Deep Learning Models for Videos

- Video Classification and Semantic Segmentation Using Deep Learning

Video Processing

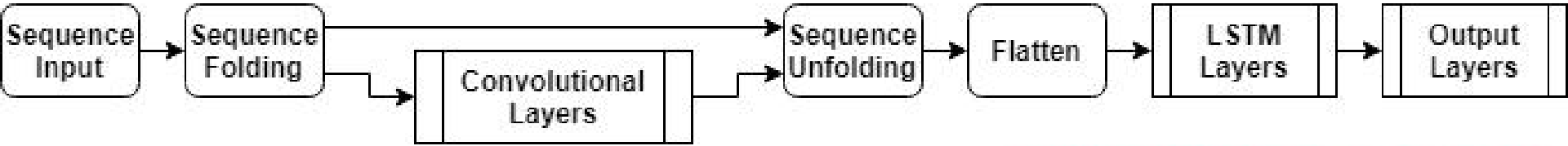
- Classify Videos Using Deep Learning
- Semantic Segmentation Using Deep Learning

Classify Videos Using Deep Learning

- This example shows how to create a network for video classification by combining a pretrained image classification model and an LSTM network.
- To create a deep learning network for video classification:
  - Convert videos to sequences of feature vectors using a pretrained convolutional neural network, such as GoogLeNet, to extract features from each frame.
  - Train an LSTM network on the sequences to predict the video labels.
  - Assemble a network that classifies videos directly by combining layers from both networks.

Classify Videos Using Deep Learning

- The following diagram illustrates the network architecture.
  - To input image sequences to the network, use a sequence input layer.
  - To use convolutional layers to extract features, that is, to apply the convolutional operations to each frame of the videos independently, use a sequence folding layer followed by the convolutional layers.
  - To restore the sequence structure and reshape the output to vector sequences, use a sequence unfolding layer and a flatten layer.
  - To classify the resulting vector sequences, include the LSTM layers followed by the output layers.



Load Pretrained Convolutional Network

- To convert frames of videos to feature vectors, use the activations of a pretrained network.
- Load a pretrained GoogLeNet model using the googlenet function. This function requires the Deep Learning Toolbox™ Model for GoogLeNet Network support package. If this support package is not installed, then the function provides a download link.

```
netCNN = googlenet;
```

Load Data

- Download the HMBD51 data set from HMDB: a large human motion database and extract the RAR file into a folder named "hmdb51_org". The data set contains about 2 GB of video data for 7000 clips over 51 classes, such as "drink", "run", and "shake_hands".
- After extracting the RAR files, use the supporting function hmdb51Files to get the file names and the labels of the videos.

```
dataFolder = "hmdb51_org";
[files,labels] = hmdb51Files(dataFolder);
```

Load Data

- Read the first video using the readVideo helper function, defined at the end of this example, and view the size of the video. The video is a H-by-W-by-C-by-S array, where H, W, C, and S are the height, width, number of channels, and number of frames of the video, respectively.

```
idx = 1;
filename = files(idx);
video = readVideo(filename);
size(video)
ans = 1×4
   240   320     3   409
```

Load Data

- View the corresponding label.

labels(idx)

ans = categorical

brush_hair

Load Data

- To view the video, use the implay function. This function expects data in the range [0,1], so you must first divide the data by 255. Alternatively, you can loop over the individual frames and use the imshow function.

```
numFrames = size(video,4);
figure
for i = 1:numFrames
    frame = video(:,:,:,i);
    imshow(frame/255);
    drawnow
end
```

Convert Frames to Feature Vectors

- Use the convolutional network as a feature extractor by getting the activations when inputting the video frames to the network. Convert the videos to sequences of feature vectors, where the feature vectors are the output of the activations function on the last pooling layer of the GoogLeNet network ("pool5-7x7_s1").
- This diagram illustrates the data flow through the network.

Convert Frames to Feature Vectors

- To read the video data and resize it to match the input size of the GoogLeNet network, use the readVideo and centerCrop helper functions, defined at the end of this example. This step can take a long time to run. After converting the videos to sequences, save the sequences in a MAT-file in the tempdir folder. If the MAT file already exists, then load the sequences from the MAT-file without reconverting them.

```
inputSize = netCNN.Layers(1).InputSize(1:2);
layerName = "pool5-7x7_s1";
tempFile = fullfile(tempdir,"hmdb51_org.mat");
if exist(tempFile,'file')
    load(tempFile,"sequences")
else
    numFiles = numel(files);
    sequences = cell(numFiles,1);
    for i = 1:numFiles
        fprintf("Reading file %d of %d...\n", i, numFiles)
        video = readVideo(files(i));
        video = centerCrop(video,inputSize);
         sequences{i,1} = activations(netCNN,video,layerName,'OutputAs','columns');
    end
    save(tempFile,"sequences","-v7.3");
end
```

Convert Frames to Feature Vectors

- View the sizes of the first few sequences. Each sequence is a D-by-S array, where D is the number of features (the output size of the pooling layer) and S is the number of frames of the video.

```
sequences(1:10)
ans = 10×1 cell array
    {1024×409 single}
    {1024×395 single}
    {1024×323 single}
    {1024×246 single}
    {1024×159 single}
    {1024×137 single}
    {1024×359 single}
    {1024×191 single}
    {1024×439 single}
    {1024×528 single}
```

Prepare Training Data

- Prepare the data for training by partitioning the data into training and validation partitions and removing any long sequences.
- Partition the data. Assign 90% of the data to the training partition and 10% to the validation partition.

