, select **Show Volume**. The 3-D snapshot matches the current rotation, zoom, and display markers in the **3-D Volume** pane. On the **Labeler** tab, click **Display Markers** to change the visibility of the scale bar and orientation axes display markers.
- 2 In the **Data Browser**, select the data source image file for which you want to publish label overlay images. You can publish label overlay images for only one data source at a time.
- 3 On the app toolbar, click **Publish** to open the **Publish** pane.

- 4 Under **Publish Format**, select either **Images** or **PDF**. Selecting **Images** publishes individual PNG files for each frame of an image series, or slice of a volume. Selecting **PDF** publishes one PDF file for the entire image file.
- 5 Specify the **Slices** information. For an image session, you can specify a range of slices for a multi-frame image sequence.

For a volume session, select the **Slice Direction** (coronal, sagittal, or transverse) along which to publish images and select either **All Slices** or **Range**. If you select **Range**, specify a range of slices. To include a 3-D snapshot, select **Include 3-D volume Snapshot**.

- 6 Click **Publish** and specify the export location.

Inspect Labeled Images (Reviewer)

Collect published label overlay images from each labeler. You can open the published PDF or PNG files using a PDF or image viewer without using MATLAB. Inspect the labeled images, and send feedback to the labelers if necessary. The labelers can update the labels and publish a new set of images for additional review. When the labels are satisfactory, the labeler exports the ground truth data and sends it to the project manager.

Export Ground Truth Data and Send to Project Manager (Labeler)

Export Ground Truth Object

When you receive approval from the reviewer, export the labeled data as a `groundTruthMedical` object to share with the project manager. On the **Labeler** tab, click **Export** and, under **Ground Truth**, select **To File**.

Send Ground Truth Data to Project Manager

Send these files to the project manager:

- MAT file containing the `groundTruthMedical` object.
- Label images containing the label masks for the data source images specified in the ground truth object. You can access the complete path to the label images in the `LabelData` property of the exported `groundTruthMedical` object. Access the `LabelData` property by loading the ground truth MAT file into the MATLAB workspace by using the `load` function. Share a copy of the label images with the project manager by sending them directly or by saving a copy in a shared network location.

Collect, Merge, and Create Training Data (Project Manager)

Collect Labeled Ground Truth Data

Collect these files from each labeler:

- MAT file containing a `groundTruthMedical` object.
- Label images containing the label masks for the data source images specified in the `groundTruthMedical` object.

You can load each object into the workspace by using the `load` function. Because the **Medical Image Labeler** always saves the ground truth object as `gTruthMed` in the exported MAT file, you must specify unique variable names when loaded each file to avoid overwriting data. For example, this code loads ground truth objects from two exported MAT files, `gTruth1.mat` and `gTruth2.mat`, that each contain a `groundTruthMedical` object, `gTruthMed`.

```
matFile1 = load("gTruth1.mat");
gTruthMed1 = matFile1.gTruthMed;
matFile2 = load("gTruth2.mat");
gTruthMed2 = matFile2.gTruthMed;
```

The `groundTruthMedical` object contains file paths in the `DataSource` and `LabelData` properties that point to the location of the data source images and the label images, respectively, on the local machine of the associated labeler.

Update File Paths

To merge the ground truth objects and create training data, you must update each ground truth object to point to the data source and label image file locations on your machine. Even if the files are saved in a shared network location, if a labeler maps a different drive letter to the shared network folder, the file path can be incorrect.

To update these paths, use the `changeFilePaths` object function. Specify the ground truth object as an input argument to this function. If the directory paths have changed, but the filenames have not, specify a string vector containing the folder names for the old and new paths. The function updates all file paths in the `groundTruthMedical` object at the specified original path. The function returns any paths that it is unable to resolve. For example, this code shows how to change the drive letter for a directory.

```
alternativePaths = ["C:\Shared\ImgFolder", "D:\Shared\ImgFolder"];
unresolvedPaths = changeFilePaths(gTruthMed1, alternativePaths);
```

If the filenames also changed, specify a string array of old and new file paths. For example, this code shows how to change the drive letter for individual files, and how to append a suffix to each filename.

```
alternativePaths = ...
{"C:\Shared\ImgFolder\Img1.png", "D:\Shared\ImgFolder\Img1_new.png"}, ...
["C:\Shared\ImgFolder\Img2.png", "D:\Shared\ImgFolder\Img2_new.png"], ...
.
.
.
["C:\Shared\ImgFolder\ImgN.png", "D:\Shared\ImgFolder\ImgN_new.png"]};
unresolvedPaths = changeFilePaths(gTruthMed1, alternativePaths);
```

By default, `changeFilePaths` updates both the data source and the label image file paths. To update the paths stored in the `DataSource` and `LabelData` properties separately, use the `propertyName` argument.

Merge Ground Truth Data

Merge the updated `groundTruthMedical` objects into one object by using the `merge` object function. For example, this code merges two objects, `gTruthMed1` and `gTruthMed2`.

```
gTruthMerged = merge(gTruthMed1, gTruthMed2);
```

Create Training Data

You can use the merged labeled ground truth data to train a semantic segmentation network. To create training data, load the data source images into an `imageDatastore`. Load the label images into a `pixelLabelDatastore`. For more details about creating training data for semantic segmentation from a `groundTruthMedical` object, see “Create Datastores for Medical Image Semantic Segmentation” on page 6-2.

See Also

`groundTruthMedical` | `changeFilePaths` | `merge`

Related Examples

- “Get Started with Medical Image Labeler” on page 1-9
- “Label 3-D Medical Image Using Medical Image Labeler” on page 5-10
- “Label 2-D Ultrasound Series Using Medical Image Labeler” on page 5-2
- “Automate Labeling in Medical Image Labeler” on page 5-20

Medical Image Segmentation

- “Create Datastores for Medical Image Semantic Segmentation” on page 6-2
- “Convert Ultrasound Image Series into Training Data for 2-D Semantic Segmentation Network” on page 6-5
- “Create Training Data for 3-D Medical Image Semantic Segmentation” on page 6-9
- “Segment Lungs from CT Scan Using Pretrained Neural Network” on page 6-14
- “Brain MRI Segmentation Using Pretrained 3-D U-Net Network” on page 6-24
- “Breast Tumor Segmentation from Ultrasound Using Deep Learning” on page 6-32
- “Cardiac Left Ventricle Segmentation from Cine-MRI Images Using U-Net Network” on page 6-40

Create Datastores for Medical Image Semantic Segmentation

Semantic segmentation deep learning networks segment a medical image by assigning a class label, such as `tumor` or `lung`, to every pixel in the image. To train a semantic segmentation network, you must have a collection of images, or data sources, and a collection of label images that contain labels for the pixels in the data source images. Manage training data for semantic segmentation by using datastores:

- Load data source images by using an `imageDatastore` object.
- Load pixel label images by using a `pixelLabelDatastore` object.
- Pair data source images and pixel label images by using a `CombinedDatastore` or a `randomPatchExtractionDatastore` object.

Medical Image Ground Truth Data

You can use the **Medical Image Labeler** app to label 2-D or 3-D medical images to generate training data for semantic segmentation networks. The app stores labeling results in a `groundTruthMedical` object, which specifies the filenames of data source and pixel label images in its `DataSource` and `LabelData` properties, respectively. The table shows how a `groundTruthMedical` object formats the data source and label image information for 2-D versus 3-D data.

Type of Data	Data Source Format	Label Data Format
2-D medical images or multiframe 2-D image series	<p>The <code>DataSource</code> property contains an <code>ImageSource</code> object that specifies 2-D images or image series in one of these formats:</p> <ul style="list-style-type: none"> • Single DICOM file. • Single NIFTI file. <p>Note A <code>groundTruthMedical</code> object can specify a combination of 2-D DICOM and NIFTI data sources.</p>	<p>The <code>LabelData</code> property contains a string array. Each element specifies the filename of the label image for the corresponding data source.</p> <ul style="list-style-type: none"> • 2-D label images are MAT files, regardless of the data source file format. • If a data source has no labels, the corresponding element of <code>LabelData</code> is an empty string, "".

Type of Data	Data Source Format	Label Data Format
3-D medical image volumes	<p>The <code>DataSource</code> property contains a <code>VolumeSource</code> object that specifies 3-D image volumes in one of these formats:</p> <ul style="list-style-type: none"> • Directory of DICOM files corresponding to one volume. • Single DICOM file. • Single NIFTI file. • Single NRRD file. <p>Note A <code>groundTruthMedical</code> object can specify a combination of 3-D DICOM, NIFTI, and NRRD data sources.</p>	<p>The <code>LabelData</code> property contains a string array. Each element specifies the filename of the label image for the corresponding data source.</p> <ul style="list-style-type: none"> • 3-D label images are NIFTI files, regardless of the data source file format. • If a data source has no labels, the corresponding element of <code>LabelData</code> is an empty string, "".

Datastores for Semantic Segmentation

You can perform medical image semantic segmentation using 2-D or 3-D deep learning networks. A 2-D network accepts 2-D input images and predicts segmentation labels using 2-D convolution kernels. The input images can be one of these sources:

- Images from 2-D modalities, such as X-ray.
- Individual frames extracted from a multiframe 2-D image series, such as an ultrasound video.
- Individual slices extracted from a 3-D image volume, such as a CT or MRI scan.

A 3-D network accepts 3-D input images and predicts segmentation labels using 3-D convolution kernels. The input images are 3-D medical volumes, such as entire CT or MRI volumes.

The benefits of 2-D networks include faster prediction speeds and lower memory requirements. Additionally, you can generate many 2-D training images from one image volume or series. Therefore, fewer scans are required to train a 2-D network that segments a volume slice-by-slice versus training a fully 3-D network. The major benefit of 3-D networks is that they use information from adjacent slices or frames to predict segmentation labels, which can produce more accurate results.

- For an example that shows how to create datastores that contain 2-D ultrasound frames, see “Convert Ultrasound Image Series into Training Data for 2-D Semantic Segmentation Network” on page 6-5.
- For an example that shows how to create, preprocess, and augment 3-D datastores for segmentation, see “Create Training Data for 3-D Medical Image Semantic Segmentation” on page 6-9.

See Also

`groundTruthMedical` | `ImageSource` | `VolumeSource` | `imageDatastore` | `pixelLabelDatastore` | `CombinedDatastore` | `randomPatchExtractionDatastore`

Related Examples

- “Convert Ultrasound Image Series into Training Data for 2-D Semantic Segmentation Network” on page 6-5
- “Create Training Data for 3-D Medical Image Semantic Segmentation” on page 6-9
- “Collaborate on Multi-Labeler Medical Image Labeling Projects” on page 5-28

Convert Ultrasound Image Series into Training Data for 2-D Semantic Segmentation Network

This example shows how to create training data for a 2-D semantic segmentation network using a `groundTruthMedical` object that contains multiframe ultrasound image series. To train a semantic segmentation network, you need pairs of data source images and label images stored in an `imageDatastore` and `pixelLabelDatastore` respectively. To train a 2-D network using data from a multiframe image series, you must convert the series into individual frames stored in separate files.

Create Ground Truth Object

You can export a `groundTruthMedical` object from the Medical Image Labeler app or create one programmatically.

This example creates a `groundTruthMedical` object by using the `createGroundTruthMed2D` helper function. The helper function is attached to this example as a supporting file. The `groundTruthMedical` object references a multiframe echocardiogram data source and its corresponding label image series. The data source and label image are stored as a single DICOM file and MAT file, respectively.

```
gTruthMed = createGroundTruthMed2D;
```

Extract Data from Ground Truth Object

Extract the data source and label image file names from the `groundTruthMedical` object.

```
dataSource = gTruthMed.DataSource.Source;
labelData = gTruthMed.LabelData;
```

Remove any data sources that are missing label images.

```
noLabelsIdx = labelData ~= "";
dataSource = dataSource(noLabelsIdx);
labelData = labelData(noLabelsIdx);
```

Extract the label definitions from the `groundTruthMedical` object. Add an additional label definition for the background region, corresponding to a pixel value of 0.

```
labelDefs = gTruthMed.LabelDefinitions;
labelDefs(2,:) = {"background",[0 1 0],0};
```

Convert Multiframe Series into 2-D Frames

The data source is a multiframe ultrasound image series stored in a single DICOM file. Convert the DICOM file into individual 2-D images, stored as MAT files, by using the `convertImageSeriesToFrames` supporting function. The supporting function is defined at the end of this example.

```
newDataSource = convertImageSeriesToFrames(dataSource);
newDataSource = string(newDataSource);
```

The label data is a multiframe image series stored in a single MAT file. Convert the single MAT file into individual MAT files for each frame by using the `convertLabelSeriesToFrames` supporting function. The supporting function is defined at the end of this example.

```
newLabelData = convertLabelSeriesToFrames(labelData);
```

Create Datastores

Load the individual MAT file data sources into a `imageDatastore`.

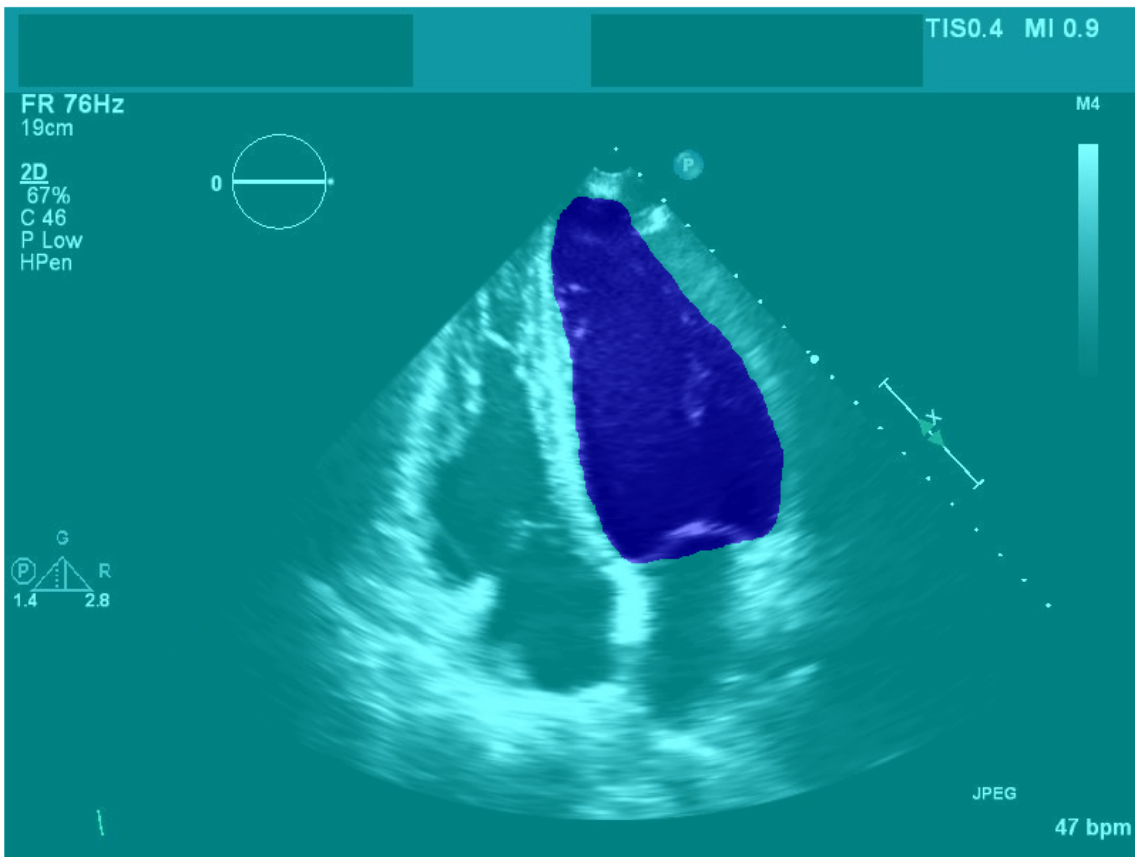
```
imds = imageDatastore(newDataSource,...
    ReadFcn=@readFramesLabels,...
    FileExtensions=[".mat",".dcm"]);
```

