

# Medical Imaging Toolbox™

User's Guide



# MATLAB®

R2022b



# How to Contact MathWorks



Latest news: [www.mathworks.com](http://www.mathworks.com)  
Sales and services: [www.mathworks.com/sales\\_and\\_services](http://www.mathworks.com/sales_and_services)  
User community: [www.mathworks.com/matlabcentral](http://www.mathworks.com/matlabcentral)  
Technical support: [www.mathworks.com/support/contact\\_us](http://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 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](http://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](http://www.mathworks.com/patents) for more information.

### **Revision History**

September 2022	Online only	New for Version 1.0 (Release 2022b)
----------------	-------------	-------------------------------------

## Getting Started

### 1

<b>Medical Imaging Toolbox Product Description</b> .....	<b>1-2</b>
<b>Read, Process, and Write 3-D Medical Images</b> .....	<b>1-3</b>
<b>Get Started with Medical Image Labeler</b> .....	<b>1-10</b>
Start New Labeling Session .....	<b>1-10</b>
Load Image Data .....	<b>1-11</b>
Create Label Definitions .....	<b>1-11</b>
Label Image .....	<b>1-12</b>
Export Labeling Results .....	<b>1-13</b>
How Medical Image Labeler Manages Ground Truth Labels .....	<b>1-14</b>
<b>Medical Image Coordinate Systems</b> .....	<b>1-15</b>
Patient Coordinate System .....	<b>1-15</b>
Intrinsic Coordinate System .....	<b>1-16</b>
<b>Introduction to Medical Imaging</b> .....	<b>1-18</b>
Common Medical Imaging Modalities .....	<b>1-18</b>
Typical Workflow for Medical Image Analysis .....	<b>1-20</b>

## Import, Export, and Spatial Referencing

### 2

<b>Read, Process, and View Ultrasound Data</b> .....	<b>2-2</b>
--	------------

## Display and Volume Rendering

### 3

<b>Visualize 3-D Medical Image Data Using Medical Image Labeler</b> .....	<b>3-2</b>
<b>Display 3-D Medical Image Data in Patient Coordinate System</b> .....	<b>3-8</b>
<b>Display Labeled Medical Image Volume in Patient Coordinates</b> .....	<b>3-12</b>
<b>Create STL Surface Model of Femur Bone for 3-D Printing</b> .....	<b>3-19</b>

## Image Preprocessing and Augmentation

### 4

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

## Medical Image Labeling

### 5

<b>Label 2-D Ultrasound Series Using Medical Image Labeler</b> .....	<b>5-2</b>
<b>Label 3-D Medical Image Using Medical Image Labeler</b> .....	<b>5-10</b>
<b>Collaborate on Multi-Labeler Medical Image Labeling Projects</b> .....	<b>5-19</b>
Create Label Definitions and Assign Data to Labelers (Project Manager) .....	<b>5-20</b>
Label Data and Publish Labels for Review (Labeler) .....	<b>5-21</b>
Inspect Labeled Images (Reviewer) .....	<b>5-23</b>
Export Ground Truth Data and Send to Project Manager (Labeler) .....	<b>5-23</b>
Collect, Merge, and Create Training Data (Project Manager) .....	<b>5-24</b>

## Medical Image Segmentation

### 6

<b>Create Datastores for Medical Image Semantic Segmentation</b> .....	<b>6-2</b>
Medical Image Ground Truth Data .....	<b>6-2</b>
Datastores for Semantic Segmentation .....	<b>6-3</b>
<b>Convert Ultrasound Image Series into Training Data for 2-D Semantic     Segmentation Network</b> .....	<b>6-5</b>
<b>Create Training Data for 3-D Medical Image Semantic Segmentation</b> ...	<b>6-9</b>
<b>Segment Lungs from CT Scan Using Pretrained Neural Network</b> .....	<b>6-14</b>
<b>Brain MRI Segmentation Using Pretrained 3-D U-Net Network</b> .....	<b>6-24</b>





# 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 1-10
- “Medical Image Coordinate Systems” on page 1-15
- “Introduction to Medical Imaging” on page 1-18

## **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 smoothing 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. This example also shows how you can use the properties and object functions of a `medicalVolume` object to work with image volumes in the intrinsic and patient coordinate systems.

### 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. 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");
```

### 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.

```
vol = 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 Center Slice in Intrinsic Coordinates

Extract the center transverse slice from the image volume and display it using intrinsic image coordinates.

Calculate the index of the center slice as half of the number of transverse slices.

```
sliceTransverse = round(medVol.NumTransverseSlices/2);
```

Extract the voxel data for the center slice by using the `extractSlice` object function. The function orients the extracted slice `imTransverse` to match the standard radiological orientation. For example, when you display an extracted transverse slice, the anterior direction always points toward the top of the image, and the left anatomical direction points toward the right side of the image.

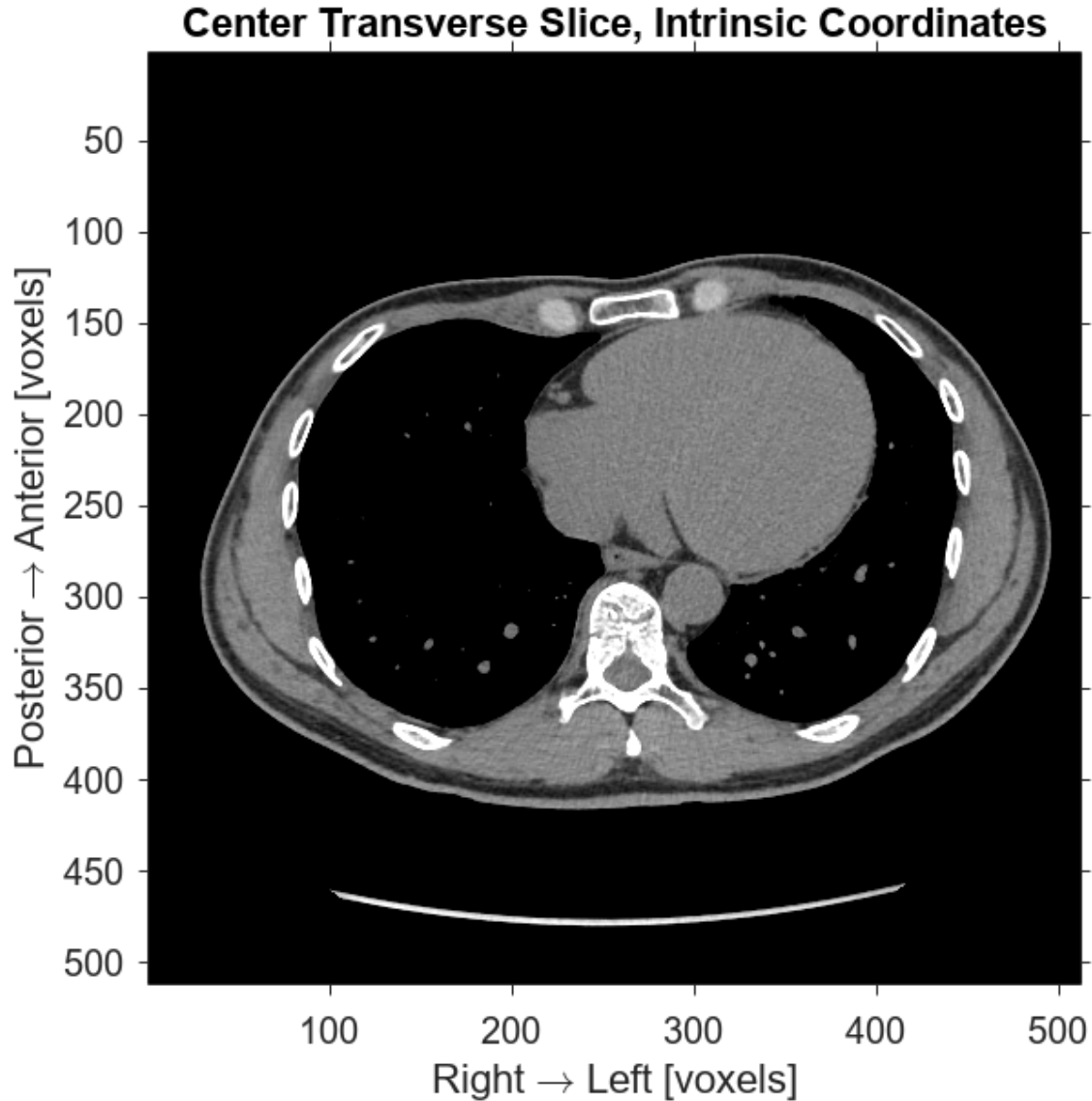
```
[imTransverse,positionTransverse,spacingsTransverse] = extractSlice(medVol,sliceTransverse,"transverse");
```

If the file metadata specifies a recommended display window to improve the contrast of the displayed image, the `WindowCenters` and `WindowWidths` properties of the `medicalVolume` object specify the center and width of the window, respectively. Calculate the minimum and maximum of the recommended display window for `medVol`.

```
windowMin = medVol.WindowCenters(1)-0.5*medVol.WindowWidths(1);  
windowMax = medVol.WindowCenters(1)+0.5*medVol.WindowWidths(1);
```

Display the extracted slice. By default, the `imshow` function displays the image in intrinsic coordinates, in voxel units.

```
imshow(imTransverse,[windowMin windowMax]);  
title("Center Transverse Slice, Intrinsic Coordinates")  
axis on  
xlabel("Right \rightarrow Left [voxels]")  
ylabel("Posterior \rightarrow Anterior [voxels]")
```



### Display Center Slice in Patient Coordinates

Display the center transverse slice in real-world units using patient coordinates.

Define the spatial referencing for the extracted slice. Get the limits of the slice in the patient coordinate system by using the `sliceLimits` object function.

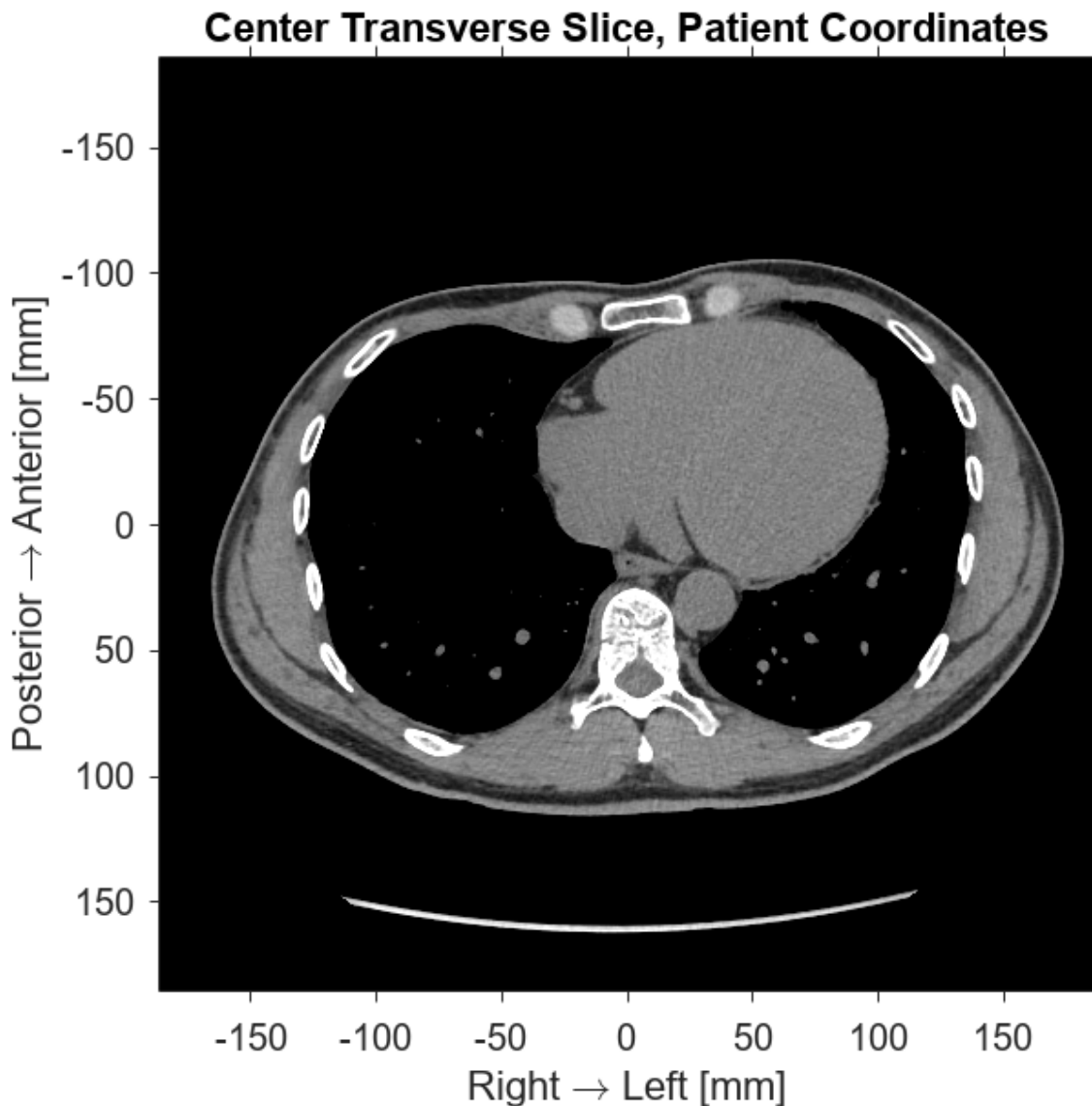
```
[XLim,YLim,ZLim] = sliceLimits(medVol,sliceTransverse,"transverse");
```

Create an `imref2d` object, `RTransverse`, that specifies the limits of the slice in patient coordinates. The `PlaneMapping` property of `medVol` maps between the transverse, coronal, and sagittal planes and the `xyz`-axes of the patient coordinate system. "transverse" is the third element of `PlaneMapping`, meaning transverse slices are normal to the `z`-axis and parallel to the `xy`-plane. Therefore, `XLim` and `YLim` define the transverse slice limits.

```
RTransverse = imref2d(size(imTransverse),XLim,YLim);
```

Display the transverse slice, specifying the `imref2d` object to plot the image in patient coordinates.

```
imshow(imTransverse,RTransverse,[windowMin windowMax])  
title("Center Transverse Slice, Patient Coordinates")  
xlabel("Right \rightarrow Left [mm]")  
ylabel("Posterior \rightarrow Anterior [mm]")
```



### 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.

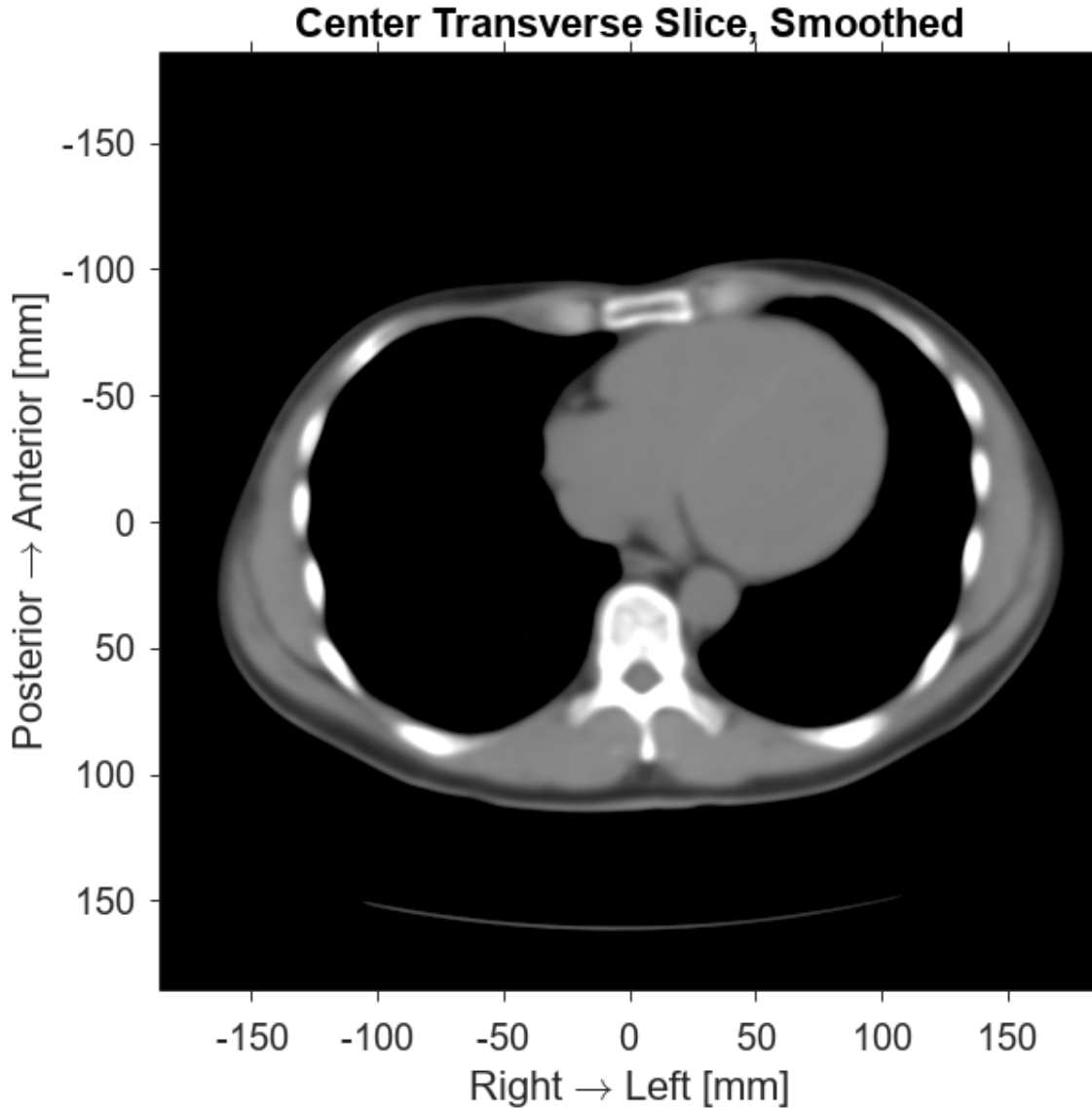
```
sigma = 2;  
volSmooth = imgaussfilt3(vol,sigma);
```

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 = volSmooth;
```

Extract and display the center slice of the smoothed voxel data. Use the same spatial referencing and display window information as the original object to plot the image in patient coordinates.

```
[imSmooth,position,spacings] = extractSlice(medVolSmooth,sliceTransverse,"transverse");  
figure  
imshow(imSmooth,RTransverse,[windowMin windowMax])  
title("Center Transverse Slice, Smoothed")  
xlabel("Right \rightarrow Left [mm]")  
ylabel("Posterior \rightarrow Anterior [mm]")
```



### 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`

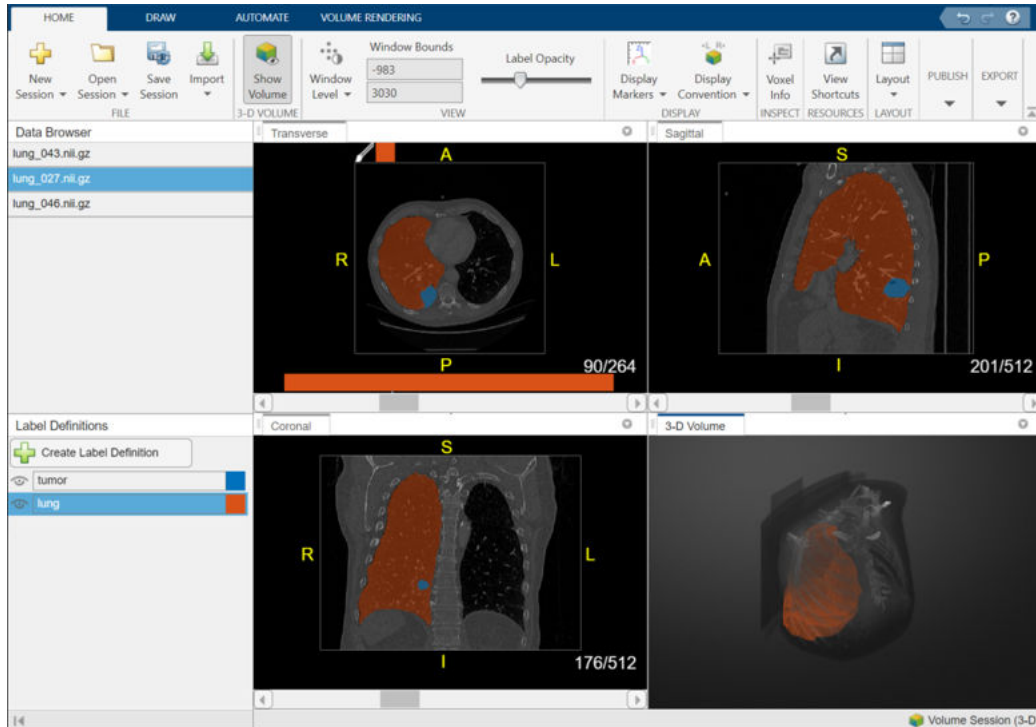
### **More About**

- “Medical Image Coordinate Systems” on page 1-15

## Get Started with Medical Image Labeler

The **Medical Image Labeler** app enables you to 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.

This tutorial provides an overview of the capabilities of the **Medical Image Labeler** and compares 2-D and 3-D image labeling.



### Start New Labeling Session

Manage labeling in the **Medical Image Labeler** using app sessions. You can create a volume session to label 3-D image volumes or an image session to label 2-D images or series of images related by time. Within one 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. Follow these steps to create a new labeling session.

- 1 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.
- 2 On the app toolstrip, click **New Session** and select **New Volume session (3-D)** or **New Image session (2-D)**.

You can also load a previous session by clicking **Open Session**.

- 3 Specify a session folder. You must specify the session folder upfront because the app automatically saves the label images as they are drawn. 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.



## Load Image Data

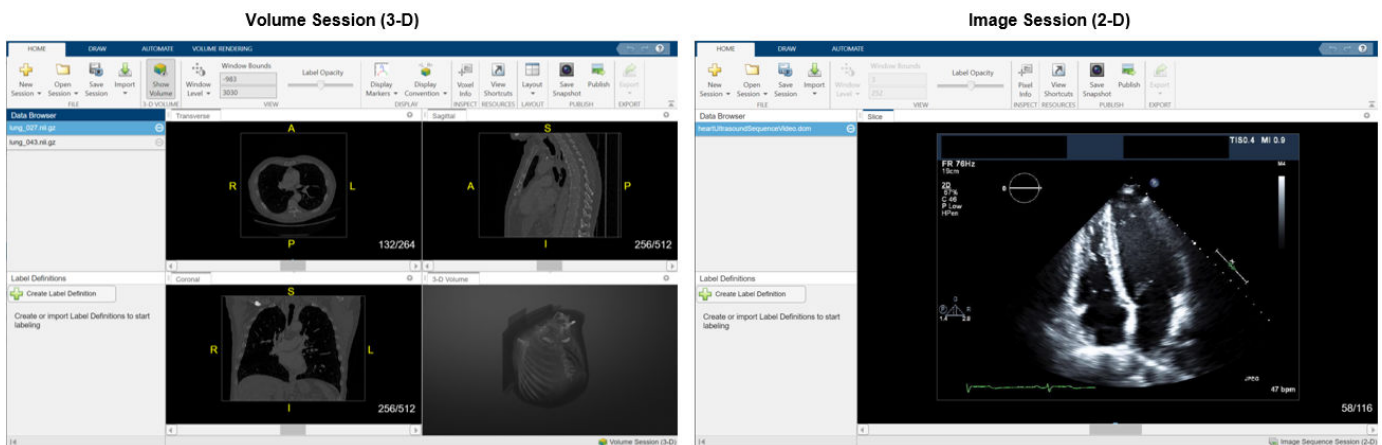
Load images from a file by clicking **Import Data** and then, under **Data**, selecting **From File**. The app supports loading these file formats:

- 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.

If you load a previous session, the app reopens the image data that was open when you closed the session.

Alternatively, you can load a `groundTruthMedical` object created programmatically or exported as a MAT file from a previous app session. You can import the `groundTruthMedical` object directly from a MAT file or from the MATLAB workspace. Importing a `groundTruthMedical` object loads the image data, label data, and label definitions stored in the object into the app. For example, import a `groundTruthMedical` object to load partially labeled data created outside the app or shared by a colleague. For more details about working on a multi-person labeling team, see “Collaborate on Multi-Labeler Medical Image Labeling Projects” on page 5-19.

The **Data Browser** pane lists all of the image files that are loaded in the app. The app displays 3-D image data in the panes for the **Transverse**, **Sagittal**, and **Coronal** slice planes and a **3-D Volume** pane. The app displays 2-D image data in the **Slice** pane.

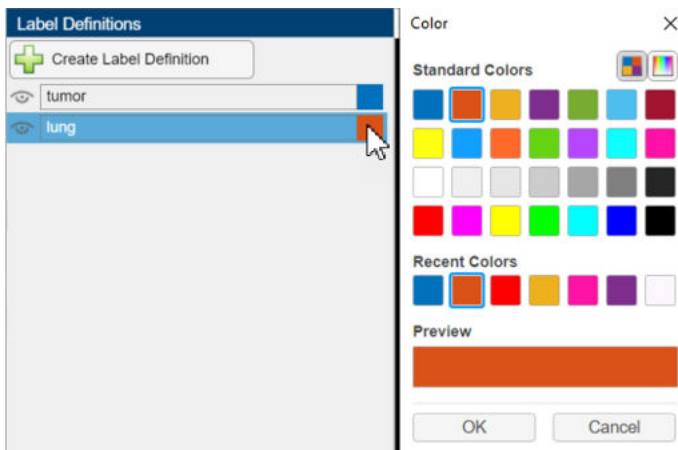


For more details about viewing image volumes in the app, see “Visualize 3-D Medical Image Data Using Medical Image Labeler” on page 3-2.

## Create Label Definitions

A label definition specifies the name, color, and order of each label assigned to the image.

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, you can 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 detail about creating a label definitions file programmatically, see the `LabelDefinitions` property of the `groundTruthMedical` object.

You must use the same label definitions across all images within a labeling session.

## Label Image

Label pixels using the tools in the **Draw** and **Automate** tabs.

### Manual Labeling

You can manually draw labels using the **Freehand**, **Assisted Freehand**, **Polygon**, and **Paintbrush** tools.

### Semi-Automated Labeling

You can 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
<b>Fill Region</b>	Flood fill label image or fill holes in label region.	

Tool	Description	Image
<b>Paint by Superpixels</b>	Manually paint within an adjustable-sized grid of pixels. To use this tool, first select <b>Paint Brush</b> and then click <b>Paint by Superpixels</b> . Each superpixel contains a cluster of similar intensity values. Adjust the <b>Superpixels Size</b> to change the size of the superpixel grid.	
<b>Trace Boundary</b>	Label connected regions that have similar intensity values. Select <b>Trace Boundary</b> in the app toolstrip, 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 <b>Threshold</b> 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.

### Labeling Examples

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

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

---

**Note** The actual pixel label data is stored 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.

Pixel label images created in an image session are saved in the `LabelData` folder as MAT files. The 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.

Voxel label images created in a volume session are saved in the `LabelData` folder as NIFTI files. The 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

- “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” on page 5-19

# 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.

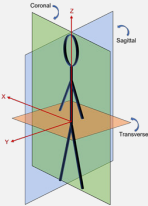



The intrinsic coordinate system defines the voxel space, and the patient coordinate system defines the anatomical space. To understand the position and orientation of intrinsic coordinates with respect to the patient, you must transform the data into the patient coordinate system. You can use the `intrinsicToWorldMapping` 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.

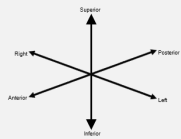
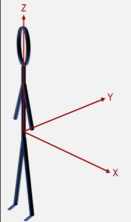
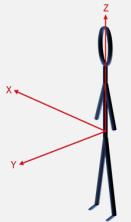
Anatomical Planes and Patient Axes	Sagittal Plane	Coronal Plane	Transverse Plane
	<ul style="list-style-type: none"> <li>• Defined by the <i>y</i>- and <i>z</i>-axes.</li> <li>• Divides the body into right and left segments.</li> </ul> 	<ul style="list-style-type: none"> <li>• Defined by the <i>x</i>-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

**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.

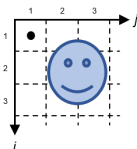
Patient Axes	DICOM	NIFTI
	<ul style="list-style-type: none"> <li>• x-axis values increase from right to left</li> <li>• y-axis values increase from anterior to posterior</li> <li>• z-axis values increase from inferior to superior</li> </ul>	<ul style="list-style-type: none"> <li>• x-axis values increase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

## Intrinsic Coordinate System

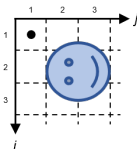
The intrinsic coordinate system describes the spatial dimensions of the patient coordinate system. Intrinsic coordinates are in units of voxels, while patient coordinates have real-world dimensions and are usually in units of millimeters. You can obtain pixel dimensions from the `PixelSpacing` property 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 3-D Medical Image Data in Patient Coordinate System” on page 3-8

## Introduction to Medical Imaging

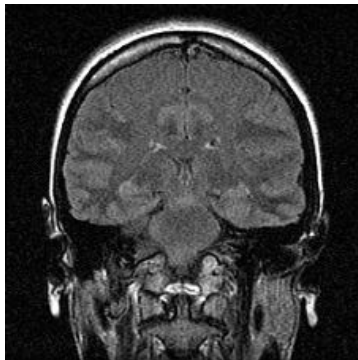
Medical imaging is the acquisition and processing of images of the human body for clinical applications. You can use medical image processing to improve the quality of medical images, for diagnosis of medical conditions, for surgical planning, or for research. Medical imaging enables the detailed, yet 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.

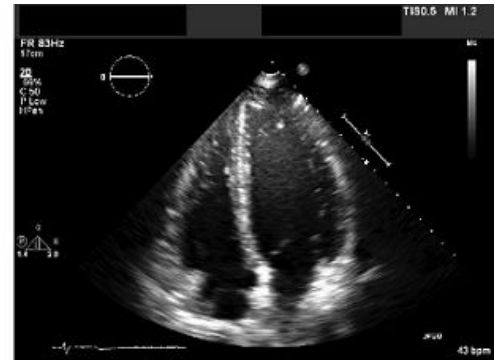
### Common Medical Imaging Modalities



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.

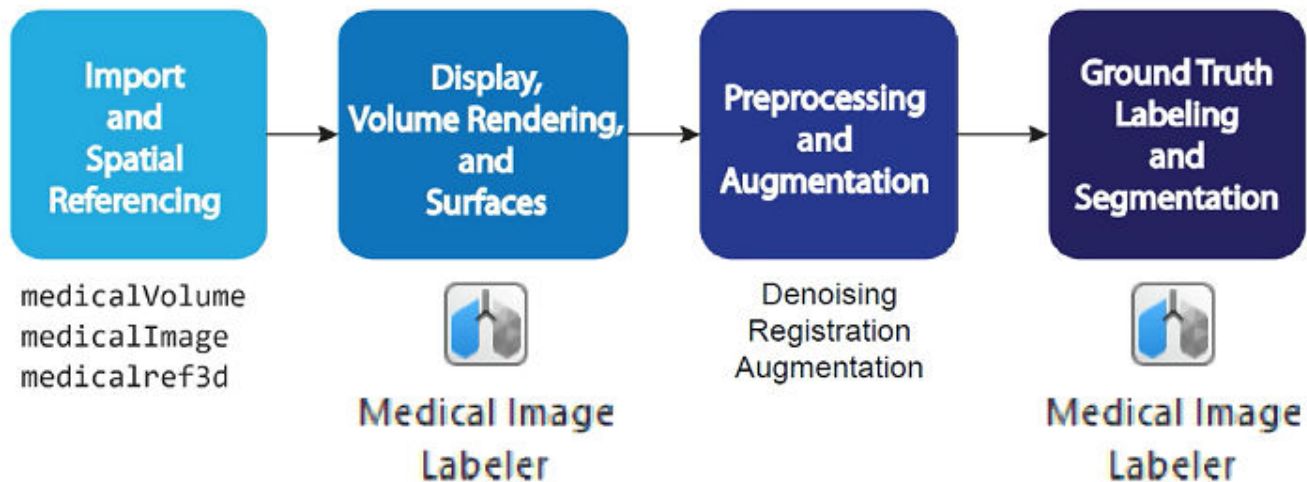
Mathematically, an ultrasound image is derived from the reflected ultrasound waves. The final output of the ultrasound scan is a sequence of 2-D images in the spatial domain. You can use a `medicalImage` object to store the pixel data and metadata for an ultrasound image sequence. Note that the emitted 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 3-2.

### **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 1-10.



# **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

The `medicalImage` object stores the image data dimensions in the order *spatial-spatial-frames-channel*, where *spatial* indicates the spatial dimensions of each frame, *frames* indicates the number of frames in the image series, and *channel* refers to the number of channels in each frame. To be compatible with the `montage` function, use the `permute` function to reorder the dimensions to *spatial-spatial-channel-frames*.