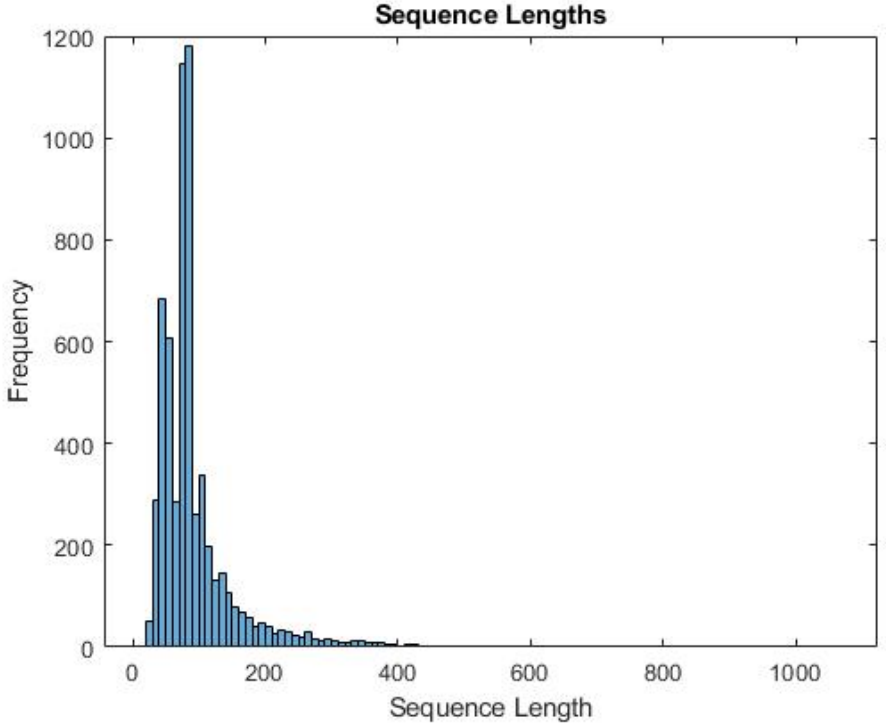
```
numObservations = numel(sequences);
idx = randperm(numObservations);
N = floor(0.9 * numObservations);
idxTrain = idx(1:N);
sequencesTrain = sequences(idxTrain);
labelsTrain = labels(idxTrain);
idxValidation = idx(N+1:end);
sequencesValidation = sequences(idxValidation);
labelsValidation = labels(idxValidation);
```

Remove Long Sequences

- Sequences that are much longer than typical sequences in the networks can introduce lots of padding into the training process. Having too much padding can negatively impact the classification accuracy. Get the sequence lengths of the training data and visualize them in a histogram of the training data.

Remove Long Sequences

```
numObservationsTrain = numel(sequencesTrain);
sequenceLengths = zeros(1,numObservationsTrain);
for i = 1:numObservationsTrain
    sequence = sequencesTrain{i};
    sequenceLengths(i) = size(sequence,2);
end
figure
histogram(sequenceLengths)
title("Sequence Lengths")
xlabel("Sequence Length")
ylabel("Frequency")
```



Remove Long Sequences

- Only a few sequences have more than 400 time steps. To improve the classification accuracy, remove the training sequences that have more than 400 time steps along with their corresponding labels.

```
maxLength = 400;
idx = sequenceLengths > maxLength;
sequencesTrain(idx) = [];
labelsTrain(idx) = [];
```

Create LSTM Network

- Next, create an LSTM network that can classify the sequences of feature vectors representing the videos.
- Define the LSTM network architecture. Specify the following network layers.
  - A sequence input layer with an input size corresponding to the feature dimension of the feature vectors
  - A BiLSTM layer with 2000 hidden units with a dropout layer afterwards. To output only one label for each sequence by setting the 'OutputMode' option of the BiLSTM layer to 'last'
  - A fully connected layer with an output size corresponding to the number of classes, a softmax layer, and a classification layer.

