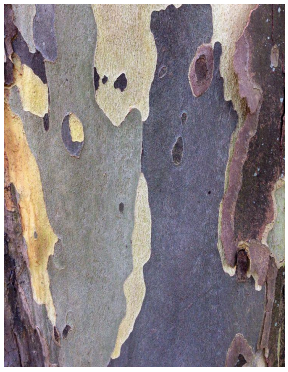
This Live Script assumes you have the following installed: Deep Learning Toolbox, Image Processing Toolbox, Computer Vision Toolbox, and Statistics and Machine Learning Toolbox. You will also need to install AlexNet from the Add-Ons module above. This script was tested on MATLAB 2019a and 2019b.

- Made by Ricky Medrano for COMP546 Pattern Recognition final project Fall 2019 semester.

Classifying Trees Based On Bark



Sycamore

Pepper

Jacaranda

```
clc, clear, close all;
load barknetmat.mat %comment if you wanto to train the network on your computer
```

Read in Images, Get Labels, and Totals

```
imds = imageDatastore('images','IncludeSubfolders', true, 'LabelSource', 'foldernames');
treeLabels = imds.Labels;
numClasses = numel(categories(imds.Labels));
numLabelsPerClass = countEachLabel(imds);
```

Verify Images Read In Properly

```
idx = randperm(numel(treeLabels),9);
out = imtile(imds,'Frames',idx,'GridSize',[3 3],'BorderSize', 25);
imshow(out);
```

Split up Images Into Training and Validation

I choose 80% of our images for training and 20% of images for testing/validation. I also make sure to randomize the split to avoid biasing the results.

```
[barkTrain,barkTest] = splitEachLabel(imds,0.8, 'randomized');
trainingLabels = barkTrain.Labels;
testingLabels = barkTest.Labels;
```

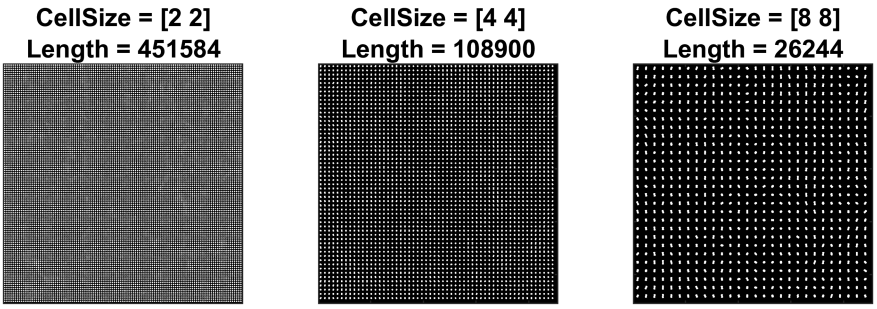
Option 1: Machine Learning - HOG Features

Histogram of Oriented Gradients

- HOG can be used as feature to classify objects based on shapes in the image
- HOG features are extracted from an image and saved in a feature vector
- Graidents are changes in intensity
- Orientation comes from magnitude of the gradients, so think of orientated gradients as vectors in space
- These are then encoded into a histogram
- The amount of shape information extracted depends on the initial pixel cell size moving across the image

Here We Test Some Different Cell Sizes To Pass to extractHOGFeatures()

```
img = readimage(imds,15);
[hog_2x2, vis2x2] = extractHOGFeatures(img,'CellSize',[2 2]);
[hog_4x4, vis4x4] = extractHOGFeatures(img,'CellSize',[4 4]);
[hog_8x8, vis8x8] = extractHOGFeatures(img,'CellSize',[8 8]);
figure;
subplot(2,3,1:3); imshow(img);
% Visualize the HOG features
subplot(2,3,4);
plot(vis2x2);
title({'CellSize = [2 2]'; ['Length = ' num2str(length(hog_2x2))]});
subplot(2,3,5);
plot(vis4x4);
title({'CellSize = [4 4]'; ['Length = ' num2str(length(hog_4x4))]});
subplot(2,3,6);
plot(vis8x8);
title({'CellSize = [8 8]'; ['Length = ' num2str(length(hog_8x8))]});
```



Choose Cell Size And Overlay with HOG Visual To Verify It's Reading The Right Amount of Shape Features for Your Specific Images

```
cellSize = [8 8];
hogFeatureSize = length(hog_8x8);
figure; imshow(img); hold on; plot(vis8x8);
```

