

2.2-1 創建和探索用於圖像分類的數據存儲

%% 創建 ImageDatastore object

% 創建一個 ImageDatastore 與所有 .tif 關聯的對象 MATLAB(R) 路徑及其子文件夾中的文件。使用文件夾名稱作為標籤名稱。

```
%%  
imds = imageDatastore(fullfile(matlabroot,'toolbox','matlab'),...  
    'IncludeSubfolders',true,'FileExtensions','.tif','LabelSource','foldernames')
```

imds =

ImageDatastore with properties:

```
    Files: {  
            '...\matlab\toolbox\matlab\demos\example.tif';  
            '...\matlab\toolbox\matlab\imagesci\corn.tif'  
          }  
    Labels: [demos; imagesci]  
    ReadSize: 1  
    ReadFcn: @readDatastoreImage  
Read information of the imds  
    imds.Files  
    imds.Labels
```

➤ 指定文件夾中之影像

Create an ImageDatastore object containing four images, and preview the first image.

```
imds =  
imageDatastore({'street1.jpg','street2.jpg','peppers.png','corn.tif'})  
imds =
```

ImageDatastore with properties:

```
    Files: {  
            '...\matlab\toolbox\matlab\demos\street1.jpg';  
            '...\matlab\toolbox\matlab\demos\street2.jpg';  
            '...\matlab\toolbox\matlab\imagesci\peppers.png'  
            ... and 1 more  
          }  
    ReadSize: 1  
    Labels: {}  
    ReadFcn: @readDatastoreImage  
  
imshow(preview(imds));
```

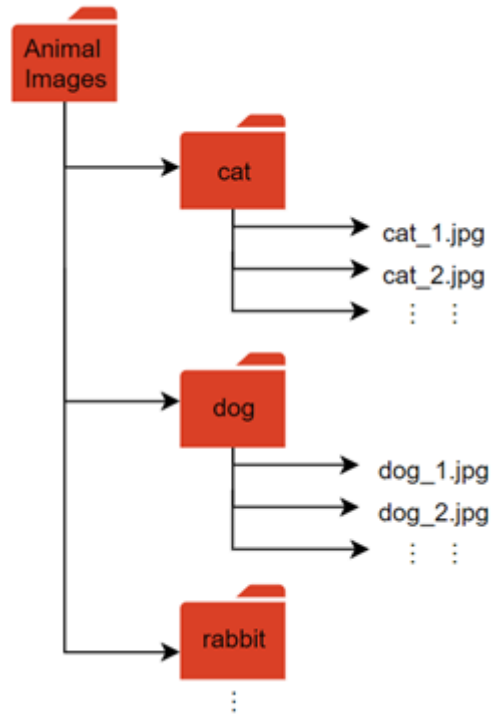


Create and Explore Datastore for Image Classification

此範例說明如何創建、讀取和擴充(augemntation)圖像數據存儲以用於訓練深度學習網絡。特別是，此範例展示瞭解如何從圖像集合創建一個 **ImageDatastore** 對象、讀取和提取數據存儲的屬性，以及創建一個增強的 **ImageDatastore** 以供在訓練期間使用。

Create Image Datastore

使用 **imageDatastore** 對象來管理無法完全放入內存的大量圖像。大量圖像集合在深度學習應用程序中很常見，這些應用程序通常涉及對數千張標記圖像的訓練。這些圖像通常存儲在一個文件夾中，子文件夾包含每個類別的圖像。

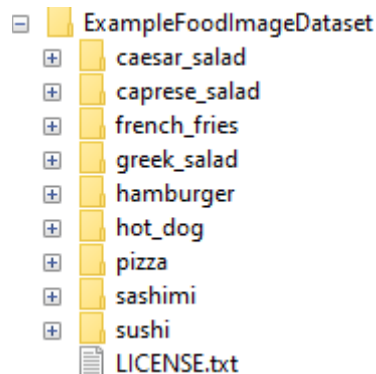


Download Data Set

此範例使用 Example Food Images 數據集，其中包含九個類別的 978 張食物照片，大小約為 77 MB。從 MathWorks 網站下載 ExampleFoodImageDataset.zip 文件，然後解壓縮該文件。

```
zipFile =  
matlab.internal.examples.downloadSupportFile('nnet','data/ExampleFoodImageDataset.zip');  
filepath = fileparts(zipFile);  
dataFolder = fullfile(filepath,'ExampleFoodImageDataset');  
unzip(zipFile,dataFolder);
```

此數據集中的圖像分為每個類別的子文件夾。



從路徑及其子文件夾中的圖像創建圖像數據存儲。使用文件夾名稱作為標籤名稱

```
foodImds = imageDatastore(dataFolder, ...  
    'IncludeSubfolders',true, ...  
    'LabelSource','foldernames');
```

Properties of Datastore

提取數據存儲的屬性。找出觀察總數。該數據集有 **978** 個觀察值，分為九個類別。

Extract the properties of the datastore.