Create LSTM Network

```
numFeatures = size(sequencesTrain{1},1);
numClasses = numel(categories(labelsTrain));

layers = [
    sequenceInputLayer(numFeatures,'Name','sequence')
    bilstmLayer(2000,'OutputMode','last','Name','bilstm')
    dropoutLayer(0.5,'Name','drop')
    fullyConnectedLayer(numClasses,'Name','fc')
    softmaxLayer('Name','softmax')
    classificationLayer('Name','classification')];
```

Specify Training Options

- Specify the training options using the trainingOptions function.
  - Set a mini-batch size 16, an initial learning rate of 0.0001, and a gradient threshold of 2 (to prevent the gradients from exploding).
  - Shuffle the data every epoch.
  - Validate the network once per epoch.
  - Display the training progress in a plot and suppress verbose output.

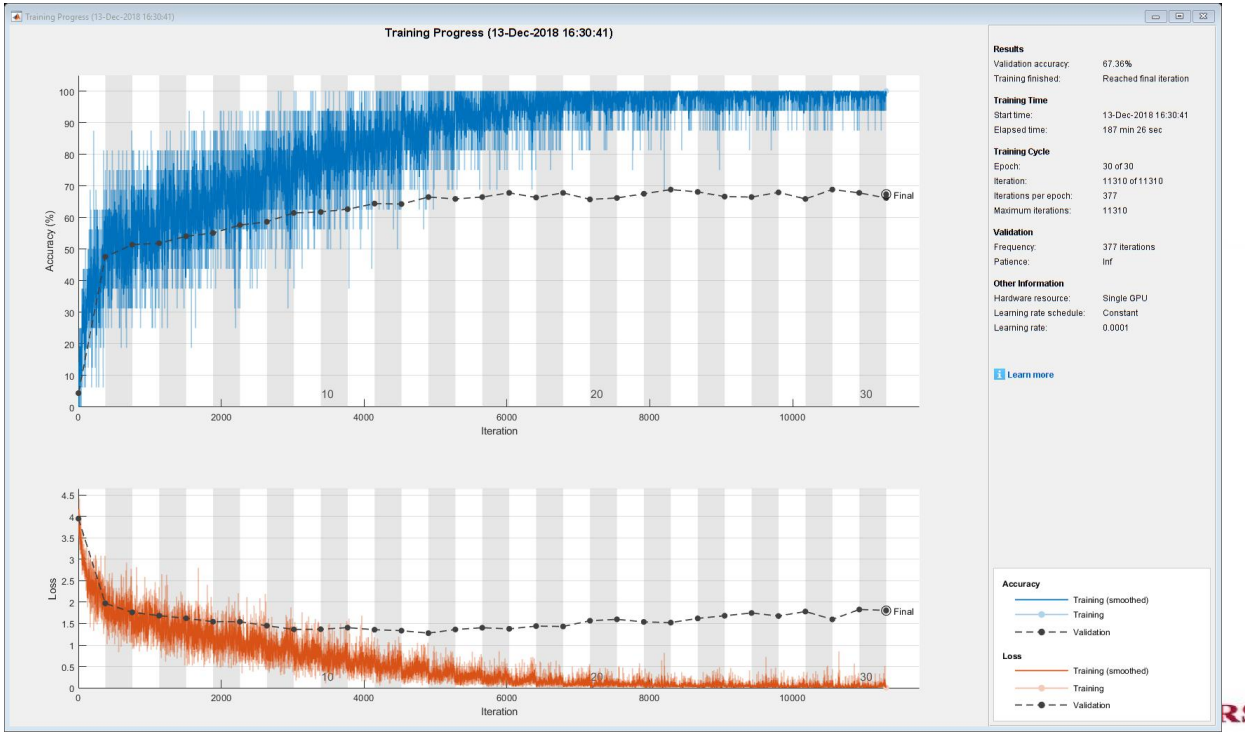
Specify Training Options

```
miniBatchSize = 16;
numObservations = numel(sequencesTrain);
numIterationsPerEpoch = floor(numObservations / miniBatchSize);
options = trainingOptions('adam', ...
    'MiniBatchSize',miniBatchSize, ...
    'InitialLearnRate',1e-4, ...
    'GradientThreshold',2, ...
    'Shuffle','every-epoch', ...
    'ValidationData',{sequencesValidation,labelsValidation}, ...
    'ValidationFrequency',numIterationsPerEpoch, ...
    'Plots','training-progress', ...
    'Verbose',false);
```

Train LSTM Network

- Train the network using the trainNetwork function. This can take a long time to run.

[netLSTM,info] = trainNetwork(sequencesTrain,labelsTrain,layers,options);



Train LSTM Network

- Calculate the classification accuracy of the network on the validation set. Use the same mini-batch size as for the training options.