Load the individual MAT file label images into a `pixelLabelDatastore` (Computer Vision Toolbox). Use the label definitions from the `groundTruthMedical` object to map pixel values to categorical labels in the datastore.

```
pxds = pixelLabelDatastore(cellstr(newLabelData),labelDefs.Name,labelDefs.PixelLabelID,...
    ReadFcn=@readFramesLabels,...
    FileExtensions=".mat");
```

Preview one image and label. Display the labeled image by using the `labeloverlay` function.

```
im = preview(imds);
label = preview(pxds);
imOverlay = labeloverlay(im,label);
imshow(imOverlay)
```



Create a CombinedDatastore that pairs each data source image with its corresponding label image.

```
trainingData = combine(imds,pxds);
```

Supporting Functions

The `convertImageSeriesToSlices` function converts a multiframe DICOM image series, such as ultrasound data, into individual 2-D frames stored in MAT files. The function returns a cell array of the new MAT filenames.

```
function newDataSource = convertImageSeriesToFrames(labelDataSource)
% Create a data directory to store MAT files
dataFileDir = fullfile(pwd,"GroundTruthData");

if ~isfolder(dataFileDir)
    mkdir(dataFileDir)
end

image = medicalImage(labelDataSource);
data3d = image.Pixels;

% Assumption that time is the third dimension
numFrames = size(data3d,3);
newDataSource = cell(numFrames,1);

for frame = 1:numFrames
    data = squeeze(data3d(:,:,frame,:));
    [~,name,~] = fileparts(labelDataSource);
    matFileName = strcat(fullfile(dataFileDir,name),"_",num2str(frame),".mat");
    save(matFileName,"data");
    newDataSource{frame} = string(matFileName);
end

end
```

The `convertLabelSeriesToFrames` function converts a multiframe image series stored in a single file into individual frames stored as separate MAT files. The function returns a cell array of the new MAT filenames.

```
function newlabelData = convertLabelSeriesToFrames(labelSource)
% Create a label directory to store MAT files
labelFileDir = fullfile(pwd,"GroundTruthLabel");

if ~isfolder(labelFileDir)
    mkdir(labelFileDir)
end

labelData = load(labelSource);

% Assumption that time is the third dimension
numFrames = size(labelData.labels,3);
newlabelData = cell(numFrames,1);

for frame = 1:numFrames
    data = squeeze(labelData.labels(:,:,frame,:));
    [~,name,~] = fileparts(labelSource);
    matFileName = strcat(fullfile(labelFileDir,name),"_",num2str(frame),".mat");
    save(matFileName,"data");
end
```

```
        newLabelData{frame} = string(matFileName);  
end  
  
end
```

The `readFramesLabels` function reads data stored in a variable named `data` from a MAT file.

```
function data = readFramesLabels(filename)  
d = load(filename);  
data = d.data;  
end
```

See Also

[groundTruthMedical](#) | [imageDatastore](#) | [pixelLabelDatastore](#) | [transform](#) | [combine](#)

Related Examples

- “Label 2-D Ultrasound Series Using Medical Image Labeler” on page 5-2
- “Create Training Data for 3-D Medical Image Semantic Segmentation” on page 6-9

Create Training Data for 3-D Medical Image Semantic Segmentation

This example shows how to create training data for a 3-D semantic segmentation network using a `groundTruthMedical` object that contains 3-D medical image data. This example also shows how to transform datastores to preprocess and augment image and pixel label data.

Create Ground Truth Object

You can export a `groundTruthMedical` object from the Medical Image Labeler app or create one programmatically.

This example creates a `groundTruthMedical` object by using the `createGroundTruthMed3D` helper function. The helper function is attached to this example as a supporting file. The ground truth object references three data sources, including one chest CT volume stored as a directory of DICOM files and two chest CT volumes stored as NIFTI files. The directory of DICOM files and its label image are attached to this example as supporting files. The `createGroundTruthMed3D` function downloads the NIFTI files and their label images from the MathWorks® website. The NIFTI files are a subset of the Medical Segmentation Decathlon data set [1 on page 6-13], and have a total file size of approximately 76 MB.

```
gTruthMed = createGroundTruthMed3D;
```

Extract Data from Ground Truth Object

Extract the data source and label image file names from the `groundTruthMedical` object.

```
dataSource = gTruthMed.DataSource.Source;
labelData = gTruthMed.LabelData;
```

Remove data sources that are missing label images.

```
noLabelsIdx = labelData~= "";
dataSource = dataSource(noLabelsIdx);
labelData = labelData(noLabelsIdx);
```

Extract the label definitions from the `groundTruthMedical` object. Add an additional label definition for the background region, corresponding to a pixel value of 0.

```
labelDefs = gTruthMed.LabelDefinitions;
labelDefs(2,:) = {"background", [0 1 0], 0};
```

Load Data Source Images into Image Datastore

A `groundTruthMedical` object can contain data sources stored as single DICOM, NIFTI, or NRRD file, or a directory of DICOM files. The `imageDatastore` object does not support reading volumetric images from a directory of DICOM files. Use the `convertMultifileDICOMs` supporting function to search `dataSource` for data sources stored as a directory of DICOM files, and convert any DICOM directories into single MAT files.

```
dataSource = convertMultifileDICOMs(dataSource);
dataSource = string(dataSource);
```

Load the updated data sources into an `imageDatastore`. Specify a custom read function, `readMedicalVolumes`, which is defined at the end of this example. The `readMedicalVolumes`

function reads data from medical volumes stored as a single MAT file, DICOM file, or NIFTI file. The .gz file extension corresponds to compressed NIFTI files.

```
imds = imageDatastore(dataSource,...  
    ReadFcn=@readMedicalVolumes,...  
    FileExtensions=[".mat",".dcm",".nii",".gz"]);
```

Load Label Images into Pixel Label Datastore

Load the label images into a pixelLabelDatastore (Computer Vision Toolbox) using `niftiread` as the read function.

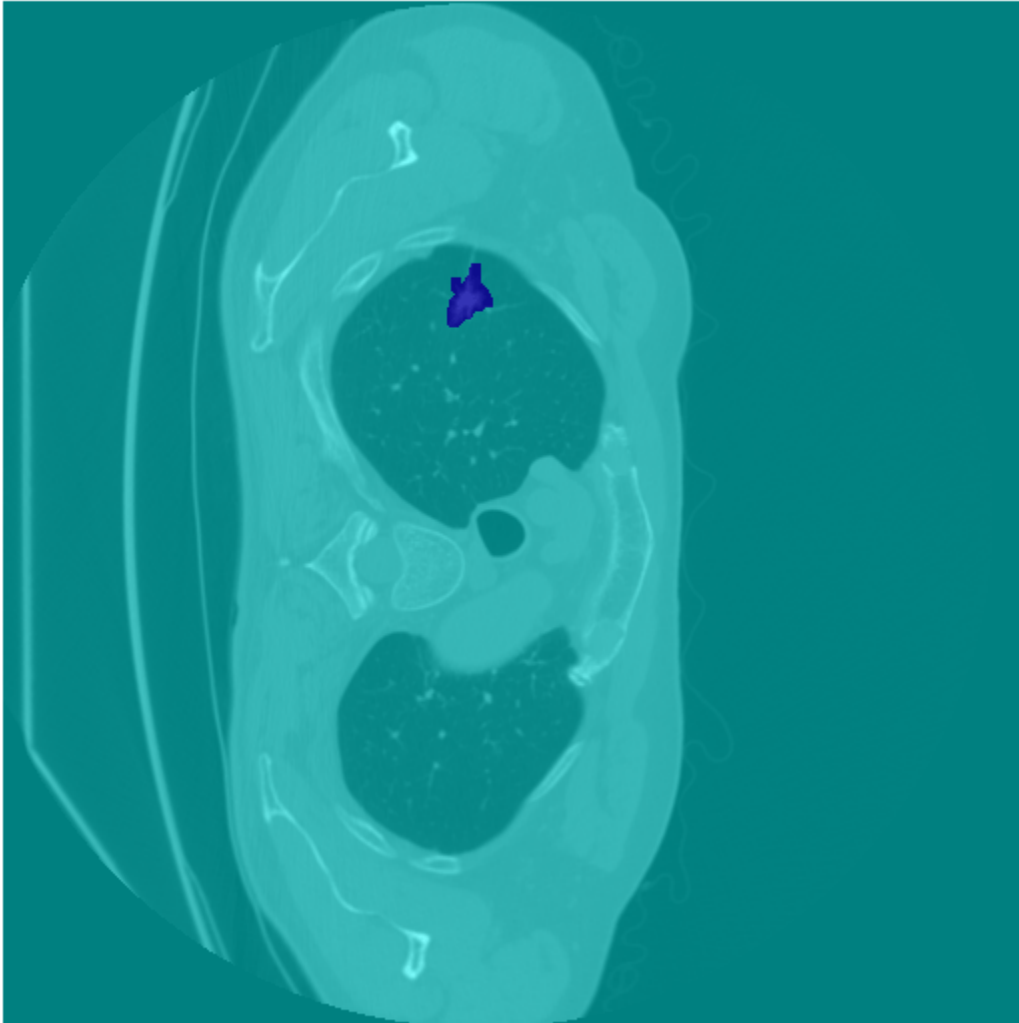
```
pxds = pixelLabelDatastore(cellstr(labelData),labelDefs.Name,labelDefs.PixelLabelID,...  
    ReadFcn=@niftiread,...  
    FileExtensions=[".nii",".gz"]);
```

Preview one image volume and its label image.

```
vol = preview(imds);  
label = preview(pxds);
```

Display one slice of the previewed volume by using the `labeloverlay` function. To better view the grayvalues of the CT slice with `labeloverlay`, first rescale the image intensities to the range [0, 1].

```
volslice = vol(:,:,100);  
volslice = rescale(volslice);  
labelslice = label(:,:,100);  
imOverlay = labeloverlay(volslice,labelslice);  
imshow(imOverlay)
```



Pair Image and Label Data

Create a `CombinedDatastore` that pairs each data source image with its corresponding label image.

```
trainingData = combine(imds,pxds);
```

Augment and Preprocess Training Data

Specify the input size of the target deep learning network.

```
targetSize = [300 300 60];
```

Augment the training data by using the `transform` function with custom operations specified by the `jitterImageIntensityAndWarp` supporting function defined at the end of this example.

```
augmentedTrainingData = transform(trainingData,@jitterImageIntensityAndWarp);
```

Preprocess the training data by using the `transform` function with custom preprocessing operation specified by the `centerCropImageAndLabel` supporting function defined at the end of this example.

```
preprocessedTrainingData = transform(augmentedTrainingData, ...
    @(data)centerCropImageAndLabel(data, targetSize));
```

Supporting Functions

The `convertMultifileDICOMs` function reads a cell array of data source file names, and converts any data sources stored as a directory of DICOM files into a single MAT file.

```
function dataSource = convertMultifileDICOMs(dataSource)
numEntries = length(dataSource);
dataFileDir = fullfile(pwd, "GroundTruthData");

if ~isfolder(dataFileDir)
    mkdir(dataFileDir)
end

for idx = 1:numEntries
    currEntry = dataSource{idx};
    % Multi-file DICOMs
    if length(currEntry) > 1
        matFileName = fileparts(currEntry(1));
        matFileName = split(matFileName, filesep);
        matFileName = replace(strtrim(matFileName(end)), " ", "_");
        matFileName = strcat(fullfile(dataFileDir, matFileName), ".mat");

        vol = medicalVolume(currEntry);
        data = vol.Voxels;

        save(matFileName, "data");
        dataSource{idx} = string(matFileName);
    end
end
end
```

The `readMedicalVolumes` function loads medical image volume data from a single MAT file, DICOM file, or NIfTI file.

```
function data = readMedicalVolumes(filename)
[~,~,ext] = fileparts(filename);
if ext == ".mat"
    d = load(filename);
    data = d.data;
else
    vol = medicalVolume(filename);
    data = vol.Voxels;
end
end
```

The `jitterImageColorAndWarp` function randomly adjusts the brightness, contrast, and gamma correction of the data source image values, and applies random geometric transformations including scaling, reflection, and rotation to the data source and pixel label images.

```
function out = jitterImageIntensityAndWarp(data)
% Unpack original data.
I = data{1};
```

```

C = data{2};

% Apply random intensity jitter.
I = jitterIntensity(I,Brightness=0.3,Contrast=0.4,Gamma=0.2);

% Define random affine transform.
tform = randomAffine3d(Scale=[0.8 1.5],XReflection=true,Rotation=[-30 30]);
rout = affineOutputView(size(I),tform);

% Transform image and pixel labels.
augmentedImage = imwarp(I,tform,"OutputView",rout);
augmentedLabel = imwarp(C,tform,"OutputView",rout);

% Return augmented data.
out = {augmentedImage,augmentedLabel};
end

```

The `centerCropImageAndLabel` function crops the data source images and pixel labels to the input size of the target deep learning network.

```

function out = centerCropImageAndLabel(data,targetSize)
    win = centerCropWindow3d(size(data{1}),targetSize);
    out{1} = imcrop3(data{1},win);
    out{2} = imcrop3(data{2},win);
end

```

References

[1] Medical Segmentation Decathlon. "Brain Tumours." Tasks. Accessed May 10, 2018. <http://medicaldecathlon.com/>.

The Medical Segmentation Decathlon data set is provided under the CC-BY-SA 4.0 license. All warranties and representations are disclaimed. See the license for details.

See Also

`groundTruthMedical` | `imageDatastore` | `pixelLabelDatastore` | `transform` | `combine`

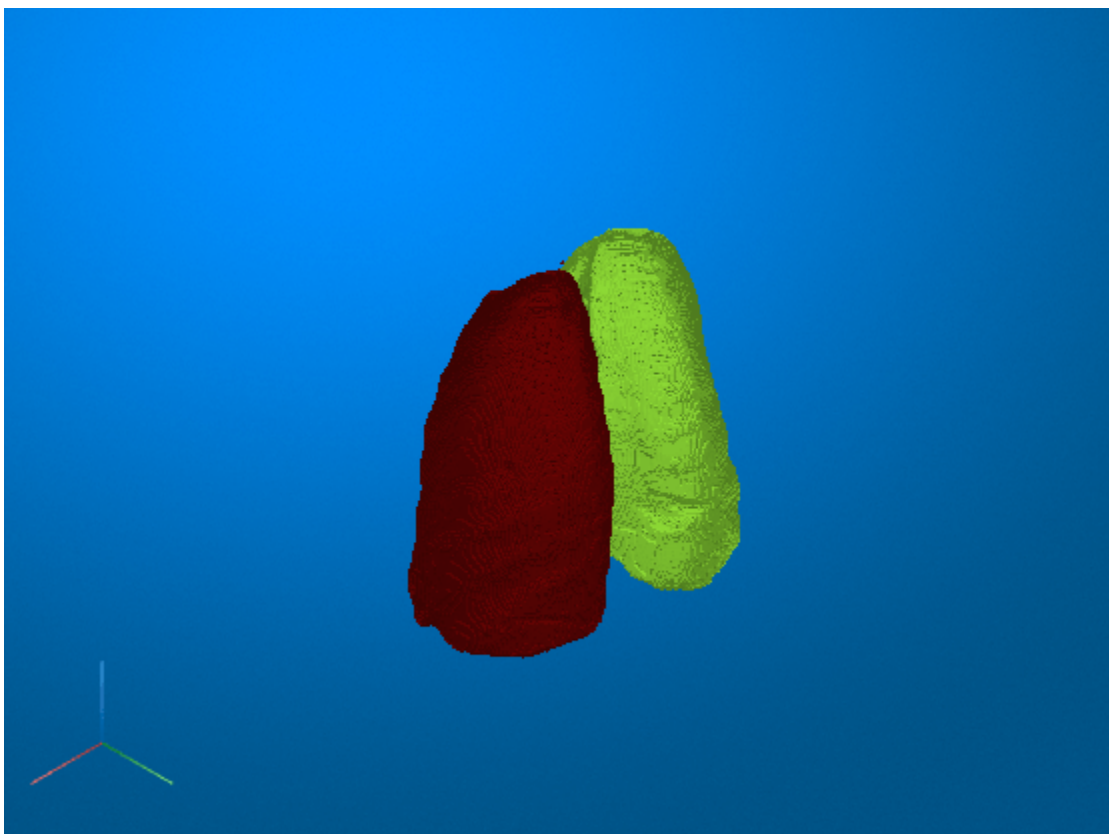
Related Examples

- "Label 3-D Medical Image Using Medical Image Labeler" on page 5-10
- "Convert Ultrasound Image Series into Training Data for 2-D Semantic Segmentation Network" on page 6-5

Segment Lungs from CT Scan Using Pretrained Neural Network

This example shows how to import a pretrained ONNX™ (Open Neural Network Exchange) 3-D U-Net [1 on page 6-22] and use it to perform semantic segmentation of the left and right lungs from a 3-D chest CT scan. Semantic segmentation associates each voxel in a 3-D image with a class label. In this example, you classify each voxel in a test data set as belonging to the left lung or right lung. For more information about semantic segmentation, see “Semantic Segmentation” (Computer Vision Toolbox).