Find the total number of observations. This data set has 978 observations split into nine classes.

```
numObs = length(foodImds.Labels)
```

```
numObs = 978
```

找出每個類別的觀察次數。您可以看到此數據集在每個類別中不包含相同數量的觀察值。

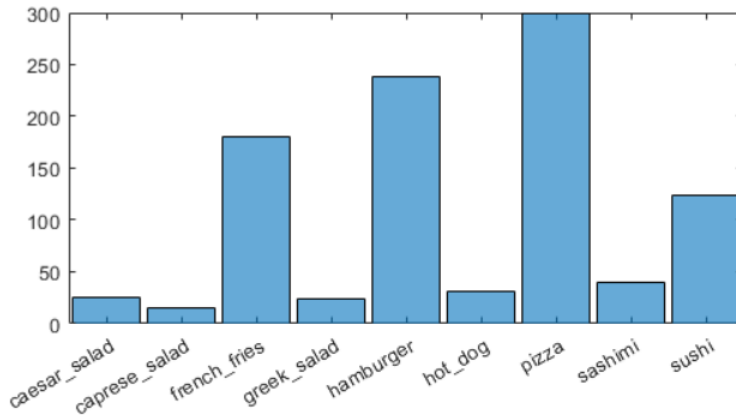
```
numObsPerClass = countEachLabel(foodImds)
```

```
numObsPerClass=9x2 table  
    Label    Count  
-----  
caesar_salad    26  
caprese_salad   15  
french_fries    181  
greek_salad     24  
hamburger       238  
hot_dog         31  
pizza           299  
sashimi         40  
sushi          124
```

您還可以使用直方圖可視化類標籤的分佈 直方圖（**foodImds.Labels**）
設置（**gca**，**'TickLabelInterpreter'**，**'無'**）

You can also visualize the distribution of the class labels using a histogram.

```
histogram(foodImds.Labels)
set(gca, 'TickLabelInterpreter', 'none')
```



Explore Datastore

探索圖像數據存儲，通過查看數據存儲中隨機選擇的圖像，來檢查數據是否符合預期。

Check that the data is as expected by viewing a random selection of images from the datastore.

```
numObsToShow = 8;
idx = randperm(numObs,numObsToShow);
imshow(imtile(foodImds.Files(idx), 'GridSize',[2 4], 'ThumbnailSize',[100 100]))
```



You can also view images that belong to a specific class.

```
class = "pizza";
idxClass = find(foodImds.Labels == class);
idx = randsample(idxClass,numObsToShow);
imshow(imtile(foodImds.Files(idx),'GridSize',[2 4],'ThumbnailSize',[100 100]));
```

To take a closer look at individual images in your datastore or folder, use the [Image Browser](#) app.

Image Augmentation

增強使您能夠訓練網絡不受圖像數據失真的影響。例如，您可以向輸入圖像添加隨機旋轉，以便網絡對旋轉的存在保持不變。`augmentedImageDatastore` 對象提供了一種方便的方法，可以將一組有限的增強應用到 2-D 圖像以解決分類問題。

定義擴充方案。該方案應用 $[-90,90]$ 度之間的隨機旋轉和 $[1,2]$ 之間的隨機縮放。增強數據存儲在訓練期間自動將圖像大小調整為 `inputSize` 值。

```
imageAugmenter = imageDataAugmenter( ...
    'RandRotation',[-90 90], ...
    'RandScale',[1 2]);

inputSize = [100 100];
```

Using the augmentation scheme, define the augmented image datastore.

```
augFoodImds = augmentedImageDatastore(inputSize,foodImds, ...
    'DataAugmentation',imageAugmenter);
```

The augmented datastore contains the same number of images as the original image datastore.

```
augFoodImds.NumObservations
```

當您使用增強圖像數據存儲作為訓練圖像的來源時，數據存儲會隨機擾動每個時期的訓練數據，其中一個時期是訓練算法對整個訓練數據集的完整傳遞。因此，每個 **epoch** 使用的數據集略有不同，但每個 **epoch** 的實際訓練圖像數量沒有變化。

Visualize Augmented Data

要用於訓練網絡的增強圖像數據之可視化。

隨機播放數據存儲 **shuffle**(洗牌)。

```
augFoodImds = shuffle(augFoodImds);
```

augmentedImageDatastore: **apply to the transform**(應用轉換) 對圖像在讀取數據存儲，並且不將轉換後的圖像存儲在內存中。因此，每次閱讀相同的圖像時，您都會看到定義的增強的隨機組合。

使用 **read** 函數讀取擴充數據存儲的一個子集。

```
subset1 = read(augFoodImds);
```

Reset the datastore to its state before calling **read** and read a subset of the datastore again.

```
reset(augFoodImds)  
subset2 = read(augFoodImds);
```

Display the two subsets of the augmented images.

```
imshow(imtile(subset1.input,'GridSize',[2 4]))  
imshow(imtile(subset2.input,'GridSize',[2 4]))
```





您可以看到兩個實例都顯示具有不同變換的相同圖像。對圖像應用變換在深度學習應用程序中很有用，因為您可以在隨機更改的圖像版本上訓練網絡。這樣做會使網絡暴露於該類圖像的不同變體，並使其能夠學習對圖像進行分類，即使它們具有不同的視覺屬性。