```
pixels = permute(medImg.Pixels,[1 2 4 3]);
```

Display the image frames side-by-side as a montage.

```
montage(pixels)
```



### Display Ultrasound Series as Video

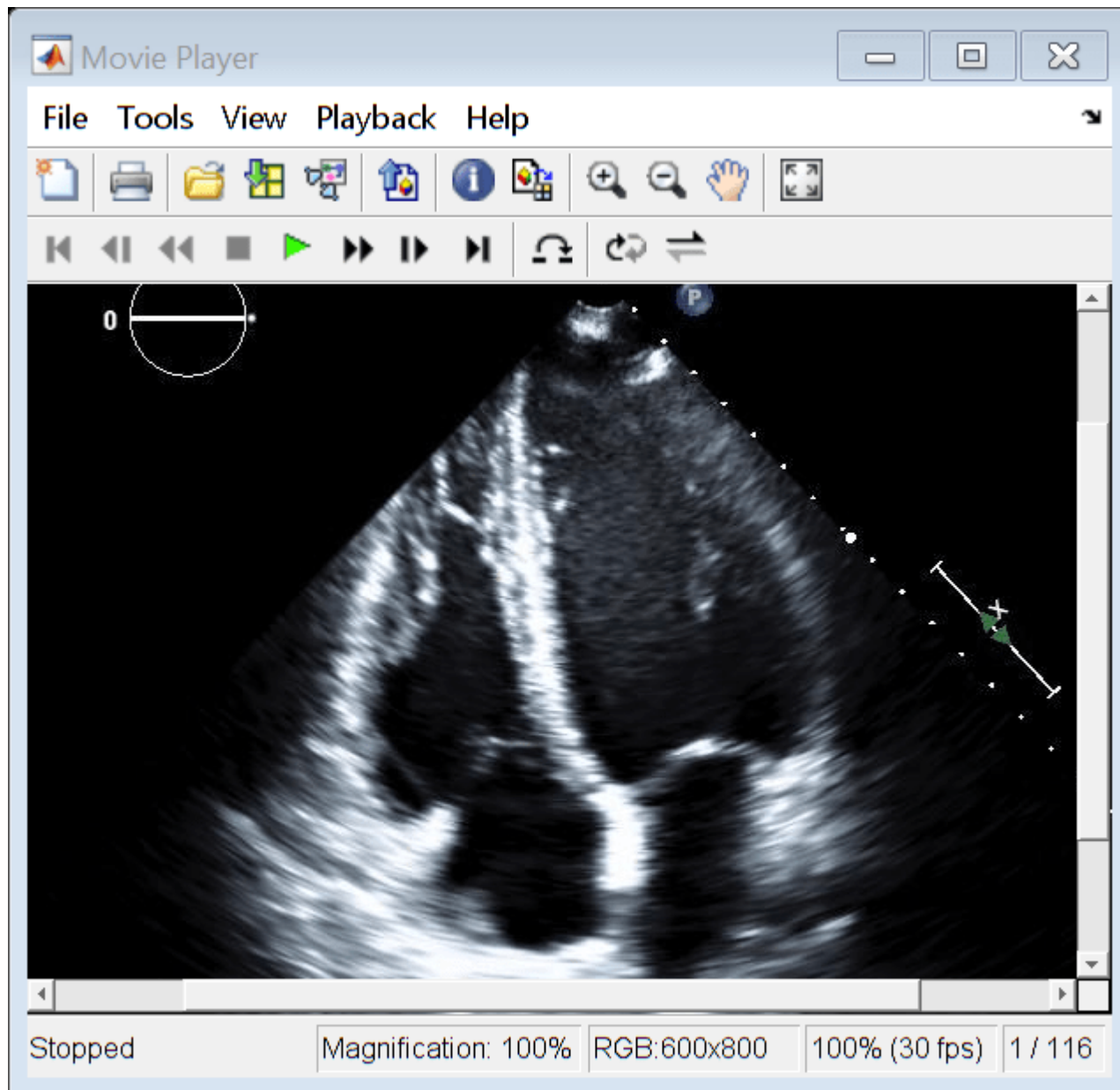
The `FrameTime` property specifies the number of milliseconds between frames. Calculate the frame rate, in frames per second.

```
fps = 1/(medImg.FrameTime/1000)
```

```
fps = 30.0003
```

View the ultrasound series as a video by using the **Video Viewer** app. Specify the frame rate at which to view the video by using the frame rate calculated from the file metadata.

```
imshow(pixels, fps)
```



### 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,c,d] = size(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

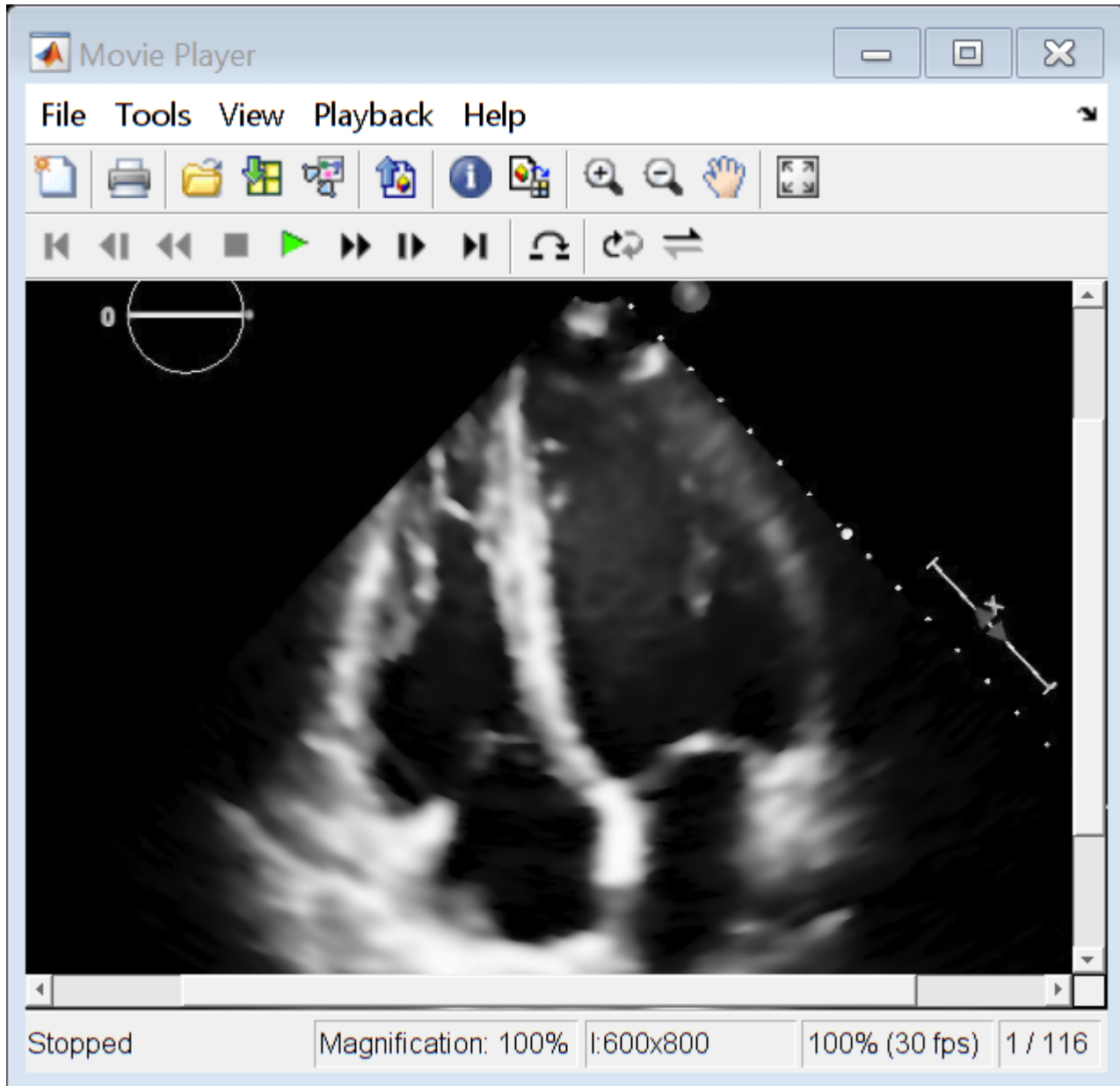
```

Display the smoothed image data as a video.

```

    implay(pixelsSmoothed, fps)

```



### Store Smoothed Data as Medical Image Object

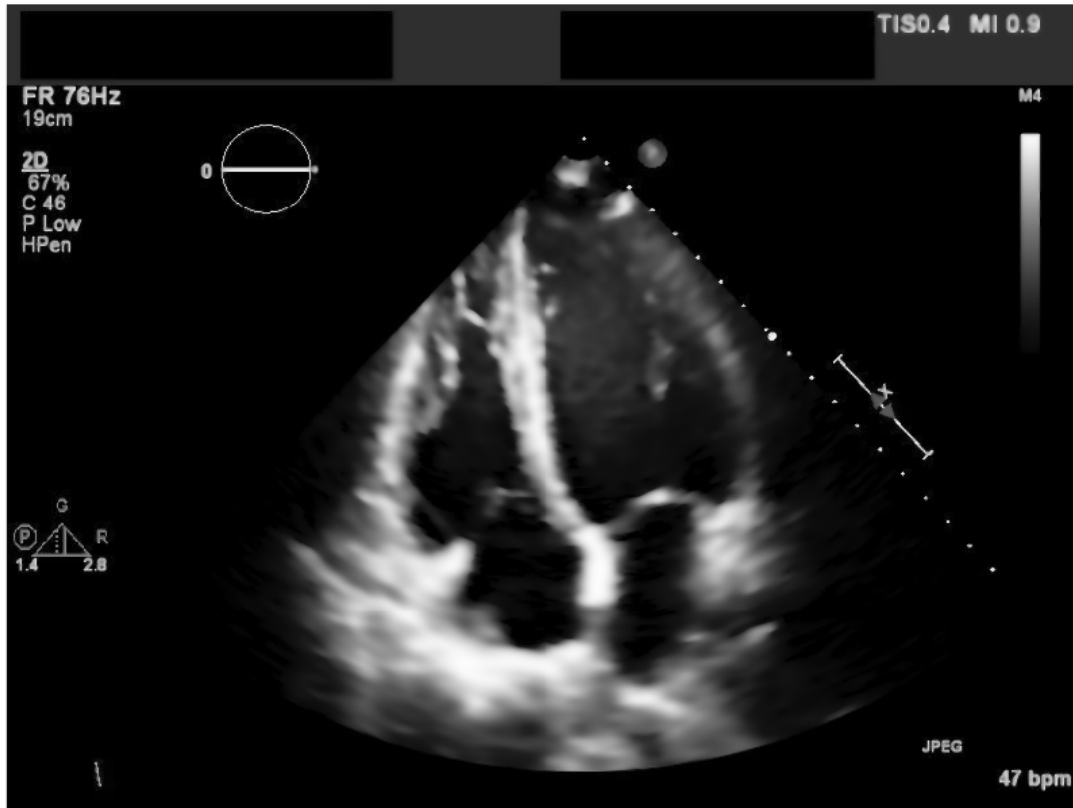
Create a new `medicalImage` object that contains the smoothed image pixels. Use the property values from the original file to maintain the correct frame information.

```
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: []
```

Display the first frame of the new medical image object to verify it contains the smoothed data.

```
imshow(medImgSmoothed.Pixels(:,:,1))
```



## 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

---

- “Visualize 3-D Medical Image Data Using Medical Image Labeler” on page 3-2
- “Display 3-D Medical Image Data in Patient Coordinate System” on page 3-8
- “Display Labeled Medical Image Volume in Patient Coordinates” on page 3-12
- “Create STL Surface Model of Femur Bone for 3-D Printing” on page 3-19

## Visualize 3-D Medical Image Data Using Medical Image Labeler

This example shows how to interactively visualize and explore 3-D medical image data using the **Medical Image Labeler** app.

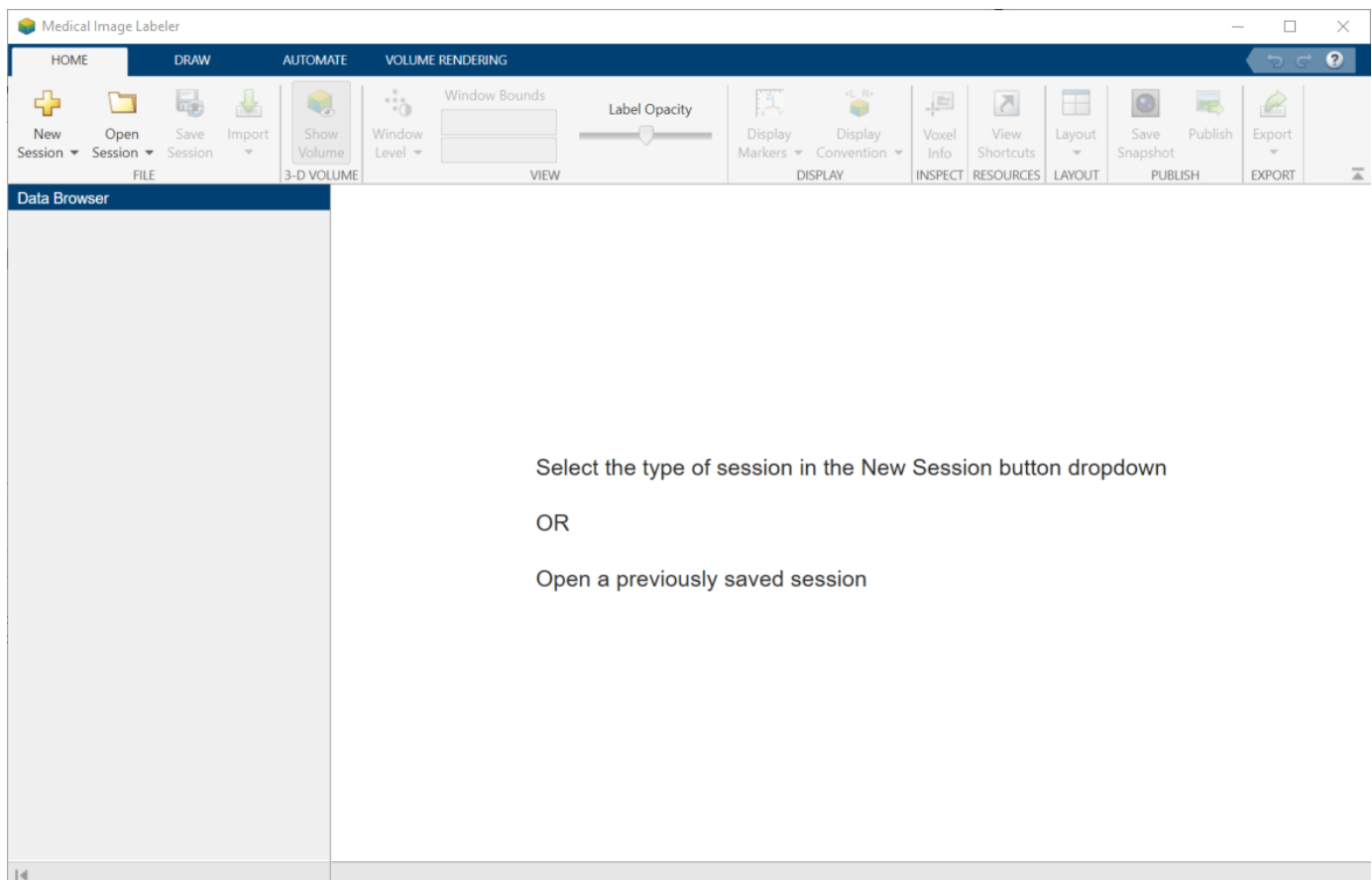
### Download Data

This example labels chest CT data from a subset of the Medical Segmentation Decathlon data set [1 on page 3-7]. 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.

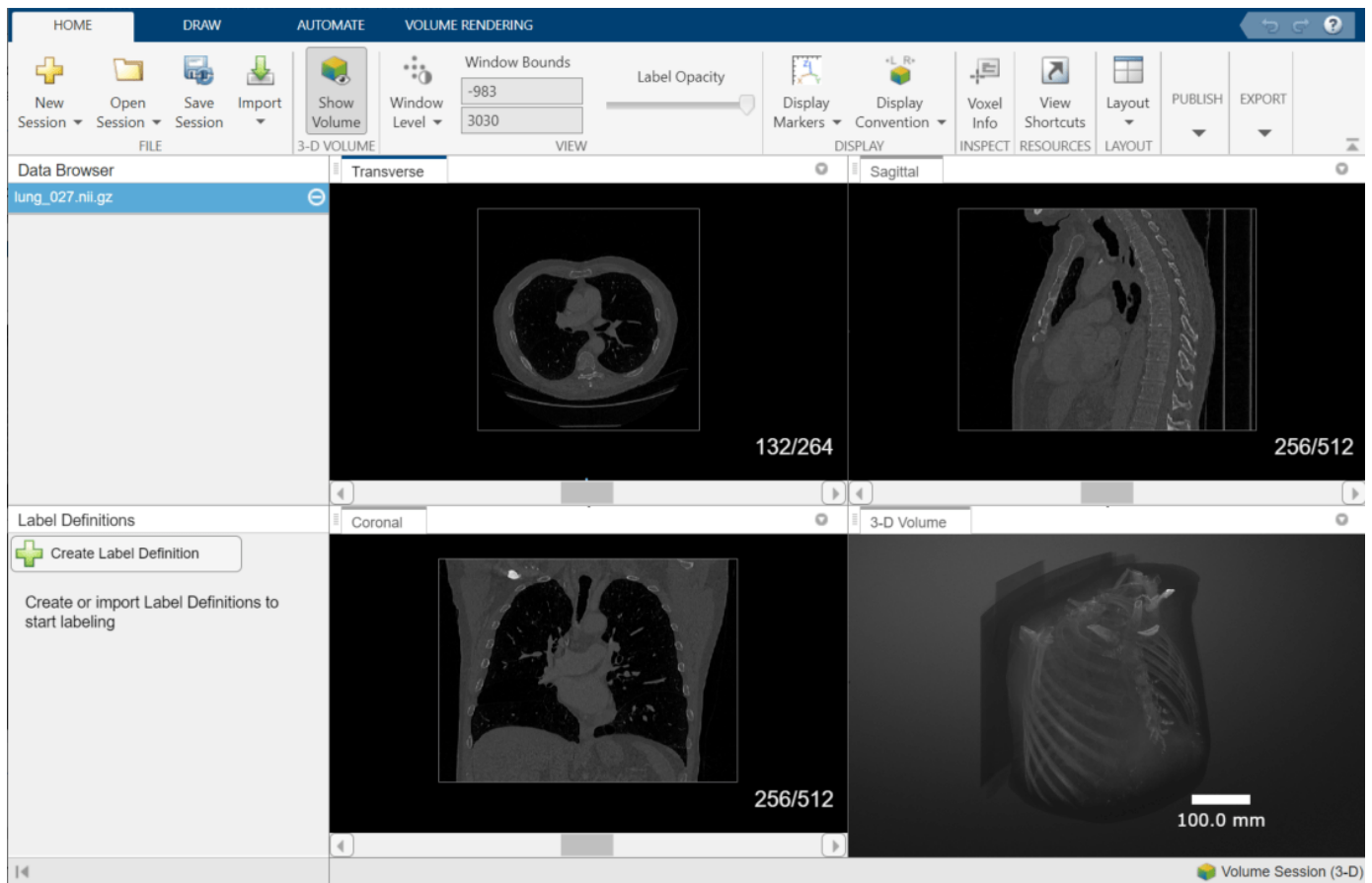


## Create New Volume Labeling Session

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

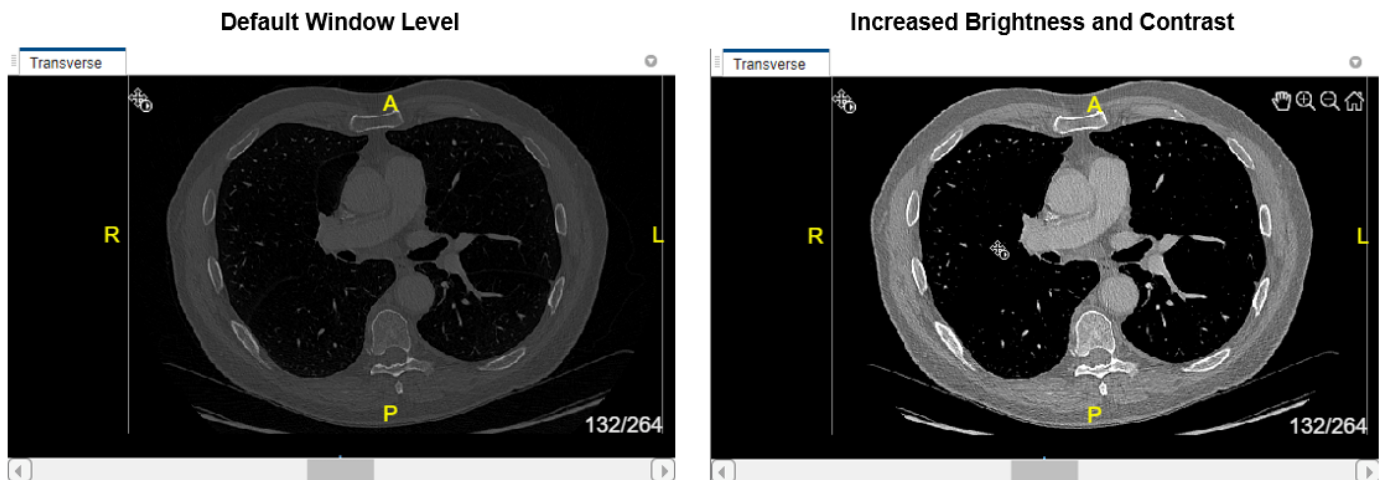
visibility of the display markers, on the app toolbar, click **Display Markers** and, under **2-D Slices**, select or clear **2D Orientation Markers**. You can zoom in on the current slice pane using the mouse scroll wheel or the zoom controls that appear when you pause on the slice pane.

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 **Home** 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**.