A challenge of applying pretrained networks is the possibility of differences between the intensity and spatial details of a new data set and the data set used to train the network. Preprocessing is typically required to format the data to match the expected network input and achieve accurate segmentation results. In this example, you standardize the spatial orientation and normalize the intensity range of a test data set before applying the pretrained network.



Download Pretrained Network

Specify the desired location of the pretrained network.

```
dataDir = fullfile(tempdir,"lungmask");  
if ~exist(dataDir,"dir")  
    mkdir(dataDir);  
end
```

Download the pretrained network from the MathWorks® website by using the `downloadTrainedNetwork` helper function. The helper function is attached to this example as a

supporting file. The network on the MathWorks website is equivalent to the R231 model, available in the LungMask GitHub repository [2 on page 6-22], converted to the ONNX format. The size of the pretrained network is approximately 11 MB.

```
lungmask_url = "https://www.mathworks.com/supportfiles/medical/pretrainedLungmaskR231Net.onnx";
downloadTrainedNetwork(lungmask_url,dataDir);
```

Import Pretrained Network

Import the ONNX network as a function by using the `importONNXFunction` (Deep Learning Toolbox) function. You can use this function to import a network with layers that the `importONNXNetwork` (Deep Learning Toolbox) function does not support. The `importONNXFunction` function requires the Deep Learning Toolbox™ Converter for ONNX Model Format support package. If this support package is not installed, then `importONNXFunction` provides a download link.

The `importONNXFunction` function imports the network and returns an `ONNXParameters` object that contains the network parameters. When you import the pretrained lung segmentation network, the function displays a warning that the `LogSoftmax` operator is not supported.

```
modelfileONNX = fullfile(dataDir,"pretrainedLungmaskR231Net.onnx");
modelfileM = "importedLungmaskFcn_R231.m";
params = importONNXFunction(modelfileONNX,modelfileM);
```

```
Function containing the imported ONNX network architecture was saved to the file importedLungmaskFcn_R231.m
To learn how to use this function, type: help importedLungmaskFcn_R231.
```

```
Warning: Unable to import some ONNX operators or attributes. They may have been replaced by 'PLACEHOLDER'.
```

```
1 operator(s) : Operator 'LogSoftmax' is not supported with its current settings or in this version.
```

Open the generated function, saved as an M file in the current directory. The function contains these lines of code that indicate that the unsupported `LogSoftmax` operator is replaced with a placeholder:

```
% PLACEHOLDER FUNCTION FOR UNSUPPORTED OPERATOR (LogSoftmax):
[Vars.x460, NumDims.x460] = PLACEHOLDER(Vars.x459);
```

In the function definition, replace the placeholder code with this code. Save the updated function as `lungmaskFcn_R231`. A copy of `lungmaskFcn_R231` with the correct code is also attached to this example as a supporting file.

```
% Replacement for PLACEHOLDER FUNCTION FOR UNSUPPORTED OPERATOR (LogSoftmax):
Vars.x460 = log(softmax(Vars.x459,'DataFormat','CSSB'));
NumDims.x460 = NumDims.x459;
```

Save the network parameters in the `ONNXParameters` object `params`. Save the parameters in a new MAT file.

```
save("lungmaskParams_R231","params");
```

Load Data

Test the pretrained lung segmentation network on a test data set. The test data is a CT chest volume from the Medical Segmentation Decathlon data set [3 on page 6-22]. Download the `MedicalVolumNIftIData` ZIP archive from the MathWorks website, then unzip the file. The ZIP file

contains two CT chest volumes and corresponding label images, stored in the NIfTI file format. The total size of the data set is approximately 76 MB.

```
zipFile = matlab.internal.examples.downloadSupportFile("medical", "MedicalVolumeNiftIData.zip");  
filePath = fileparts(zipFile);  
unzip(zipFile, filePath)  
dataFolder = fullfile(filePath, "MedicalVolumeNiftIData");
```

Specify the file name of the first CT volume.

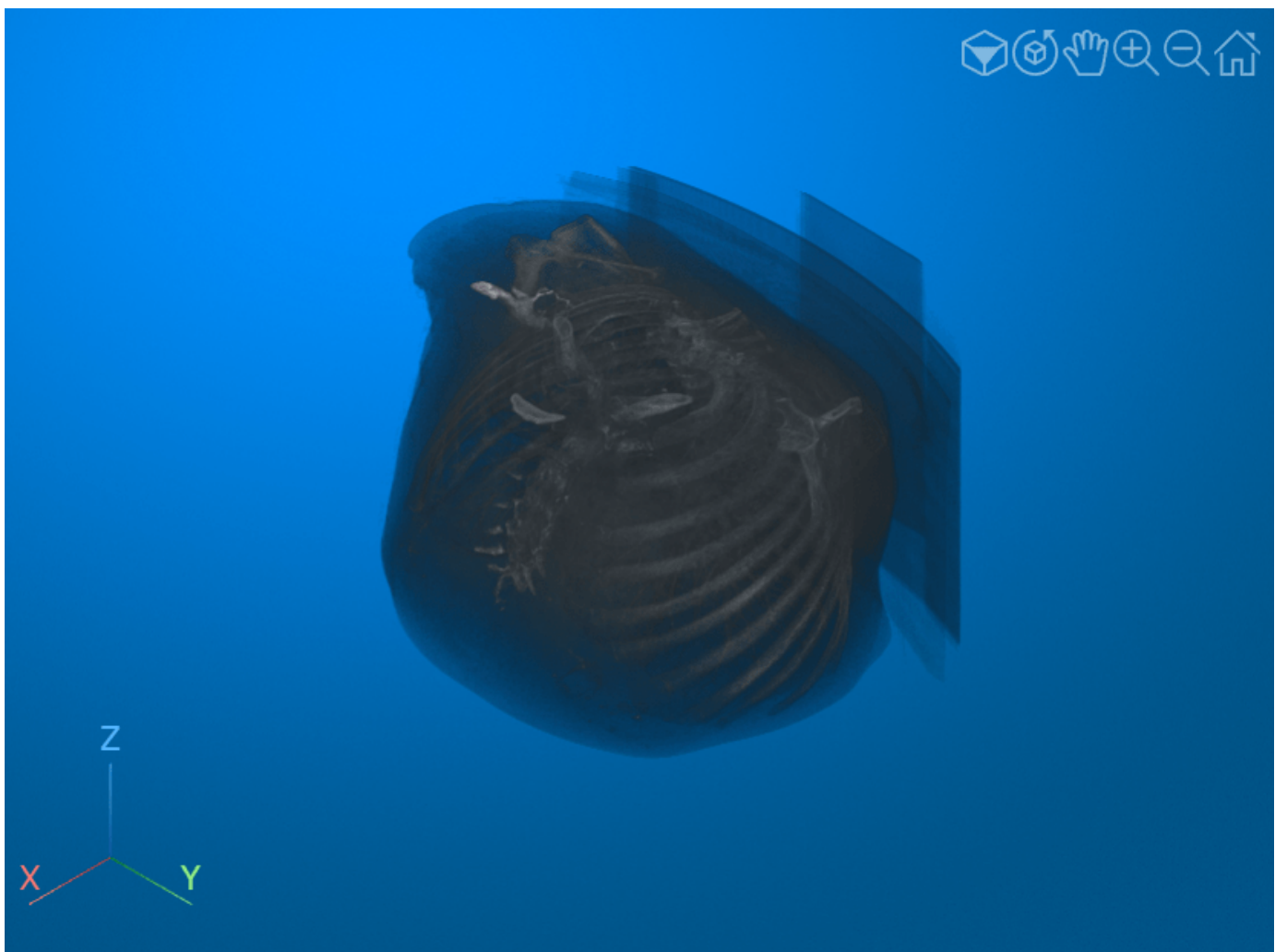
```
fileName = fullfile(dataFolder, "lung_027.nii.gz");
```

Create a `medicalVolume` object for the CT volume.

```
medVol = medicalVolume(fileName);
```

Extract and display the voxel data from the `medicalVolume` object.

```
V = medVol.Voxels;  
volshow(V, RenderingStyle="GradientOpacity");
```



Preprocess Test Data

Preprocess the test data to match the expected orientation and intensity range of the pretrained network.

Rotate the test image volume in the transverse plane to match the expected input orientation for the pretrained network. The network was trained using data oriented with the patient bed at the bottom of the image, so the test data must be oriented in the same direction. If you change the test data, you need to apply an appropriate spatial transformation to match the expected orientation for the network.

```
rotationAxis = [0 0 1];  
volAligned = imrotate3(V,90,rotationAxis);
```

Display a slice of the rotated volume to check the updated orientation.

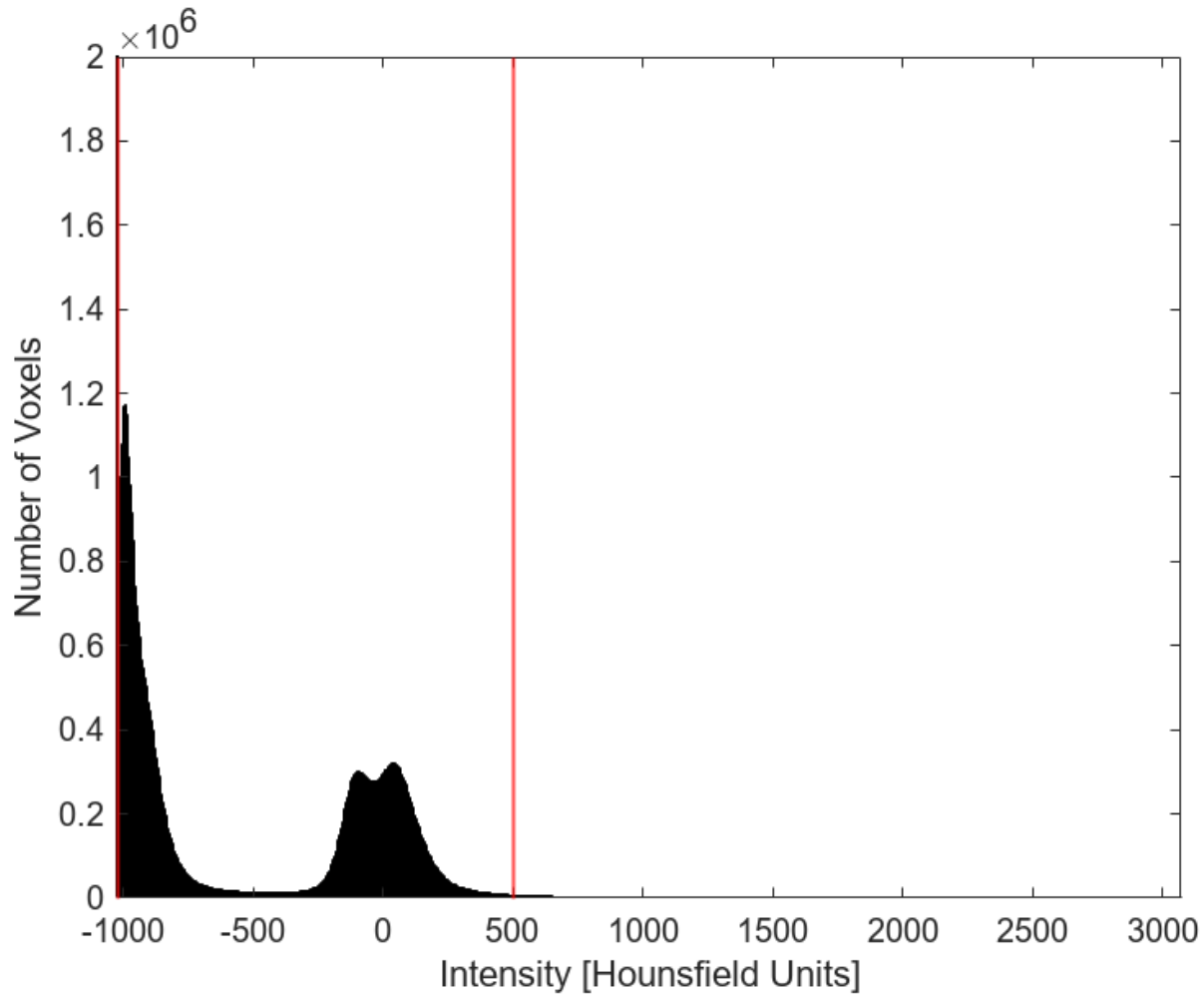
```
imshow(volAligned(:,:,150),[])
```



Use intensity normalization to rescale the range of voxel intensities in the region of interest to the range $[0, 1]$, which is the range that the pretrained network expects. The first step in intensity normalization is to determine the range of intensity values within the region of interest. The values are in Hounsfield units. To determine the thresholds for the intensity range, plot a histogram of the voxel intensity values. Set the x - and y -limits of the histogram plot based on the minimum and maximum values. The histogram has two large peaks. The first peak corresponds to background pixels outside the body of the patient and air in the lungs. The second peak corresponds to soft tissue such as the heart and stomach.

```
figure
histogram(V)
xlim([min(V,[],"all") max(V,[],"all")])
ylim([0 2e6])
xlabel("Intensity [Hounsfield Units]")
```

```
ylabel("Number of Voxels")
xline([-1024 500], "red", LineWidth=1)
```



To limit the intensities to the region containing the majority of the tissue in the region of interest, select the thresholds for the intensity range as -1024 and 500.

```
th = [-1024 500];
```

Apply the preprocessLungCT helper function to further preprocess the test image volume. The helper function is attached to this example as a supporting file. The preprocessLungCT function performs these steps:

- 1 Resize each 2-D slice along the transverse dimension to the target size, `imSize`. Decreasing the number of voxels can improve prediction speed. Set the target size to 256-by-256 voxels.
- 2 Crop the voxel intensities to the range specified by the thresholds in `th`.
- 3 Normalize the updated voxel intensities to the range [0, 1].

```
imSize = [256 256];
volInp = preprocessLungCT(volAligned, imSize, th);
```

Segment Test Data and Postprocess Predicted Labels

Segment the test CT volume by using the `lungSeg` helper function. The helper function is attached to this example as a supporting file. The `lungSeg` function predicts the segmentation mask by performing inference on the pretrained network and postprocesses the network output to obtain the segmentation mask.

To decrease the required computational time, the `lungSeg` function performs inference on the slices of a volume in batches. Specify the batch size as eight slices using the `batchSize` name-value argument of `lungSeg`. Increasing the batch size increases the speed of inference, but requires more memory. If you run out of memory, try decreasing the batch size.