創建數據存儲對象後，使用 **Deep Network Designer** 應用程序或 **trainNetwork** 函數來訓練圖像分類網絡。有關示例，請參閱使用預訓練網絡進行遷移學習。

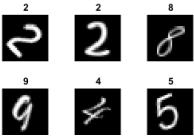

有關為深度學習應用程序預處理圖像的更多信息，請參閱為深度學習預處理圖像。您還可以通過使用變換和組合功能應用更高級的增強，例如不同級別的亮度或飽和度。有關詳細信息，請參閱用於深度學習的數據存儲。

Copyright 2020 The MathWorks, Inc.



Data Sets for Deep Learning









Use these data sets to get started with deep learning applications.

Image Data Sets


Data Set	Description	Task
<div>Digits</div> <div></div>	<p>The digits data set consists of 10,000 synthetic grayscale images of handwritten digits. Each image is 28-by-28 pixels and has an associated label denoting which digit the image represents (0–9). Each image has been rotated by a certain angle. When loading the images as arrays, you can also load the rotation angle of the image.</p> <p>Load the digits data as in-memory numeric arrays using the <code>digitTrain4DArrayData</code> and <code>digitTest4DArrayData</code> functions.</p> <pre>[XTrain,YTrain,anglesTrain] = digitTrain4DArrayData; [XTest,YTest,anglesTest] = digitTest4DArrayData;</pre>	Image classification and image regression
	<p>For examples showing how to process this data for deep learning, see Monitor Deep Learning Training Progress and Train Convolutional Neural Network for Regression.</p>	
	<p>Load the digits data as an image datastore using the <code>imageDatastore</code> function and specify the folder containing the image data.</p> <pre>dataFolder = fullfile(toolboxdir('nnet'),'nndemos','nndatasets','DigitDataset'); imds = imageDatastore(dataFolder, ... 'IncludeSubfolders',true, ... 'LabelSource','foldernames');</pre> <p>For an example showing how to process this data for deep learning, see Create Simple Deep Learning Network for Classification.</p>	Image classification
<div>MNIST</div> <div></div> <div>(Representative example)</div>	<p>The MNIST data set consists of 70,000 handwritten digits split into training and test partitions of 60,000 and 10,000 images, respectively. Each image is 28-by-28 pixels and has an associated label denoting which digit the image represents (0–9).</p> <p>Download the MNIST files from http://yann.lecun.com/exdb/mnist/ and load the data set into the workspace. To load the data from the files as MATLAB arrays, place the files in the working directory then use the helper functions <code>processImagesMNIST</code> and <code>processLabelsMNIST</code>, which are used in the example Train Variational Autoencoder (VAE) to Generate Images.</p> <pre>oldpath = addpath(fullfile(matlabroot,'examples','nnet','main')); filenameImagesTrain = 'train-images-idx3-ubyte.gz'; filenameLabelsTrain = 'train-labels-idx1-ubyte.gz'; filenameImagesTest = 't10k-images-idx3-ubyte.gz'; filenameLabelsTest = 't10k-labels-idx1-ubyte.gz'; XTrain = processImagesMNIST(filenameImagesTrain); YTrain = processLabelsMNIST(filenameLabelsTrain); XTest = processImagesMNIST(filenameImagesTest); YTest = processLabelsMNIST(filenameLabelsTest);</pre> <p>For an example showing how to process this data for deep learning, see Train Variational Autoencoder (VAE) to Generate Images.</p> <p>To restore the path, use the <code>path</code> function.</p> <pre>path(oldpath);</pre>	Image classification

Data Set	Description	Task
<p>Omniglot</p> 	<p>The Omniglot data set contains character sets for 50 alphabets, divided into 30 sets for training and 20 sets for testing. Each alphabet contains a number of characters, from 14 for Ojibwe (Canadian Aboriginal syllabics) to 55 for Tifinagh. Finally, each character has 20 handwritten observations.</p> <p>Download and extract the Omniglot data set [1] from https://github.com/brendenlake/omniglot. Set downloadFolder to the location of the data.</p> <pre> downloadFolder = tempdir; url = "https://github.com/brendenlake/omniglot/raw/master/python"; urlTrain = url + "/images_background.zip"; urlTest = url + "/images_evaluation.zip"; filenameTrain = fullfile(downloadFolder,"images_background.zip"); filenameTest = fullfile(downloadFolder,"images_evaluation.zip"); dataFolderTrain = fullfile(downloadFolder,"images_background"); dataFolderTest = fullfile(downloadFolder,"images_evaluation"); if ~exist(dataFolderTrain,"dir") fprintf("Downloading Omniglot training data set (4.5 MB)... ") websave(filenameTrain,urlTrain); unzip(filenameTrain,downloadFolder); fprintf("Done.\n") end if ~exist(dataFolderTest,"dir") fprintf("Downloading Omniglot test data (3.2 MB)... ") websave(filenameTest,urlTest); unzip(filenameTest,downloadFolder); fprintf("Done.\n") end </pre> <p>To load the training and test data as image datastores, use the imageDatastore function. Specify the labels manually by extracting the labels from the file names and setting the Labels property.</p> <pre> imdsTrain = imageDatastore(dataFolderTrain, ... 'IncludeSubfolders',true, ... 'LabelSource','none'); files = imdsTrain.Files; parts = split(files,filesep); labels = join(parts(:,(end-2):(end-1)),'_'); imdsTrain.Labels = categorical(labels); imdsTest = imageDatastore(dataFolderTest, ... 'IncludeSubfolders',true, ... 'LabelSource','none'); files = imdsTest.Files; parts = split(files,filesep); labels = join(parts(:,(end-2):(end-1)),'_'); imdsTest.Labels = categorical(labels); </pre> <p>For an example showing how to process this data for deep learning, see Train a Siamese Network to Compare Images.</p>	Image similarity