Extract Hog Features From Training Image Set

```
numImages = numel(barkTrain.Files);
HOGtrainingFeatures = zeros(numImages, hogFeatureSize, 'single');
for i = 1:numImages
    img = readimage(barkTrain, i);
    img = rgb2gray(img);
    img = imbinarize(img);
    HOGtrainingFeatures(i, :) = extractHOGFeatures(img, 'CellSize', cellSize);
end
```

Train Multi-Class SVM Classifier Using Extracted HOG Features on Training Set

```
SVM_HOG_Classifier = fitcecoc(HOGtrainingFeatures, trainingLabels,"FitPosterior",true);
```

Extract Hog Features From Test Image Set

```
numImages = numel(barkTest.Files);
HOGtestFeatures = zeros(numImages, hogFeatureSize, 'single');
for i = 1:numImages
    img = readimage(barkTest, i);
    img1 = rgb2gray(img);
    img = imbinarize(img);
    HOGtestFeatures(i, :) = extractHOGFeatures(img, 'CellSize', cellSize);
end
```

Classify/Predict the Test Images

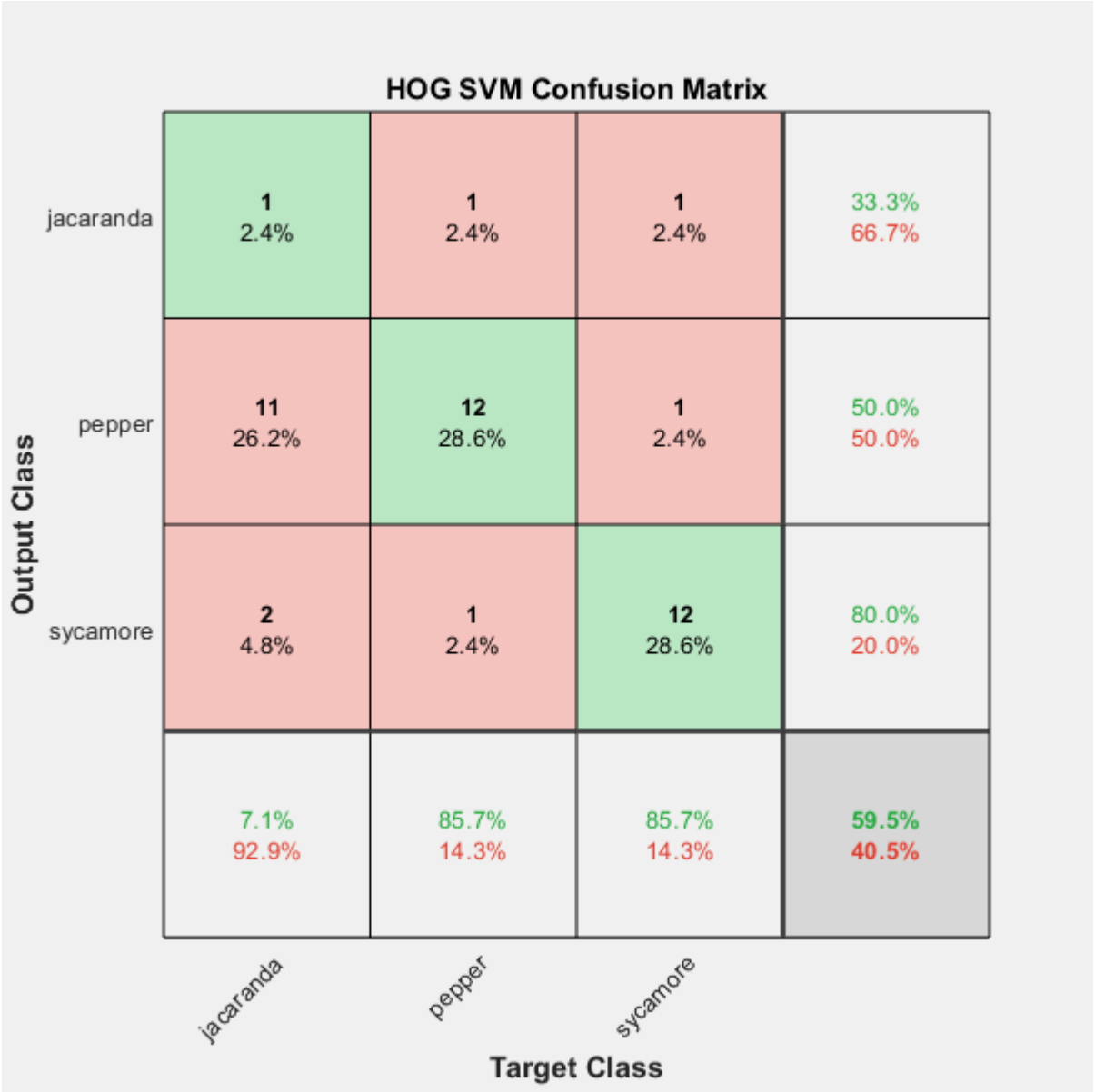
```
[predicted_HOG_Labels, probs, ~] = predict(SVM_HOG_Classifier, HOGtestFeatures);
```