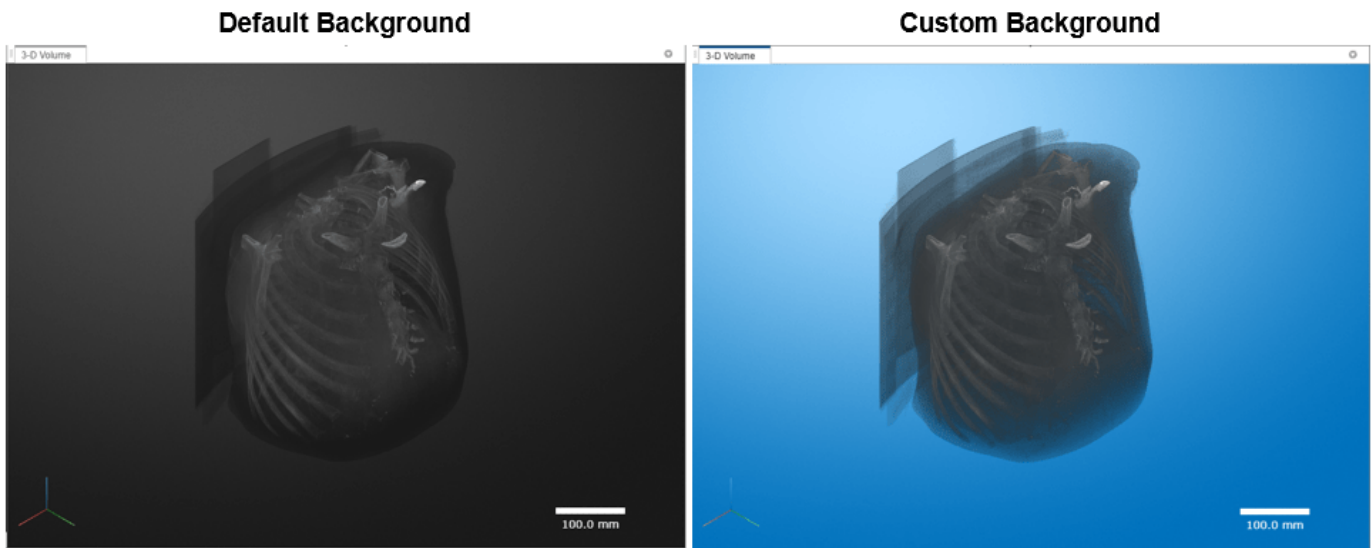


#### View Data as 3-D Volume

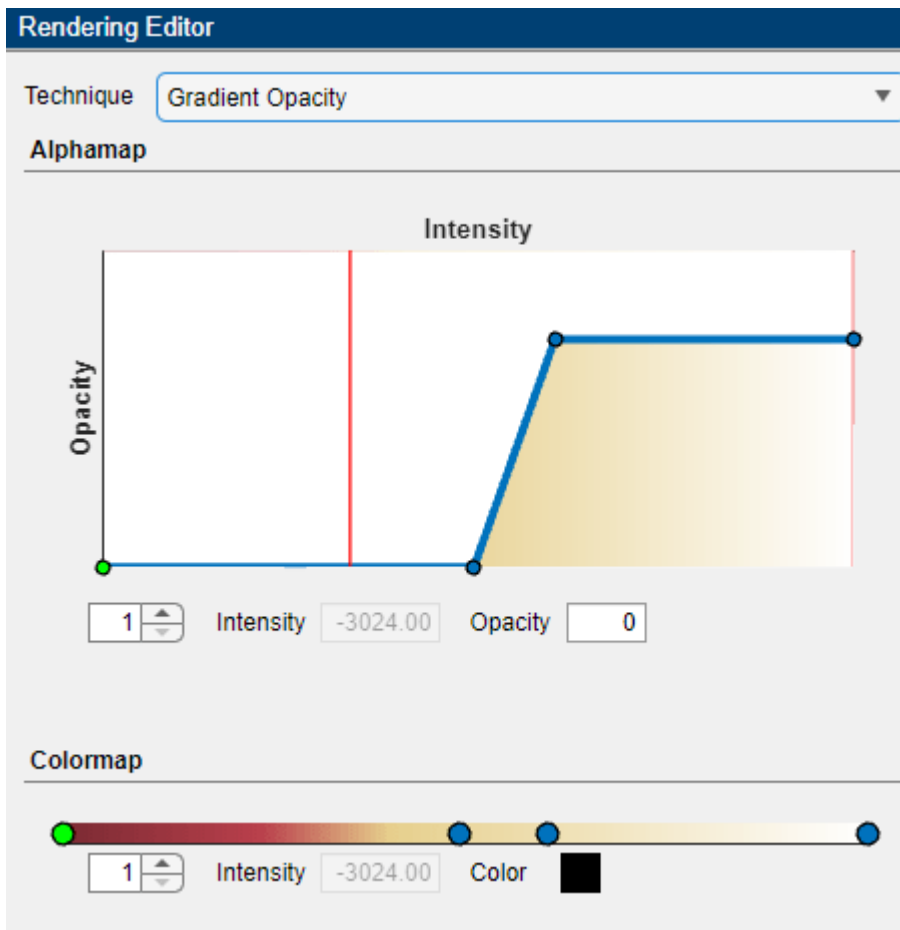
The app displays the data as a 3-D volume in the **3-D Volume** pane. You can toggle the visibility of the volume display using the **Show Volume** button on the **Home** tab of the app toolbar.

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 **Home** tab of the app toolbar, click **Display Markers**, and under **3-D Volume**, select or clear **3D Orientation Axes** and **Scale Bars**.





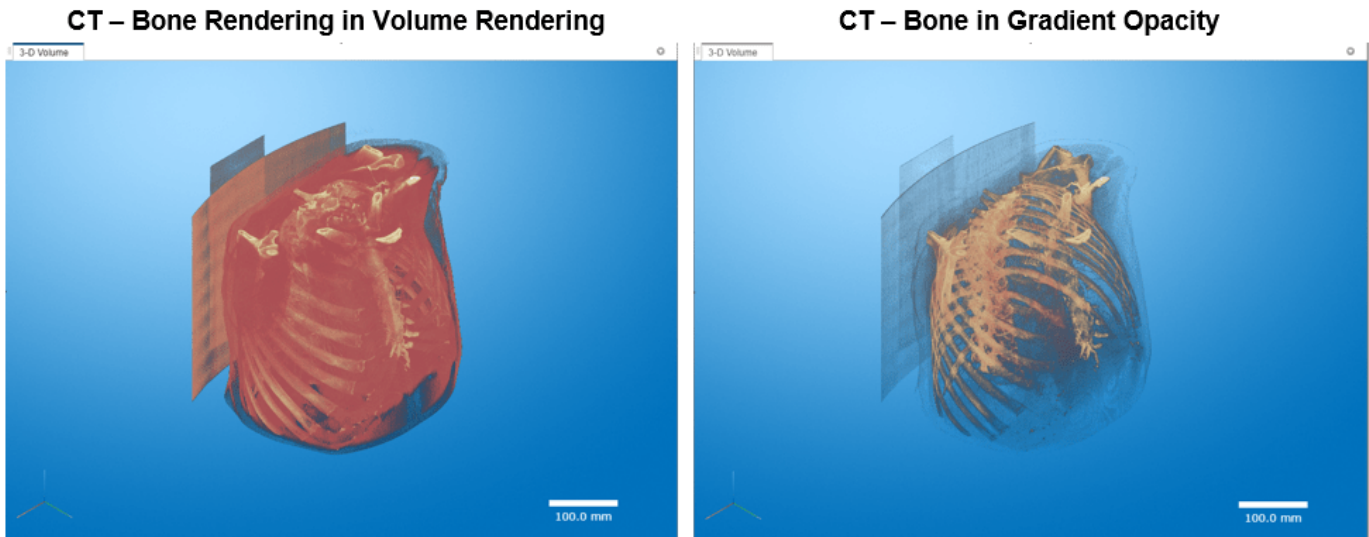
You can change how the app displays voxel intensity data by selecting one of several built-in rendering styles. On the **Volume Rendering** tab, select **CT - Bone** from the **Rendering Presets** gallery to use an alphamap and colormap recommended for highlighting bone tissue in computed tomography (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**.



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 **Home** 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.

### 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

**Medical Image Labeler**

### Related Examples

- "Display 3-D Medical Image Data in Patient Coordinate System" on page 3-8
- "Display Labeled Medical Image Volume in Patient Coordinates" on page 3-12

## Display 3-D Medical Image Data in Patient Coordinate System

This example shows how to display volumetric CT data in the patient coordinate system using the `volshow` function. You can use 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.

### 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 medical image 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 0.7285-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 Without Spatial Referencing

Extract the voxel data from the `medicalVolume` object.

```
V = medVol.Voxels;
```

Visualize the image volume by using the `volshow` function. The image volume appears squished in the transverse direction. The x-, y- and z-axis display markers are aligned with the first, second, and third dimensions of the image array, not the xyz-axes of the patient coordinate system.

```
volIntrinsic = volshow(V,RenderingStyle="MaximumIntensityProjection");
```



By default, `volshow` plots the volume in the intrinsic coordinate system, which assumes cubic voxels with uniform spatial dimensions. Therefore, if your image data is isotropic, then you can display the image in intrinsic coordinates without distortions.

The `Transformation` property of the object created by `volshow` controls whether a transformation is applied to the intrinsic coordinates. By default, the transformation value is a 4-by-4 identity matrix.

```
volIntrinsic.Transformation
```

```
ans =  
  affinetform3d with properties:  
    Dimensionality: 3  
           A: [4x4 double]
```

```
volIntrinsic.Transformation.A
```

```
ans = 4x4
```

```
1    0    0    0
0    1    0    0
0    0    1    0
0    0    0    1
```

You can display the volume in patient coordinates by specifying the `Transformation` property as an `affinetform3d` object that maps between the intrinsic and patient coordinate systems.

#### Define Spatial Referencing

The `VolumeGeometry` property of the medical volume object contains a `medicalref3d` object that specifies the spatial referencing for the volume. Extract the `medicalref3d` object for the chest CT volume.

```
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
```

Calculate the transformation that maps between the intrinsic and patient coordinate systems by using the `intrinsicToWorldMapping` object function. The function returns an `affinetform3d` object, `tform`.

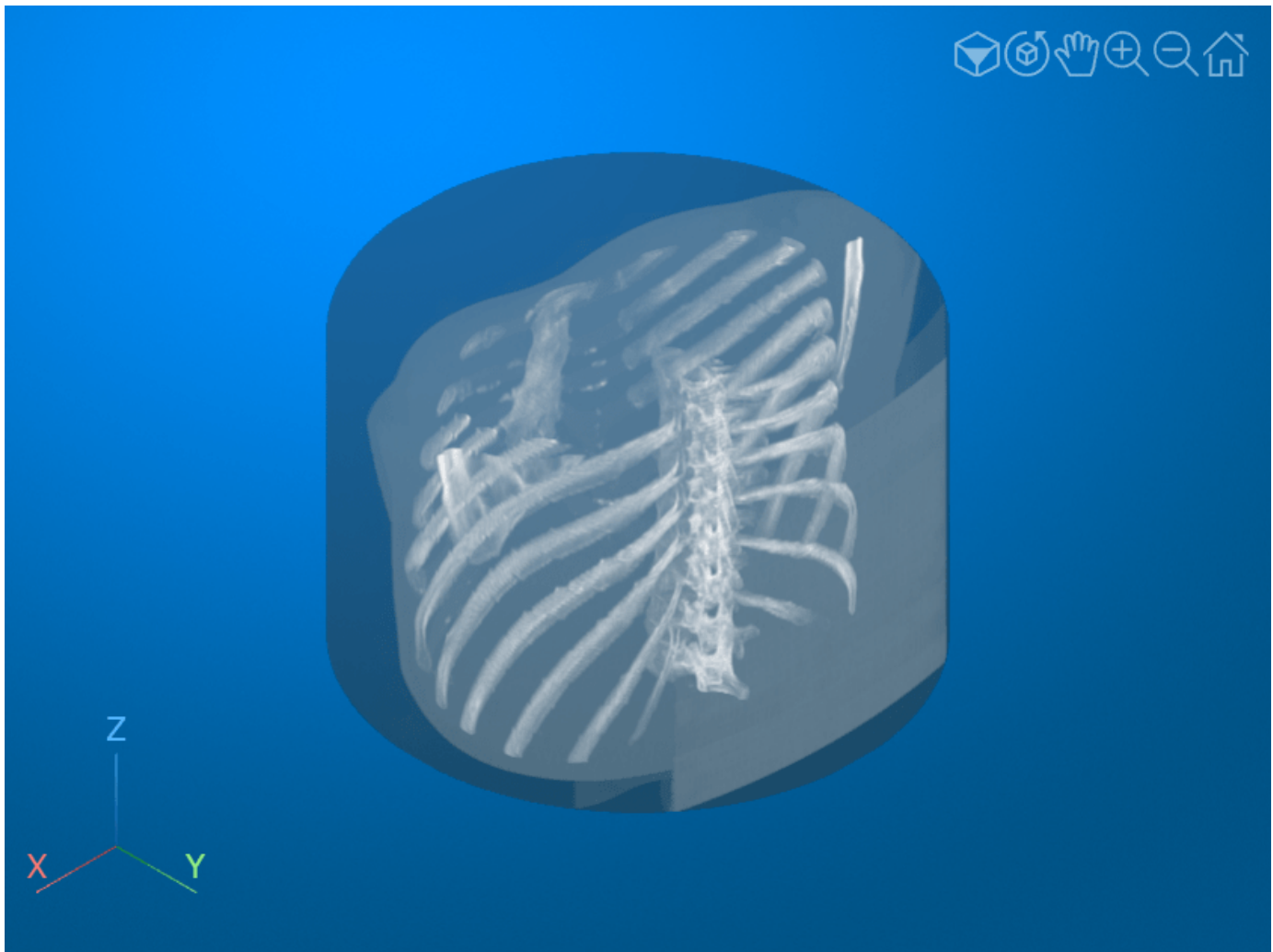
```
tform = intrinsicToWorldMapping(R)
```

```
tform =
  affinetform3d with properties:
    Dimensionality: 3
    A: [4x4 double]
```

#### Display Image Volume with Spatial Referencing

Visualize the image volume by using the `volshow` function, specifying the `Transformation` property as `tform` to include the spatial referencing information. The image volume no longer appears squished in the transverse direction. The transformed volume is also rotated so that the x-, y-, and z-axis display markers are aligned with the patient coordinate axes and point in the directions specified by the `PatientCoordinateSystem` property of `R`.

```
volPatient = volshow(V,Transformation=tform,RenderingStyle="MaximumIntensityProjection");
```



### See Also

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

### Related Examples

- “Display Labeled Medical Image Volume in Patient Coordinates” on page 3-12

## Display Labeled Medical Image Volume in Patient Coordinates

This example shows how to display labeled 3-D medical image volumes by using the `volshow` function. You can visualize labels as an overlay by using the `OverlayData` property of the `Volume` object created by `volshow`. Additionally, you can use 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.

### Download Image Volume Data

This example uses a subset of the Medical Segmentation Decathlon data set [1 on page 3-18]. The subset of data includes two CT chest volumes and corresponding label images, stored in the NIFTI file format. 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)
dataFolder = fullfile(filepath, "MedicalVolumeNIFTIData");
```

Specify the filenames of one CT volume and its label image.

```
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: [512x512x129 single]
  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
```

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.

```
R = medVolData.VolumeGeometry;
```



Calculate the transformation that maps between the intrinsic and patient coordinate systems by using the `intrinsicToWorldMapping` object function. The function returns an `affinetform3d` object, `tform`.

```
tform = intrinsicToWorldMapping(R);
```

### 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);
```

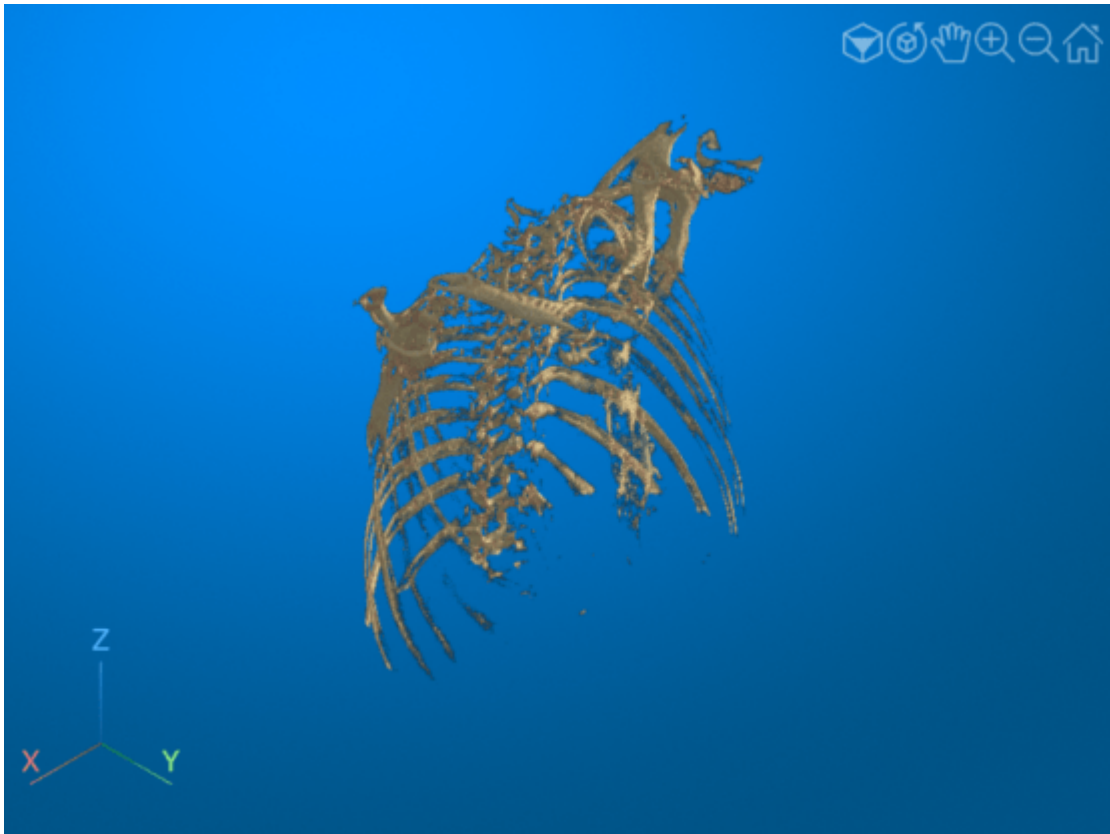
### Display CT Volume with Tumor Overlay

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 for this volume have been tuned 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);
```

View the CT volume with the custom colormap and transparency map. The `volshow` function creates a `Volume` object, `vol`. Specify the `Transformation` property of `vol` as `tform` to include the spatial referencing information and plot the image in patient coordinates. Drag the cursor to rotate the volume. Use the scroll wheel to zoom in and out of the volume.

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



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 **Y** 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`.

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

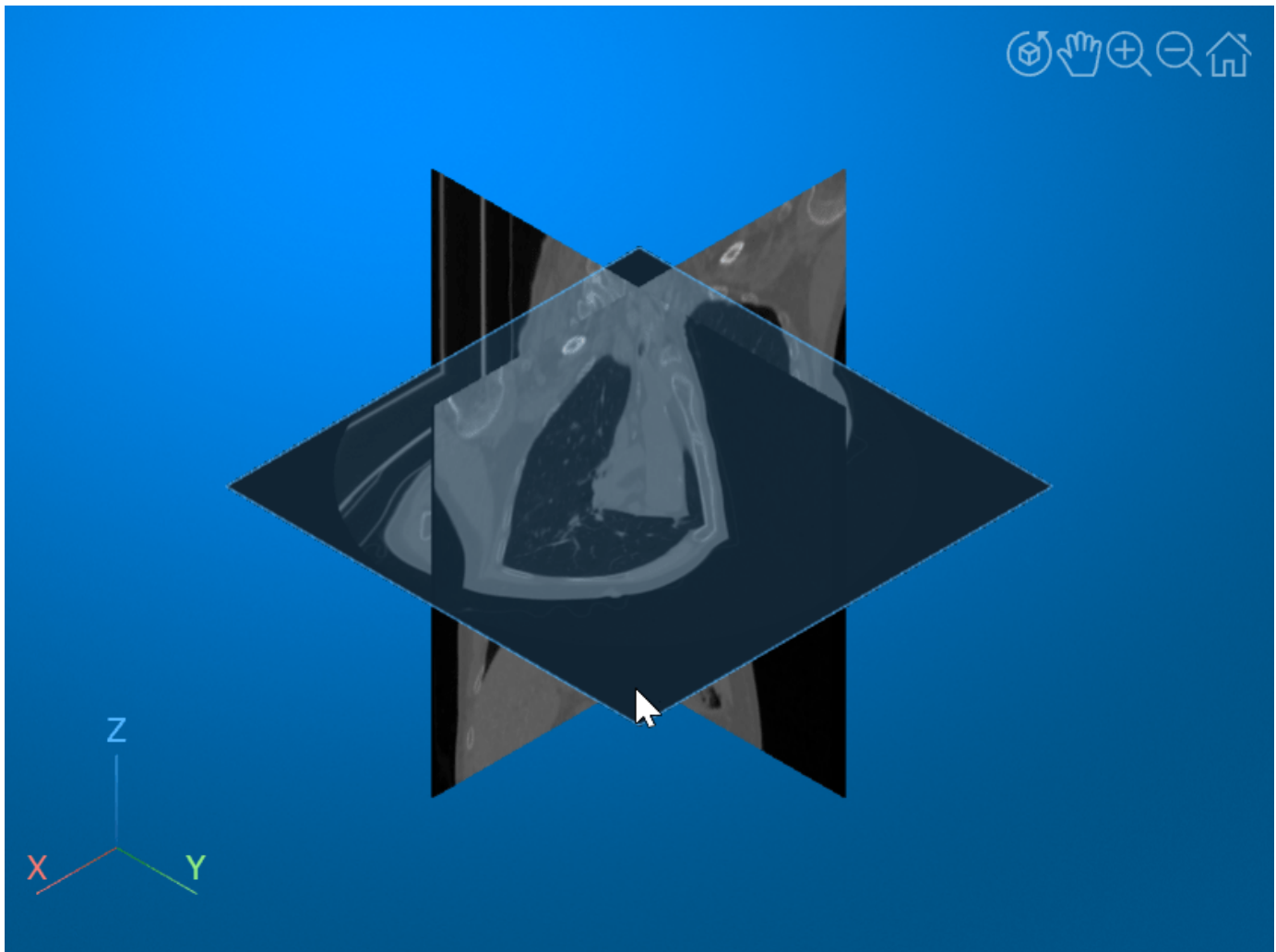


#### Display CT Volume as Slice Planes

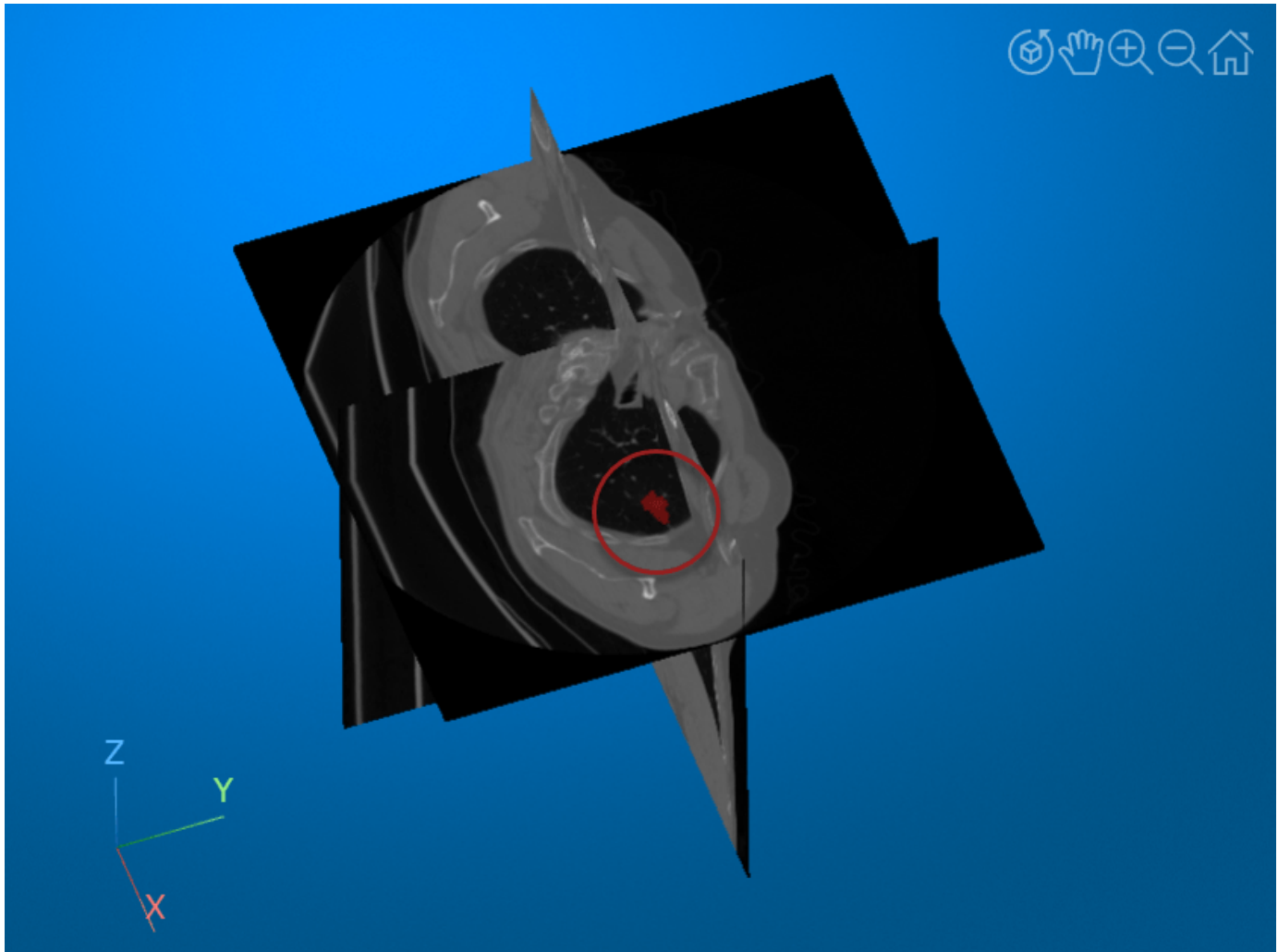
Visualize the CT volume and label overlay as slice planes. Use the `RenderingStyle` name-value argument to specify the rendering style as "SlicePlanes". Specify the tumor label overlay using the `OverlayData` name-value argument.