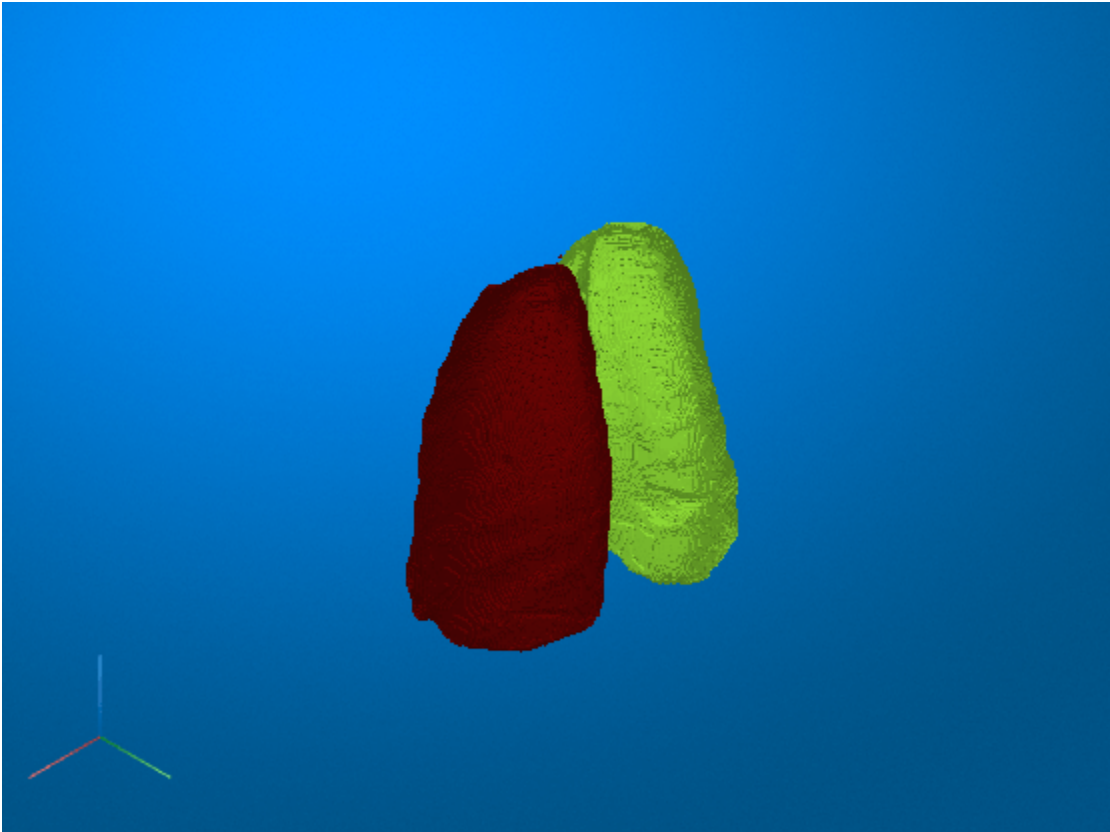
During postprocessing, the `lungSeg` helper function applies a mode filter to the network output to smooth the segmentation labels using the `modefilt` function. You can set the size of the mode filter by using the `modeFilt` name-value argument of `lungSeg`. The default filter size is `[9 9 9]`.

```
labelOut = lungSeg(volInp, batchSize=8);
```

Display Predicted Segmentation Labels

Display the segmentation results by using the `volshow` function. Use the `OverlayData` argument to plot the predicted segmentation labels. To focus on the label data, use the `Alphamap` argument to set the opacity of the image volume to 0 and the `OverlayAlphamap` argument to set the opacity of the labels to 0.9.

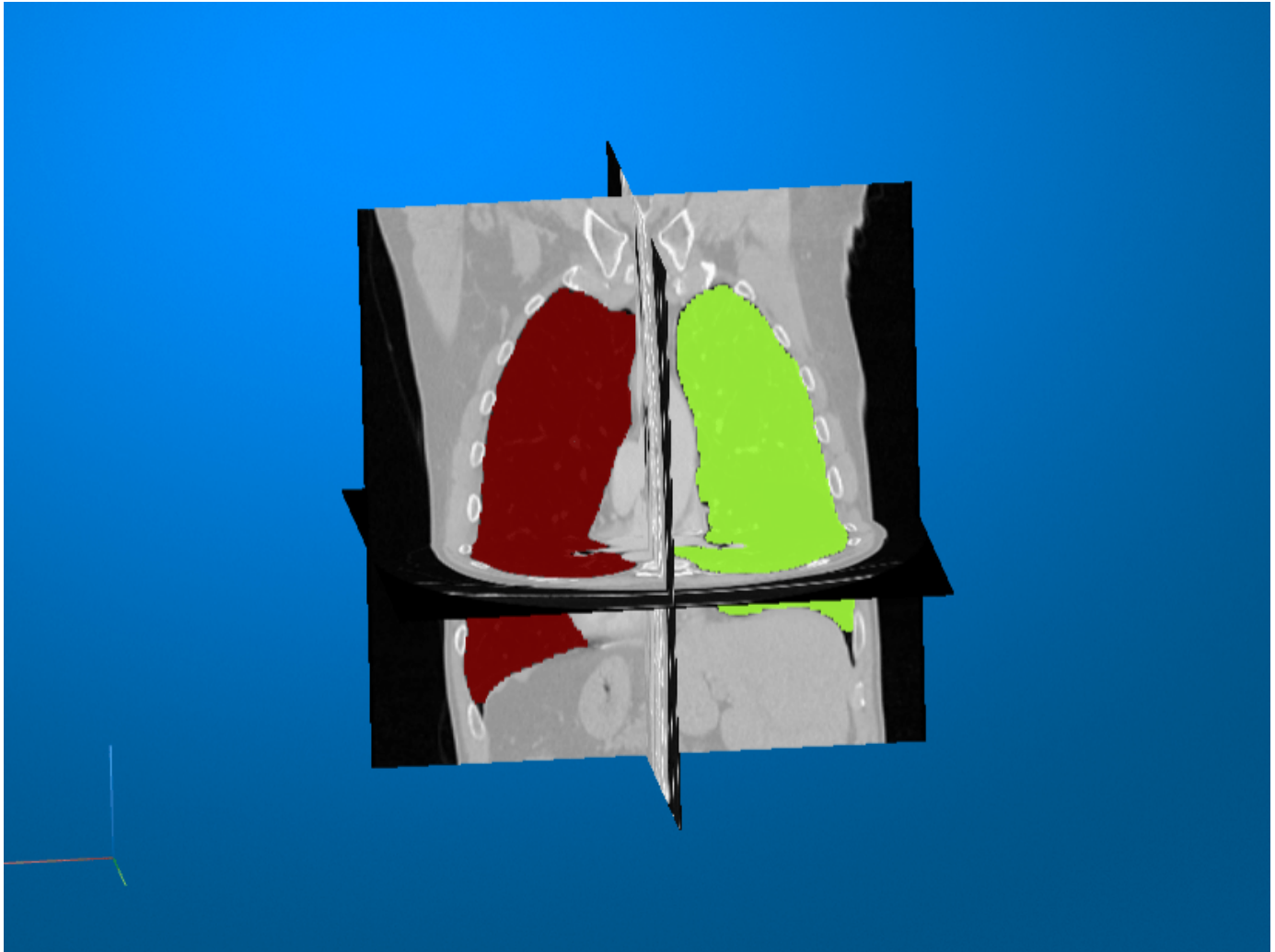
```
volshow(volInp, OverlayData=labelOut, ...  
        Alphamap=0, ...  
        OverlayAlphamap=0.9, ...  
        RenderingStyle="GradientOpacity");
```



You can also display the preprocessed test volume as slice planes with the predicted segmentation labels as an overlay by setting the `RenderingStyle` name-value argument to `"SlicePlanes"`. Specify the lung segmentation label using the `OverlayData` name-value argument.

```
volshow(volInp,OverlayData=labelOut,...  
        OverlayAlphamap=0.9,...  
        RenderingStyle="SlicePlanes");
```

Click and drag the mouse to rotate the volume. To scroll in a plane, pause on the slice you want to investigate until it becomes highlighted, then click and drag. The left and right lung segmentation masks are visible in the slices for which they are defined.



References

- [1] Hofmanninger, Johannes, Florian Prayer, Jeanny Pan, Sebastian Röhrich, Helmut Prosch, and Georg Langs. "Automatic Lung Segmentation in Routine Imaging Is Primarily a Data Diversity Problem, Not a Methodology Problem." *European Radiology Experimental* 4, no. 1 (December 2020): 50. <https://doi.org/10.1186/s41747-020-00173-2>.
- [2] GitHub. "Automated Lung Segmentation in CT under Presence of Severe Pathologies." Accessed July 21, 2022. <https://github.com/JoHof/lungmask>.

[3] Medical Segmentation Decathlon. "Lung." Tasks. Accessed May 10, 2018. <http://medicaldecathlon.com>. The Medical Segmentation Decathlon data set is provided under the CC-BY-SA 4.0 license. All warranties and representations are disclaimed. See the license for details.

See Also

`importONNXFunction` | `modelfilt` | `volshow`

Related Examples

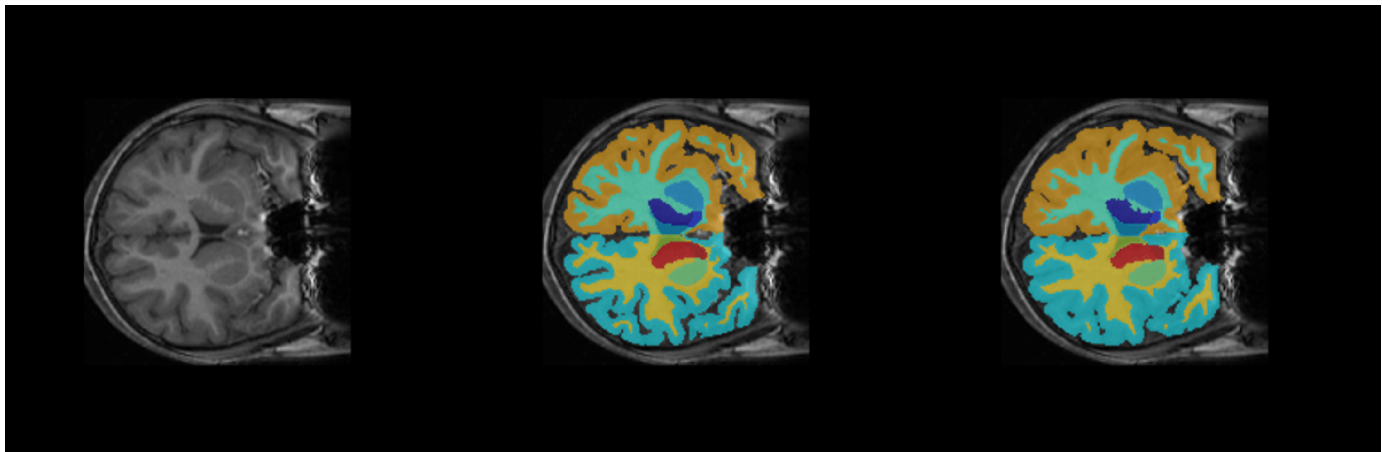
- "Segment Lungs from 3-D Chest Scan"
- "Brain MRI Segmentation Using Pretrained 3-D U-Net Network" on page 6-24

Brain MRI Segmentation Using Pretrained 3-D U-Net Network

This example shows how to segment a brain MRI using a deep neural network.

Segmentation of brain scans enables the visualization of individual brain structures. Brain segmentation is also commonly used for quantitative volumetric and shape analyses to characterize healthy and diseased populations. Manual segmentation by clinical experts is considered the highest standard in segmentation. However, the process is extremely time-consuming and not practical for labeling large data sets. Additionally, labeling requires expertise in neuroanatomy and is prone to errors and limitations in interrater and intrarater reproducibility. Trained segmentation algorithms, such as convolutional neural networks, have the potential to automate the labeling of large clinical data sets.

In this example, you use the pretrained SynthSeg neural network [1 on page 6-30], a 3-D U-Net for brain MRI segmentation. SynthSeg can be used to segment brain scans of any contrast and resolution without retraining or fine-tuning. SynthSeg is also robust to a wide array of subject populations, from young and healthy to aging and diseased subjects, and a wide array of scan conditions, such as white matter lesions, with or without preprocessing, including bias field corruption, skull stripping, intensity normalization, and template registration.



2-D Slice from Input Volume

Predicted Segmentation Map

Ground Truth

Download Brain MRI and Label Data

This example uses a subset of the CANDI data set [2 on page 6-30] [3 on page 6-30]. The subset consists of a brain MRI volume and the corresponding ground truth label volume for one patient. Both files are in the NIfTI file format. The total size of the data files is ~2.5 MB.

Run this code to download the dataset from the MathWorks® website and unzip the downloaded folder.

```
zipFile = matlab.internal.examples.downloadSupportFile("image","data/brainSegData.zip");
filepath = fileparts(zipFile);
unzip(zipFile,filepath)
```

The dataDir folder contains the downloaded and unzipped dataset.

```
dataDir = fullfile(filepath,"brainSegData");
```

Load Pretrained Network

This example uses a pretrained TensorFlow-Keras convolutional neural network. Download the pretrained network from the MathWorks® website by using the helper function `downloadTrainedNetwork`. The helper function is attached to this example as a supporting file. The size of the pretrained network is approximately 51 MB.

```
trainedBrainCANDINetwork_url = "https://www.mathworks.com/supportfiles/" + ...
    "image/data/trainedBrainSynthSegNetwork.h5";
downloadTrainedNetwork(trainedBrainCANDINetwork_url,dataDir);
```

Load Test Data

Read the metadata from the brain MRI volume by using the `niftiinfo` function. Read the brain MRI volume by using the `niftiread` function.

```
imFile = fullfile(dataDir,"anat.nii.gz");
metaData = niftiinfo(imFile);
vol = niftiread(metaData);
```

In this example, you segment the brain into 32 classes corresponding to anatomical structures. Read the names and numeric identifiers for each class label by using the `getBrainCANDISegmentationLabels` helper function. The helper function is attached to this example as a supporting file.

```
labelDirs = fullfile(dataDir,"groundTruth");
[classNames,labelIDs] = getBrainCANDISegmentationLabels;
```

Preprocess Test Data

Preprocess the MRI volume by using the `preProcessBrainCANDIData` helper function. The helper function is attached to this example as a supporting file. The helper function performs these steps:

- Resampling — If `resample` is `true`, resample the data to the isotropic voxel size 1-by-1-by-1 mm. By default, `resample` is `false` and the function does not perform resampling. To test the pretrained network on images with a different voxel size, set `resample` to `true` if the input is not isotropic.
- Alignment — Rotate the volume to a standardized RAS orientation.
- Cropping — Crop the volume to a maximum size of 192 voxels in each dimension.
- Normalization — Normalize the intensity values of the volume to values in the range [0, 1], which improves the contrast.

```
resample = false;
cropSize = 192;
[volProc,cropIdx,imSize] = preProcessBrainCANDIData(vol,metaData,cropSize,resample);
inputSize = size(volProc);
```

Convert the preprocessed MRI volume into a formatted deep learning array with the SSSCB (spatial, spatial, spatial, channel, batch) format by using `dLarray` (Deep Learning Toolbox).

```
volDL = dLarray(volProc,"SSSCB");
```

Define Network Architecture

Import the network layers from the downloaded model file of the pretrained network using the `importKerasLayers` (Deep Learning Toolbox) function. The `importKerasLayers` function

requires the Deep Learning Toolbox™ Converter for TensorFlow Models support package. If this support package is not installed, then `importKerasLayers` provides a download link. Specify `ImportWeights` as `true` to import the layers using the weights from the same HDF5 file. The function returns a `layerGraph` (Deep Learning Toolbox) object.