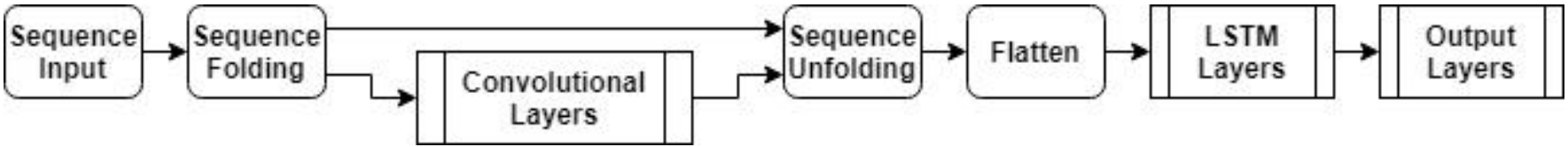
```
YPred = classify(netLSTM,sequencesValidation,'MiniBatchSize',miniBatchSize);
YValidation = labelsValidation;
accuracy = mean(YPred == YValidation)
accuracy = 0.6647
```

Assemble Video Classification Network

- To create a network that classifies videos directly, assemble a network using layers from both of the created networks. Use the layers from the convolutional network to transform the videos into vector sequences and the layers from the LSTM network to classify the vector sequences.

Assemble Video Classification Network

- The following diagram illustrates the network architecture.
  - To input image sequences to the network, use a sequence input layer.
  - To use convolutional layers to extract features, that is, to apply the convolutional operations to each frame of the videos independently, use a sequence folding layer followed by the convolutional layers.
  - To restore the sequence structure and reshape the output to vector sequences, use a sequence unfolding layer and a flatten layer.
  - To classify the resulting vector sequences, include the LSTM layers followed by the output layers.



Add Convolutional Layers

- First, create a layer graph of the GoogLeNet network.

```
cnnLayers = layerGraph(netCNN);
```

- Remove the input layer ("data") and the layers after the pooling layer used for the activations ("pool5-drop_7x7_s1", "loss3-classifier", "prob", and "output").

```
layerNames = ["data" "pool5-drop_7x7_s1" "loss3-classifier" "prob" "output"];
cnnLayers = removeLayers(cnnLayers,layerNames);
```

Add Sequence Input Layer

- Create a sequence input layer that accepts image sequences containing images of the same input size as the GoogLeNet network. To normalize the images using the same average image as the GoogLeNet network, set the 'Normalization' option of the sequence input layer to 'zerocenter' and the 'Mean' option to the average image of the input layer of GoogLeNet.

```
inputSize = netCNN.Layers(1).InputSize(1:2);
averageImage = netCNN.Layers(1).Mean;
inputLayer = sequenceInputLayer([inputSize 3], ...
    'Normalization','zerocenter', ...
    'Mean',averageImage, ...
    'Name','input');
```

Add Sequence Input Layer

- Add the sequence input layer to the layer graph. To apply the convolutional layers to the images of the sequences independently, remove the sequence structure of the image sequences by including a sequence folding layer between the sequence input layer and the convolutional layers. Connect the output of the sequence folding layer to the input of the first convolutional layer ("conv1-7x7_s2").

```
layers = [
    inputLayer
    sequenceFoldingLayer('Name','fold')];
lgraph = addLayers(cnnLayers,layers);
lgraph = connectLayers(lgraph,"fold/out","conv1-7x7_s2");
```

Add LSTM Layers

- Add the LSTM layers to the layer graph by removing the sequence input layer of the LSTM network. To restore the sequence structure removed by the sequence folding layer, include a sequence unfolding layer after the convolution layers. The LSTM layers expect sequences of vectors. To reshape the output of the sequence unfolding layer to vector sequences, include a flatten layer after the sequence unfolding layer.
- Take the layers from the LSTM network and remove the sequence input layer.

```
lstmLayers = netLSTM.Layers;
lstmLayers(1) = [];
```

Add LSTM Layers

- Add the sequence folding layer, the flatten layer, and the LSTM layers to the layer graph. Connect the last convolutional layer ("pool5-7x7_s1") to the input of the sequence unfolding layer ("unfold/in").

```
layers = [
    sequenceUnfoldingLayer('Name','unfold')
    flattenLayer('Name','flatten')
    lstmLayers];
lgraph = addLayers(lgraph,layers);
lgraph = connectLayers(lgraph,"pool5-7x7_s1","unfold/in");
```

Add LSTM Layers

- To enable the unfolding layer to restore the sequence structure, connect the "miniBatchSize" output of the sequence folding layer to the corresponding input of the sequence unfolding layer.

```
lgraph = connectLayers(lgraph,"fold/miniBatchSize","unfold/miniBatchSize");
```

Assemble Network

- Check that the network is valid using the analyzeNetwork function.

analyzeNetwork(lgraph)

- Assemble the network so that it is ready for prediction using the assembleNetwork function.

net = assembleNetwork(lgraph)

net =

DAGNetwork with properties:

Layers: [148×1 nnet.cnn.layer.Layer]

Connections: [175×2 table]

Classify Using New Data

- Read and center-crop the video "pushup.mp4" using the same steps as before.

```
filename = "pushup.mp4";
video = readVideo(filename);
```

Classify Using New Data

- To view the video, use the implay function (requires Image Processing Toolbox). This function expects data in the range [0,1], so you must first divide the data by 255. Alternatively, you can loop over the individual frames and use the imshow function.