Data Set	Description	Task
<p>Flowers</p>  <p>Image credits: [3] [4] [5] [6]</p>	<p>The Flowers data set contains 3670 images of flowers belonging to five classes (<i>daisy</i>, <i>dandelion</i>, <i>roses</i>, <i>sunflowers</i>, and <i>tulips</i>).</p> <p>Download and extract the Flowers data set [2] from http://download.tensorflow.org/example_images/flower_photos.tgz. The data set is about 218 MB. Depending on your internet connection, the download process can take some time. Set <code>downloadFolder</code> to the location of the data.</p> <pre> url = 'http://download.tensorflow.org/example_images/flower_photos.tgz'; downloadFolder = tempdir; filename = fullfile(downloadFolder, 'flower_dataset.tgz'); dataFolder = fullfile(downloadFolder, 'flower_photos'); if ~exist(dataFolder, 'dir') fprintf("Downloading Flowers data set (218 MB)... ") websave(filename,url); untar(filename,downloadFolder) fprintf("Done.\n") end </pre> <p>Load the data as an image datastore using the <code>imageDatastore</code> function and specify the folder containing the image data.</p> <pre> imds = imageDatastore(dataFolder, ... 'IncludeSubfolders',true, ... 'LabelSource', 'foldernames'); </pre> <p>For an example showing how to process this data for deep learning, see Train Generative Adversarial Network (GAN).</p>	Image classification
<p>Example Food Images</p> 	<p>The Example Food Images data set contains 978 photographs of food in nine classes (<i>caesar_salad</i>, <i>caprese_salad</i>, <i>french_fries</i>, <i>greek_salad</i>, <i>hamburger</i>, <i>hot_dog</i>, <i>pizza</i>, <i>sashimi</i>, and <i>sushi</i>).</p> <p>Download the Example Food Images data set using the <code>downloadSupportFile</code> function and extract the images using the <code>unzip</code> function. This data set is about 77 MB. Depending on your internet connection, the download process can take some time.</p> <pre> fprintf("Downloading Example Food Image data set (77 MB)... ") filename = matlab.internal.examples.downloadSupportFile('nnet', ... 'data/ExampleFoodImageDataset.zip'); fprintf("Done.\n") filepath = fileparts(filename); dataFolder = fullfile(filepath, 'ExampleFoodImageDataset'); unzip(filename,dataFolder); </pre> <p>For an example showing how to process this data for deep learning, see View Network Behavior Using tsne.</p>	Image classification

Data Set	Description	Task
<p>CIFAR-10</p> <div>     </div> <p>(Representative example)</p>	<p>The CIFAR-10 data set contains 60,000 color images of size 32-by-32 pixels, belonging to 10 classes (<i>airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck</i>).</p> <p>There are 6000 images per class and the data set is split into a training set with 50,000 images and a test set with 10,000 images. This data set is one of the most widely used data sets for testing new image classification models.</p> <p>Download and extract the CIFAR-10 data set [7] from https://www.cs.toronto.edu/%7Ekriz/cifar-10-matlab.tar.gz. The data set is about 175 MB. Depending on your internet connection, the download process can take some time. Set downloadFolder to the location of the data.</p> <pre>url = 'https://www.cs.toronto.edu/~kriz/cifar-10-matlab.tar.gz'; downloadFolder = tempdir; filename = fullfile(downloadFolder, 'cifar-10-matlab.tar.gz'); dataFolder = fullfile(downloadFolder, 'cifar-10-batches-mat'); if ~exist(dataFolder, 'dir') fprintf("Downloading CIFAR-10 dataset (175 MB)... "); websave(filename, url); untar(filename, downloadFolder); fprintf("Done.\n") end</pre> <p>Convert the data to numeric arrays using the helper function loadCIFARData, which is used in the example Train Residual Network for Image Classification.</p> <pre>oldpath = addpath(fullfile(matlabroot, 'examples', 'nnet', 'main')); [XTrain, YTrain, XValidation, YValidation] = loadCIFARData(downloadFolder);</pre> <p>For an example showing how to process this data for deep learning, see Train Residual Network for Image Classification.</p> <p>To restore the path, use the path function.</p> <pre>path(oldpath);</pre>	Image classification
<p>MathWorks® Merch</p> <div>     </div>	<p>This is a small data set containing 75 images of MathWorks merchandise, belonging to five different classes (<i>cap, cube, playing cards, screwdriver, and torch</i>). You can use this data set to try out transfer learning and image classification quickly.</p> <p>The images are of size 227-by-227-by-3.</p> <p>Extract the MathWorks Merch data set.</p> <pre>filename = 'MerchData.zip'; dataFolder = fullfile(tempdir, 'MerchData'); if ~exist(dataFolder, 'dir') unzip(filename, tempdir); end</pre> <p>Load the data as an image datastore using the imageDatastore function and specify the folder containing the image data.</p> <pre>imds = imageDatastore(dataFolder, ... 'IncludeSubfolders', true, ... 'LabelSource', 'foldernames');</pre> <p>For examples showing how to process this data for deep learning, see Get Started with Transfer Learning and Train Deep Learning Network to Classify New Images.</p>	Image classification