The Keras network contains some layers that the Deep Learning Toolbox™ does not support. The `importKerasLayers` function displays a warning and replaces the unsupported layers with placeholder layers.

```
modelFile = fullfile(dataDir, "trainedBrainSynthSegNetwork.h5");
lgraph = importKerasLayers(modelFile, ImportWeights=true, ImageInputSize=inputSize);
```

```
Warning: Imported layers have no output layer because the model does not specify a loss function
```

```
Warning: Unable to import some Keras layers, because they are not supported by the Deep Learning
```

To replace the placeholder layers in the imported network, first identify the names of the layers to replace. Find the placeholder layers using `findPlaceholderLayers` (Deep Learning Toolbox).

```
placeholderLayers = findPlaceholderLayers(lgraph)
```

```
placeholderLayers =
  PlaceholderLayer with properties:
      Name: 'UNET_prediction'
  KerasConfiguration: [1x1 struct]
      Weights: []
      InputLabels: {''}
      OutputLabels: {''}
```

```
Learnable Parameters
  No properties.
```

```
State Parameters
  No properties.
```

```
Show all properties
```

Define existing layers with the same configurations as the imported Keras layers.

```
sf = softmaxLayer;
```

Replace the placeholder layers with existing layers using `replaceLayer` (Deep Learning Toolbox).

```
lgraph = replaceLayer(lgraph, "UNET_prediction", sf);
```

Convert the network to a `dlNetwork` (Deep Learning Toolbox) object.

```
net = dlNetwork(lgraph);
```

Display the updated layer graph information.

```
layerGraph(net)
ans =
  LayerGraph with properties:
      InputNames: {'UNET_input'}
```

```

OutputNames: {1×0 cell}
  Layers: [60×1 nnet.cnn.layer.Layer]
Connections: [63×2 table]

```

Predict Using Test Data

Predict Network Output

Predict the segmentation output for the preprocessed MRI volume. The segmentation output `predictIm` contains 32 channels corresponding to the segmentation label classes, such as "background", "leftCerebralCortex", "rightThalamus". The `predictIm` output assigns confidence scores to each voxel for every class. The confidence scores reflect the likelihood of the voxel being part of the corresponding class. This prediction is different from the final semantic segmentation output, which assigns each voxel to exactly one class.

```
predictIm = predict(net,volDL);
```

Test Time Augmentation

This example uses *test time augmentation* to improve segmentation accuracy. In general, augmentation applies random transformations to an image to increase the variability of a data set. You can use augmentation before network training to increase the size of the training data set. Test time augmentation applies random transformations to test images to create multiple versions of the test image. You can then pass each version of the test image to the network for prediction. The network calculates the overall segmentation result as the average prediction for all versions of the test image. Test time augmentation improves segmentation accuracy by averaging out random errors in the individual network predictions.

By default, this example flips the MRI volume in the left-right direction, resulting in a flipped volume `flippedData`. The network output for the flipped volume is `flipPredictIm`. Set `flipVal` to `false` to skip the test time augmentation and speed up prediction.

```

flipVal = ;
if flipVal
    flippedData = fliplr(volProc);
    flippedData = flip(flippedData,2);
    flippedData = flip(flippedData,1);
    flippedData = dlarray(flippedData,"SSSCB");
    flipPredictIm = predict(net,flippedData);
else
    flipPredictIm = [];
end

```

Postprocess Segmentation Prediction

To get the final segmentation maps, postprocess the network output by using the `postProcessBrainCANDIData` helper function. The helper function is attached to this example as a supporting file. The `postProcessBrainCANDIData` function performs these steps:

- Smoothing — Apply a 3-D Gaussian smoothing filter to reduce noise in the predicted segmentation masks.
- Morphological Filtering — Keep only the largest connected component of predicted segmentation masks to remove additional noise.

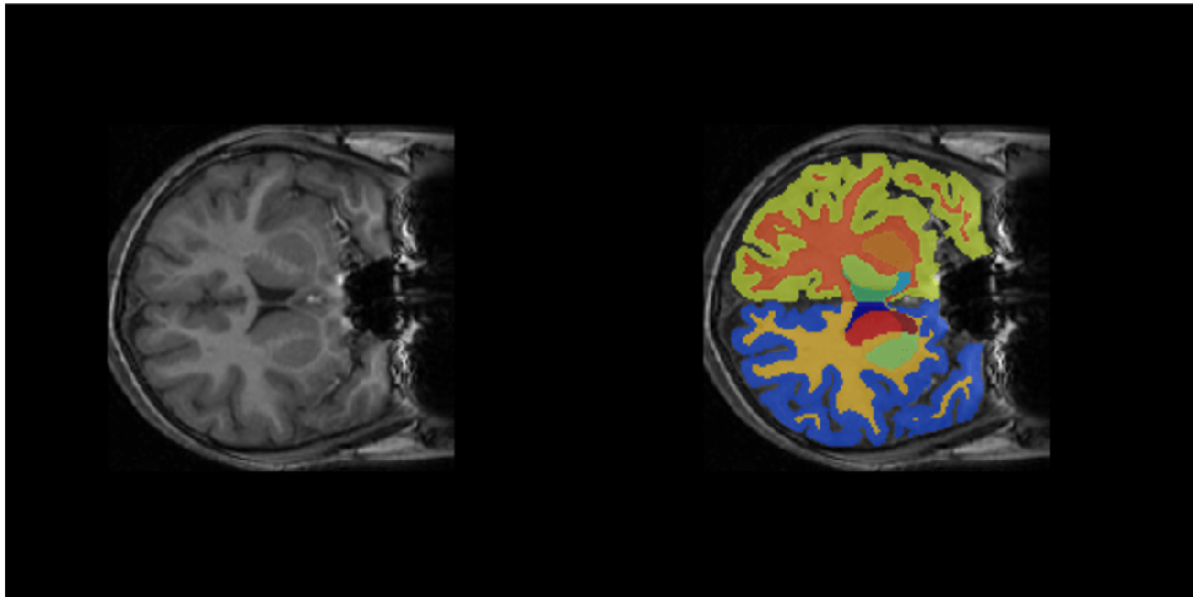
- Segmentation — Assign each voxel to the label class with the greatest confidence score for that voxel.
- Resizing — Resize the segmentation map to the original input volume size. Resizing the label image allows you to visualize the labels as an overlay on the grayscale MRI volume.
- Alignment — Rotate the segmentation map back to the orientation of the original input MRI volume.

The final segmentation result, `predictedSegMaps`, is a 3-D categorical array the same size as the original input volume. Each element corresponds to one voxel and has one categorical label.

```
predictedSegMaps = postProcessBrainCANDIData(predictIm,flipPredictIm,imSize, ...
      cropIdx,metaData,classNames,labelIDs);
```

Overlay a slice from the predicted segmentation map on a corresponding slice from the input volume using the `labeloverlay` function. Include all the brain structure labels except the background label.

```
sliceIdx = 80;
testSlice = rescale(vol(:,:,sliceIdx));
predSegMap = predictedSegMaps(:,:,sliceIdx);
B = labeloverlay(testSlice,predSegMap,"IncludedLabels",2:32);
figure
montage({testSlice,B})
```



Quantify Segmentation Accuracy

Measure the segmentation accuracy by comparing the predicted segmentation labels with the ground truth labels drawn by clinical experts.

Create a `pixelLabelDatastore` (Computer Vision Toolbox) to store the labels. Because the NIFTI file format is a nonstandard image format, you must use a NIFTI file reader to read the pixel label data. You can use the helper NIFTI file reader, `niftiReader`, defined at the bottom of this example.

```
pxds = pixelLabelDatastore(labelDirs,classNames,labelIDs,FileExtensions=".gz",...
    ReadFcn=@niftiReader);
```

Read the ground truth labels from the pixel label datastore.

```
groundTruthLabel = read(pxds);
groundTruthLabel = groundTruthLabel{1};
```

Measure the segmentation accuracy using the dice function. This function computes the Dice index between the predicted and ground truth segmentations.

```
diceResult = zeros(length(classNames),1);
for j = 1:length(classNames)
    diceResult(j)= dice(groundTruthLabel==classNames(j),...
        predictedSegMaps==classNames(j));
end
```

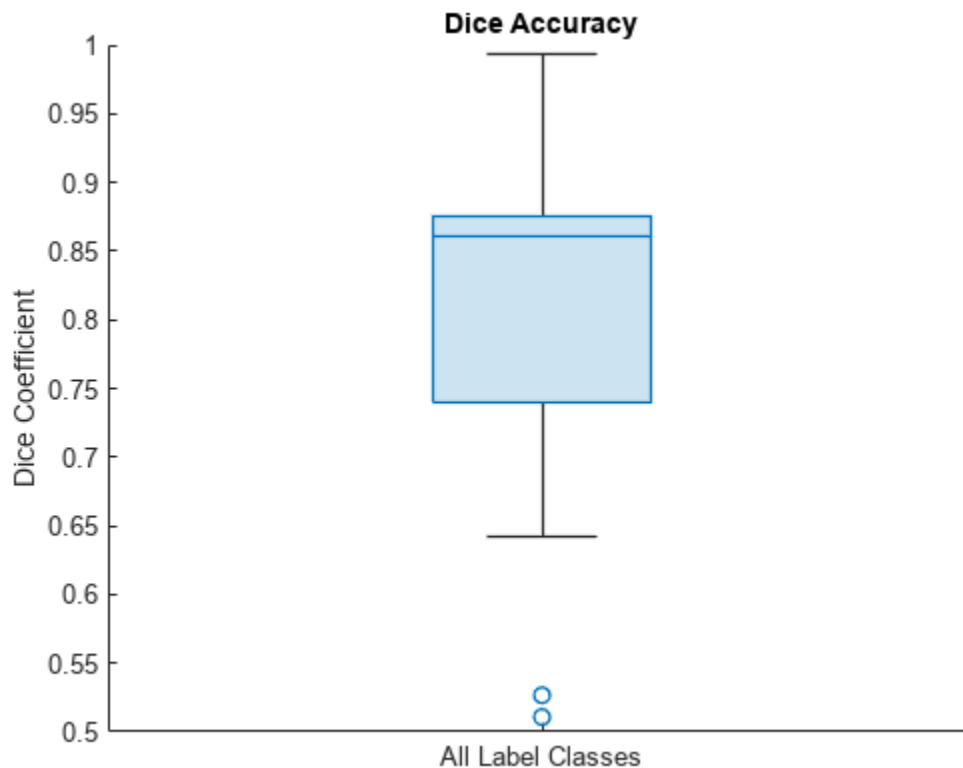
Calculate the average Dice index across all labels for the MRI volume.

```
meanDiceScore = mean(diceResult);
disp("Average Dice score across all labels = " +num2str(meanDiceScore))
```

Average Dice score across all labels = 0.80793

Visualize statistics about the Dice indices across all the label classes as a box chart. The middle blue line in the plot shows the median Dice index. The upper and lower bounds of the blue box indicate the 25th and 75th percentiles, respectively. Black whiskers extend to the most extreme data points that are not outliers.

```
figure
boxchart(diceResult)
title("Dice Accuracy")
xticklabels("All Label Classes")
ylabel("Dice Coefficient")
```



Supporting Functions

The `niftiReader` helper function reads a NIFTI file in a datastore.

```
function data = niftiReader(filename)
    data = niftiread(filename);
    data = uint8(data);
end
```

References

[1] Billot, Benjamin, Douglas N. Greve, Oula Puonti, Axel Thielscher, Koen Van Leemput, Bruce Fischl, Adrian V. Dalca, and Juan Eugenio Iglesias. "SynthSeg: Domain Randomisation for Segmentation of Brain Scans of Any Contrast and Resolution." *ArXiv:2107.09559 [Cs, Eess]*, December 21, 2021. <http://arxiv.org/abs/2107.09559>.

[2] "NITRC: CANDI Share: Schizophrenia Bulletin 2008: Tool/Resource Info." Accessed October 17, 2022. https://www.nitrc.org/projects/cs_schizbull08/.

[3] Frazier, J. A., S. M. Hodge, J. L. Breeze, A. J. Giuliano, J. E. Terry, C. M. Moore, D. N. Kennedy, et al. "Diagnostic and Sex Effects on Limbic Volumes in Early-Onset Bipolar Disorder and Schizophrenia." *Schizophrenia Bulletin* 34, no. 1 (October 27, 2007): 37-46. <https://doi.org/10.1093/schbul/sbm120>.

See Also

niftiread | importKerasLayers | findPlaceholderLayers | pixelLabelDatastore | dice | boxchart

Related Examples

- "Segment Lungs from CT Scan Using Pretrained Neural Network" on page 6-14
- "Breast Tumor Segmentation from Ultrasound Using Deep Learning" on page 6-32
- "3-D Brain Tumor Segmentation Using Deep Learning"

Breast Tumor Segmentation from Ultrasound Using Deep Learning

This example shows how to perform semantic segmentation of breast tumors from 2-D ultrasound images using a deep neural network.

Semantic segmentation involves assigning a class to each pixel in a 2-D image. In this example, you perform breast tumor segmentation using the DeepLab v3+ architecture. A common challenge of medical image segmentation is class imbalance. In segmentation, class imbalance means the size of the region of interest, such as a tumor, is small relative to the image background, resulting in many more pixels in the background class. This example addresses class imbalance by using a custom Tversky loss [1 on page 6-39]. The Tversky loss is an asymmetric similarity measure that is a generalization of the Dice index and the Jaccard index.

Load Pretrained Network

Create a folder in which to store the pretrained network and image data set. In this example, a folder named `BreastSegmentation` created within the `tempdir` directory has been used as `dataDir`. Download the pretrained DeepLab v3+ network and test image by using the `downloadTrainedNetwork` helper function. The helper function is attached to this example as a supporting file. You can use the pretrained network to run the example without waiting for training to complete.

```
dataDir = fullfile(tempdir,"BreastSegmentation");
if ~exist(dataDir,"dir")
    mkdir(dataDir)
end
pretrainedNetwork_url = "https://www.mathworks.com/supportfiles/"+ ...
    "image/data/breastTumorDeepLabV3.tar.gz";
downloadTrainedNetwork(pretrainedNetwork_url,dataDir);
```

Unzip the TAR GZ file completely. Load the pretrained network into a variable called `trainedNet`.

```
gunzip(fullfile(dataDir,"breastTumorDeepLabV3.tar.gz"),dataDir);
untar(fullfile(dataDir,"breastTumorDeepLabV3.tar"),dataDir);
exampleDir = fullfile(dataDir,"breastTumorDeepLabV3");
load(fullfile(exampleDir,"breast_seg_deepLabV3.mat"));
```

Read the test ultrasound image and resize the image to the input size of the pretrained network.

```
imTest = imread(fullfile(exampleDir,"breastUltrasoundImg.png"));
imSize = [256 256];
imTest = imresize(imTest,imSize);
```

Predict the tumor segmentation mask for the test image.

```
segmentedImg = semanticseg(imTest,trainedNet);
```

Display the test image and the test image with the predicted tumor label overlay as a montage.

```
overlayImg = labeloverlay(imTest,segmentedImg,Transparency=0.7,IncludedLabels="tumor", ...
    Colormap="hsv");
montage({imTest,overlayImg});
```



Download Data Set

This example uses the Breast Ultrasound Images (BUSI) data set [2 on page 6-39]. The BUSI data set contains 2-D ultrasound images stored in the PNG file format. The total size of the data set is 197 MB. The data set contains 133 normal scans, 487 scans with benign tumors, and 210 scans with malignant tumors. This example uses images from the tumor groups only. Each ultrasound image has a corresponding tumor mask image. The tumor mask labels have been reviewed by clinical radiologists [2 on page 6-39].

Run this code to download the dataset from the MathWorks® website and unzip the downloaded folder.

```
zipFile = matlab.internal.examples.downloadSupportFile("image","data/Dataset_BUSI.zip");
filepath = fileparts(zipFile);
unzip(zipFile,filepath)
```

The `imageDir` folder contains the downloaded and unzipped dataset.

```
imageDir = fullfile(filepath,"Dataset_BUSI_with_GT");
```

Load Data

Create an `imageDatastore` object to read and manage the ultrasound image data. Label each image as normal, benign, or malignant according to the name of its folder.

```
imds = imageDatastore(imageDir,IncludeSubfolders=true,LabelSource="foldernames");
```

Remove files whose names contain "mask" to remove label images from the datastore. The image datastore now contains only the grayscale ultrasound images.

```
imds = subset(imds,find(~contains(imds.Files,"mask")));
```

Create a `pixelLabelDatastore` (Computer Vision Toolbox) object to store the labels.