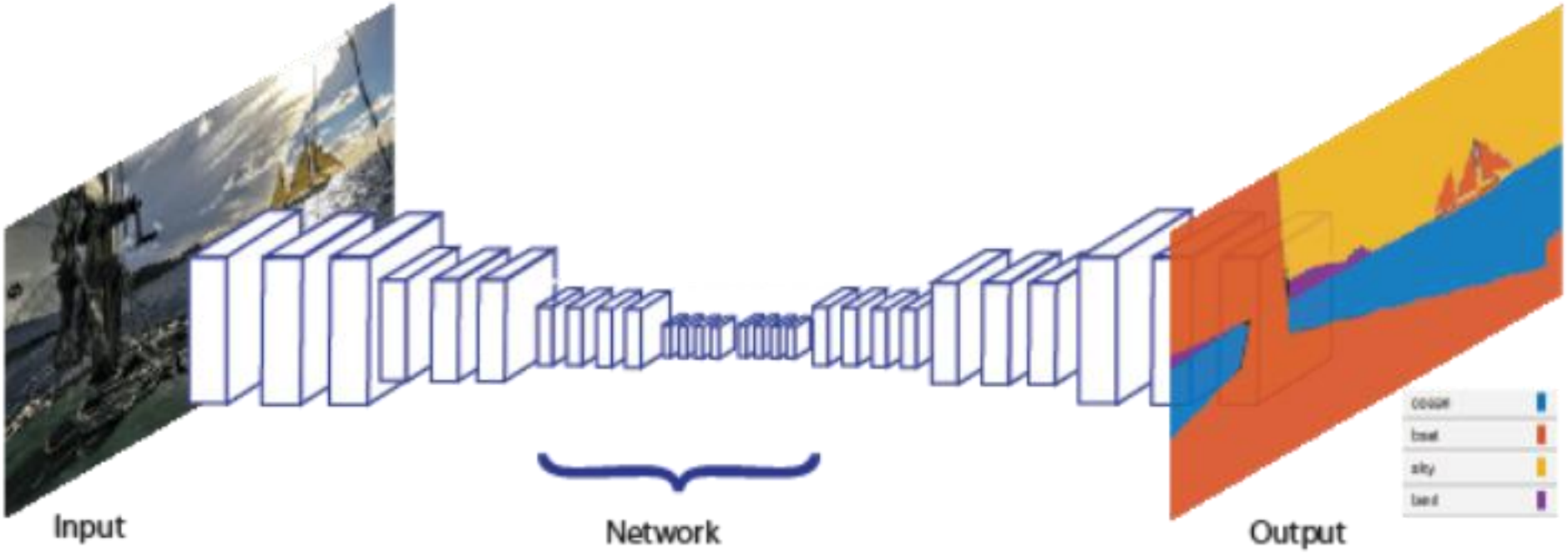
```
numFrames = size(video,4);
figure
for i = 1:numFrames
    frame = video(:,:,:,i);
    imshow(frame/255);
    drawnow
end
```

Classify Using New Data

- Classify the video using the assembled network. The classify function expects a cell array containing the input videos, so you must input a 1-by-1 cell array containing the video.

```
video = centerCrop(video,inputSize)
YPred = classify(net,{video})
YPred = categorical
    pushup
```

Semantic Segmentation Using Deep Learning



Semantic Segmentation Using Deep Learning

- A semantic segmentation network classifies every pixel in an image, resulting in an image that is segmented by class. Applications for semantic segmentation include road segmentation for autonomous driving and cancer cell segmentation for medical diagnosis.
- To illustrate the training procedure, this example trains Deeplab v3+ , one type of convolutional neural network (CNN) designed for semantic image segmentation. Other types of networks for semantic segmentation include fully convolutional networks (FCN), SegNet, and U-Net. The training procedure shown here can be applied to those networks too.

Semantic Segmentation Using Deep Learning

- This example uses the CamVid dataset from the University of Cambridge for training. This dataset is a collection of images containing street-level views obtained while driving. The dataset provides pixel-level labels for 32 semantic classes including car, pedestrian, and road.
- This example creates the Deeplab v3+ network with weights initialized from a pre-trained Resnet-18 network. ResNet-18 is an efficient network that is well suited for applications with limited processing resources. Other pretrained networks such as MobileNet v2 or ResNet-50 can also be used depending on application requirements.
- To get a pretrained Resnet-18, install resnet18. After installation is complete, run the following code to verify that the installation is correct.

```
resnet18();
```