```
volSlice = volshow(medVolData.Voxels,OverlayData=medvolLabels.Voxels,Transformation=tform,Render
```

To scroll through the transverse slices, pause the cursor on the transverse slice until it highlights in blue, then drag the cursor along the z-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 3-D Medical Image Data in Patient Coordinate System" on page 3-8

## 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.

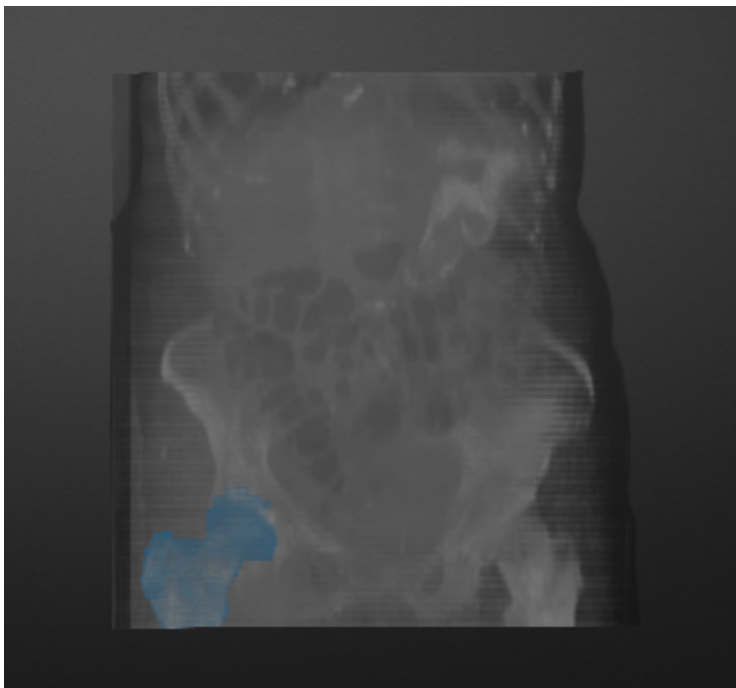
3-D printing creates physical models of computer-generated surface models. The typical workflow for 3-D printing anatomical structures from medical image volumes includes these steps:

- 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 3-25] 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.



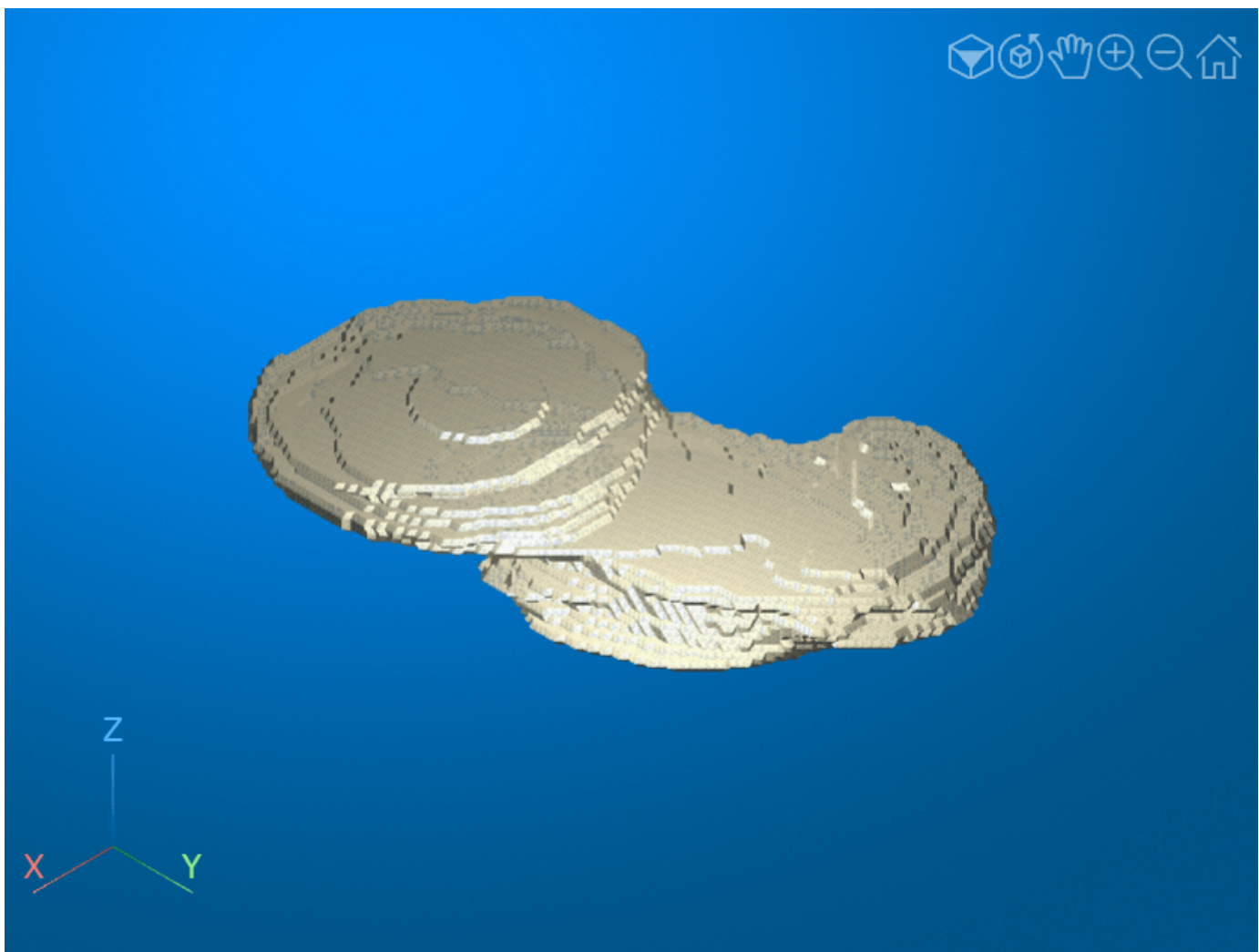
```
[faces,vertices] = extractIsosurface(medVol.Voxels,isovalue);
```

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=0.5);
```



### 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);
```

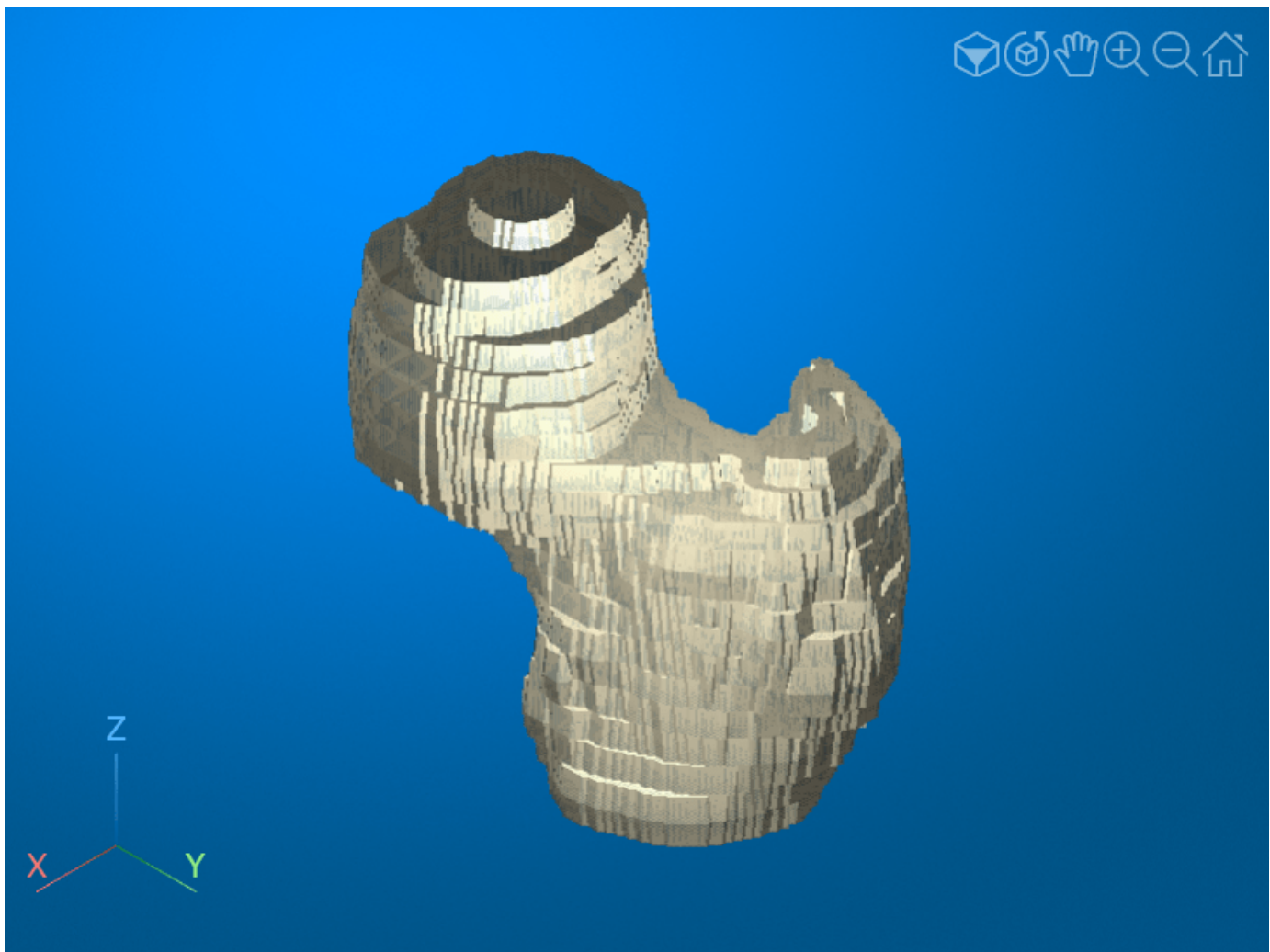
```
[X,Y,Z] = intrinsicToWorld(R,I,J,K);  
verticesPatientCoords = [X Y Z];
```

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: [21950×3 double]  
        ConnectivityList: [43896×3 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`.

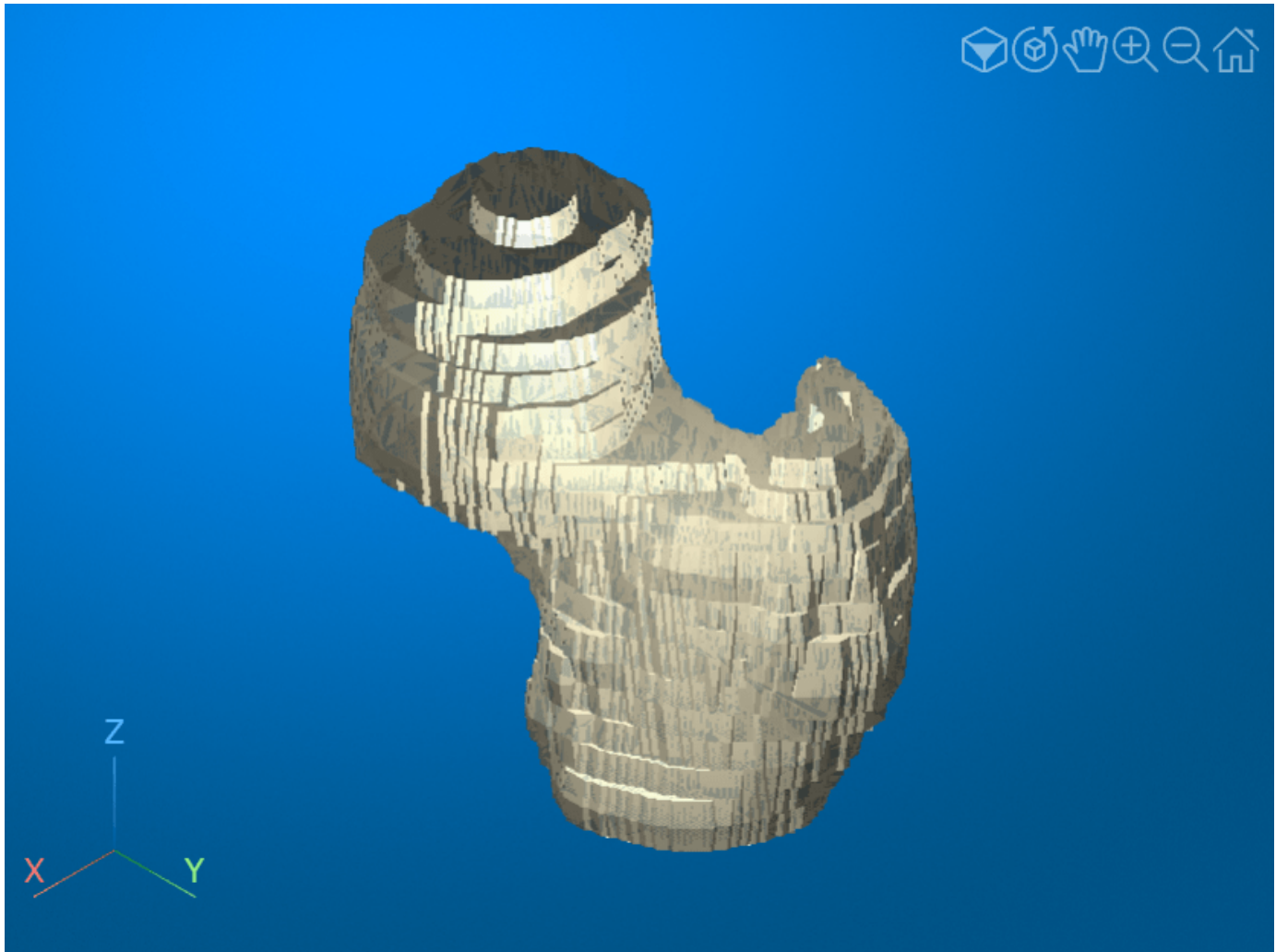
```
pReduced = reducepatch(faces,verticesPatientCoords,0.5);
```

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: [10976×3 double]  
        ConnectivityList: [21948×3 double]
```

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

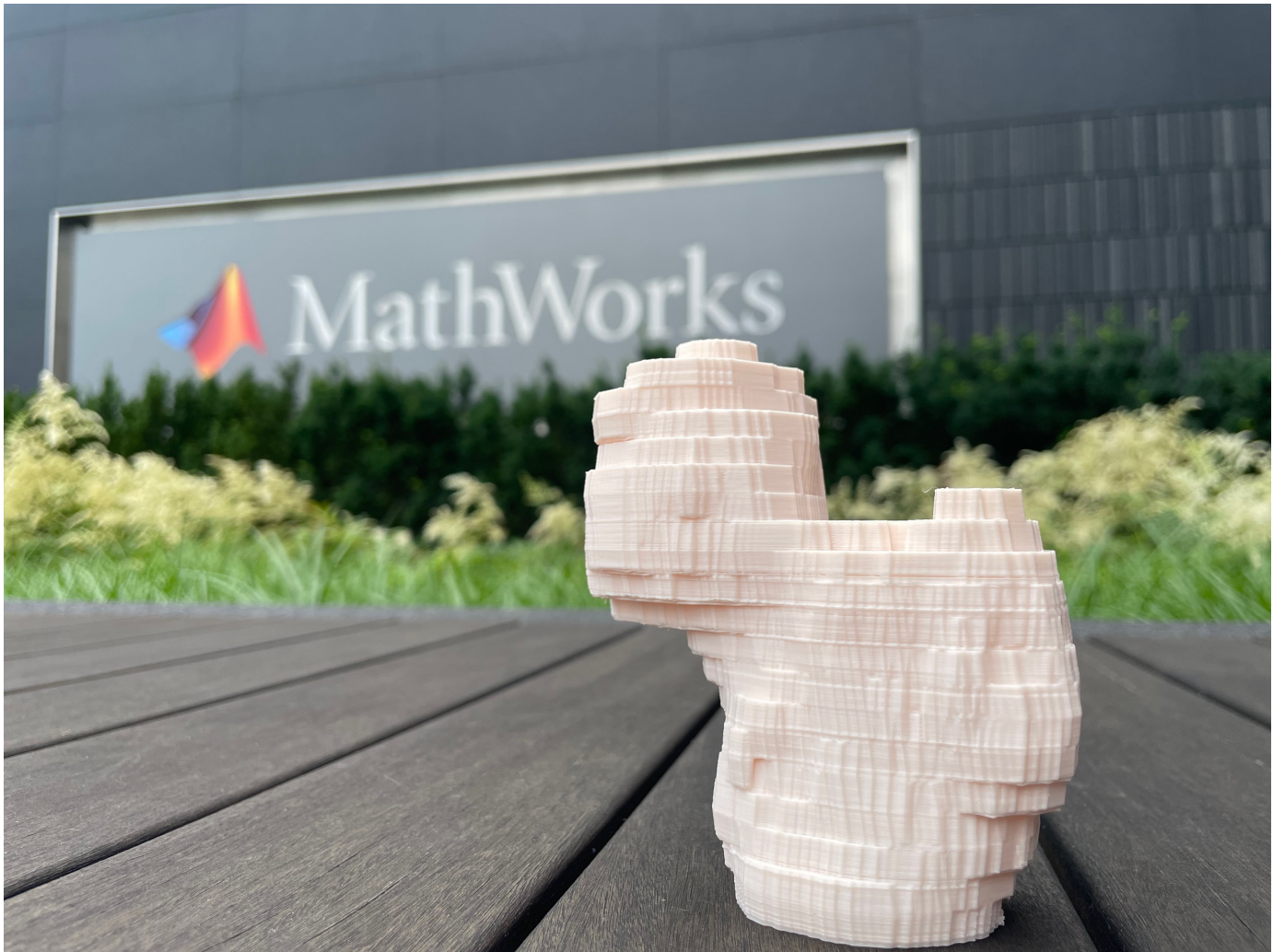
```
viewerPatient = viewer3d;  
obj = images.ui.graphics3d.Surface(viewerPatient,Data=triReduced,Color=[0.88 0.84 0.71],Alpha=0.9)
```



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 3-D Medical Image Data in Patient Coordinate System” on page 3-8

# 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 the 2-D or 3-D medical images in a data set. Registration enables you to align images of different patients, or images of the same patient acquired at different times, on different scanners, or using different imaging modalities. Medical Imaging Toolbox provides functions for rigid, similarity, and deformable image registration and rigid surface registration. 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

- “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

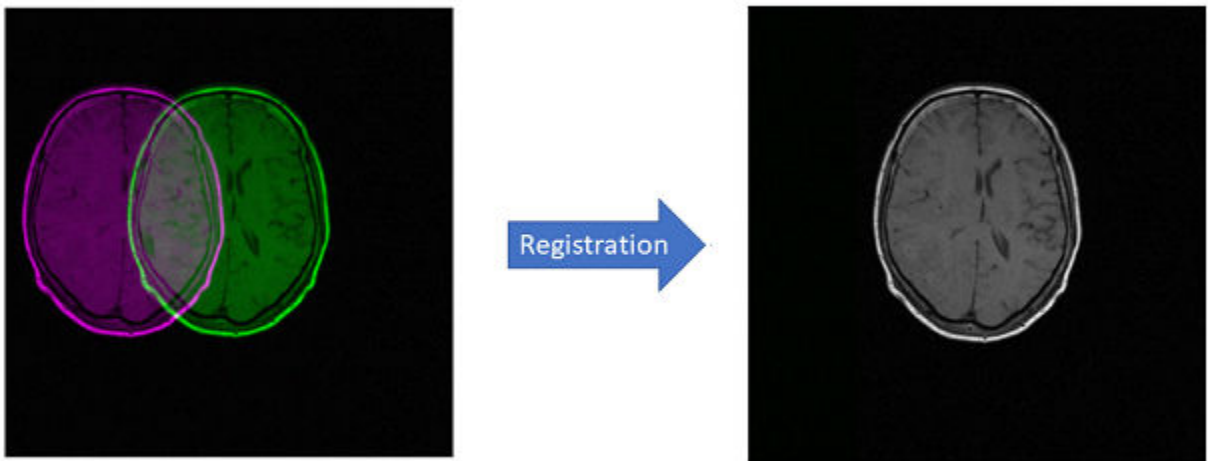
## Medical Image Registration

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.

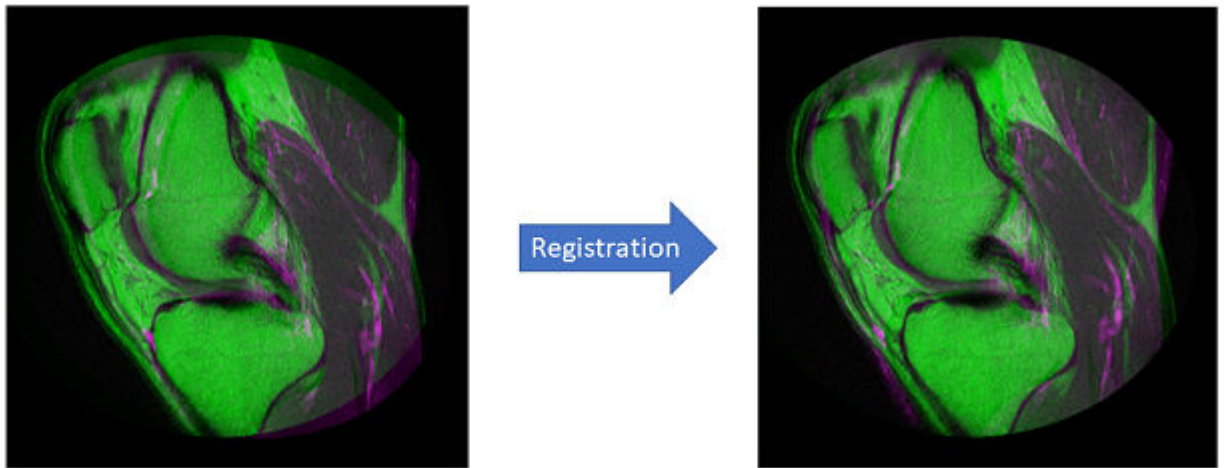
### Scenarios for Medical Image Registration

#### 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.

<b>Function</b>	<b>Type of Misalignment</b>	<b>Type of Input</b>	<b>Function Details</b>	<b>Method</b>
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

# 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. To register images with scaling differences using `imregmoment`, you must define the *spatial referencing* for the images. Spatial referencing maps the row, column, and slice indices of the image array to the patient coordinate system. You can use the `medicalVolume` object to automatically import image volumes and spatial referencing metadata from an image file.

In this example, you use `medicalVolume` and `imregmoment` to register multimodal MRI and CT image 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: [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: []
```

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. Therefore, the CT volume contains more voxels than the MRI scan for the same region of interest.

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

```
movingCTVolume =
    medicalVolume with properties:
```

```
        Voxels: [512x512x28 single]
    VolumeGeometry: [1x1 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;
```

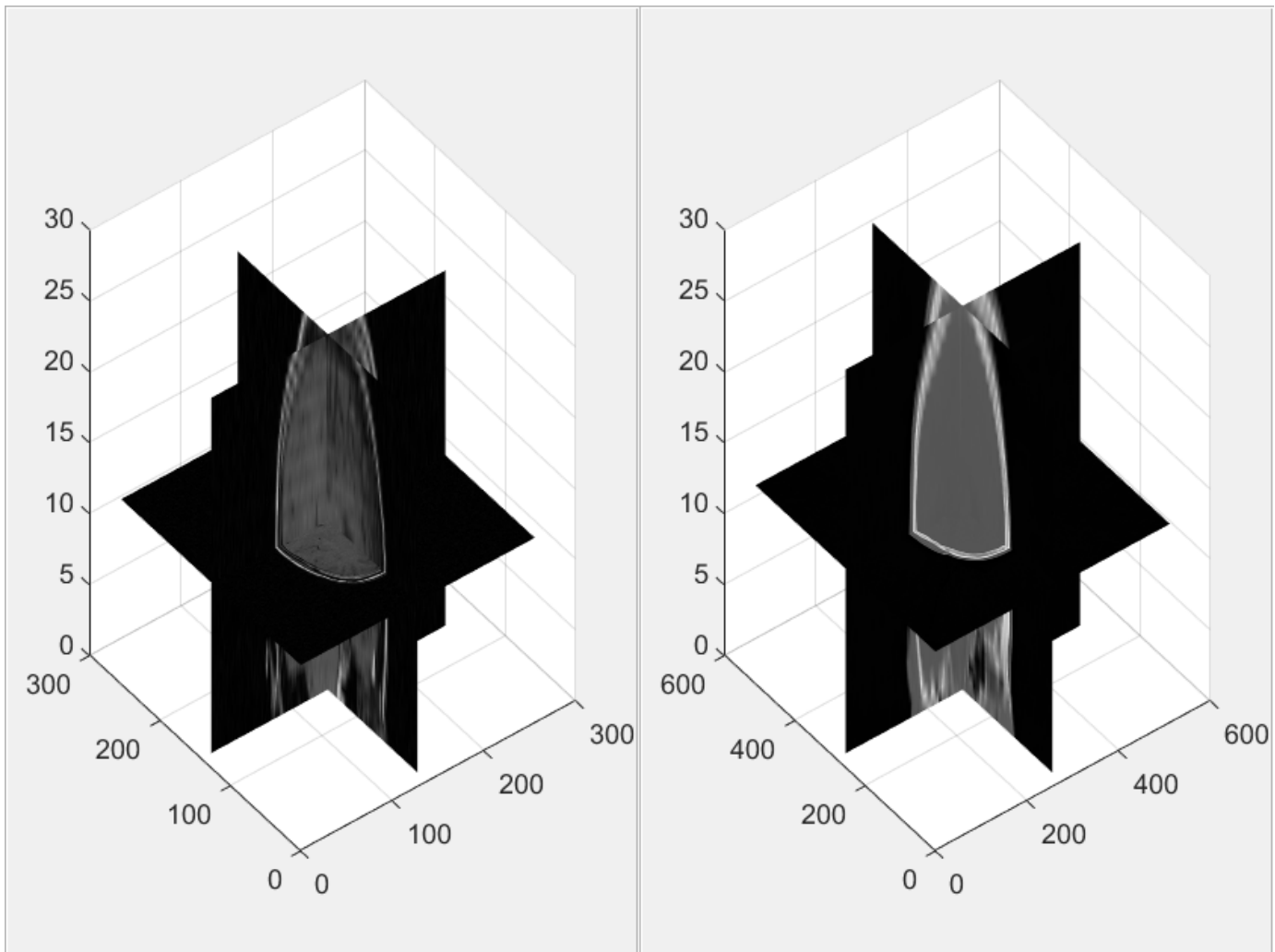
### Display Unregistered Images

Use the `helperVolumeRegistration` helper function to judge image alignment. The output axes remain in `syn` if you interactively rotate the view.

The plot is in intrinsic coordinates, in units of voxels. The axes limits are different between the two images because of the differences in image array size.

```
helperVolumeRegistration(fixedMRIVoxels,movingCTVoxels);
```





You can also use `imshowpair` to check alignment in single slices from the fixed and moving volumes. Use `imshowpair` to view the middle transverse slice of each volume.

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

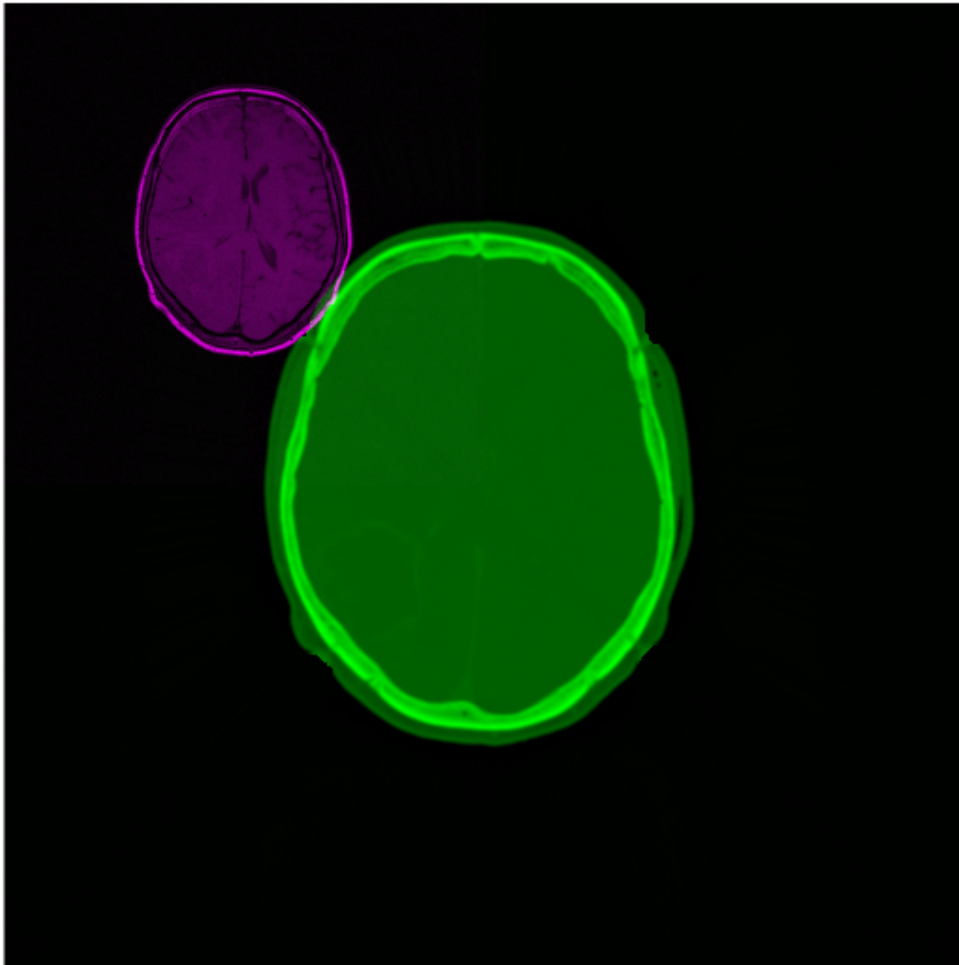
```
fixedVolumeSize = fixedMRIVolume.VolumeGeometry.VolumeSize;
movingVolumeSize = movingCTVolume.VolumeGeometry.VolumeSize;
```

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

Next, display the image slices 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. The images are misaligned due to translations, rotations, and scaling.

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

### Unregistered Transverse Slice



#### Register Images

To obtain accurate scaling results using `imregmoment`, you must specify spatial referencing information for each volume. Use the spatial referencing information stored in the `medicalVolume` objects to define corresponding `imref3d` spatial referencing objects. Specify the image array dimensions and voxel sizes using the `VolumeSize` and `VoxelSpacing` properties, respectively, of the `medicalVolume` objects.