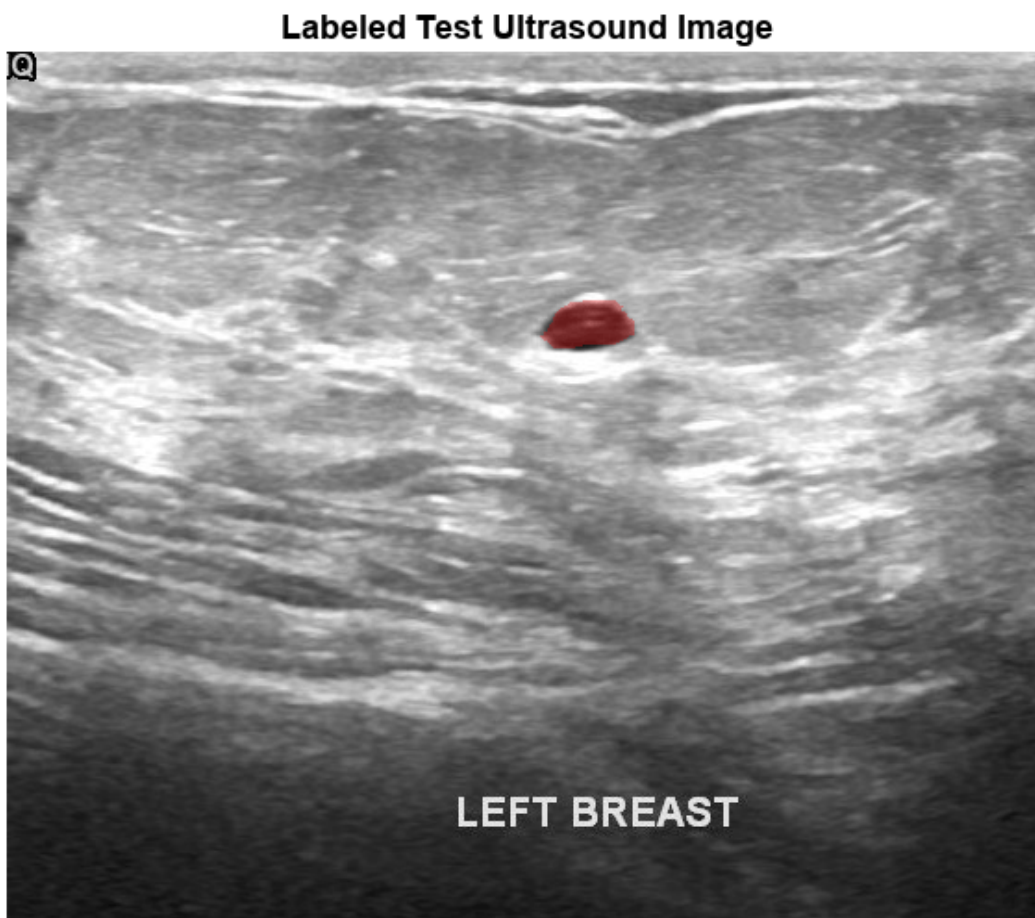
```
classNames = ["tumor", "background"];  
labelIDs = [1 0];  
numClasses = numel(classNames);  
pxds = pixelLabelDatastore(imageDir, classNames, labelIDs, IncludeSubfolders=true);
```

Include only the subset of files whose names contain `"_mask.png"` in the datastore. The pixel label datastore now contains only the tumor mask images.

```
pxds = subset(pxds, contains(pxds.Files, "_mask.png"));
```

Preview one image with a tumor mask overlay.

```
testImage = preview(imds);  
mask = preview(pxds);  
B = labeloverlay(testImage, mask, Transparency=0.7, IncludedLabels="tumor", ...  
    Colormap="hsv");  
imshow(B)  
title("Labeled Test Ultrasound Image")
```



Combine the image datastore and the pixel label datastore to create a `CombinedDatastore` object.

```
dsCombined = combine(imds,pxds);
```

Prepare Data for Training

Partition Data into Training, Validation, and Test Sets

Split the combined datastore into data sets for training, validation, and testing. Allocate 80% of the data for training, 10% for validation, and the remaining 10% for testing. Determine the indices to include in each set by using the `splitLabels` (Computer Vision Toolbox) function. To exclude images in the `normal` class without tumor images, use the image datastore labels as input and set the `Exclude` name-value argument to `"normal"`.

```
idxSet = splitLabels(imds.Labels,[0.8,0.1],"randomized",Exclude="normal");
dsTrain = subset(dsCombined,idxSet{1});
dsVal = subset(dsCombined,idxSet{2});
dsTest = subset(dsCombined,idxSet{3});
```

Augment Training and Validation Data

Augment the training and validation data by using the `transform` function with custom preprocessing operations specified by the `transformBreastTumorImageAndLabels` helper function. The helper function is attached to the example as a supporting file. The `transformBreastTumorImageAndLabels` function performs these operations:

- 1 Convert the ultrasound images from RGB to grayscale.
- 2 Augment the intensity of the grayscale images by using the `jitterIntensity` function.
- 3 Resize the images to 256-by-256 pixels.

```
tdsTrain = transform(dsTrain,@transformBreastTumorImageAndLabels,IncludeInfo=true);
tdsVal = transform(dsVal,@transformBreastTumorImageAndLabels,IncludeInfo=true);
```

Define Network Architecture

This example uses the DeepLab v3+ network. DeepLab v3+ consists of a series of convolution layers with a skip connection, one maxpool layer, and one averagepool layer. The network also has a batch normalization layer before each ReLU layer.

Create a DeepLab v3+ network based on ResNet-50 by using the `deeplabv3plusLayers` (Computer Vision Toolbox) function. Setting the base network as ResNet-50 requires the Deep Learning Toolbox™ Model for ResNet-50 Network support package. If this support package is not installed, then the function provides a download link.

Define the input size of the network as 256-by-256-by-3. Specify the number of classes as two for background and tumor.

```
imageSize = [256 256 3];
lgraph = deeplabv3plusLayers(imageSize,numClasses,"resnet50");
```

Because the preprocessed ultrasound images are grayscale, replace the original input layer with a 256-by-256 input layer.

```
newInputLayer = imageInputLayer(imageSize(1:2),Name="newInputLayer");
lgraph = replaceLayer(lgraph,lgraph.Layers(1).Name,newInputLayer);
```

Replace the first 2-D convolution layer with a new 2-D convolution layer to match the size of the new input layer.

```
newConvLayer = convolution2dLayer([7 7],64,Stride=2,Padding=[3 3 3 3],Name="newConv1");
lgraph = replaceLayer(lgraph,lgraph.Layers(2).Name,newConvLayer);
```

To better segment smaller tumor regions and reduce the influence of larger background regions, use a custom Tversky pixel classification layer. For more details about using a custom Tversky layer, see “Define Custom Pixel Classification Layer with Tversky Loss” (Deep Learning Toolbox). Replace the pixel classification layer with the Tversky pixel classification layer. The `alpha` and `beta` weighting factors control the contribution of false positives and false negatives, respectively, to the loss function. The `alpha` and `beta` values used in this example were selected using trial and error for the target data set. Generally, specifying the `beta` value greater than the `alpha` value is useful for training images with small objects and large background regions.

```
alpha = 0.01;
beta = 0.99;
pxLayer = tverskyPixelClassificationLayer("tverskyLoss",alpha,beta);
lgraph = replaceLayer(lgraph,"classification",pxLayer);
```

Alternatively, you can modify the DeepLab v3+ network by using the Deep Network Designer (Deep Learning Toolbox) from Deep Learning Toolbox.

Use the **Analyze** tool in the Deep Network Designer (Deep Learning Toolbox) to analyze the DeepLab v3+ network.

```
deepNetworkDesigner(lgraph)
```

Specify Training Options

Train the network using the `adam` optimization solver. Specify the hyperparameter settings using the `trainingOptions` (Deep Learning Toolbox) function. Set the learning rate to `1e-3` over the span of training. You can experiment with the mini-batch size based on your GPU memory. Batch normalization layers are less effective for smaller values of the mini-batch size. Tune the initial learning rate based on the mini-batch size.

```
options = trainingOptions("adam", ...
    ExecutionEnvironment="gpu", ...
    InitialLearnRate=1e-3, ...
    ValidationData=tdsVal, ...
    MaxEpochs=300, ...
    MiniBatchSize=16, ...
    VerboseFrequency=20, ...
    Plots="training-progress");
```

Train Network

To train the network, set the `doTraining` variable to `true`. Train the model using the `trainNetwork` (Deep Learning Toolbox) function.

Train on a GPU if one is available. Using a GPU requires Parallel Computing Toolbox™ and a CUDA® enabled NVIDIA® GPU. For more information, see “GPU Computing Requirements” (Parallel Computing Toolbox). Training takes about four hours on a single-GPU system with an NVIDIA™ Titan Xp GPU and can take longer depending on your GPU hardware.

```
doTraining = ;
if doTraining
```

```

    [trainedNet,info] = trainNetwork(tdsTrain,lgraph,options);
    modelDateTime = string(datetime("now",Format="yyyy-MM-dd-HH-mm-ss"));
    save("breastTumorDeepLabv3-"+modelDateTime+".mat","trainedNet");
end

```

Predict Using New Data

Preprocess Test Data

Prepare the test data by using the `transform` function with custom preprocessing operations specified by the `transformBreastTumorImageResize` helper function. This helper function is attached to the example as a supporting file. The `transformBreastTumorImageResize` function converts images from RGB to grayscale and resizes the images to 256-by-256 pixels.

```
dsTest = transform(dsTest,@transformBreastTumorImageResize,IncludeInfo=true);
```

Segment Test Data

Use the trained network for semantic segmentation of the test data set.

```
pxdsResults = semanticseg(dsTest,trainedNet,Verbose=true);
```

```
Running semantic segmentation network
-----
* Processed 65 images.
```

Evaluate Segmentation Accuracy

Evaluate the network-predicted segmentation results against the ground truth pixel label tumor masks.

```
metrics = evaluateSemanticSegmentation(pxdsResults,dsTest,Verbose=true);
```

```
Evaluating semantic segmentation results
-----
* Selected metrics: global accuracy, class accuracy, IoU, weighted IoU, BF score.
* Processed 65 images.
* Finalizing... Done.
* Data set metrics:
```

<u>GlobalAccuracy</u>	<u>MeanAccuracy</u>	<u>MeanIoU</u>	<u>WeightedIoU</u>	<u>MeanBFScore</u>
0.97588	0.96236	0.86931	0.9565	0.57667

Measure the segmentation accuracy using the `evaluateBreastTumorDiceAccuracy` helper function. This helper function computes the Dice index between the predicted and ground truth segmentations using the `dice` function. The helper function is attached to the example as a supporting file.

```
[diceTumor,diceBackground,numTestImgs] = evaluateBreastTumorDiceAccuracy(pxdsResults,dsTest);
```

Calculate the average Dice index across the set of test images.

```
disp("Average Dice score of background across "+num2str(numTestImgs)+ ...
    " test images = "+num2str(mean(diceBackground)))
```

```
Average Dice score of background across 65 test images = 0.98581
```

```
disp("Average Dice score of tumor across "+num2str(numTestImgs)+ ...
    " test images = "+num2str(mean(diceTumor)))
```

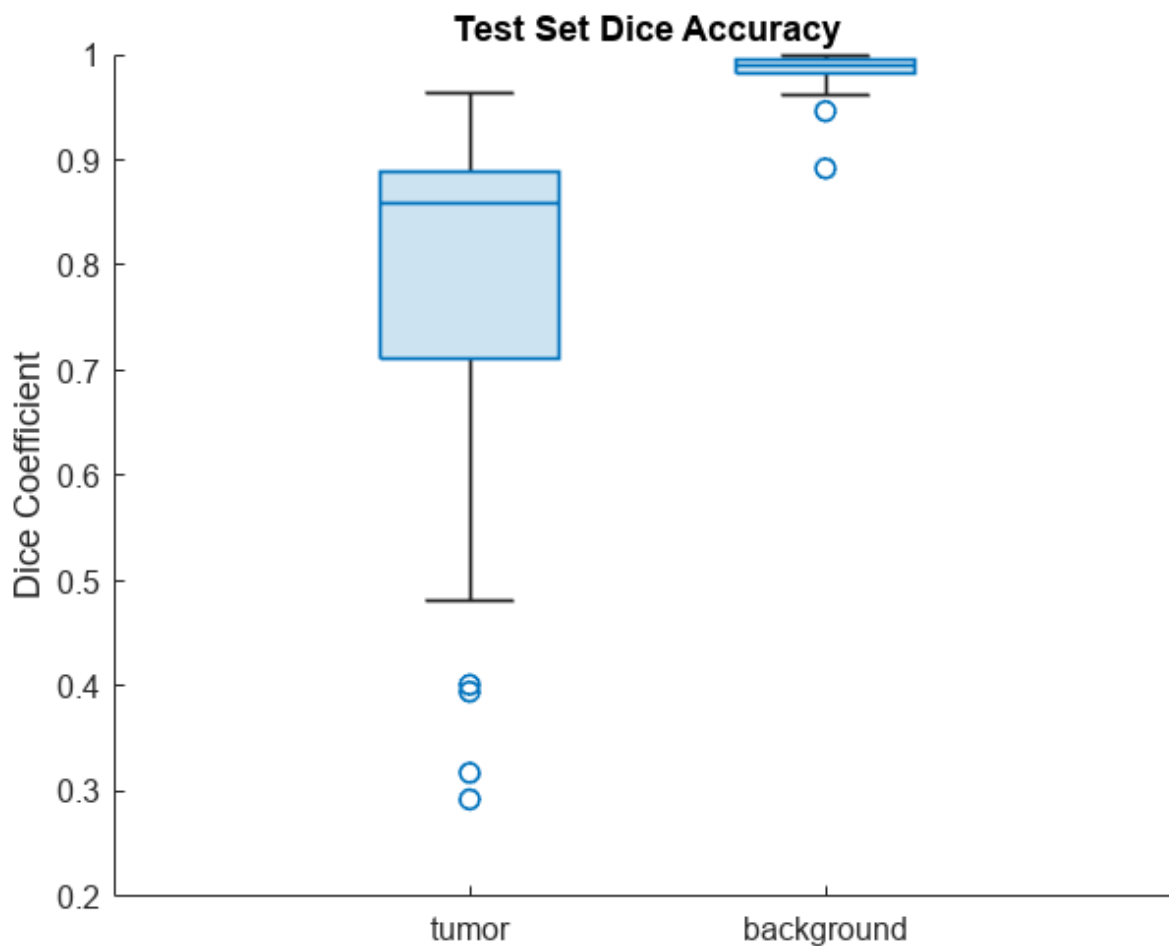
Average Dice score of tumor across 65 test images = 0.78588

```
disp("Median Dice score of tumor across "+num2str(numTestImgs)+ ...
    " test images = "+num2str(median(diceTumor)))
```

Median Dice score of tumor across 65 test images = 0.85888

Visualize statistics about the Dice scores as a box chart. The middle blue line in the plot shows the median Dice index. The upper and lower bounds of the blue box indicate the 25th and 75th percentiles, respectively. Black whiskers extend to the most extreme data points that are not outliers.

```
figure
boxchart([diceTumor diceBackground])
title("Test Set Dice Accuracy")
xticklabels(classNamees)
ylabel("Dice Coefficient")
```



References

[1] Salehi, Seyed Sadegh Mohseni, Deniz Erdogmus, and Ali Gholipour. "Tversky Loss Function for Image Segmentation Using 3D Fully Convolutional Deep Networks." In *Machine Learning in Medical Imaging*, edited by Qian Wang, Yinghuan Shi, Heung-Il Suk, and Kenji Suzuki, 10541:379-87. Cham: Springer International Publishing, 2017. https://doi.org/10.1007/978-3-319-67389-9_44.