Semantic Segmentation Using Deep Learning

- In addition, download a pretrained version of DeepLab v3+. The pretrained model allows you to run the entire example without having to wait for training to complete.

```
pretrainedURL =
'https://www.mathworks.com/supportfiles/vision/data/deeplabv3plusResnet18CamVid.mat';
pretrainedFolder = fullfile(tempdir,'pretrainedNetwork');
pretrainedNetwork = fullfile(pretrainedFolder,'deeplabv3plusResnet18CamVid.mat');
if ~exist(pretrainedNetwork,'file')
    mkdir(pretrainedFolder);
    disp('Downloading pretrained network (58 MB)...');
    websave(pretrainedNetwork,pretrainedURL);
end
```

Semantic Segmentation Using Deep Learning

- Download CamVid Dataset
- Download the CamVid dataset from the following URLs.

imageURL =
'http://web4.cs.ucl.ac.uk/staff/g.brostow/MotionSegRecData/files/701_StillsRaw_full.zip';

labelURL =
'http://web4.cs.ucl.ac.uk/staff/g.brostow/MotionSegRecData/data/LabeledApproved_full.zip';

Semantic Segmentation Using Deep Learning

- Use imageDatastore to load CamVid images. The imageDatastore enables you to efficiently load a large collection of images on disk.

```
imgDir = fullfile(outputFolder,'images','701_StillsRaw_full');
imds = imageDatastore(imgDir);
```

Display one of the images.

```
I = readimage(imds,559);
I = histeq(I);
imshow(I)
```

Load CamVid Pixel-Labeled Images

- Use pixelLabelDatastore to load CamVid pixel label image data. A pixelLabelDatastore encapsulates the pixel label data and the label ID to a class name mapping. We make training easier, we group the 32 original classes in CamVid to 11 classes. Specify these classes.

```
classes = [
    "Sky"
    "Building"
    "Pole"
    "Road"
    "Pavement"
    "Tree"
    "SignSymbol"
    "Fence"
    "Car"
    "Pedestrian"
    "Bicyclist"
    ];
```

Load CamVid Pixel-Labeled Images

- To reduce 32 classes into 11, multiple classes from the original dataset are grouped together. For example, "Car" is a combination of "Car", "SUVPickupTruck", "Truck_Bus", "Train", and "OtherMoving". Return the grouped label IDs by using the supporting function camvidPixelLabelIDs, which is listed at the end of this example.

```
labelIDs = camvidPixelLabelIDs();
```

- Use the classes and label IDs to create the pixelLabelDatastore.

```
labelDir = fullfile(outputFolder,'labels');
pxds = pixelLabelDatastore(labelDir,classes,labelIDs);
```

Load CamVid Pixel-Labeled Images

- Read and display one of the pixel-labeled images by overlaying it on top of an image.

```
C = readimage(pxds,559);
cmap = camvidColorMap;
B = labeloverlay(I,C,'ColorMap',cmap);
imshow(B)
pixelLabelColorbar(cmap,classes);
```

Analyze Dataset Statistics

- To see the distribution of class labels in the CamVid dataset, use countEachLabel. This function counts the number of pixels by class label.

tbl = countEachLabel(pxds)

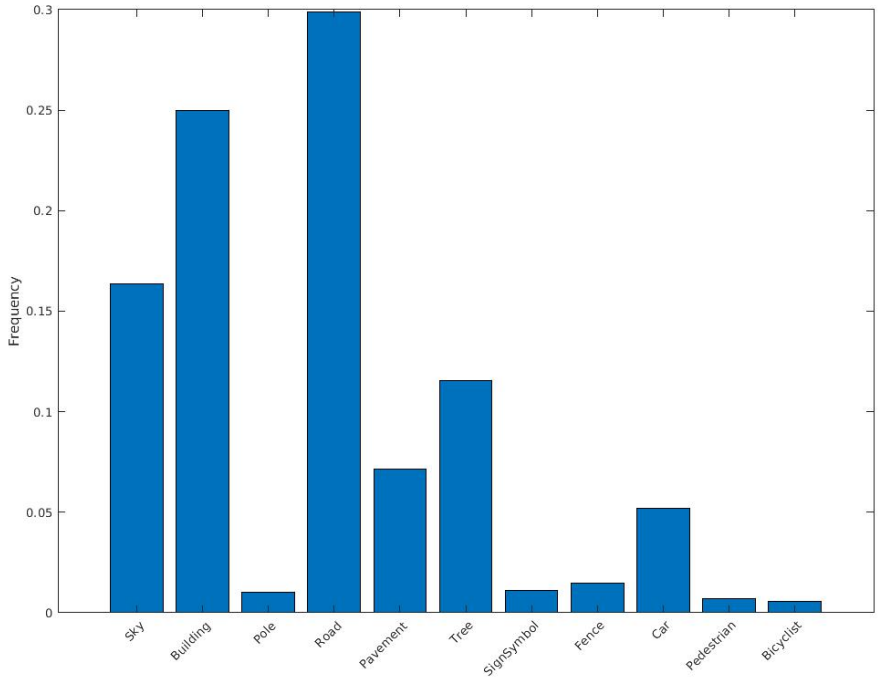
tbl=11×3 table

Name	PixelCount	ImagePixelCount
{'Sky' }	7.6801e+07	4.8315e+08
{'Building' }	1.1737e+08	4.8315e+08
{'Pole' }	4.7987e+06	4.8315e+08
{'Road' }	1.4054e+08	4.8453e+08
{'Pavement' }	3.3614e+07	4.7209e+08
{'Tree' }	5.4259e+07	4.479e+08