Data Set	Description	Task
CamVid 	<p>The CamVid data set is a collection of images containing street-level views obtained from cars being driven. The data set is useful for training networks that perform semantic segmentation of images and provides pixel-level labels for 32 semantic classes, including <i>car</i>, <i>pedestrian</i>, and <i>road</i>.</p> <p>The images are of size 720-by-960-by-3.</p> <p>Download and extract the CamVid data set [8] from http://web4.cs.ucl.ac.uk/staff/g.brostow/MotionSegRecData. The data set is about 573 MB. Depending on your internet connection, the download process can take some time. Set downloadFolder to the location of the data.</p> <pre> downloadFolder = tempdir; url = "http://web4.cs.ucl.ac.uk/staff/g.brostow/MotionSegRecData" urlImages = url + "/files/701_StillsRaw_full.zip"; urlLabels = url + "/data/LabeledApproved_full.zip"; dataFolder = fullfile(downloadFolder,'CamVid'); dataFolderImages = fullfile(dataFolder,'images'); dataFolderLabels = fullfile(dataFolder,'labels'); filenameLabels = fullfile(dataFolder,'labels.zip'); filenameImages = fullfile(dataFolder,'images.zip'); if ~exist(filenameLabels, 'file') ~exist(filenameImages, 'file') mkdir(dataFolder) fprintf("Downloading CamVid data set images (557 MB)... "); websave(filenameImages, urlImages); unzip(filenameImages, dataFolderImages); fprintf("Done.\n") fprintf("Downloading CamVid data set labels (16 MB)... "); websave(filenameLabels, urlLabels); unzip(filenameLabels, dataFolderLabels); fprintf("Done.\n") end </pre> <p>Load the data as a pixel label datastore using the <code>pixelLabelDatastore</code> function and specify the folder containing the label data, the classes, and the label IDs. To make training easier, group the 32 original classes in the data set into 11 classes. To get the label IDs, use the helper function <code>camvidPixelLabelIDs</code>, which is used in the example Semantic Segmentation Using Deep Learning.</p> <pre> oldpath = addpath(fullfile(matlabroot,'examples','deeplearning_shared','main')); imds = imageDatastore(dataFolderImages,'IncludeSubfolders',true); classes = ["Sky" "Building" "Pole" "Road" "Pavement" "Tree" ... "SignSymbol" "Fence" "Car" "Pedestrian" "Bicyclist"]; labelIDs = camvidPixelLabelIDs; pxds = pixelLabelDatastore(dataFolderLabels,classes,labelIDs); </pre> <p>For an example showing how to process this data for deep learning, see Semantic Segmentation Using Deep Learning.</p> <p>To restore the path, use the <code>path</code> function.</p> <pre> path(oldpath); </pre>	Semantic segmentation