```
Rfixed = imref3d(fixedVolumeSize, fixedMRIVolume.VoxelSpacing(2), fixedMRIVolume.VoxelSpacing(1),  
Rmoving = imref3d(movingVolumeSize, movingCTVolume.VoxelSpacing(2), movingCTVolume.VoxelSpacing(1))
```

Use `imregmoment` to register the moving volume to the fixed volume. Specify the `MedianThresholdBitmap` name-value argument as `true`, which is appropriate for multimodal images .

```
[geomtform,movingCTRegisteredVoxels] = imregmoment(movingCTVoxels,Rmoving,fixedMRIVoxels,Rfixed,I
```

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 to the fixed MRI volume.

```
geomtform.T
```

```
ans = 4x4
```

```
    0.5229    -0.0021     0.0007         0
    0.0020     0.5229     0.0033         0
   -0.0014   -0.0063     1.0000         0
   -4.8094  -16.0063   -1.3481     1.0000
```

The `movingRegisteredVoxels` output contains the registered CT image. The `imregmoment` function resamples the registered image coordinates using the voxel grid of the fixed image. Therefore, the registered volume and fixed volume are the same size and have voxel-to-voxel correspondence.

```
whos movingCTRegisteredVoxels fixedMRIVoxels
```

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

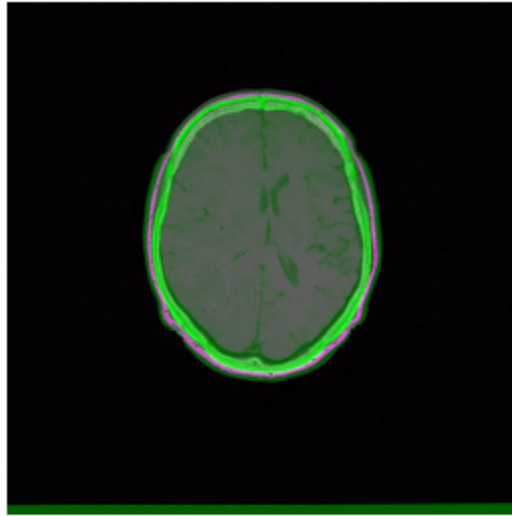
### Display Registered Volumes

To check the registration results, use `imshowpair` to view the middle transverse slices of the fixed and registered volumes. The images are aligned and scaled to the same size.

```
figure
```

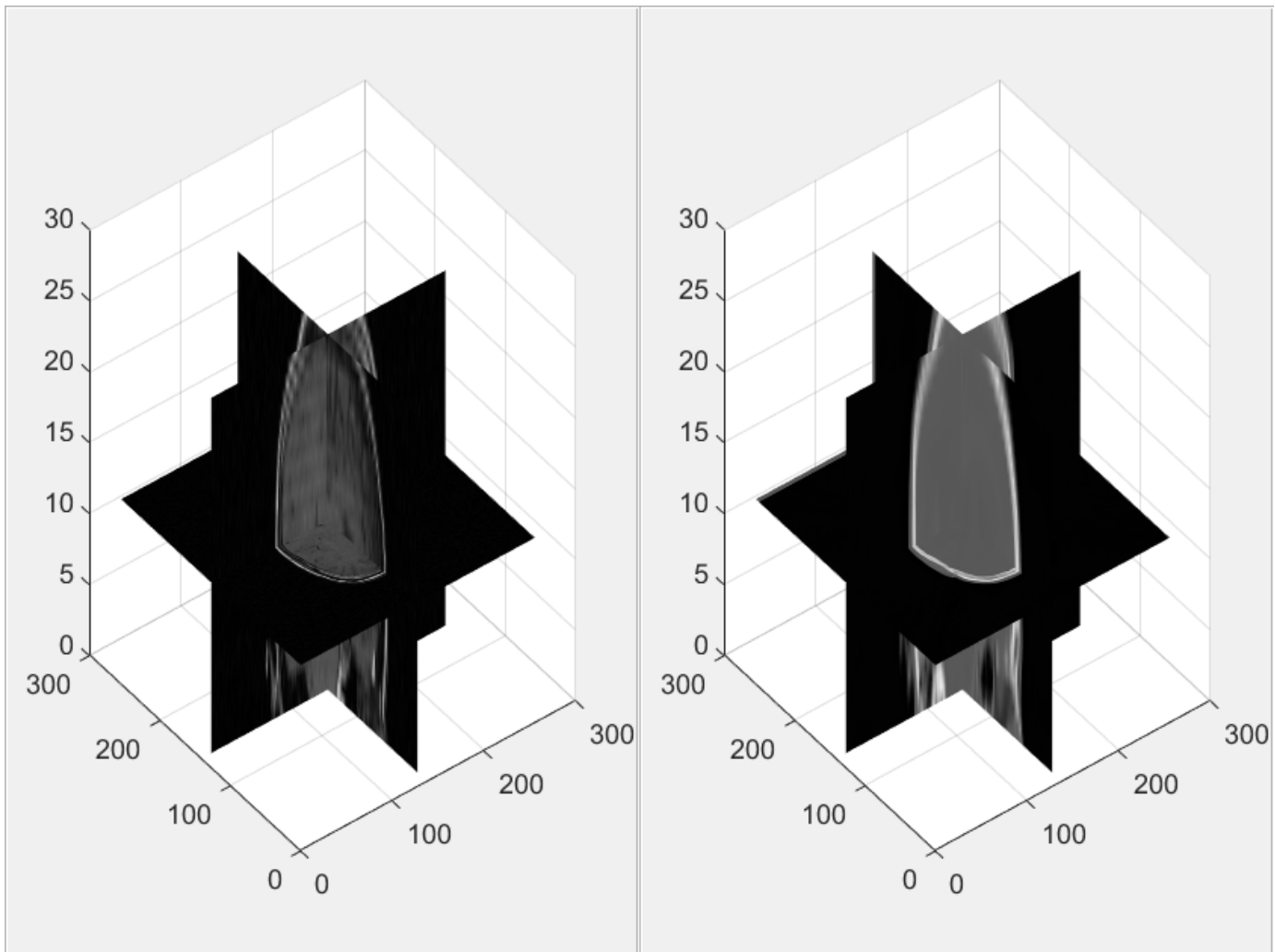
```
imshowpair(movingCTRegisteredVoxels(:,:,centerFixed(3)),fixedMRIVoxels(:,:,centerFixed(3)))
title("Registered Transverse Slice")
```

### Registered Transverse Slice



Use `helperVolumeRegistration` to view the results of the 3-D registration. The axes limits are the same, because of scaling the CT volume during registration.

```
helperVolumeRegistration(fixedMRIVoxels,movingCTRegisteredVoxels);
```



### 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 containing spatial details. 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:
```

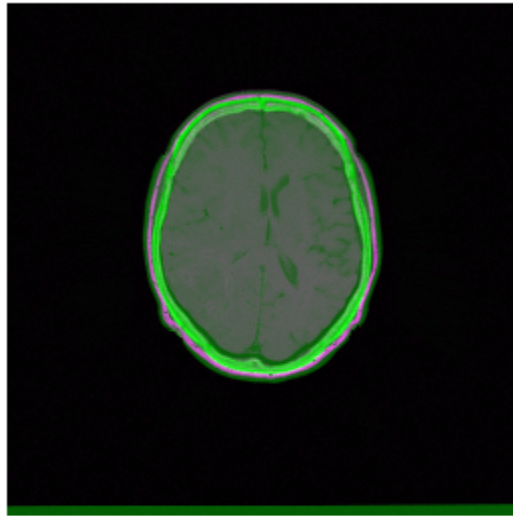
```
        Voxels: [256x256x26 single]
    VolumeGeometry: [1x1 medicalref3d]
        SpatialUnits: "unknown"
        Orientation: "unknown"
        VoxelSpacing: [1.2500 1.2500 4]
        NormalVector: [0 0 1]
    NumCoronalSlices: []
```

```
NumSagittalSlices: []
NumTransverseSlices: []
PlaneMapping: ["unknown" "unknown" "unknown"]
Modality: "unknown"
WindowCenters: []
WindowWidths: []
```

Verify that the new `medicalVolume` object contains CT voxel data that is aligned with the MRI image by using `imshowpair`.

```
figure
imshowpair(movingRegisteredVolume.Voxels(:,:,centerFixed(3)), fixedMRIVoxels(:,:,centerFixed(3)))
title("Axial Slice of Registered Volume")
```

### Axial Slice of Registered Volume



### See Also

`imregmoment` | `medicalref3d` | `medicalVolume`

### Related Examples

- “Medical Image Registration” on page 4-5

# Medical Image Labeling

---

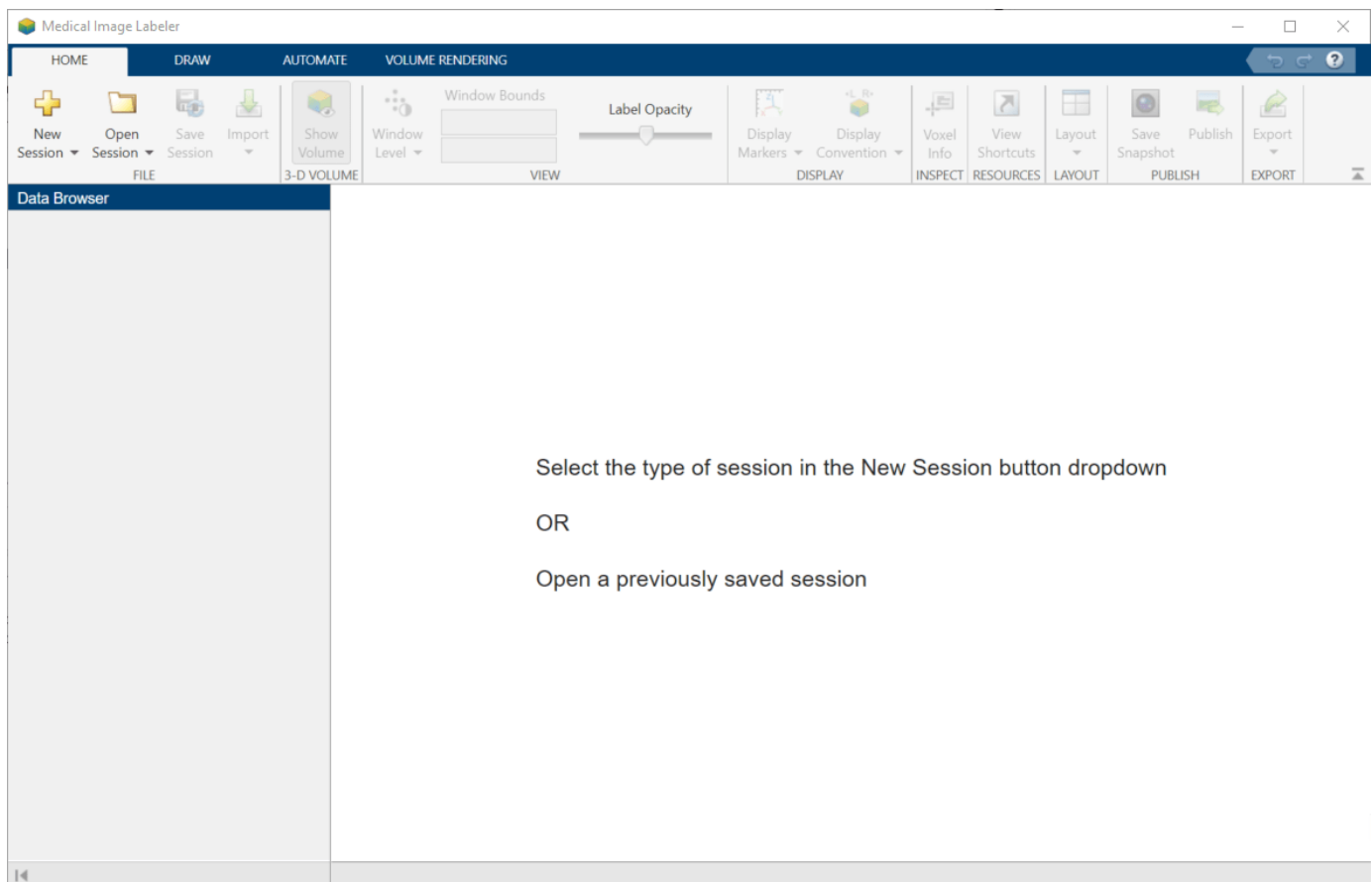
- “Label 2-D Ultrasound Series Using Medical Image Labeler” on page 5-2
- “Label 3-D Medical Image Using Medical Image Labeler” on page 5-10
- “Collaborate on Multi-Labeler Medical Image Labeling Projects” on page 5-19

## 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

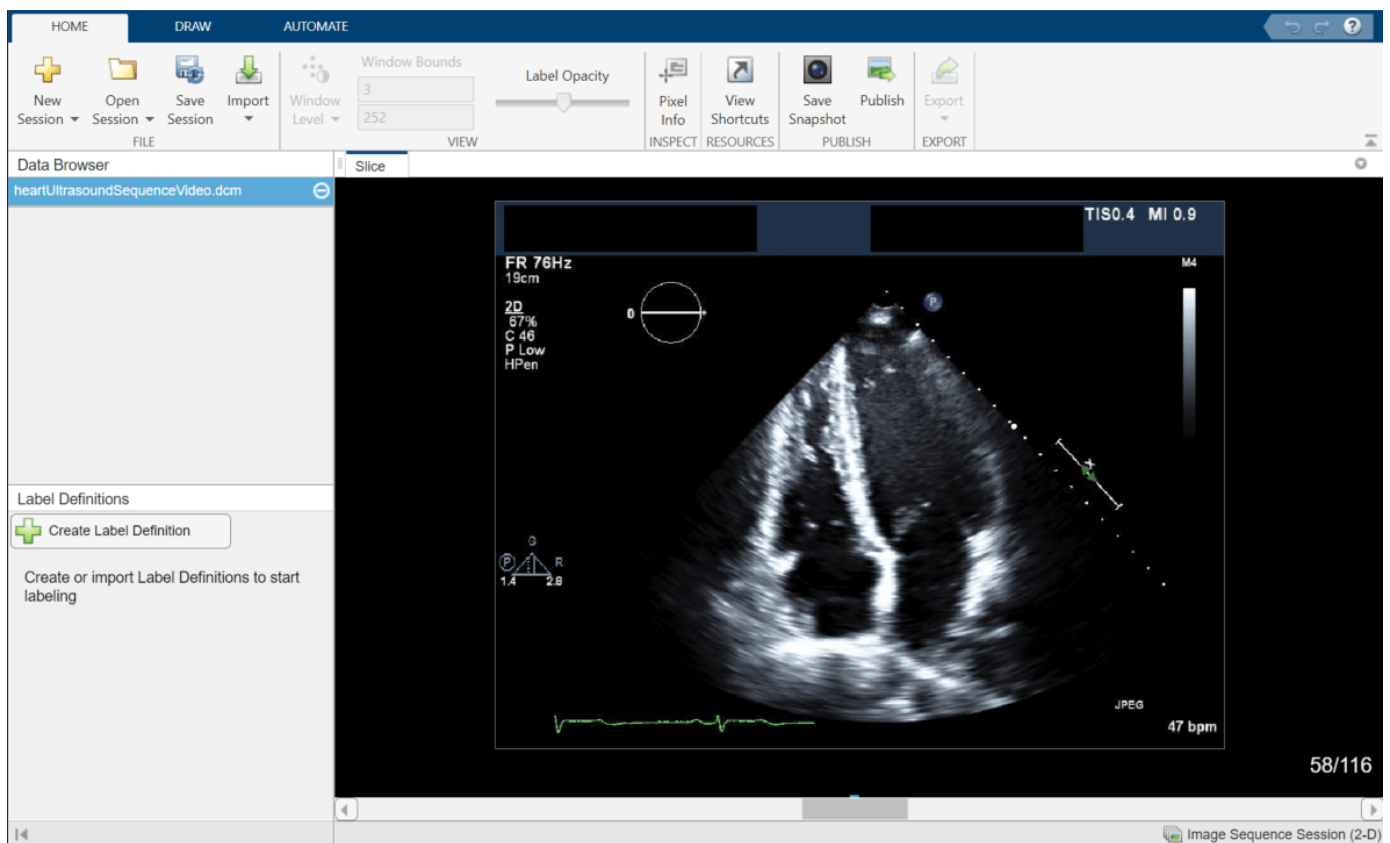
To start a new 2-D labeling session, on the app toolstrip, click **New Session** and select **New Image session (2-D)**. In the Create a new session folder dialog box, specify a location in which to save the new session folder by entering a path or selecting **Browse** and navigating to your desired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

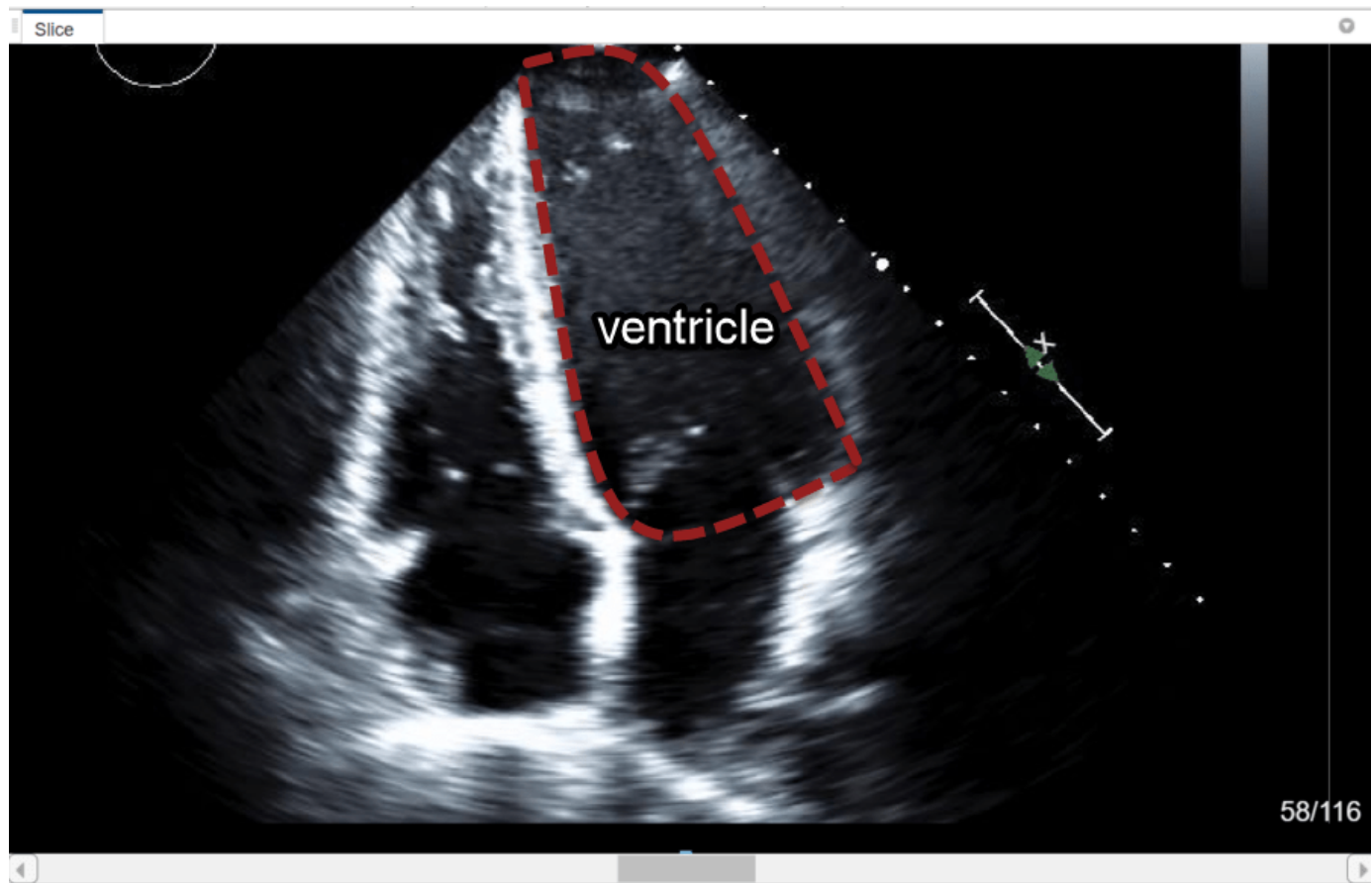
The name of the imported image is visible in the **Data Browser** pane. The no labels symbol  next to the file name indicates that the image file does not contain any pixel labels. You can import multiple 2-D image files into an image session. All files imported into a single app session must label the 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.



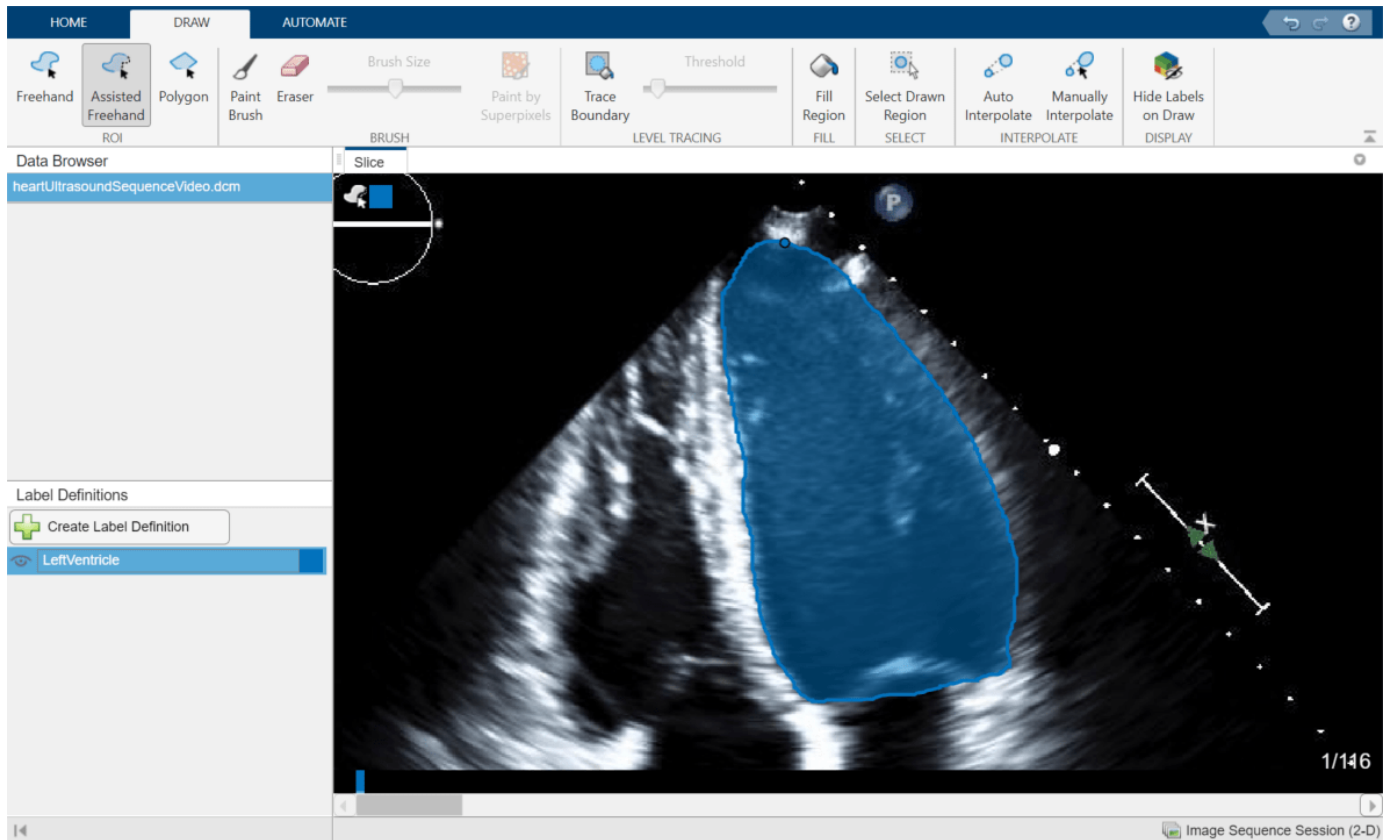
### Create Label Definitions

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.