Analyze Dataset Statistics

- Visualize the pixel counts by class.

```
frequency = tbl.PixelCount/sum(tbl.PixelCount);
bar(1:numel(classes),frequency)
xticks(1:numel(classes))
xticklabels(tbl.Name)
xtickangle(45)
ylabel('Frequency')
```



Prepare Training, Validation, and Test Sets

- Deeplab v3+ is trained using 60% of the images from the dataset. The rest of the images are split evenly in 20% and 20% for validation and testing respectively. The following code randomly splits the image and pixel label data into a training, validation and test set.

```
[imdsTrain, imdsVal, imdsTest, pxdsTrain, pxdsVal, pxdsTest] =
partitionCamVidData(imds,pxds);
```

- The 60/20/20 split results in the following number of training, validation and test images:

```
numTrainingImages = numel(imdsTrain.Files) % numTrainingImages = 421
numValImages = numel(imdsVal.Files) % numValImages = 140
numTestingImages = numel(imdsTest.Files) % numTestingImages = 140
```

Create the Network

- Use the deeplabv3plusLayers function to create a DeepLab v3+ network based on ResNet-18. Choosing the best network for your application requires empirical analysis and is another level of hyperparameter tuning. For example, you can experiment with different base networks such as ResNet-50 or MobileNet v2, or you can try other semantic segmentation network architectures such as SegNet, fully convolutional networks (FCN), or U-Net.

```
% Specify the network image size. This is typically the same as the traing image sizes.
imageSize = [720 960 3];
% Specify the number of classes.
numClasses = numel(classes);
% Create DeepLab v3+.
lgraph = deeplabv3plusLayers(imageSize, numClasses, "resnet18");
```

Balance Classes Using Class Weighting

- As shown earlier, the classes in CamVid are not balanced. To improve training, you can use class weighting to balance the classes. Use the pixel label counts computed earlier with countEachLabel and calculate the median frequency class weights.

```
imageFreq = tbl.PixelCount ./ tbl.ImagePixelCount;
classWeights = median(imageFreq) ./ imageFreq
```

classWeights = 11×1

0.3182

0.2082

5.0924

0.1744

0.7103 ...

Balance Classes Using Class Weighting

- Specify the class weights using a pixelClassificationLayer.

```
pxLayer = pixelClassificationLayer('Name','labels','Classes',tbl.Name,'ClassWeights',classWeights);
lgraph = replaceLayer(lgraph,"classification",pxLayer);
```

Select Training Options

- The optimization algorithm used for training is stochastic gradient descent with momentum (SGDM).

```
% Define validation data.
dsVal = combine(imdsVal,pxdsVal);
% Define training options.
options = trainingOptions('sgdm', ...
    'LearnRateSchedule','piecewise',...
    'LearnRateDropPeriod',10,...
    'LearnRateDropFactor',0.3,...
    'Momentum',0.9, ...
    'InitialLearnRate',1e-3, ...
    'L2Regularization',0.005, ...
    'ValidationData',dsVal,...
    'MaxEpochs',30, ...
    'MiniBatchSize',8, ...
    'Shuffle','every-epoch', ...
    'CheckpointPath', tempdir, ...
    'VerboseFrequency',2,...
    'Plots','training-progress',...
    'ValidationPatience', 4);
```

Data Augmentation

- Data augmentation is used to improve network accuracy by randomly transforming the original data during training. By using data augmentation, you can add more variety to the training data without increasing the number of labeled training samples. To apply the same random transformation to both image and pixel label data use datastore combine and transform. First, combine imdsTrain and pxdsTrain.

```
dsTrain = combine(imdsTrain, pxdsTrain);
```

- Next, use datastore transform to apply the desired data augmentation defined in the supporting function augmentImageAndLabel. Here, random left/right reflection and random X/Y translation of +/- 10 pixels is used for data augmentation.

```
xTrans = [-10 10];
yTrans = [-10 10];
dsTrain = transform(dsTrain, @(data)augmentImageAndLabel(data,xTrans,yTrans));
```

Start Training

- Start training using trainNetwork (Deep Learning Toolbox) if the doTraining flag is true. Otherwise, load a pretrained network.
- Note: The training was verified on an NVIDIA™ Titan X with 12 GB of GPU memory. If your GPU has less memory, you may run out of memory during training. If this happens, try setting 'MiniBatchSize' to 1 in trainingOptions, or reducing the network input and resizing the training data.

```
doTraining = false;
if doTraining
    [net, info] = trainNetwork(dsTrain,lgraph,options);
else
    data = load(pretrainedNetwork);
    net = data.net;
end
```