Data Set	Description	Task
Vehicle 	<p>The Vehicle data set consists of 295 images containing one or two labeled instances of a vehicle. This small data set is useful for exploring the YOLO-v2 training procedure, but in practice, more labeled images are needed to train a robust detector.</p> <p>The images are of size 720-by-960-by-3.</p> <p>Extract the Vehicle data set. Set dataFolder to the location of the data.</p> <pre>filename = 'vehicleDatasetImages.zip'; dataFolder = fullfile(tempdir,'vehicleImages'); if ~exist(dataFolder,'dir') unzip(filename,tempdir); end</pre> <p>Load the data set of as a table of file names and bounding boxes from the extracted MAT file and convert the file names to absolute file paths.</p> <pre>data = load('vehicleDatasetGroundTruth.mat'); vehicleDataset = data.vehicleDataset; vehicleDataset.imageFilename = fullfile(tempdir,vehicleDataset.imageFilename);</pre> <p>Create an image datastore containing the images and a box label datastore containing the bounding boxes using the imageDatastore and boxLabelDatastore functions, respectively. Combine the resulting datastores using the combine function.</p> <pre>filenamesImages = vehicleDataset.imageFilename; tblBoxes = vehicleDataset(:, 'vehicle'); imds = imageDatastore(filenamesImages); blbs = boxLabelDatastore(tblBoxes); cbs = combine(imds,blbs);</pre> <p>For an example showing how to process this data for deep learning, see Object Detection Using YOLO v2 Deep Learning.</p>	Object detection
RIT-18 	<p>The RIT-18 data set contains image data captured by a drone over Hamlin Beach State Park, in New York state. The data contains labeled training, validation, and test sets, with 18 object class labels including <i>road markings</i>, <i>tree</i>, and <i>building</i>.</p> <p>Download the RIT-18 data set [9] from https://www.cis.rit.edu/%7Ermk6217/rit18_data.mat. The data set is about 3 GB. Depending on your internet connection, the download process can take some time. Set downloadFolder to the location of the data.</p> <pre>downloadFolder = tempdir; url = 'http://www.cis.rit.edu/~rmk6217/rit18_data.mat'; filename = fullfile(downloadFolder,'rit18_data.mat'); if ~exist(filename,'file') fprintf("Downloading Hamlin Beach data set (3 GB)... "); websave(filename,url); fprintf("Done.\n") end</pre> <p>For an example showing how to process this data for deep learning, see Semantic Segmentation of Multispectral Images Using Deep Learning.</p>	Semantic segmentation

List of Deep Learning Layers

This page provides a list of deep learning layers in MATLAB®.

To learn how to create networks from layers for different tasks, see the following examples.






Task	Learn More
Create deep learning networks for image classification or regression.	Create Simple Deep Learning Network for Classification Train Convolutional Neural Network for Regression Train Residual Network for Image Classification
Create deep learning networks for sequence and time series data.	Sequence Classification Using Deep Learning Time Series Forecasting Using Deep Learning
Create deep learning network for audio data.	Speech Command Recognition Using Deep Learning
Create deep learning network for text data.	Classify Text Data Using Deep Learning Generate Text Using Deep Learning

Deep Learning Layers






Use the following functions to create different layer types. Alternatively, use the [Deep Network Designer](#) app to create networks interactively.


To learn how to define your own custom layers, see [Define Custom Deep Learning Layers](#).

Input Layers









Layer	Description
 imageInputLayer	An image input layer inputs 2-D images to a network and applies data normalization.
 image3dInputLayer	A 3-D image input layer inputs 3-D images or volumes to a network and applies data normalization.
 sequenceInputLayer	A sequence input layer inputs sequence data to a network.
 featureInputLayer	A feature input layer inputs feature data to a network and applies data normalization. Use this layer when you have a data set of numeric scalars representing features (data without spatial or time dimensions).
 roiInputLayer (Computer Vision Toolbox)	An ROI input layer inputs images to a Fast R-CNN object detection network.

Convolution and Fully Connected Layers








Layer	Description
 convolution2dLayer	A 2-D convolutional layer applies sliding convolutional filters to the input.
 convolution3dLayer	A 3-D convolutional layer applies sliding cuboidal convolution filters to three-dimensional input.
 groupedConvolution2dLayer	A 2-D grouped convolutional layer separates the input channels into groups and applies sliding convolutional filters. Use grouped convolutional layers for channel-wise separable (also known as depth-wise separable) convolution.
 transposedConv2dLayer	A transposed 2-D convolution layer upsamples feature maps.
 transposedConv3dLayer	A transposed 3-D convolution layer upsamples three-dimensional feature maps.

Layer	Description
 <code>fullyConnectedLayer</code>	A fully connected layer multiplies the input by a weight matrix and then adds a bias vector.

Sequence Layers











Layer	Description
 <code>sequenceInputLayer</code>	A sequence input layer inputs sequence data to a network.
 <code>lstmLayer</code>	An LSTM layer learns long-term dependencies between time steps in time series and sequence data.
 <code>biLstmLayer</code>	A bidirectional LSTM (BiLSTM) layer learns bidirectional long-term dependencies between time steps of time series or sequence data. These dependencies can be useful when you want the network to learn from the complete time series at each time step.
 <code>gruLayer</code>	A GRU layer learns dependencies between time steps in time series and sequence data.
 <code>sequenceFoldingLayer</code>	A sequence folding layer converts a batch of image sequences to a batch of images. Use a sequence folding layer to perform convolution operations on time steps of image sequences independently.
 <code>sequenceUnfoldingLayer</code>	A sequence unfolding layer restores the sequence structure of the input data after sequence folding.
 <code>flattenLayer</code>	A flatten layer collapses the spatial dimensions of the input into the channel dimension.
 <code>wordEmbeddingLayer</code> (Text Analytics Toolbox)	A word embedding layer maps word indices to vectors.