[2] Al-Dhabyani, Walid, Mohammed Gomaa, Hussien Khaled, and Aly Fahmy. "Dataset of Breast Ultrasound Images." *Data in Brief* 28 (February 2020): 104863. <https://doi.org/10.1016/j.dib.2019.104863>.

See Also

`imageDatastore` | `pixelLabelDatastore` | `subset` | `combine` | `transform` | `deeplabv3plusLayers` | `semanticseg` | `evaluateSemanticSegmentation`

Related Examples

- "Segment Lungs from CT Scan Using Pretrained Neural Network" on page 6-14
- "Brain MRI Segmentation Using Pretrained 3-D U-Net Network" on page 6-24
- "3-D Brain Tumor Segmentation Using Deep Learning"

More About

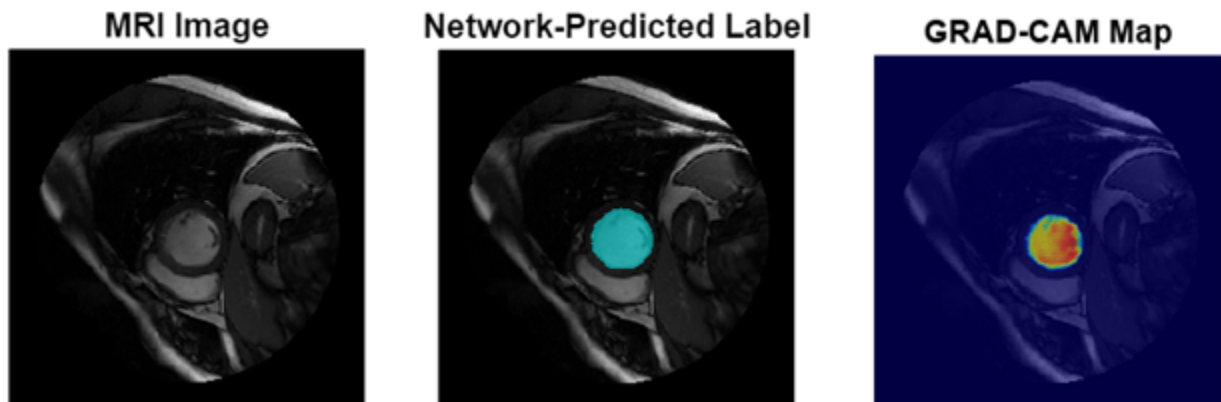
- "Datastores for Deep Learning" (Deep Learning Toolbox)
- "Define Custom Pixel Classification Layer with Tversky Loss" (Computer Vision Toolbox)
- "GPU Computing Requirements" (Parallel Computing Toolbox)

Cardiac Left Ventricle Segmentation from Cine-MRI Images Using U-Net Network

This example shows how to perform semantic segmentation of the left ventricle from 2-D cardiac MRI images using U-Net.

Semantic segmentation associates each pixel in an image with a class label. Segmentation of cardiac MRI images is useful for detecting abnormalities in heart structure and function. A common challenge of medical image segmentation is *class imbalance*, meaning the region of interest is small relative to the image background. Therefore, the training images contain many more background pixels than labeled pixels, which can limit classification accuracy. In this example, you address class imbalance by using a generalized Dice loss function [1 on page 6-51]. You also use the gradient-weighted class activation mapping (Grad-CAM) deep learning explainability technique to determine which regions of an image are important for the pixel classification decision.

This figure shows an example of a cine-MRI image before segmentation, the network-predicted segmentation map, and the corresponding Grad-CAM map.



Load Pretrained Network

Download the pretrained U-Net network by using the `downloadTrainedNetwork` helper function. The helper function is attached to this example as a supporting file. You can use this pretrained network to run the example without training the network.

```
exampleDir = fullfile(tempdir, "cardiacMR");
if ~isfolder(exampleDir)
    mkdir(exampleDir);
end

trainedNetworkURL = "https://ssd.mathworks.com/supportfiles" + ...
    "/medical/pretrainedLeftVentricleSegmentation.zip";
downloadTrainedNetwork(trainedNetworkURL, exampleDir);
```

Load the network.

```
data = load(fullfile(exampleDir, "pretrainedLeftVentricleSegmentationModel.mat"));
trainedNet = data.trainedNet;
```

Perform Semantic Segmentation

Use the pretrained network to predict the left ventricle segmentation mask for a test image.

Download Data Set

This example uses a subset of the Sunnybrook Cardiac Data data set [2 on page 6-51,3 on page 6-52]. The subset consists of 45 cine-MRI images and their corresponding ground truth label images. The MRI images were acquired from multiple patients with various cardiac pathologies. The ground truth label images were manually drawn by experts [2]. The MRI images are in the DICOM file format and the label images are in the PNG file format. The total size of the subset of data is ~105 MB.

Download the data set from the MathWorks® website and unzip the downloaded folder.

```
zipFile = matlab.internal.examples.downloadSupportFile("medical","CardiacMRI.zip");
filepath = fileparts(zipFile);
unzip(zipFile,filepath)
```

The `imageDir` folder contains the downloaded and unzipped data set.

```
imageDir = fullfile(filepath,"Cardiac MRI");
```

Predict Left Ventricle Mask

Read an image from the data set and preprocess the image by using the `preprocessImage` on page 6-50 helper function, which is defined at the end of this example. The helper function resizes MRI images to the input size of the network and converts them from grayscale to three-channel images.

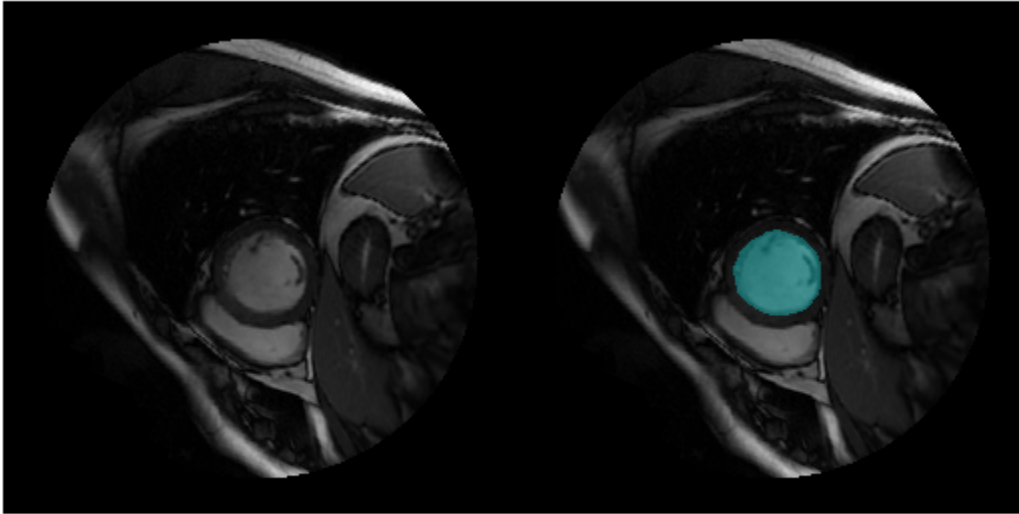
```
testImg = dicomread(fullfile(imageDir,"images","SC-HF-I-01","SC-HF-I-01_rawdcm_099.dcm"));
trainingSize = [256 256 3];
data = preprocessImage(testImg,trainingSize);
testImg = data{1};
```

Predict the left ventricle segmentation mask for the test image using the pretrained network by using the `semanticseg` (Computer Vision Toolbox) function.

```
segmentedImg = semanticseg(testImg,trainedNet);
```

Display the test image and an overlay with the predicted mask as a montage.

```
overlayImg = labeloverlay(mat2gray(testImg),segmentedImg, ...
    Transparency=0.7, ...
    IncludedLabels="LeftVentricle");
imshowpair(mat2gray(testImg),overlayImg,"montage");
```



Prepare Data for Training

Create an `imageDatastore` object to read and manage the MRI images.

```
dataFolder = fullfile(imageDir,"images");
imds = imageDatastore(dataFolder,...
    IncludeSubfolders=true,...
    FileExtensions=".dcm",...
    ReadFcn=@dicomread);
```

Create a `pixelLabelDatastore` (Computer Vision Toolbox) object to read and manage the label images.

```
labelFolder = fullfile(imageDir,"labels");
classNames = ["Background","LeftVentricle"];
pixIDs = [0,1];

pxds = pixelLabelDatastore(labelFolder,classNames,pixIDs,...
    IncludeSubfolders=true,...
    FileExtensions=".png");
```

Preprocess the data by using the `transform` function with custom operations specified by the `preprocessImage` on page 6-50 helper function, which is defined at the end of this example. The helper function resizes the MRI images to the input size of the network and converts them from grayscale to three-channel images.

```
timds = transform(imds,@(img) preprocessImage(img,trainingSize));
```

Preprocess the label images by using the `transform` function with custom operations specified by the `preprocessLabels` on page 6-51 helper function, which is defined at the end of this example. The helper function resizes the label images to the input size of the network.

```
tpxds = transform(pxds,@(img) preprocessLabels(img,trainingSize));
```

Combine the transformed image and pixel label datastores to create a CombinedDatastore object.

```
combinedDS = combine(timds, tpxds);
```

Partition Data for Training, Validation, and Testing

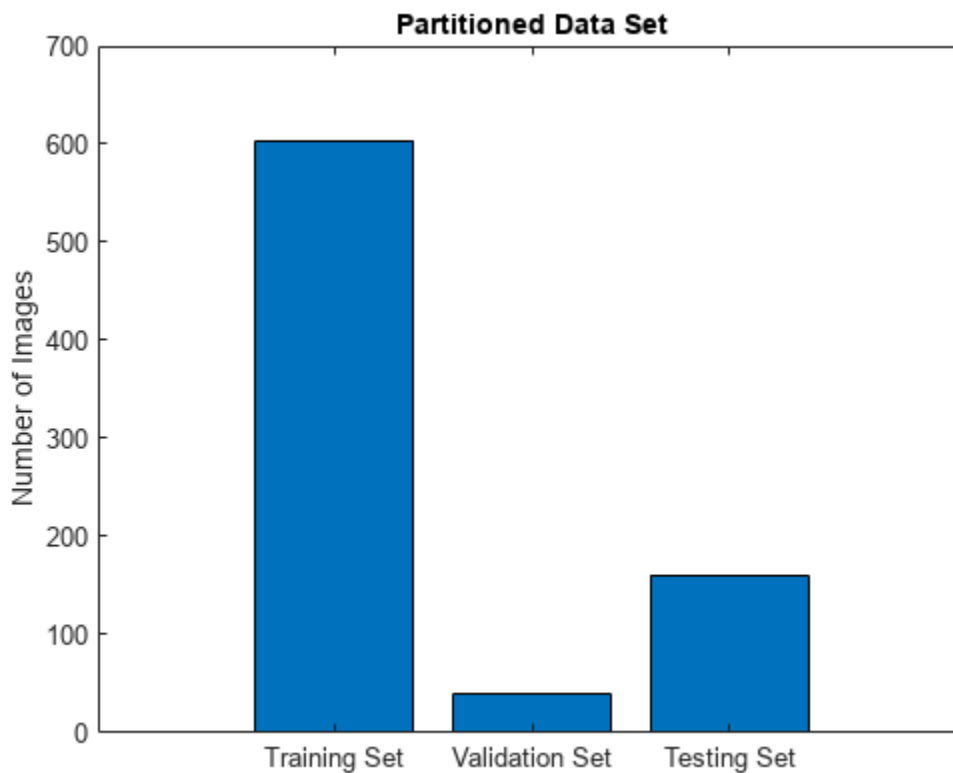
Split the combined datastore into data sets for training, validation, and testing. Allocate 75% of the data for training, 5% for validation, and the remaining 20% for testing.

```
numImages = numel(imds.Files);  
numTrain = round(0.75*numImages);  
numVal = round(0.05*numImages);  
numTest = round(0.2*numImages);
```

```
shuffledIndices = randperm(numImages);  
dsTrain = subset(combinedDS, shuffledIndices(1:numTrain));  
dsVal = subset(combinedDS, shuffledIndices(numTrain+1:numTrain+numVal));  
dsTest = subset(combinedDS, shuffledIndices(numTrain+numVal+1:end));
```

Visualize the number of images in the training, validation, and testing subsets.

```
figure  
bar([numTrain, numVal, numTest])  
title("Partitioned Data Set")  
xticklabels({"Training Set", "Validation Set", "Testing Set"})  
ylabel("Number of Images")
```



Augment Training Data

Augment the training data by using the `transform` function with custom operations specified by the `augmentDataForLVSegmentation` on page 6-51 helper function, which is defined at the end of this example. The helper function applies random rotations, translations, and reflections to the MRI images and corresponding ground truth labels.

```
dsTrain = transform(dsTrain,@(data) augmentDataForLVSegmentation(data));
```

Measure Label Imbalance

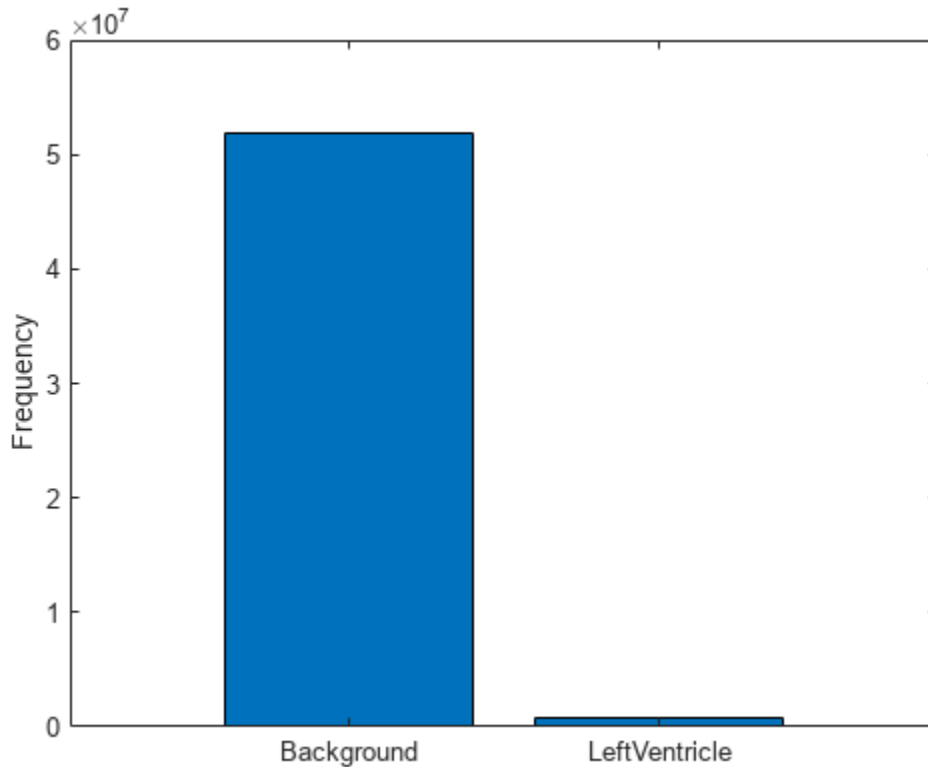
To measure the distribution of class labels in the data set, use the `countEachLabel` function to count the background pixels and the labeled ventricle pixels.

```
pixelLabelCount = countEachLabel(pxds)
```

```
pixelLabelCount=2x3 table
      Name      PixelCount      ImagePixelCount
-----
{'Background' } 5.1901e+07      5.2756e+07
{'LeftVentricle'} 8.5594e+05      5.2756e+07
```

Visualize the labels by class. The image contains many more background pixels than labeled ventricle pixels. The label imbalance can bias the training of the network. You address this imbalance when you design the network.

```
figure
bar(categorical(pixelLabelCount.Name),pixelLabelCount.PixelCount)
ylabel("Frequency")
```



Define Network Architecture

This example uses a U-Net network for semantic segmentation. Create a U-Net network with an input size of 256-by-256-by-3 that classifies pixels into two categories corresponding to the background and left ventricle.

```
numClasses = length(classNames);
net = unetLayers(trainingSize,numClasses);
```

Replace the input network layer with an `imageInputLayer` (Deep Learning Toolbox) object that normalizes image values between 0 and 1000 to the range [0, 1]. Values less than 0 are set to 0 and values greater than 1000 are set to 1000.

```
inputlayer = imageInputLayer(trainingSize, ...
    Normalization="rescale-zero-one", ...
    Min=0, ...
    Max=1000, ...
    Name="input");
net = replaceLayer(net,net.Layers(1).Name,inputlayer);
```

To address the class imbalance between the smaller ventricle regions and larger background, this example uses a `dicePixelClassificationLayer` (Computer Vision Toolbox) object. Replace the pixel classification layer with the Dice pixel classification layer.

```
pxLayer = dicePixelClassificationLayer(Name="labels");
net = replaceLayer(net,net.Layers(end).Name,pxLayer);
```

Specify Training Options

Specify the training options by using the `trainingOptions` (Deep Learning Toolbox) function. Train the network using the `adam` optimization solver. Set the learning rate to 0.001 over the span of training. You can experiment with the mini-batch size based on your GPU memory. Batch normalization layers are less effective for smaller values of the mini-batch size. Tune the initial learning rate based on the mini-batch size.

```
options = trainingOptions("adam", ...
    InitialLearnRate=0.0002, ...
    GradientDecayFactor=0.999, ...
    L2Regularization=0.0005, ...
    MaxEpochs=100, ...
    MiniBatchSize=32, ...
    Shuffle="every-epoch", ...
    Verbose=false, ...
    VerboseFrequency=100, ...
    ValidationData=dsVal, ...
    Plots="training-progress", ...
    ExecutionEnvironment="auto", ...
    ResetInputNormalization=false);
```

Train Network

To train the network, set the `doTraining` variable to `true`. Train the network by using the `trainNetwork` (Deep Learning Toolbox) function.