Test Network on One Image

- As a quick sanity check, run the trained network on one test image.

```
I = readimage(imdsTest,35);
C = semanticseg(I, net);
```

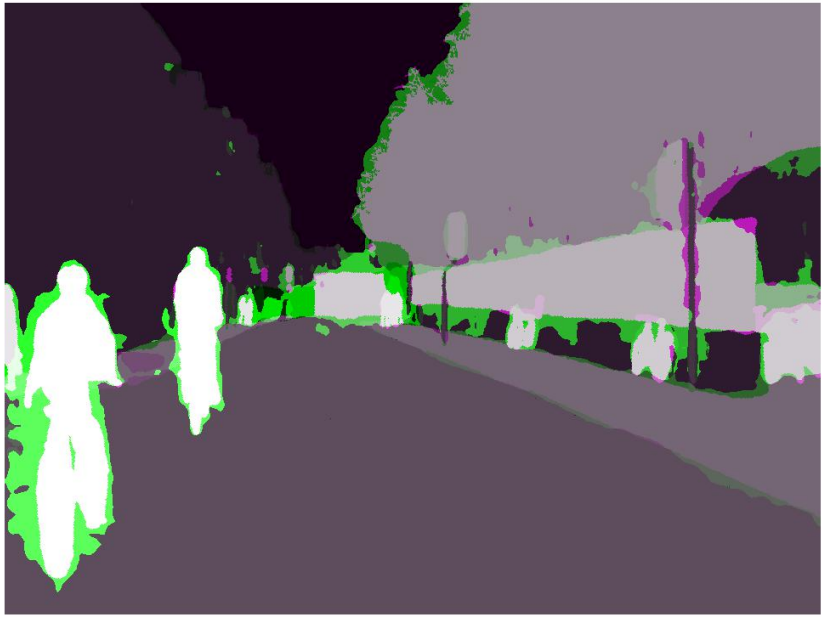
- Display the results.

```
B = labeloverlay(I,C,'Colormap',cmap, ...
                     'Transparency',0.4);
imshow(B)
pixelLabelColorbar(cmap, classes);
```

Test Network on One Image

- Compare the results in C with the expected ground truth stored in pxdsTest. The green and magenta regions highlight areas where the segmentation results differ from the expected ground truth.

```
expectedResult = readimage(pxdsTest,35);
actual = uint8(C);
expected = uint8(expectedResult);
imshowpair(actual, expected)
```



Test Network on One Image

- The amount of overlap per class can be measured using the intersection-over-union (IoU) metric, also known as the Jaccard index. Use the jaccard function to measure IoU.

```
iou = jaccard(C,expectedResult);
table(classes,iou)
```

ans=11×2 table

classes	iou
"Sky"	0.91837
"Building"	0.84479
"Pole"	0.31203
"Road"	0.93698
"Pavement"	0.82838 ...

Evaluate Trained Network

- To measure accuracy for multiple test images, runsemanticseg on the entire test set. A mini-batch size of 4 is used to reduce memory usage while segmenting images. You can increase or decrease this value based on the amount of GPU memory you have on your system.

```
pxdsResults = semanticseg(imdsTest,net, ...
    'MiniBatchSize',4, ...
    'WriteLocation',tempdir, ...
    'Verbose',false);
```

Evaluate Trained Network

- semanticseg returns the results for the test set as a pixelLabelDatastore object. Use evaluateSemanticSegmentation to measure semantic segmentation metrics on the test set results.

```
metrics = evaluateSemanticSegmentation(pxdsResults,pxdsTest,'Verbose',false);
```

- evaluateSemanticSegmentation returns various metrics for the entire dataset, for individual classes, and for each test image. To see the dataset level metrics, inspect metrics.DataSetMetrics .

```
metrics.DataSetMetrics
```

ans=1×5 table

GlobalAccuracy	MeanAccuracy	MeanIoU	WeightedIoU	MeanBFScore
0.87695	0.85392	0.6302	0.80851	0.65051

Evaluate Trained Network

metrics.ClassMetrics

ans=11×3 table

	Accuracy	IoU	MeanBFScore
Sky	0.93112	0.90209	0.8952
Building	0.78453	0.76098	0.58511
Pole	0.71586	0.21477	0.51439
Road	0.93024	0.91465	0.76696
Pavement	0.88466	0.70571	0.70919
Tree	0.87377	0.76323	0.70875
SignSymbol	0.79358	0.39309	0.48302
Fence	0.81507	0.46484	0.48566
Car	0.90956	0.76799	0.69233
Pedestrian	0.87629	0.4366	0.60792
Bicyclist	0.87844	0.60829	0.55089