Activation Layers




Layer	Description
 <code>reluLayer</code>	A ReLU layer performs a threshold operation to each element of the input, where any value less than zero is set to zero.
 <code>leakyReluLayer</code>	A leaky ReLU layer performs a threshold operation, where any input value less than zero is multiplied by a fixed scalar.
 <code>clippedReluLayer</code>	A clipped ReLU layer performs a threshold operation, where any input value less than zero is set to zero and any value above the <i>clipping ceiling</i> is set to that clipping ceiling.
 <code>eluLayer</code>	An ELU activation layer performs the identity operation on positive inputs and an exponential nonlinearity on negative inputs.
 <code>tanhLayer</code>	A hyperbolic tangent (tanh) activation layer applies the tanh function on the layer inputs.
 <code>swishLayer</code>	A swish activation layer applies the swish function on the layer inputs.
 <code>preluLayer</code> (Custom layer example)	A PReLU layer performs a threshold operation, where for each channel, any input value less than zero is multiplied by a scalar learned at training time.







Normalization, Dropout, and Cropping Layers

Layer	Description
-------	-------------






Layer	Description
 <code>batchNormalizationLayer</code>	A batch normalization layer normalizes a mini-batch of data across all observations for each channel independently. To speed up training of the convolutional neural network and reduce the sensitivity to network initialization, use batch normalization layers between convolutional layers and nonlinearities, such as ReLU layers.
 <code>groupNormalizationLayer</code>	A group normalization layer normalizes a mini-batch of data across grouped subsets of channels for each observation independently. To speed up training of the convolutional neural network and reduce the sensitivity to network initialization, use group normalization layers between convolutional layers and nonlinearities, such as ReLU layers.
 <code>instanceNormalizationLayer</code>	An instance normalization layer normalizes a mini-batch of data across each channel for each observation independently. To improve the convergence of training the convolutional neural network and reduce the sensitivity to network hyperparameters, use instance normalization layers between convolutional layers and nonlinearities, such as ReLU layers.
 <code>layerNormalizationLayer</code>	A layer normalization layer normalizes a mini-batch of data across all channels for each observation independently. To speed up training of recurrent and multi-layer perceptron neural networks and reduce the sensitivity to network initialization, use layer normalization layers after the learnable layers, such as LSTM and fully connected layers.
 <code>crossChannelNormalizationLayer</code>	A channel-wise local response (cross-channel) normalization layer carries out channel-wise normalization.
 <code>dropoutLayer</code>	A dropout layer randomly sets input elements to zero with a given probability.
 <code>crop2dLayer</code>	A 2-D crop layer applies 2-D cropping to the input.
 <code>crop3dLayer</code>	A 3-D crop layer crops a 3-D volume to the size of the input feature map.
 <code>resize2dLayer</code> (Image Processing Toolbox)	A 2-D resize layer resizes 2-D input by a scale factor, to a specified height and width, or to the size of a reference input feature map.
 <code>resize3dLayer</code> (Image Processing Toolbox)	A 3-D resize layer resizes 3-D input by a scale factor, to a specified height, width, and depth, or to the size of a reference input feature map.

Pooling and Unpooling Layers







Layer	Description
 <code>averagePooling2dLayer</code>	An average pooling layer performs downsampling by dividing the input into rectangular pooling regions and computing the average values of each region.
 <code>averagePooling3dLayer</code>	A 3-D average pooling layer performs downsampling by dividing three-dimensional input into cuboidal pooling regions and computing the average values of each region.
 <code>globalAveragePooling2dLayer</code>	A global average pooling layer performs downsampling by computing the mean of the height and width dimensions of the input.







Layer	Description
 <code>globalAveragePooling3dLayer</code>	A 3-D global average pooling layer performs downsampling by computing the mean of the height, width, and depth dimensions of the input.
 <code>maxPooling2dLayer</code>	A max pooling layer performs downsampling by dividing the input into rectangular pooling regions, and computing the maximum of each region.
 <code>maxPooling3dLayer</code>	A 3-D max pooling layer performs downsampling by dividing three-dimensional input into cuboidal pooling regions, and computing the maximum of each region.
 <code>globalMaxPooling2dLayer</code>	A global max pooling layer performs downsampling by computing the maximum of the height and width dimensions of the input.
 <code>globalMaxPooling3dLayer</code>	A 3-D global max pooling layer performs downsampling by computing the maximum of the height, width, and depth dimensions of the input.
 <code>maxUnpooling2dLayer</code>	A max unpooling layer unpool the output of a max pooling layer.

Combination Layers



Layer	Description
 <code>additionLayer</code>	An addition layer adds inputs from multiple neural network layers element-wise.
 <code>multiplicationLayer</code>	A multiplication layer multiplies inputs from multiple neural network layers element-wise.
 <code>depthConcatenationLayer</code>	A depth concatenation layer takes inputs that have the same height and width and concatenates them along the third dimension (the channel dimension).
 <code>concatenationLayer</code>	A concatenation layer takes inputs and concatenates them along a specified dimension. The inputs must have the same size in all dimensions except the concatenation dimension.
 <code>weightedAdditionLayer</code> (Custom layer example)	A weighted addition layer scales and adds inputs from multiple neural network layers element-wise.