Train on a GPU if one is available. Using a GPU requires a Parallel Computing Toolbox™ license and a CUDA®-enabled NVIDIA® GPU.

```
doTraining = ;
if doTraining
    trainedNet = trainNetwork(dsTrain,net,options);
    modelDateTime = string(datetime("now",Format="yyyy-MM-dd-HH-mm-ss"));
    save(fullfile(exampleDir,"trainedLeftVentricleSegmentation-" ...
        +modelDateTime+".mat"),"trainedNet");
end
```

end

Test Network

Segment each image in the test data set by using the trained network.

```
resultsDir = fullfile(exampleDir,"Results");

if ~isfolder(resultsDir)
    mkdir(resultsDir)
end

pxdsResults = semanticseg(dsTest,trainedNet,...
    WriteLocation=resultsDir,...
    Verbose=true,...
    MiniBatchSize=1);
```

Running semantic segmentation network

```
-----
* Processed 161 images.
```


Evaluate Segmentation Metrics

Evaluate the network by calculating performance metrics using the `evaluateSemanticSegmentation` (Computer Vision Toolbox) function. The function computes metrics that compare the labels that the network predicts in `pxdsResults` to the ground truth labels in `pxdsTest`.

```
pxdsTest = dsTest.UnderlyingDatastores{2};
metrics = evaluateSemanticSegmentation(pxdsResults,pxdsTest);
```

```
Evaluating semantic segmentation results
```

```
-----
* Selected metrics: global accuracy, class accuracy, IoU, weighted IoU, BF score.
* Processed 161 images.
* Finalizing... Done.
* Data set metrics:
```

<u>GlobalAccuracy</u>	<u>MeanAccuracy</u>	<u>MeanIoU</u>	<u>WeightedIoU</u>	<u>MeanBFScore</u>
0.99829	0.97752	0.9539	0.99667	0.96747

View the metrics by class by querying the `ClassMetrics` property of `metrics`.

```
metrics.ClassMetrics
```

```
ans=2x3 table
```

	<u>Accuracy</u>	<u>IoU</u>	<u>MeanBFScore</u>
Background	0.99907	0.99826	0.995
LeftVentricle	0.95598	0.90953	0.93994

Evaluate Dice Score

Evaluate the segmentation accuracy by calculating the Dice score between the predicted and ground truth label images. For each test image, calculate the Dice score for the background label and the ventricle label by using the `dice` function.

```
reset(pxdsTest);
reset(pxdsResults);

diceScore = zeros(numTest,numClasses);
for idx = 1:numTest

    prediction = read(pxdsResults);
    groundTruth = read(pxdsTest);

    diceScore(idx,1) = dice(prediction{1}==classNames(1),groundTruth{1}==classNames(1));
    diceScore(idx,2) = dice(prediction{1}==classNames(2),groundTruth{1}==classNames(2));
end
```

Calculate the mean Dice score over all test images and report the mean values in a table.

```
meanDiceScore = mean(diceScore);
diceTable = array2table(meanDiceScore', ...
    VariableNames="Mean Dice Score", ...
    RowNames=classNames)
```

```

diceTable=2x1 table
                Mean Dice Score
-----
Background      0.99913
LeftVentricle   0.9282

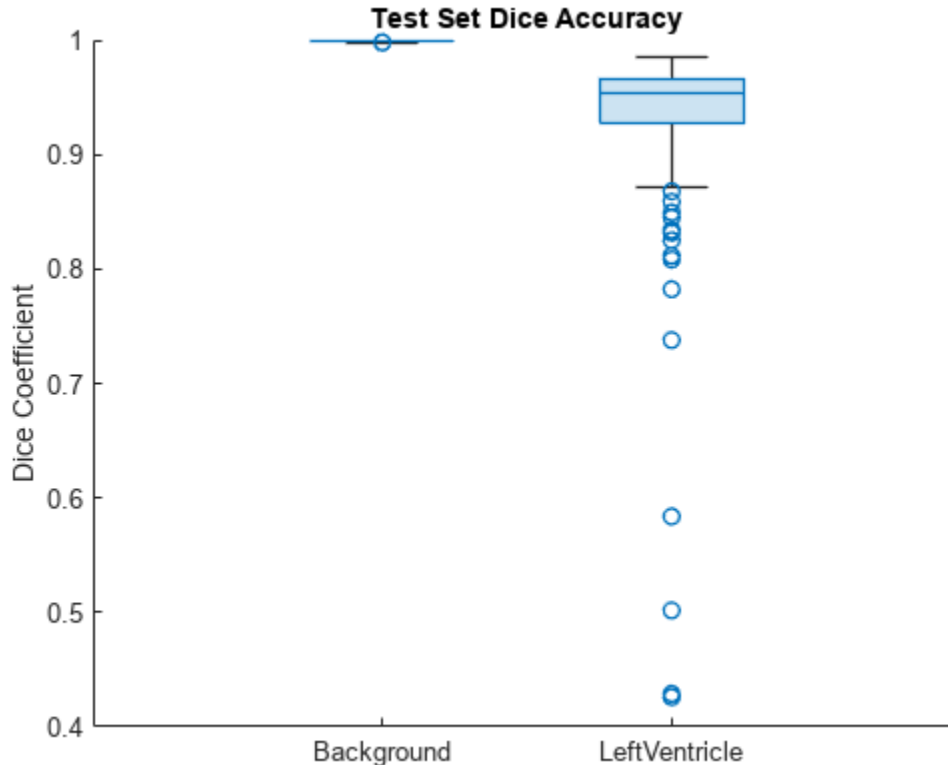
```

Visualize the Dice scores for each class as a box chart. The middle blue line in the plot shows the median Dice score. The upper and lower bounds of the blue box indicate the 25th and 75th percentiles, respectively. Black whiskers extend to the most extreme data points that are not outliers.

```

figure
boxchart(diceScore)
title("Test Set Dice Accuracy")
xticklabels(classNames)
ylabel("Dice Coefficient")

```



Explainability

By using explainability methods like Grad-CAM, you can see which areas of an input image the network uses to make its pixel classifications. Use Grad-CAM to show which areas of a test MRI image the network uses to segment the left ventricle.

Load an image from the test data set and preprocess it using the same operations you use to preprocess the training data. The `preprocessImage` on page 6-50 helper function is defined at the end of this example.

```
testImg = dicomread(fullfile(imageDir,"images","SC-HF-I-01","SC-HF-I-01_rawdcm_099.dcm"));
data = preprocessImage(testImg,trainingSize);
testImg = data{1};
```

Load the corresponding ground truth label image and preprocess it using the same operations you use to preprocess the training data. The `preprocessLabels` on page 6-51 function is defined at the end of this example.

```
testGroundTruth = imread(fullfile(imageDir,"labels","SC-HF-I-01","SC-HF-I-01gtmask0099.png"));
data = preprocessLabels({testGroundTruth}, trainingSize);
testGroundTruth = data{1};
```

Segment the test image using the trained network.

```
prediction = semanticseg(testImg,trainedNet);
```

To use Grad-CAM, you must select a feature layer from which to extract the feature map and a reduction layer from which to extract the output activations. Use `analyzeNetwork` (Deep Learning Toolbox) to find the layers to use with Grad-CAM. In this example, you use the final ReLU layer as the feature layer and the softmax layer as the reduction layer.

```
analyzeNetwork(trainedNet)
featureLayer = "Decoder-Stage-4-Conv-2";
reductionLayer = "Softmax-Layer";
```

Compute the Grad-CAM map for the test image by using the `gradCAM` (Deep Learning Toolbox) function.

```
gradCAMMap = gradCAM(trainedNet,testImg,classNames,...
    ReductionLayer=reductionLayer,...
    FeatureLayer=featureLayer);
```

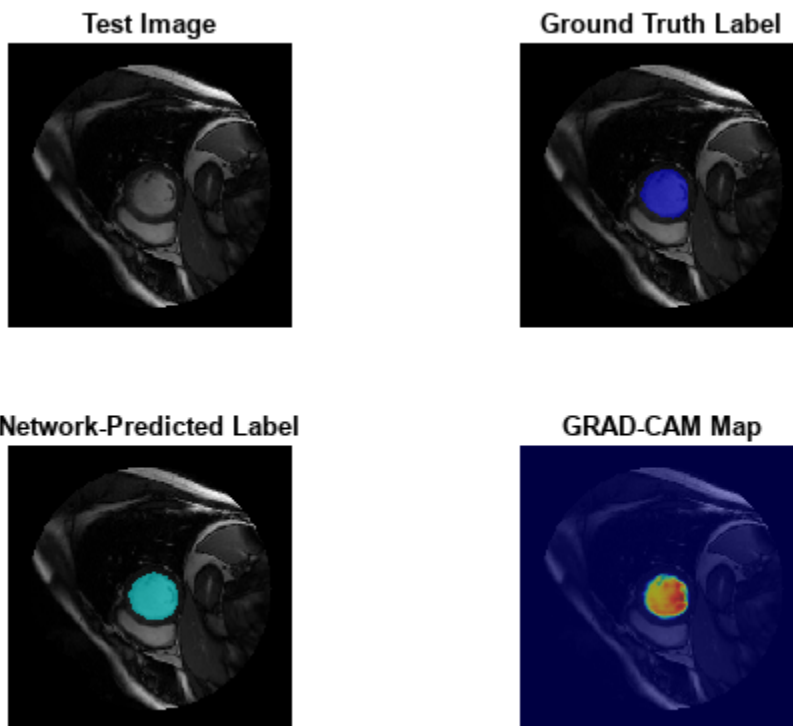
Visualize the test image, the ground truth labels, the network-predicted labels, and the Grad-CAM map for the ventricle. As expected, the area within the ground truth ventricle mask contributes most strongly to the network prediction of the ventricle label.

```
figure
tiledlayout(2,2)
nexttile
imshow(mat2gray(testImg))
title("Test Image")

nexttile
imshow(labeloverlay(mat2gray(testImg),testGroundTruth))
title("Ground Truth Label")

nexttile
imshow(labeloverlay(mat2gray(testImg),prediction,IncludedLabels="LeftVentricle"))
title("Network-Predicted Label")

nexttile
imshow(mat2gray(testImg))
hold on
imagesc(gradCAMMap(:,:,2),AlphaData=0.5)
title("GRAD-CAM Map")
colormap jet
```



Supporting Functions

The `preprocessImage` helper function preprocesses the MRI images using these steps:

- 1 Resize the input image to the target size of the network.
- 2 Convert grayscale images to three channel images.
- 3 Return the preprocessed image in a cell array.

```
function out = preprocessImage(img,targetSize)
% Copyright 2023 The MathWorks, Inc.

    targetSize = targetSize(1:2);
    img = imresize(img,targetSize);

    if size(img,3) == 1
        img = repmat(img,[1 1 3]);
    end

    out = {img};

end
```

The `preprocessLabels` helper function preprocesses label images using these steps:

- 1 Resize the input label image to the target size of the network. The function uses nearest neighbor interpolation so that the output is a binary image without partial decimal values.
- 2 Return the preprocessed image in a cell array.

```
function out = preprocessLabels(labels, targetSize)
% Copyright 2023 The MathWorks, Inc.

    targetSize = targetSize(1:2);
    labels = imresize(labels{1},targetSize,"nearest");

    out = {labels};

end
```

The `augmentDataForLVSegmentation` helper function randomly applies these augmentations to each input image and its corresponding label image. The function returns the output data in a cell array.

- Random rotation between 0 to 180 degrees.
- Random translation along the x- and y-axes of -10 to 10 pixels.
- Random reflection to flip the image in the x-axis.

```
function out = augmentDataForLVSegmentation(data)
% Copyright 2023 The MathWorks, Inc.

    img = data{1};
    labels = data{2};
    inputSize = size(img,[1 2]);

    tform = randomAffine2d(...
        Rotation=[-5 5],...
        XTranslation=[-10 10],...
        YTranslation=[-10 10]);

    sameAsInput = affineOutputView(inputSize,tform,BoundsStyle="sameAsInput");
    img = imwarp(img,tform,"linear",OutputView=sameAsInput);
    labels = imwarp(labels,tform,"nearest",OutputView=sameAsInput);

    out = {img,labels};

end
```

References

[1] Milletari, Fausto, Nassir Navab, and Seyed-Ahmad Ahmadi. "V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation." In *2016 Fourth International Conference on 3D Vision (3DV)*, 565–71. Stanford, CA, USA: IEEE, 2016. <https://doi.org/10.1109/3DV.2016.79>.

[2] Radau, Perry, Yingli Lu, Kim Connelly, Gideon Paul, Alexander J Dick, and Graham A Wright. "Evaluation Framework for Algorithms Segmenting Short Axis Cardiac MRI." *The MIDAS Journal*, July 9, 2009. <https://doi.org/10.54294/g80ruo>.

[3] "Sunnybrook Cardiac Data - Cardiac Atlas Project." Accessed January 10, 2023. <http://www.cardiacatlas.org/studies/sunnybrook-cardiac-data/>.

See Also

`imageDatastore` | `pixelLabelDatastore` | `subset` | `combine` | `transform` | `unetLayers` | `semanticseg` | `evaluateSemanticSegmentation` | `gradCAM`

Related Examples

- "Segment Lungs from CT Scan Using Pretrained Neural Network" on page 6-14
- "Brain MRI Segmentation Using Pretrained 3-D U-Net Network" on page 6-24
- "Breast Tumor Segmentation from Ultrasound Using Deep Learning" on page 6-32
- "3-D Brain Tumor Segmentation Using Deep Learning"

More About

- "Datastores for Deep Learning" (Deep Learning Toolbox)