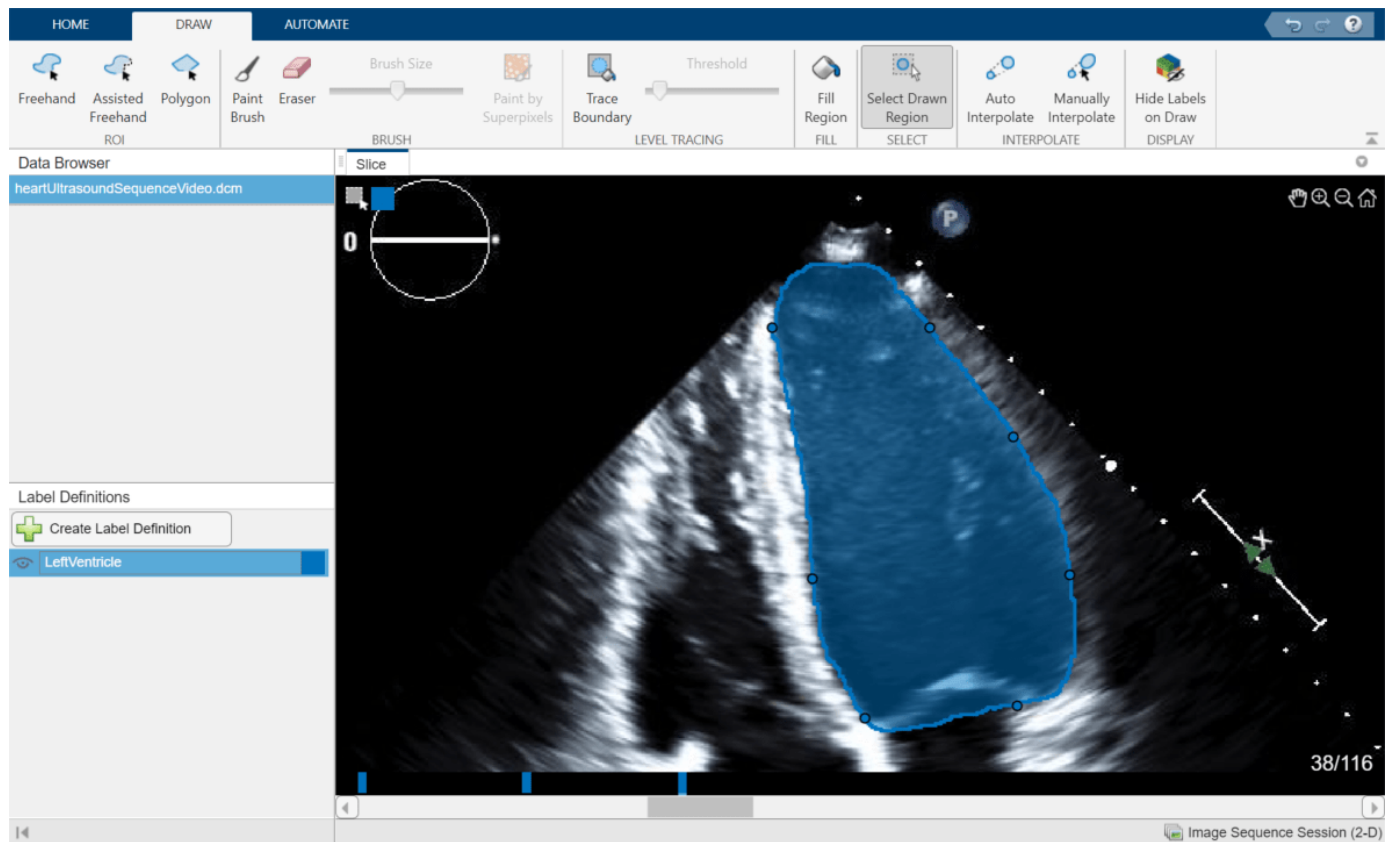


In addition to the drawing tools, you can add or refine label regions using the tools in the **Automate** tab of the app toolstrip. The app provides automation algorithms including **Active Contours**, **Adaptive Threshold**, **Dilate**, and **Erode**. To apply an algorithm, click **Algorithm Parameters** 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

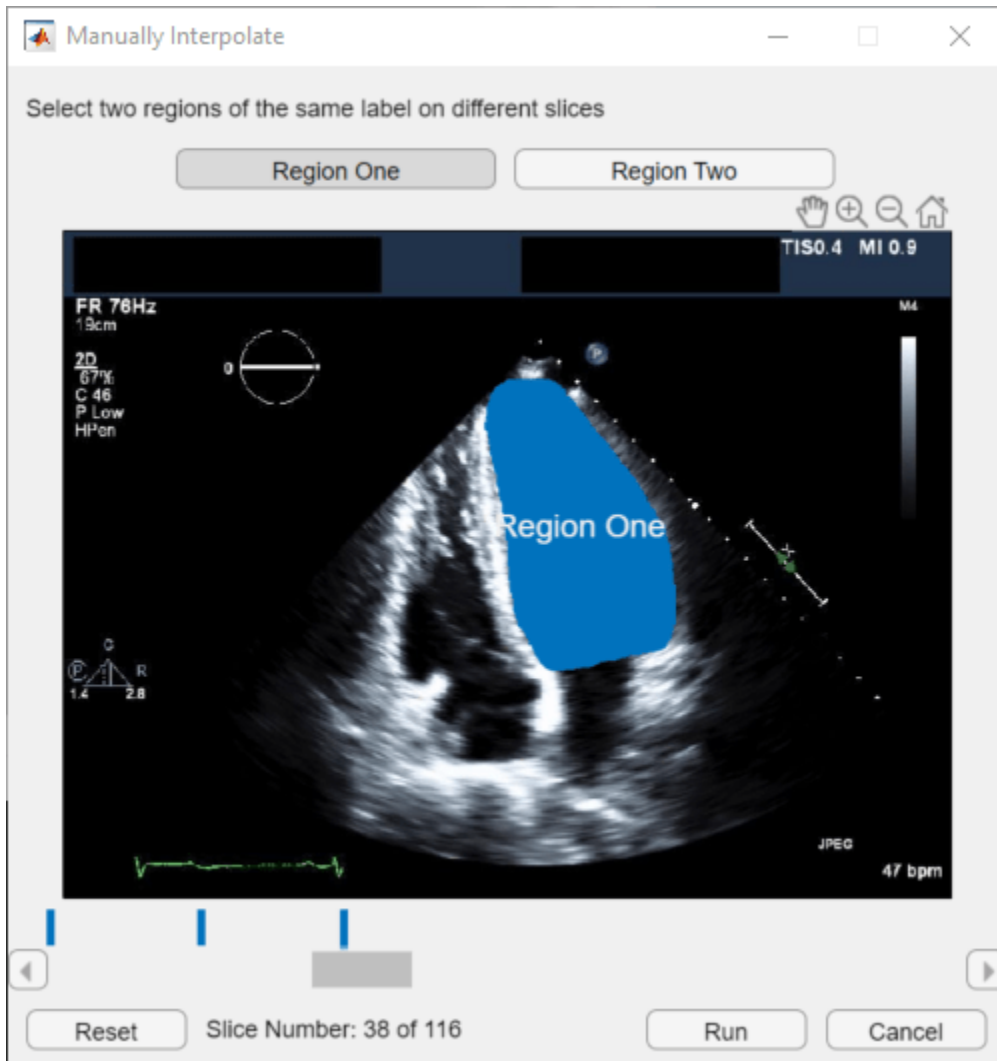
You can move through the image series and draw labels frame-by-frame, but the **Medical Image Labeler** app also provides interpolation tools in the **Draw** tab that can help you label an object between frames. Interpolation is most suitable between frames where the region of interest has a similar 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 Draw 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.

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 **Home** 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 1-10
- “Label 3-D Medical Image Using Medical Image Labeler” on page 5-10

## 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.

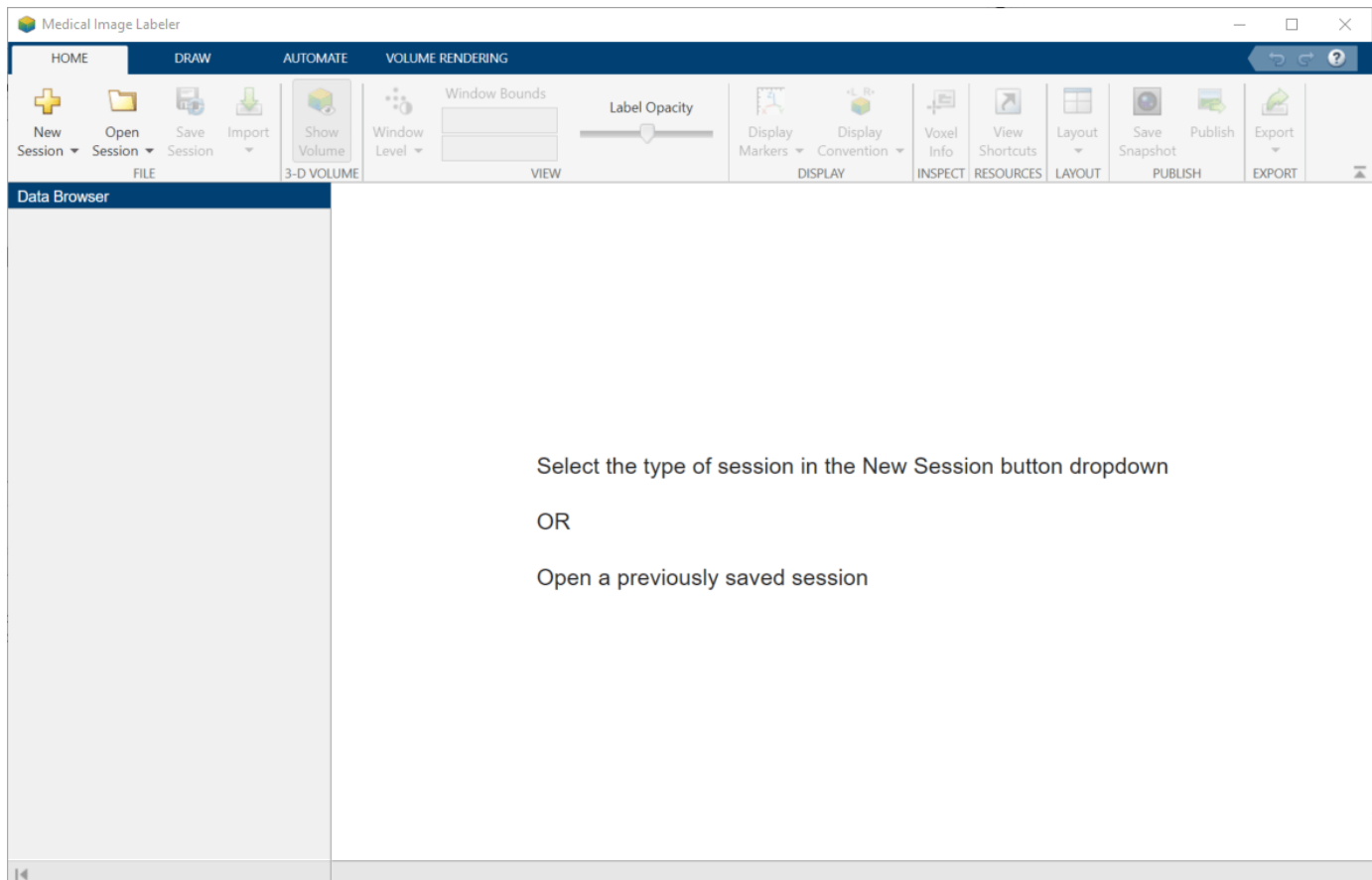
### Download Data to Label

This example labels chest CT data from a subset of the Medical Segmentation Decathlon data set [1 on page 5-18]. 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.






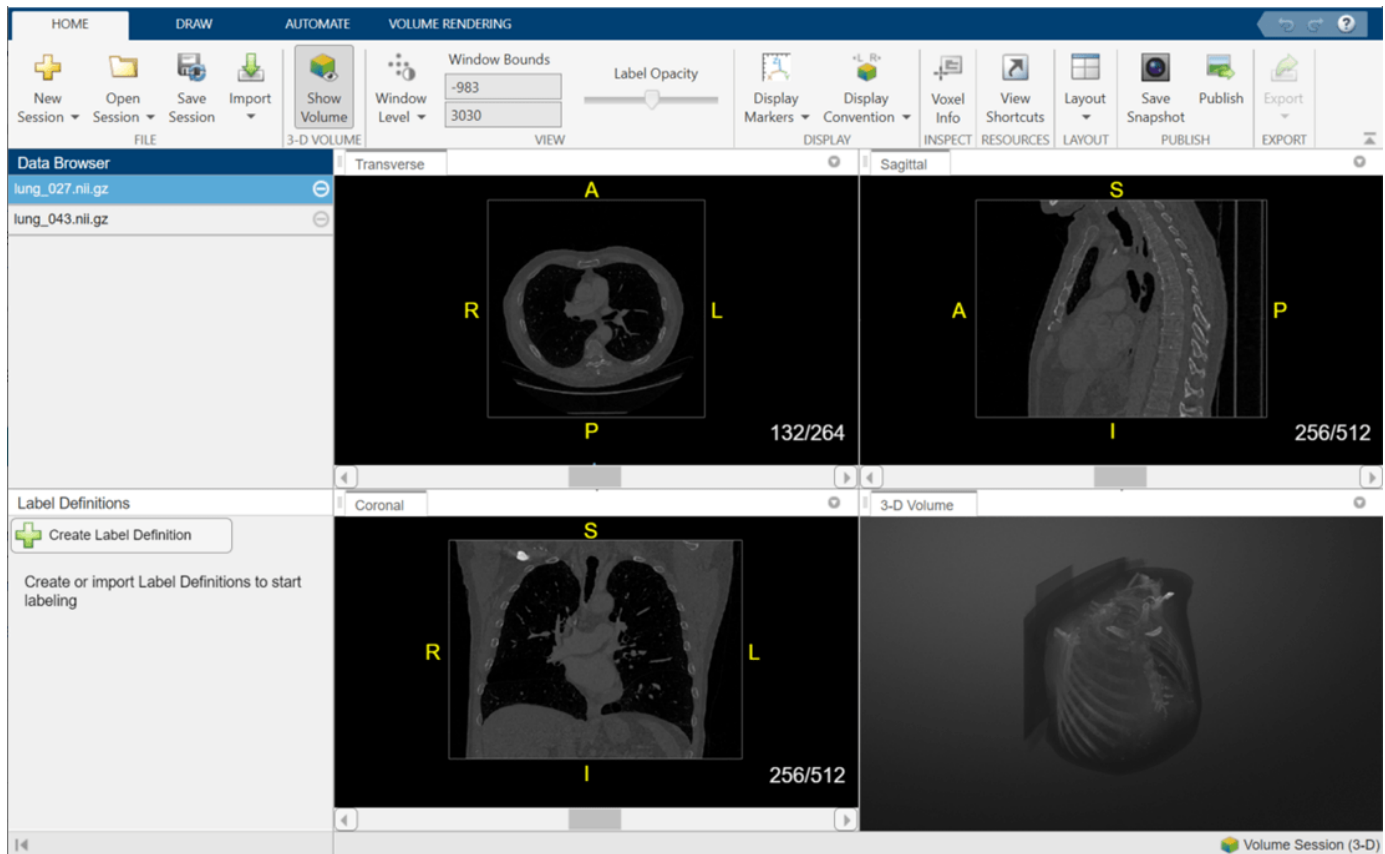
## Create New Volume Labeling Session

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 **Home** 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.

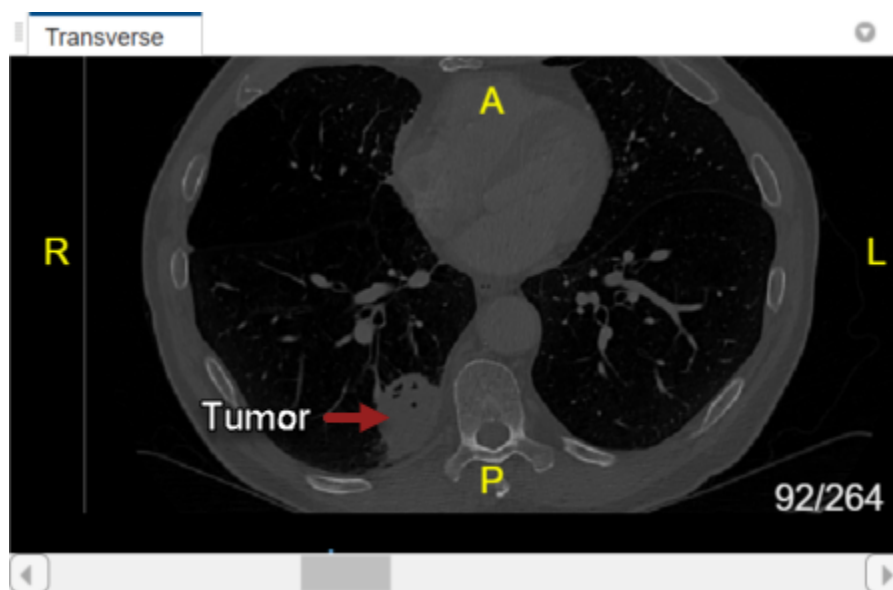
By default, the app displays the scan using the **Radiological** display convention, with the left side of the patient on the right side of the image. To display the left side of the patient on the left side of the image, on the app toolbar, click **Display Convention** and select **Neurological**.

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

**Window Level** tool in the **Home** 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

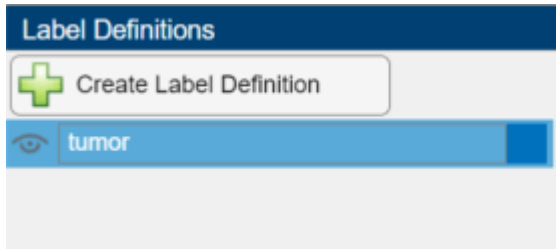
bounds are displayed in the app toolbar under **Window Bounds**. Click  to deactivate the tool. Changing the display window does not modify the image data.

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 78 and 105 in the **Transverse** slice pane.



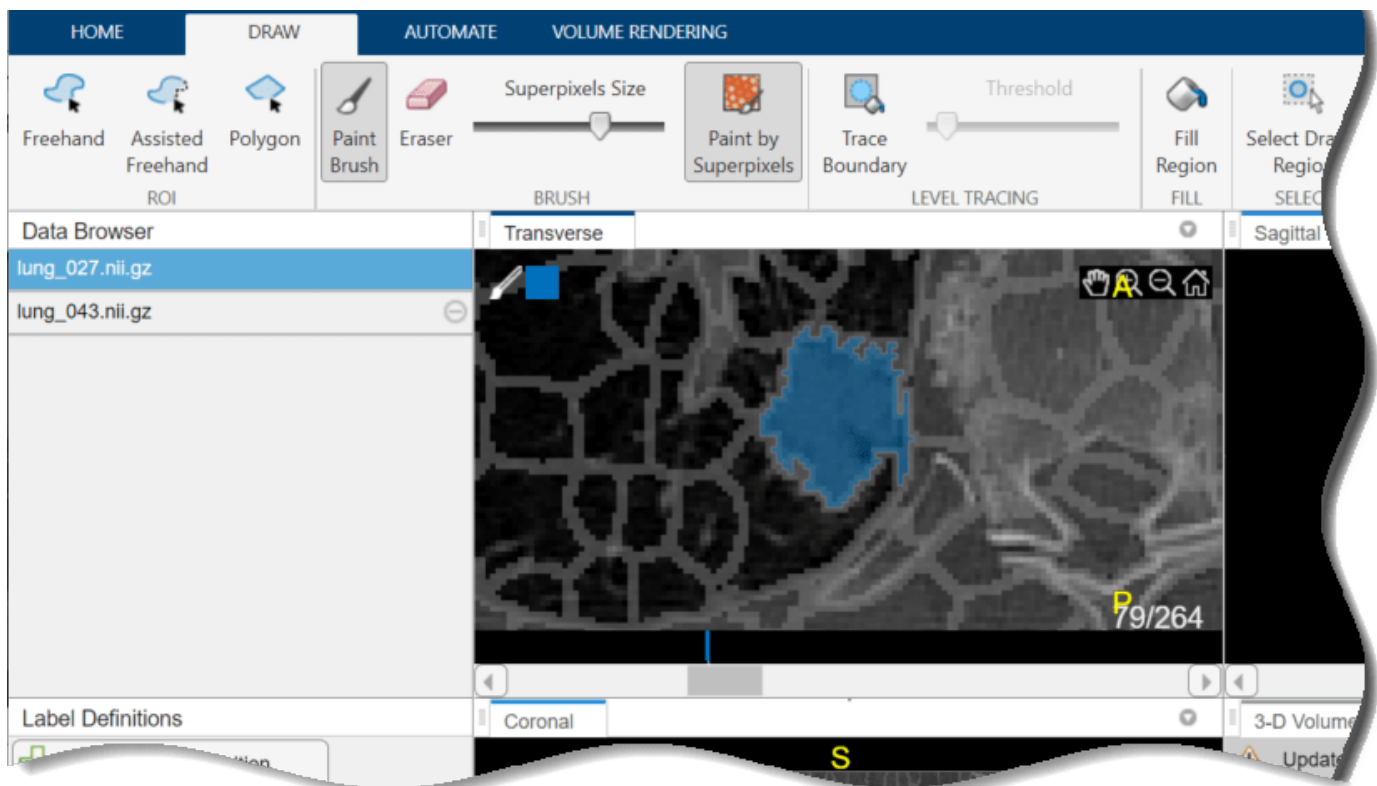
## Create Label Definitions

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 78. 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.

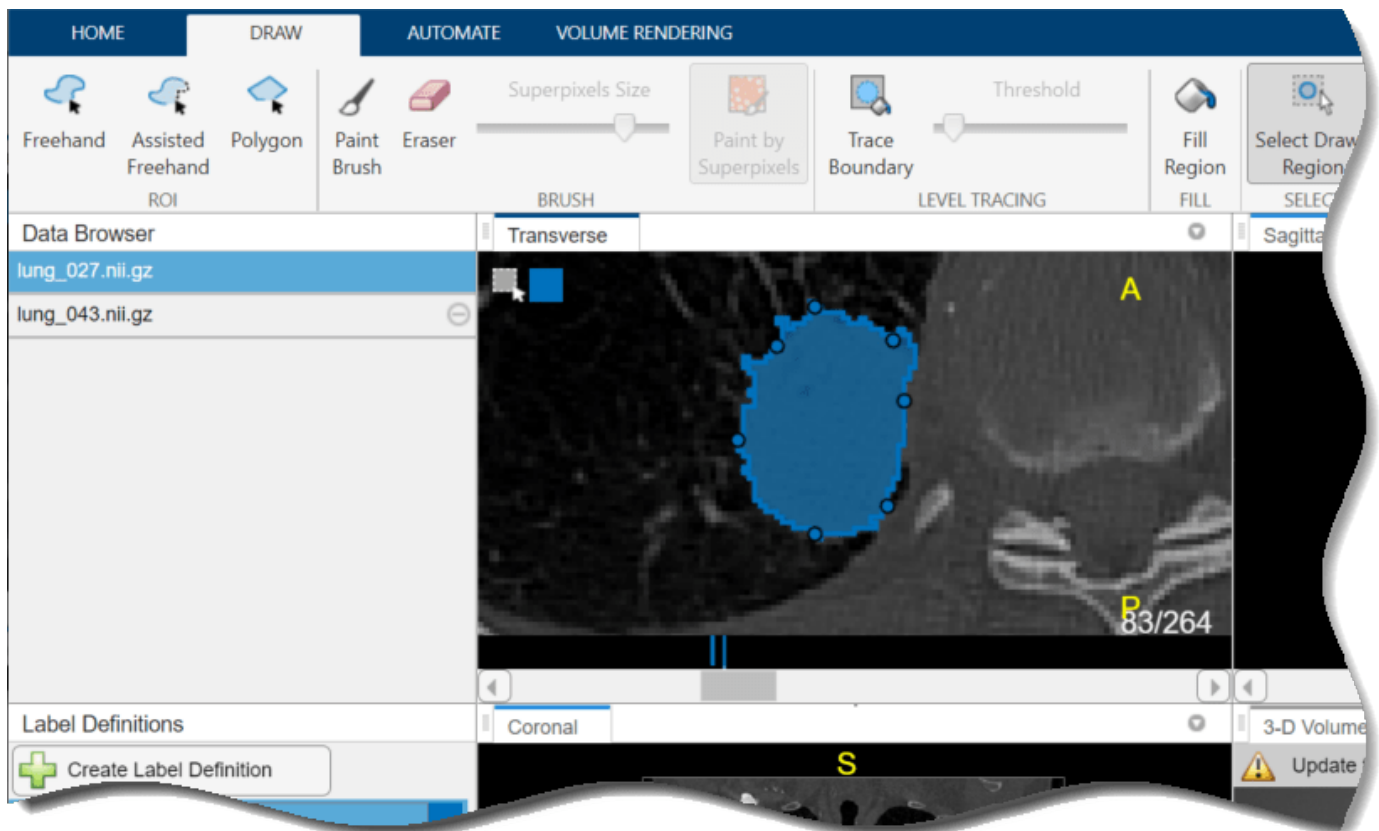


In addition to the drawing tools, you can add or refine label regions using the tools in the **Automate** tab of the app toolstrip. The app provides slice-based automation algorithms including **Active Contours**, **Adaptive Threshold**, **Dilate**, and **Erode**, as well as volume-based algorithms including **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.

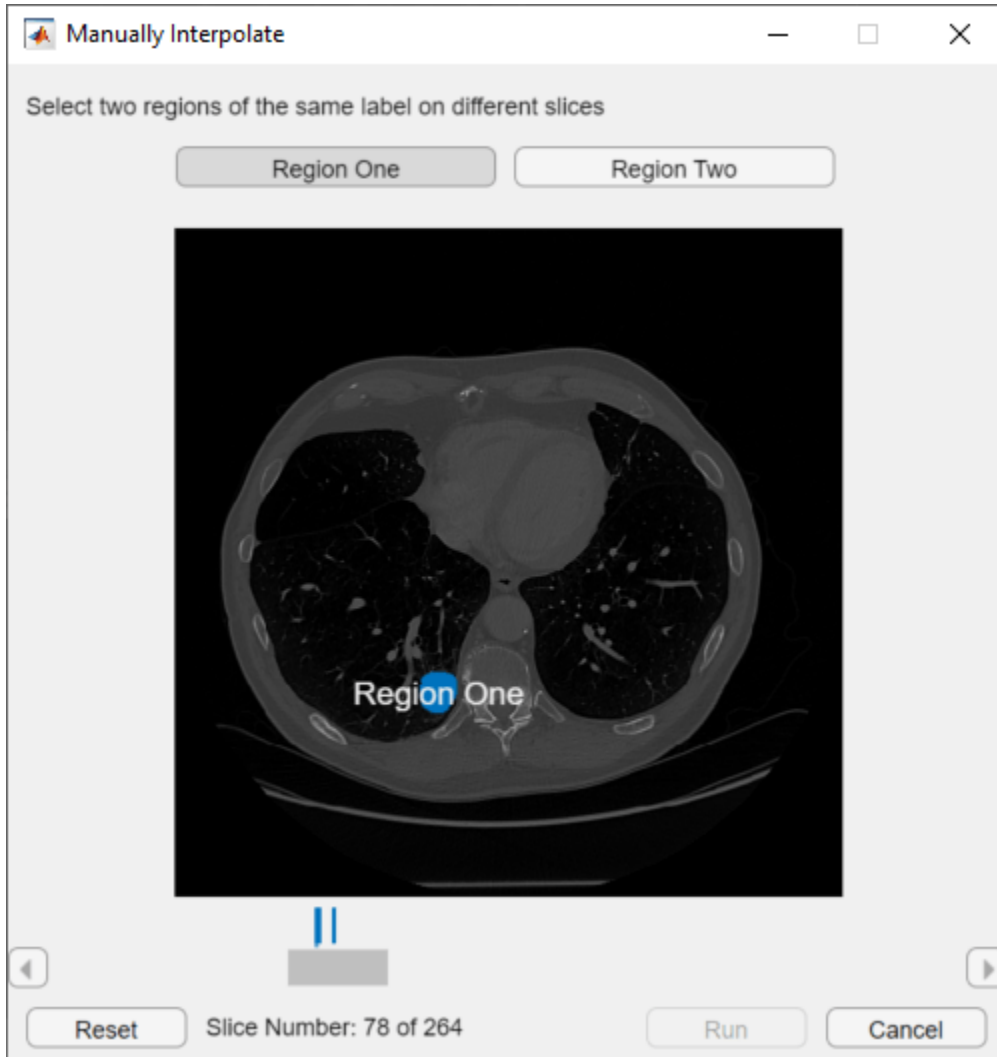
To use interpolation, you must first manually label a region on two slices. You have already labeled the tumor in slice 78. Use the same process to label the tumor in slice 83. If the **Auto Interpolate** button is not active, make sure the labeled region is selected by clicking **Select Drawn Region** in the app toolstrip 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 78 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 78, 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 **Home** 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.

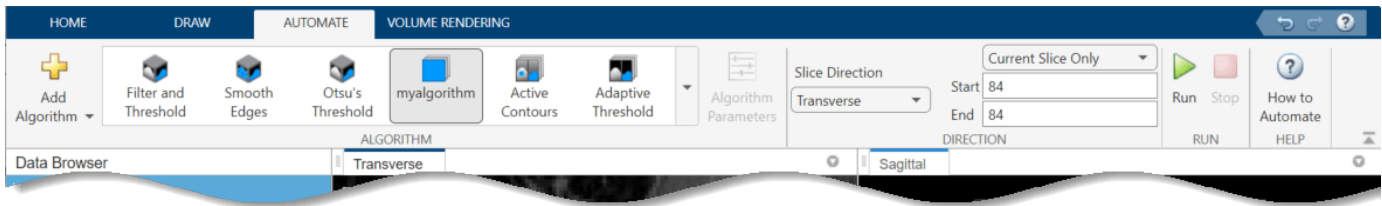
- Remove labels from individual pixels — In the **Draw** tab, use the **Eraser** tool.
- Remove labels from one connected region in a slice — Click **Select Drawn Region** and, in the slice pane, select the region from which you want to remove labels. Press **Delete**, or right-click and select **Delete**,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

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.