Object Detection Layers









Layer	Description
 <code>roiInputLayer</code> (Computer Vision Toolbox)	An ROI input layer inputs images to a Fast R-CNN object detection network.
 <code>roiMaxPooling2dLayer</code> (Computer Vision Toolbox)	An ROI max pooling layer outputs fixed size feature maps for every rectangular ROI within the input feature map. Use this layer to create a Fast or Faster R-CNN object detection network.
 <code>roiAlignLayer</code> (Computer Vision Toolbox)	An ROI align layer outputs fixed size feature maps for every rectangular ROI within an input feature map. Use this layer to create a Mask-RCNN network.
 <code>anchorBoxLayer</code> (Computer Vision Toolbox)	An anchor box layer stores anchor boxes for a feature map used in object detection networks.
 <code>regionProposalLayer</code> (Computer Vision Toolbox)	A region proposal layer outputs bounding boxes around potential objects in an image as part of the region proposal network (RPN) within Faster R-CNN.
 <code>ssdMergeLayer</code> (Computer Vision Toolbox)	An SSD merge layer merges the outputs of feature maps for subsequent regression and classification loss computation.






Layer	Description
 spaceToDepthLayer (Image Processing Toolbox)	A space to depth layer permutes the spatial blocks of the input into the depth dimension. Use this layer when you need to combine feature maps of different size without discarding any feature data.
 depthToSpace2dLayer (Image Processing Toolbox)	A 2-D depth to space layer permutes data from the depth dimension into blocks of 2-D spatial data.
 rpnSoftmaxLayer (Computer Vision Toolbox)	A region proposal network (RPN) softmax layer applies a softmax activation function to the input. Use this layer to create a Faster R-CNN object detection network.
 focalLossLayer (Computer Vision Toolbox)	A focal loss layer predicts object classes using focal loss.
 rpnClassificationLayer (Computer Vision Toolbox)	A region proposal network (RPN) classification layer classifies image regions as either <i>object</i> or <i>background</i> by using a cross entropy loss function. Use this layer to create a Faster R-CNN object detection network.
 rcnnBoxRegressionLayer (Computer Vision Toolbox)	A box regression layer refines bounding box locations by using a smooth L1 loss function. Use this layer to create a Fast or Faster R-CNN object detection network.

Generative Adversarial Network Layers

Layer	Description
 projectAndReshapeLayer (Custom layer example)	A project and reshape layer takes as input 1-by-1-by-numLatentInputs arrays and converts them to images of the specified size. Use project and reshape layers to reshape the noise input to GANs.
 embedAndReshapeLayer (Custom layer example)	An embed and reshape layer takes as input numeric indices of categorical elements and converts them to images of the specified size. Use embed and reshape layers to input categorical data into conditional GANs.

Output Layers

Layer	Description
 softmaxLayer	A softmax layer applies a softmax function to the input.
 sigmoidLayer	A sigmoid layer applies a sigmoid function to the input such that the output is bounded in the interval (0,1).
 classificationLayer	A classification layer computes the cross-entropy loss for classification and weighted classification tasks with mutually exclusive classes.
 regressionLayer	A regression layer computes the half-mean-squared-error loss for regression tasks.
 pixelClassificationLayer (Computer Vision Toolbox)	A pixel classification layer provides a categorical label for each image pixel or voxel.
 dicePixelClassificationLayer (Computer Vision Toolbox)	A Dice pixel classification layer provides a categorical label for each image pixel or voxel using generalized Dice loss.
 focalLossLayer (Computer Vision Toolbox)	A focal loss layer predicts object classes using focal loss.
 rpnSoftmaxLayer (Computer Vision Toolbox)	A region proposal network (RPN) softmax layer applies a softmax activation function to the input. Use this layer to create a Faster R-CNN object detection network.

Layer	Description
 rpnClassificationLayer (Computer Vision Toolbox)	A region proposal network (RPN) classification layer classifies image regions as either <i>object</i> or <i>background</i> by using a cross entropy loss function. Use this layer to create a Faster R-CNN object detection network.
 rcnnBoxRegressionLayer (Computer Vision Toolbox)	A box regression layer refines bounding box locations by using a smooth L1 loss function. Use this layer to create a Fast or Faster R-CNN object detection network.
 tverskyPixelClassificationLayer (Custom layer example)	A Tversky pixel classification layer provides a categorical label for each image pixel or voxel using Tversky loss.
 sseClassificationLayer (Custom layer example)	A classification SSE layer computes the sum of squares error loss for classification problems.
 maeRegressionLayer (Custom layer example)	A regression MAE layer computes the mean absolute error loss for regression problems.

See Also

[Deep Network Designer](#) | [trainingOptions](#) | [trainNetwork](#)

Related Topics

- [Build Networks with Deep Network Designer](#)
- [Specify Layers of Convolutional Neural Network](#)
- [Set Up Parameters and Train Convolutional Neural Network](#)
- [Define Custom Deep Learning Layers](#)
- [Create Simple Deep Learning Network for Classification](#)
- [Sequence Classification Using Deep Learning](#)
- [Pretrained Deep Neural Networks](#)
- [Deep Learning Tips and Tricks](#)