View Some Predicted Labels

```
idx = randperm(numel(barkTest.Files),4);
figure
for i = 1:4
    subplot(2,2,i)
    I = readimage(barkTest,idx(i));
    imshow(I);
    label = predicted_HOG_Labels(idx(i));
    title(string(label) + ", " + num2str(100+(100*max(probs(idx(i),:))),3) + "%");
end
```

HOG SVM Confusion Matrix

```
plotconfusion(testingLabels,predicted_HOG_Labels,'HOG SVM');
```



Option 2: Machine Learning - Bag of Words with SURF Features

Speeded Up Robost Feature (SURF)

- Detects local features
- Local features - a pattern in an image such as a point, edge, or patch, that differs from its immediate surroundings
- Invariant to scale and rotation
- Bag of Words function in Matlab extacts SURF feature descriptors under the hood
- SURF descriptors used to train a multiclass SVM classifier

Train SVM Classifier on Bag of Words Features

```
bagOfWords = bagOfFeatures(barkTrain, 'PointSelection','Detector');
```

```
Creating Bag-Of-Features.
-------------------------
* Image category 1: jacaranda
* Image category 2: pepper
* Image category 3: sycamore
* Selecting feature point locations using the Detector method.
* Extracting SURF features from the selected feature point locations.
** detectSURFFeatures is used to detect key points for feature extraction.

* Extracting features from 162 images...done. Extracted 49220 features.

* Keeping 80 percent of the strongest features from each category.

* Balancing the number of features across all image categories to improve clustering.
** Image category 3 has the least number of strongest features: 9008.
** Using the strongest 9008 features from each of the other image categories.

* Using K-Means clustering to create a 500 word visual vocabulary.
* Number of features          : 27024
* Number of clusters (K)      : 500

* Initializing cluster centers...100.00%.
* Clustering...completed 14/100 iterations (~0.07 seconds/iteration)...converged in 14 iterations.

* Finished creating Bag-Of-Features
```

```
bagOfWordsClassifier = trainImageCategoryClassifier(barkTrain, bagOfWords);
```

```
Training an image category classifier for 3 categories.
--------------------------------------------------------
* Category 1: jacaranda
* Category 2: pepper
* Category 3: sycamore

* Encoding features for 162 images...done.

* Finished training the category classifier. Use evaluate to test the classifier on a test set.
```

Evalute Bag of Words Classifier on Testing Set

```
[BoWConfMatrix, ~,predLabel, ~] = evaluate(bagOfWordsClassifier, barkTest);
```

```
Evaluating image category classifier for 3 categories.
-------------------------------------------------------

* Category 1: jacaranda
* Category 2: pepper
* Category 3: sycamore

* Evaluating 42 images...done.

* Finished evaluating all the test sets.

* The confusion matrix for this test set is:


                         PREDICTED
KNOWN        | jacaranda   pepper   sycamore
---------------------------------------------
jacaranda    | 0.71        0.29     0.00
```