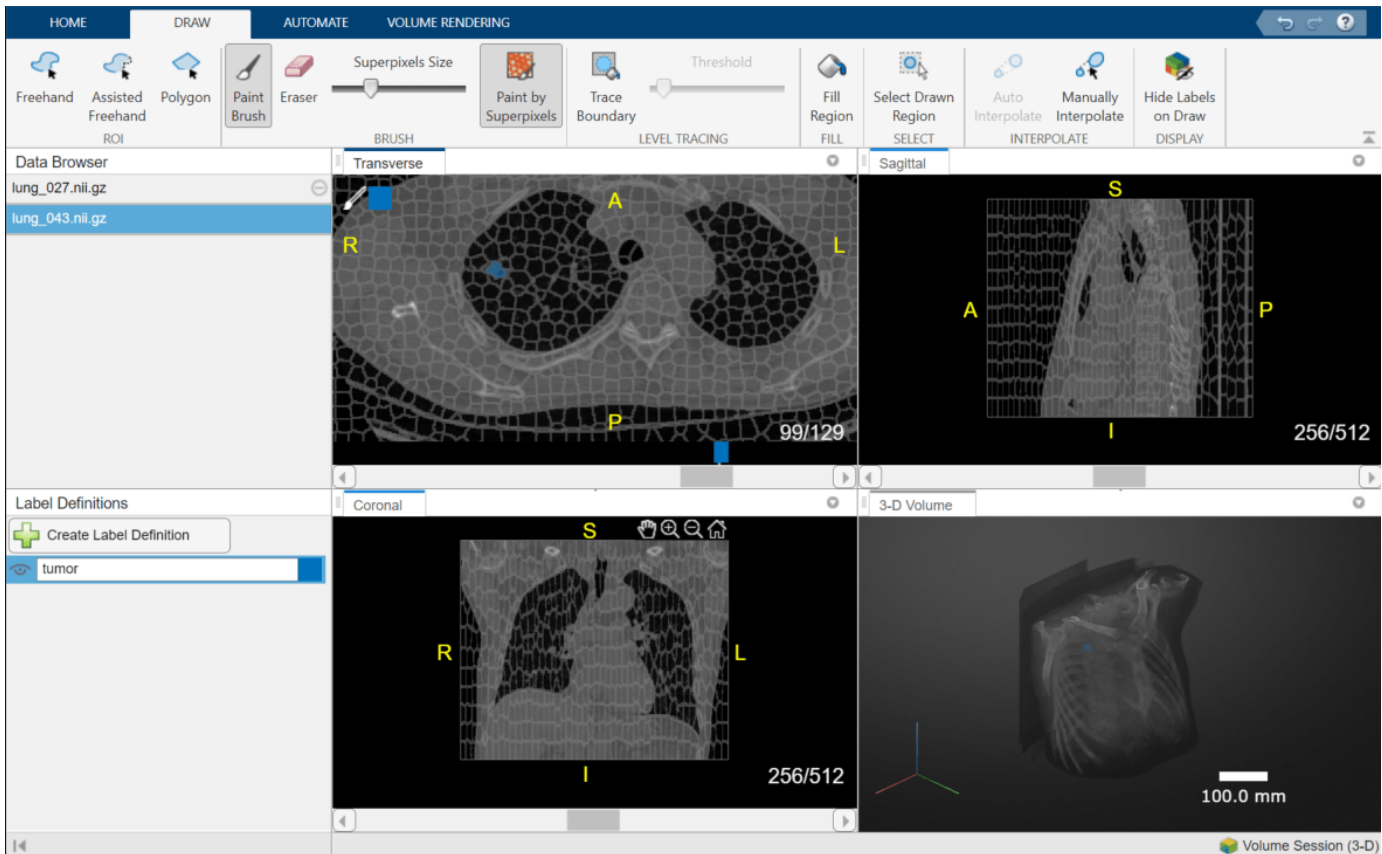


After testing your function on a single slice, you can run it on all of the slices, or a subset of the slices, in the selected direction. You can run it from the current slice to the end (the highest numbered slice) or from the current slice back to the beginning (slice 1). You can also specify a range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

### Label Next Image Volume

For this example, the labels for `lung_001.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 **Home** 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

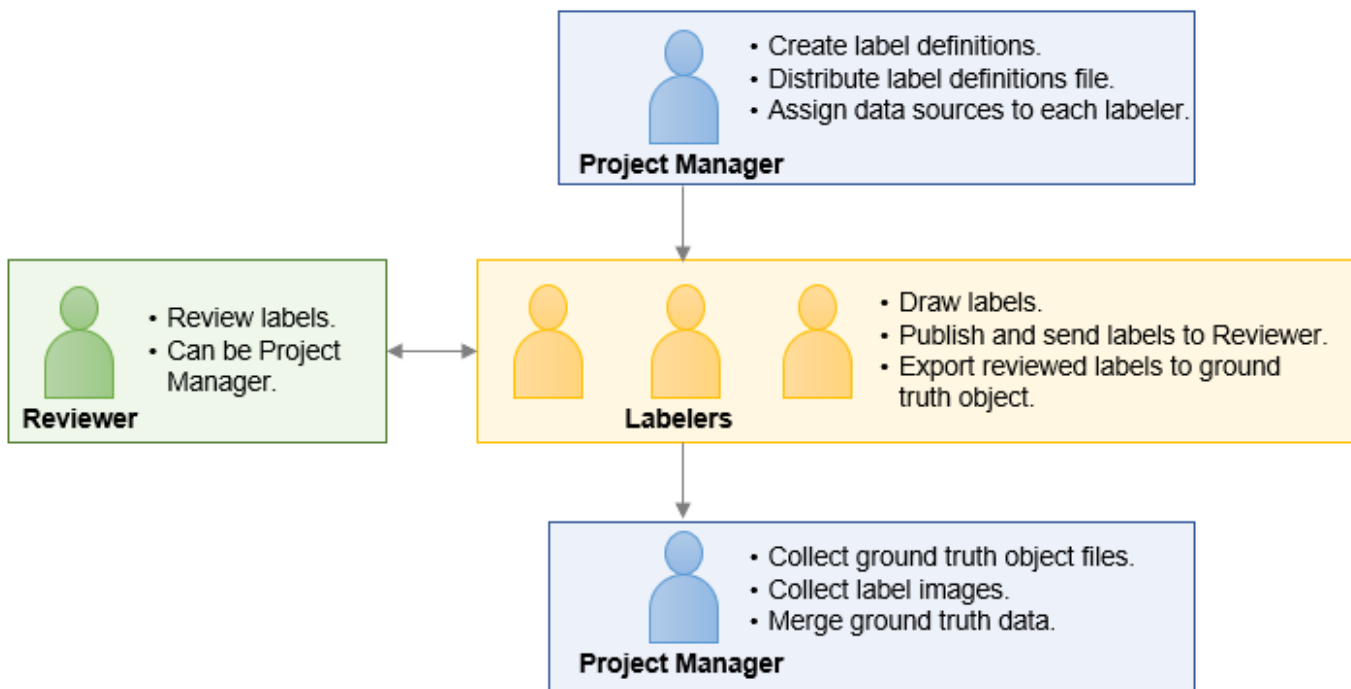
- "Get Started with Medical Image Labeler" on page 1-10
- "Label 2-D Ultrasound Series Using Medical Image Labeler" on page 5-2



## 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).

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 original intensity images to be labeled are data source images. 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 5-20
- 2 “Label Data and Publish Labels for Review (Labeler)” on page 5-21
- 3 “Export Ground Truth Data and Send to Project Manager (Labeler)” on page 5-23
- 4 “Inspect Labeled Images (Reviewer)” on page 5-23
- 5 “Collect, Merge, and Create Training Data (Project Manager)” on page 5-24

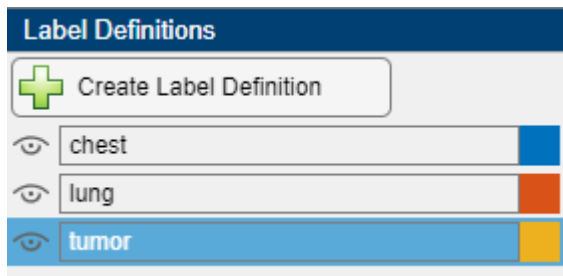
## Create Label Definitions and Assign Data to Labelers (Project Manager)

### 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`).

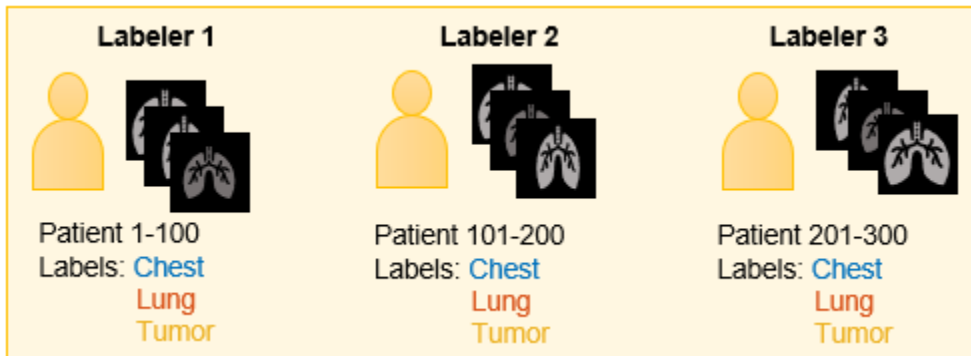


- 5 Click **Export** and, under **Label Definitions**, select **To File** to export the label definitions to a MAT file.

### 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.



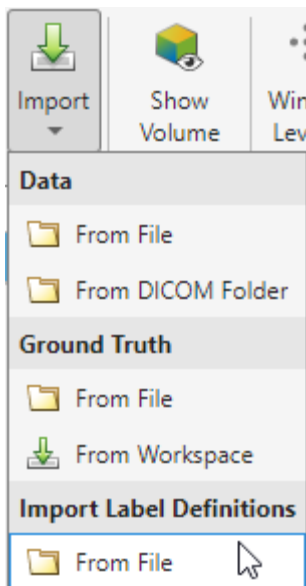
## Label Data and Publish Labels for Review (Labeler)

### 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 **Home** tab of the app toolbar, use the **Label Opacity** slider to set the transparency of the labels.
  - c On the **Home** 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 **Home** 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 **Home** 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 **Home** 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 1-10
- “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





# 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

## 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"> <li>• Single DICOM file.</li> <li>• Single NIFTI file.</li> </ul> <p><b>Note</b> 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"> <li>• 2-D label images are MAT files, regardless of the data source file format.</li> <li>• If a data source has no labels, the corresponding element of <code>LabelData</code> is an empty string, "".</li> </ul>

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"> <li>• Directory of DICOM files corresponding to one volume.</li> <li>• Single DICOM file.</li> <li>• Single NIFTI file.</li> <li>• Single NRRD file.</li> </ul> <p><b>Note</b> 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"> <li>• 3-D label images are NIFTI files, regardless of the data source file format.</li> <li>• If a data source has no labels, the corresponding element of <code>LabelData</code> is an empty string, "".</li> </ul>

## 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 5-19

## 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 a multiframe ultrasound image series. To train a semantic segmentation network, you must have pairs of data source images and label images, stored in an `imageDatastore` object and `pixelLabelDatastore` object, 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. For illustrative purposes, 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 images 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 filenames from the `groundTruthMedical` object.

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

Remove any data sources that are missing label images, if present.

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

Extract the label definitions from the `groundTruthMedical` object. Add a 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 each frame of the image series into an individual MAT file 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 an `imageDatastore` object.

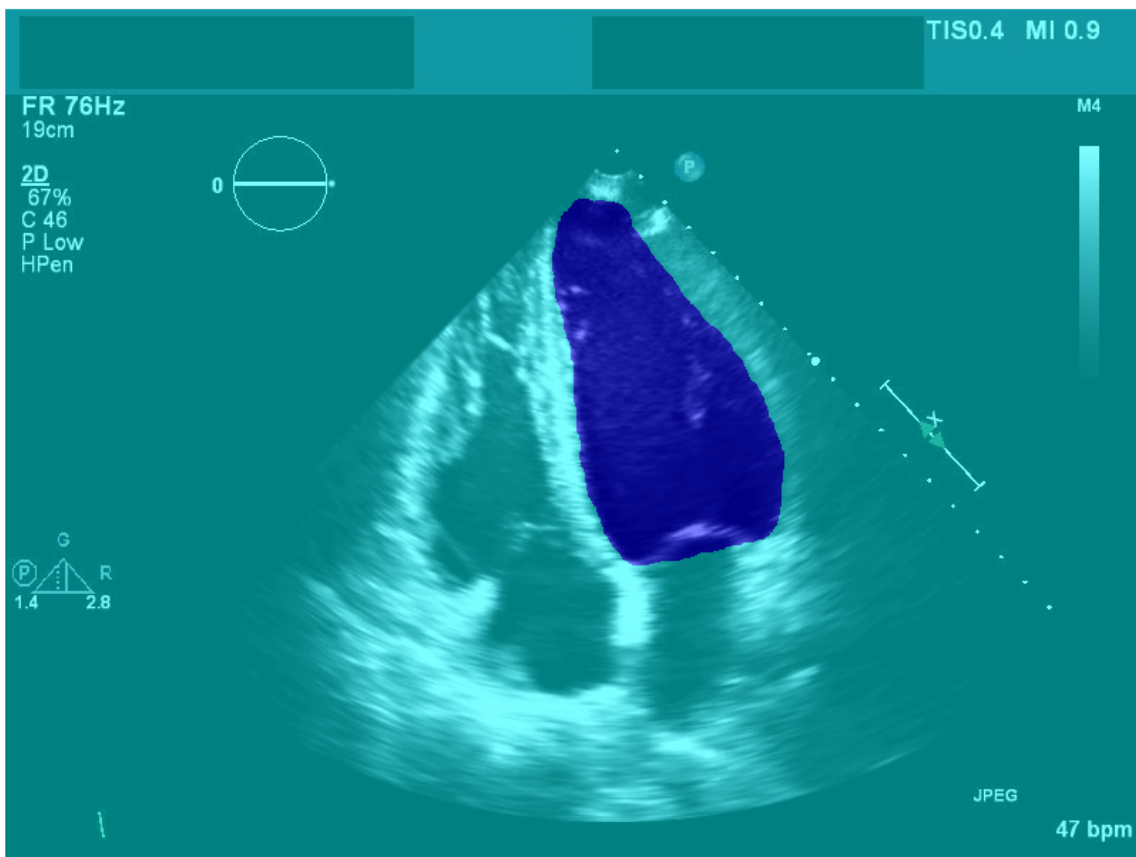
```
imds = imageDatastore(newDataSource, ...
    ReadFcn=@readFramesLabels, ...
    FileExtensions=[".mat", ".dcm"]);
```

Load the individual MAT file label images into a `pixelLabelDatastore` (Computer Vision Toolbox) object. 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 its corresponding 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 object 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 separate 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);

% Assume that the third dimension contains frames taken at different times
numFrames = size(labelData.labels,3);
newlabelData = cell(numFrames,1);

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

```
    save(matFileName, "data");  
    newlabelData{frame} = string(matFileName);  
end  
  
end
```

The `readFramesLabels` function reads data from the `data` variable of 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, consisting of 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 filenames from the `groundTruthMedical` object.

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

Remove data sources that are missing label images.

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

Extract the label definitions from the `groundTruthMedical` object. Add a 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 files, or as a directory of DICOM files. Because 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` object. 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) object using `niftiread` as the read function.

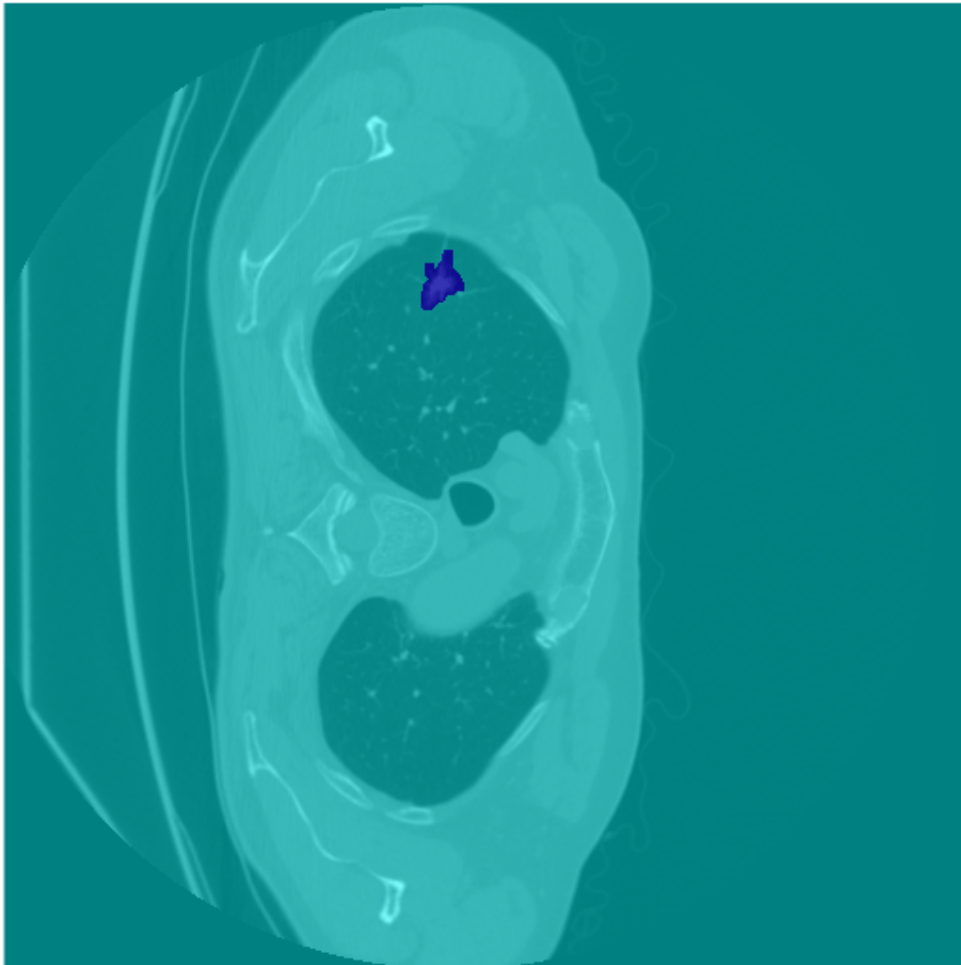
```
pxds = pixelLabelDatastore(cellstr(labelData), labelDefs.Name, labelDefs.PixelLabelID, ...  
    ReadFcn=@niftiread, ...  
    FileExtensions=[".nii", ".gz"]);
```

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 intensity values of the CT slice with `labeloverlay`, 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` object 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 operations specified by the `centerCropImageAndLabel` supporting function defined at the end of this example. Depending on your application and network, your preprocessing workflow may require additional steps, such as resampling or registration. For more information about preprocessing, see “Medical Image Preprocessing” on page 4-2.

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

### Supporting Functions

The `convertMultifileDICOMs` function reads a cell array of data source filenames, 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 `jitterImageIntensityAndWarp` 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. "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

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

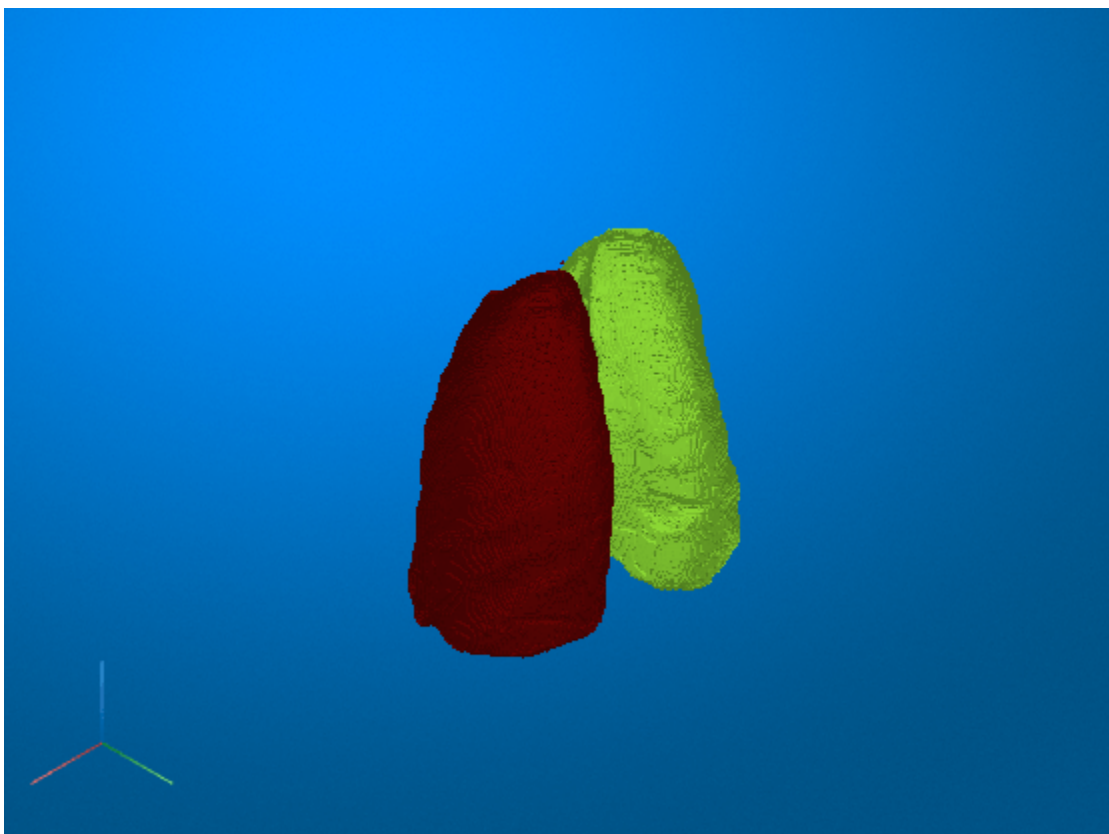
## Related Examples

- "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 of MATLAB.
```

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.

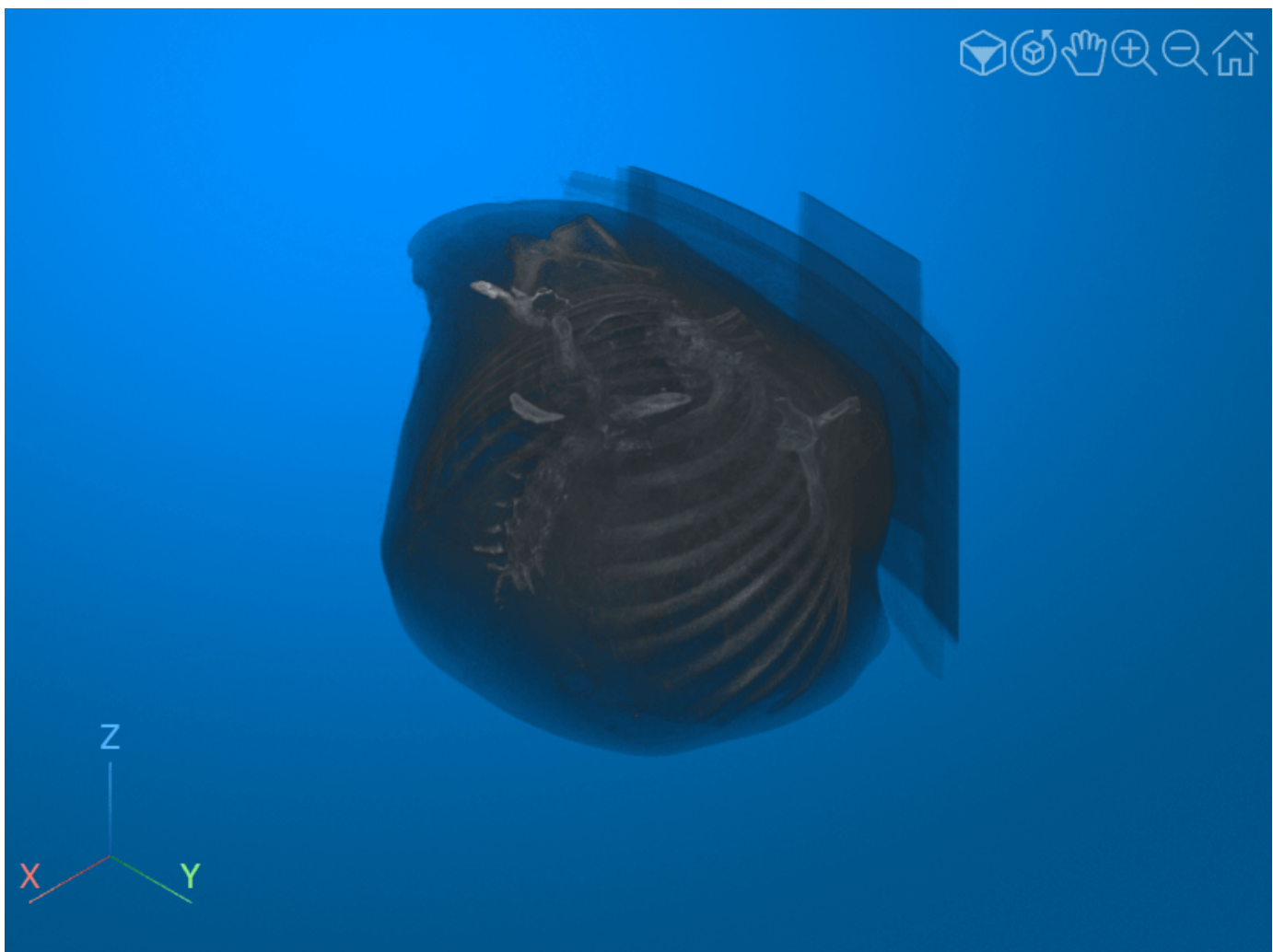
```
fileName = fullfile(dataFolder, "lung_027.nii.gz");
```

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.

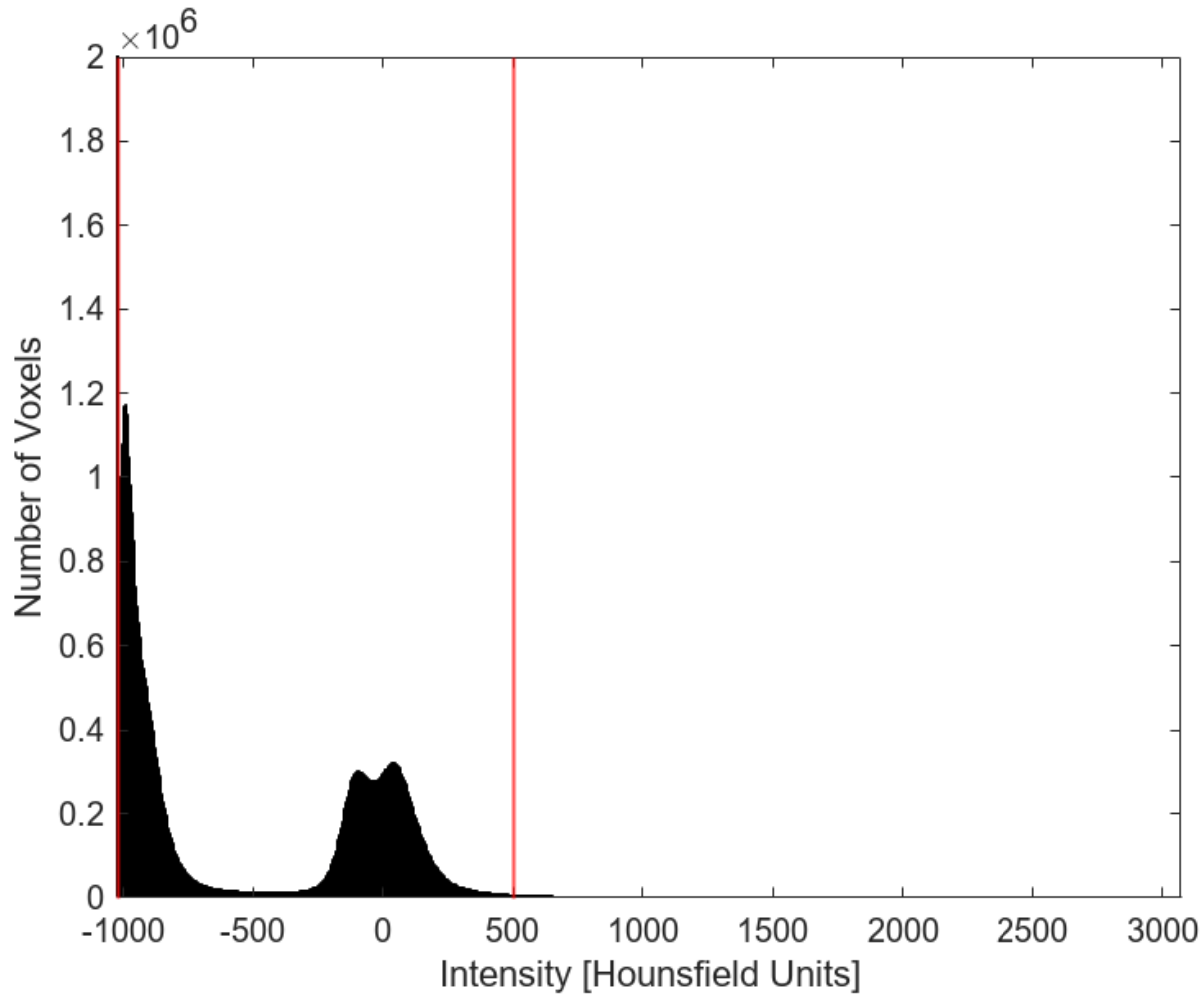
```
imshow(volAligned(:,:,150),[])
```



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.

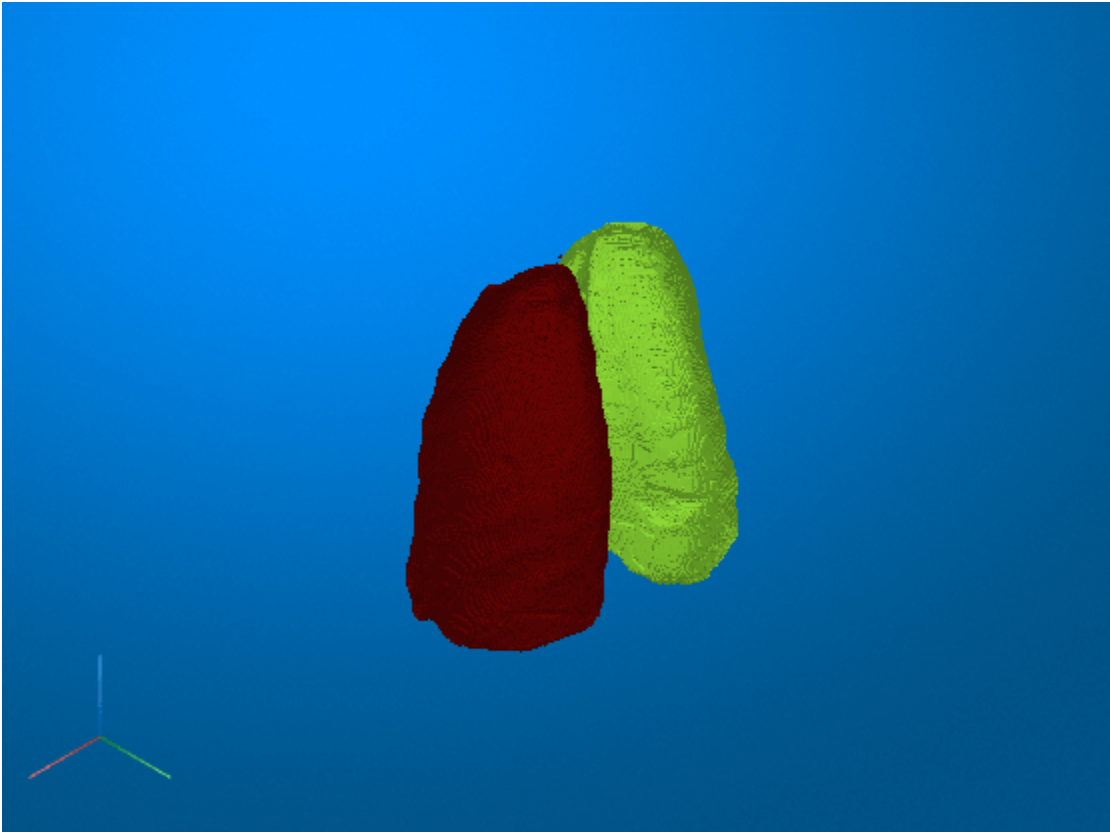
During postprocessing, the `lungSeg` helper function applies a mode filter to the network output to smooth the segmentation labels using the `modefilt` function. You can set the size of the mode filter by using the `modeFilt` name-value argument of `lungSeg`. The default filter size is `[9 9 9]`.

```
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.

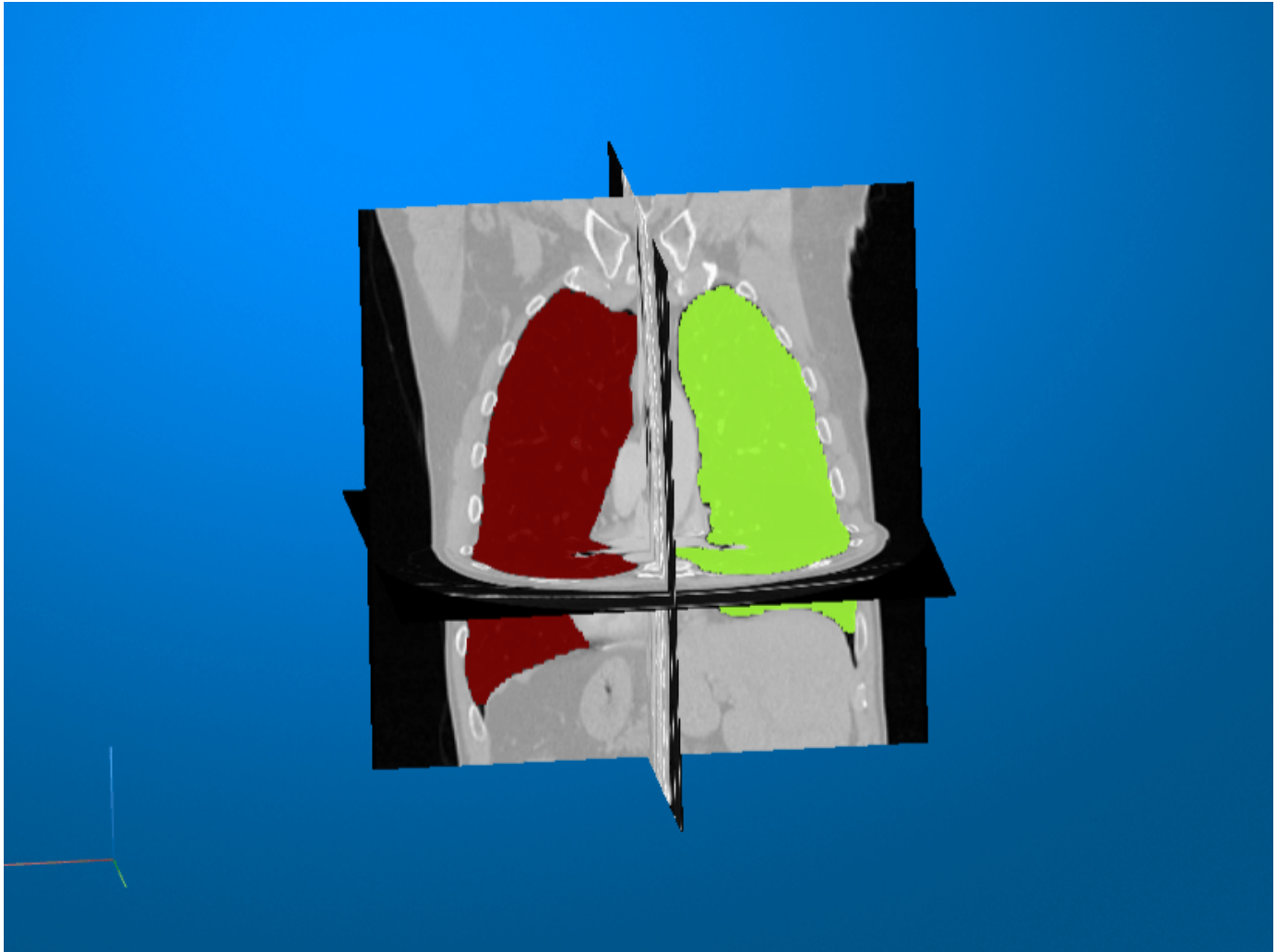
```
volshow(volInp, OverlayData=labelOut, ...  
        Alphamap=0, ...  
        OverlayAlphamap=0.9, ...  
        RenderingStyle="GradientOpacity");
```



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

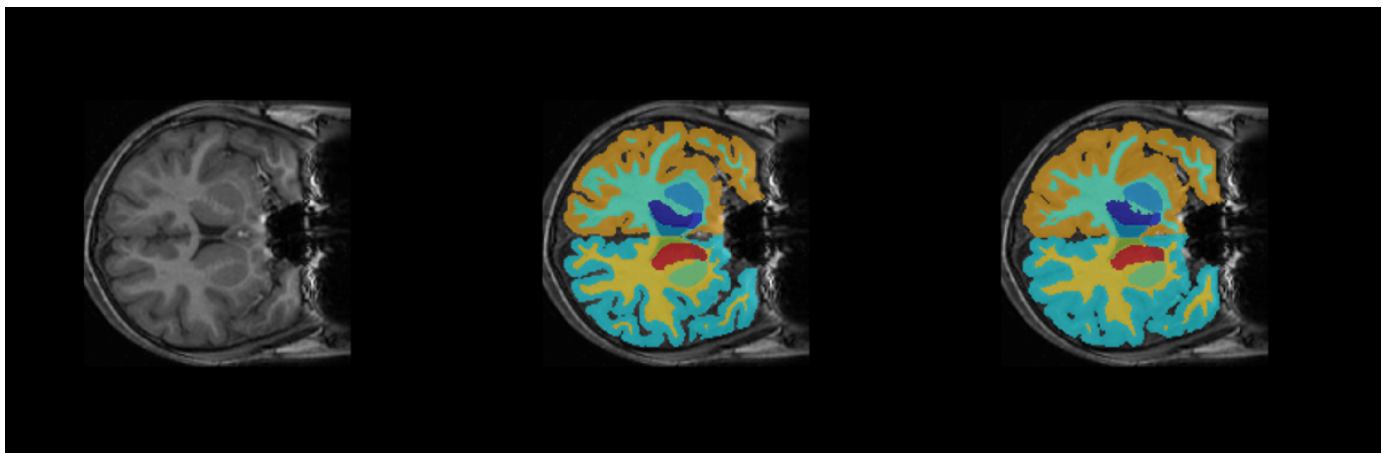
- "Segment Lungs from 3-D Chest Scan"
- "Brain MRI Segmentation Using Pretrained 3-D U-Net Network" on page 6-24

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

This example shows how to segment 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 6-31], 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 6-31] [3 on page 6-31]. 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 ~32 MB.

Create a folder in which to store the data set. In this example, a folder named `brainSegData` created within the `tempdir` directory has been used as `dataDir`.

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

To download the data, go to the UMass Medical CANDI website. Click the **Download** buttons next to "**Segmentation File (16.0 MB)**" and "**Anatomical Scan (16.0 MB)**" to download the ground truth



label and MRI scan file, respectively. Save the anatomical scan in the folder specified in `dataDir`. Save the segmentation file in a folder named `groundTruth` created within the folder specified in `dataDir`.

### 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/trainedBrainSy";
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");
metaData = niftiinfo(imFile);
X = 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;
[X1,cropIdx,imSize] = preProcessBrainCANDIData(X,metaData,cropSize,resample);
inputSize = size(X1);
```

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).

```
X2 = dLarray(X1,"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 contains no loss information. The
```

```
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: []  
  
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:
```

```

Layers: [60x1 nnet.cnn.layer.Layer]
Connections: [63x2 table]
InputNames: {'UNET_input'}
OutputNames: {1x0 cell}

```

## 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,X2);
```

### 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(X1);
    flippedData = flip(flippedData,2);
    flippedData = flip(flippedData,1);
    flippedData = darray(flippedData,"SSCB");
    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.

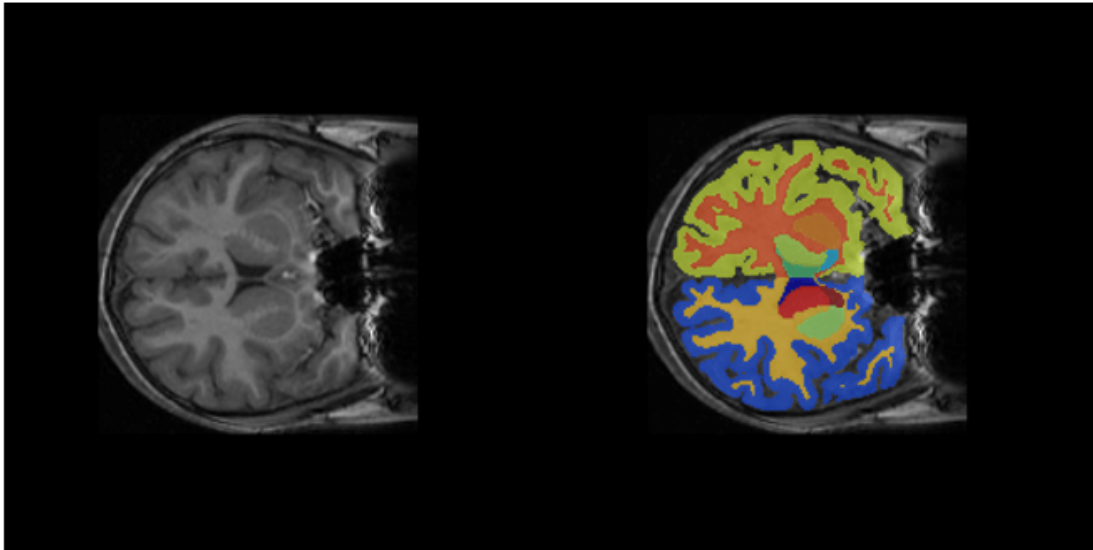
- Segmentation — Assign each voxel to the label class with the greatest confidence score for that voxel.
- Resizing — Resize the segmentation map to the original input volume size. Resizing the label image allows you to visualize the labels as an overlay on the grayscale MRI volume.
- Alignment — Rotate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

```
B = labeloverlay(rescale(X(:,:,80)),predictedSegMaps(:,:,80),"IncludedLabels",2:32);
figure
montage({rescale(X(:,:,80)), 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=".nii",...
    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.

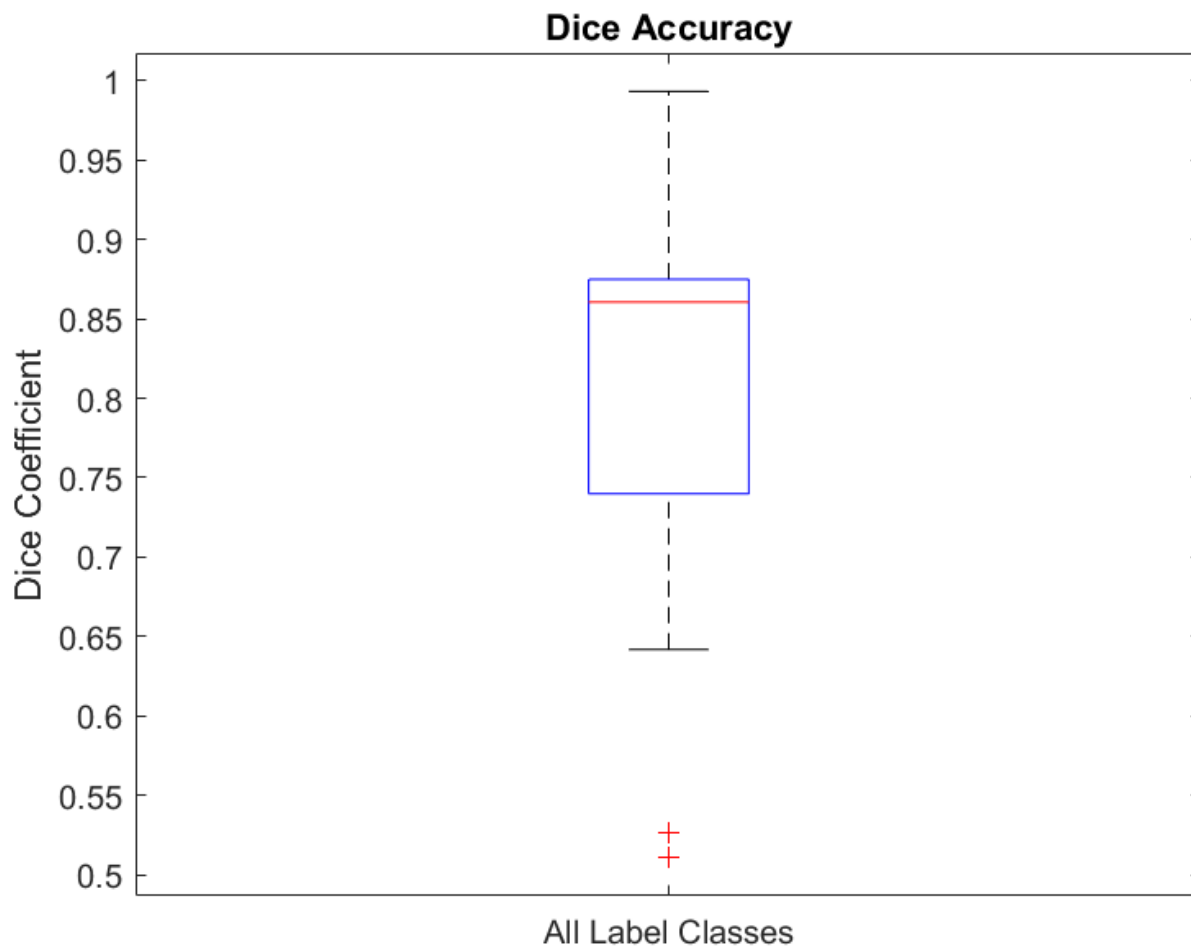
```
diceResult = zeros(length(classNames),1);  
for j = 1:length(classNames)  
    diceResult(j) = dice(groundTruthLabel==classNames(j), ...  
        predictedSegMaps==classNames(j));  
end
```

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
```

This figure shows a boxplot that visualizes statistics about the Dice indices across all the label classes. The red lines in the plot show 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.



If you have a Statistics and Machine Learning Toolbox™ license, then you can use the `boxplot` (Statistics and Machine Learning Toolbox) function to visualize statistics about the Dice indices. To create a boxplot, set `createBoxplot` to `true`.

```
createBoxplot =  ;
if createBoxplot
    figure
    boxplot(diceResult)
    title("Dice Accuracy")
    xticklabels("All Label Classes")
    ylabel("Dice Coefficient")
end
```

### Supporting Functions

The `niftiReader` helper function is a custom read function for reading a NIfTI file in a datastore.

```
function data = niftiReader(filename)
% Read nifti file and convert to data type uint8.
```

```
% Copyright 2022 The MathWorks, Inc.
```

```
    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] Child and Adolescent NeuroDevelopment Initiative, "1. HC\_001" (2008). *CANDI Neuroimaging Data*. Dataset 1. [https://escholarship.umassmed.edu/cs\\_schizbull08/1](https://escholarship.umassmed.edu/cs_schizbull08/1).

[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](#) | [boxplot](#)

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

Unzip the TAR GZ file completely. Load the pretrained network.

```
gunzip(fullfile(dataDir,"breastTumorDeepLabV3.tar.gz"),dataDir);
untar(fullfile(dataDir,"breastTumorDeepLabV3.tar"),dataDir);
exampleDir = fullfile(dataDir,"breastTumorDeepLabV3");
pretrained = load(fullfile(exampleDir,"breast_seg_deepLabV3.mat"));
trainedNet = pretrained.trainedNet;
```

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");
imshowpair(imTest,overlayImg,"montage");
```





### 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].

Go to the website of Prof. Aly Fahmy and click "**Breast Ultrasound Images Dataset (Dataset BUSI)**" and download the BUSI data in the folder specified by `dataDir`.

### Load Data

If the downloaded data set is zipped, unzip it in the folder specified by the `dataDir` variable. After you unzip the data set, the folder specified by `dataDir` contains a folder named `Dataset_BUSI_with_GT`.

```
if isfile(fullfile(dataDir, "Dataset_BUSI.zip"))
    unzip(fullfile(dataDir, "Dataset_BUSI.zip"), dataDir);
end
```

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.

```
imageDir = fullfile(dataDir, "Dataset_BUSI_with_GT");
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.

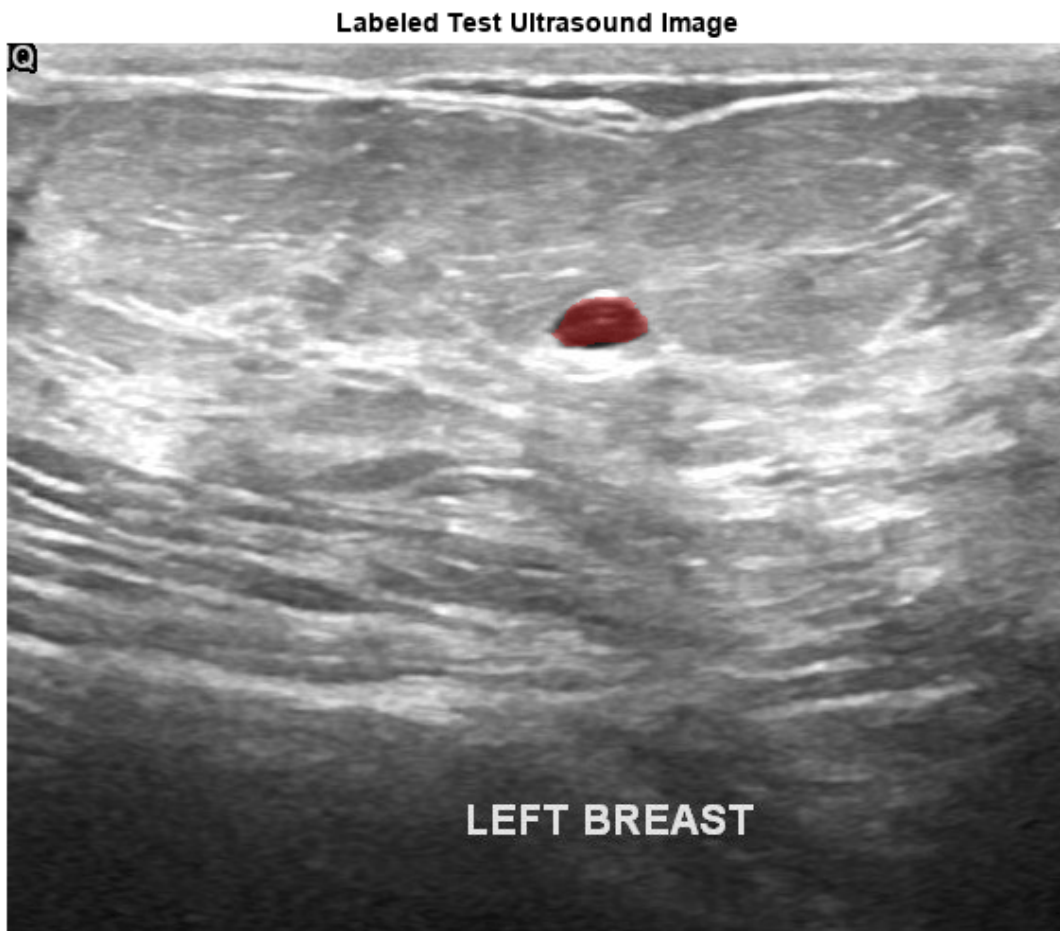
```
classNames = ["tumor", "background"];  
labelIDs = [1 0];  
numClasses = numel(classNames);  
pxds = pixelLabelDatastore(imageDir, classNames, labelIDs, IncludeSubfolders=true);
```

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 option 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 a Parallel Computing Toolbox™ license and CUDA®-enabled NVIDIA™ GPU. Training takes about four hours on a single-GPU system with 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.

```
tdsTest = transform(dsTest,@transformBreastTumorImageResize,IncludeInfo=true);
```

### Segment Test Data

Use the trained network for semantic segmentation of the test data set.

```
pxdsResults = semanticseg(tdsTest,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,tdsTest,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>
0.93925	0.89078	0.73802	0.90067	0.54716

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,tdsTest);
```

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 = 0.96338
```

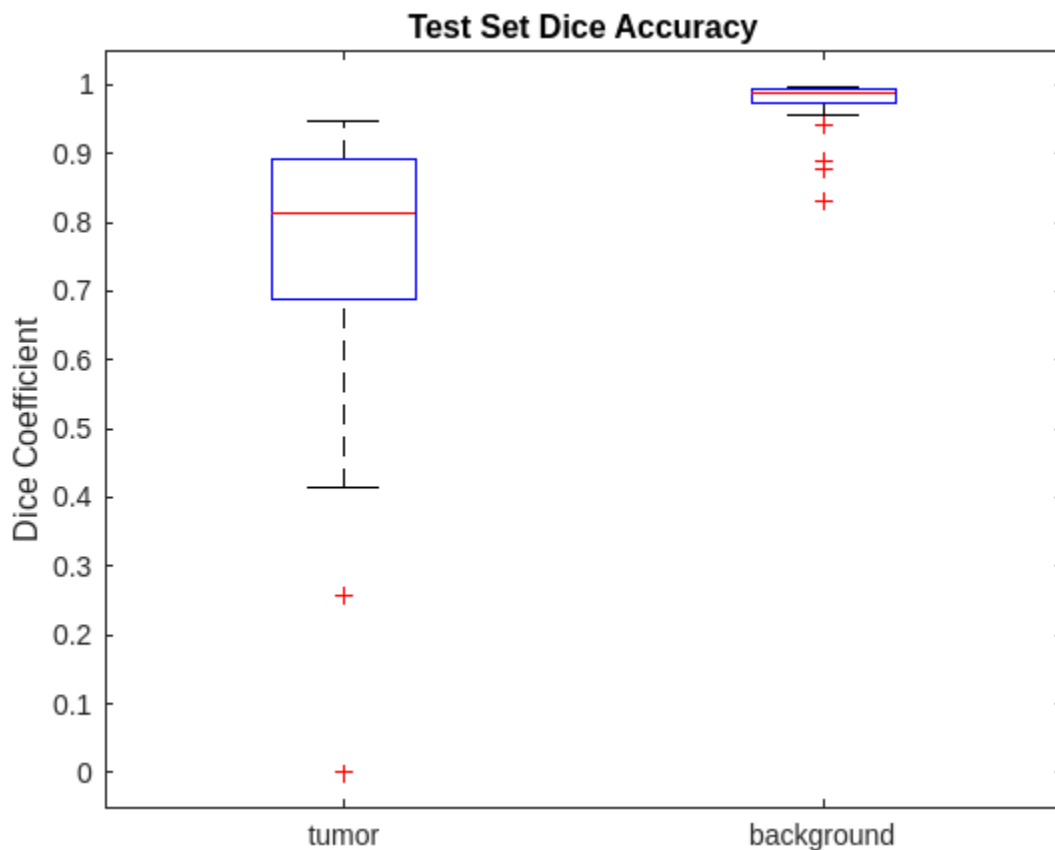
```
disp("Average Dice score of tumor across "+num2str(numTestImgs)+ ...
    " test images = "+num2str(mean(diceTumor)))
```

Average Dice score of tumor across 65 test images = 0.6867

```
disp("Median Dice score of tumor across "+num2str(numTestImgs)+ ...
    " test images = "+num2str(median(diceTumor)))
```

Median Dice score of tumor across 65 test images = 0.76217

This figure shows a boxplot that visualizes statistics about the Dice indices across the set of 65 test samples. The red lines in the plot show the median Dice index for the classes. 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.



If you have a Statistics and Machine Learning Toolbox™ license, then you can use the `boxplot` (Statistics and Machine Learning Toolbox) function to visualize statistics about the Dice indices across all your test images. To create a boxplot, set `createBoxplot` to `true`.

```
createBoxplot =  ;
if createBoxplot
    figure
    diceResult = [diceTumor diceBackground];
    boxplot(diceResult)
    title("Test Set Dice Accuracy")
    xticklabels(classNames)
```

```
    ylabel("Dice Coefficient")  
end
```

## 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](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)