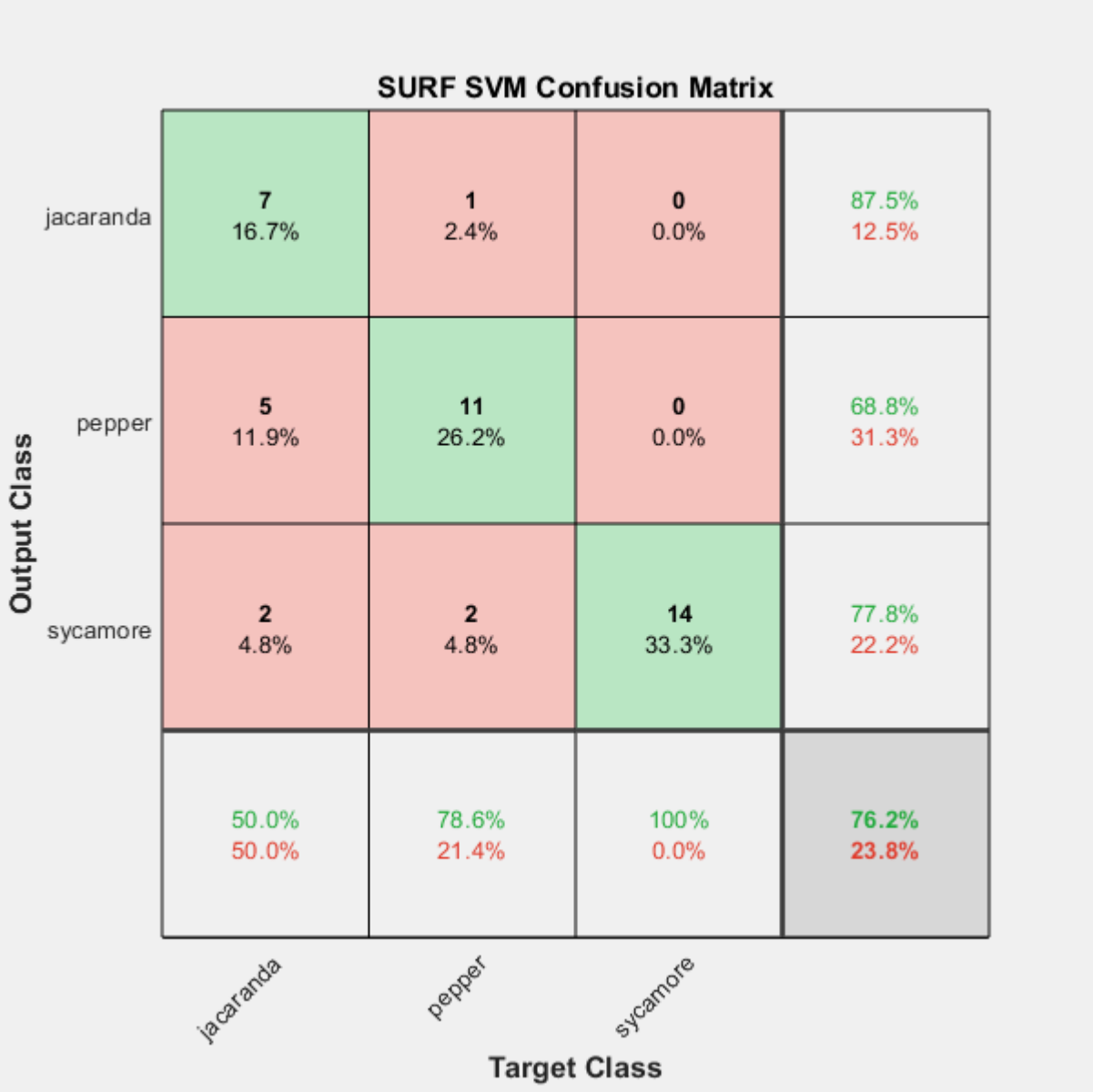
```
pepper       | 0.14        0.71     0.14
sycamore     | 0.07        0.07     0.86

* Average Accuracy is 0.76.
```

Do Some Necessary Processing of Labels

```
test = cell(size(predLabel,1),1);
for i = 1:size(predLabel,1)
    if predLabel(i) == 1
        test{i} = 'jacaranda';
    elseif predLabel(i) == 2
        test{i} = 'pepper';
    else
        test{i} = 'sycamore';
    end
end
predBoWlabels = categorical(test);
mean(diag(BoWConfMatrix));
```

Bag of Words SVM Confusion Matrix

```
plotconfusion(testingLabels,predBoWlabels,'SURF SVM');
```



Option 3: Machine Learning - Texture Enhancement

Texture Filtering By Local Entropy

- Entropy is a statistical measure of randomness
- Calculated as -sum(p*log2(p)) where p is the normalized histogram counts for the pixel values
- Local entropy is calculated from a size 9 neighborhood, by default with entropyfilt(), which produces a textured image

Before the next step was run, I texturized the original imageset with the entropyfilt() function and resized the images properly. This was done by running the Matlab app Image Batch Processor and

giving it my "resize227227Texture.m" file. These new texturized images were then saved into the folder "imagesWithTexture"

Create New Image DataStore with Textured Images from entropyfilt()

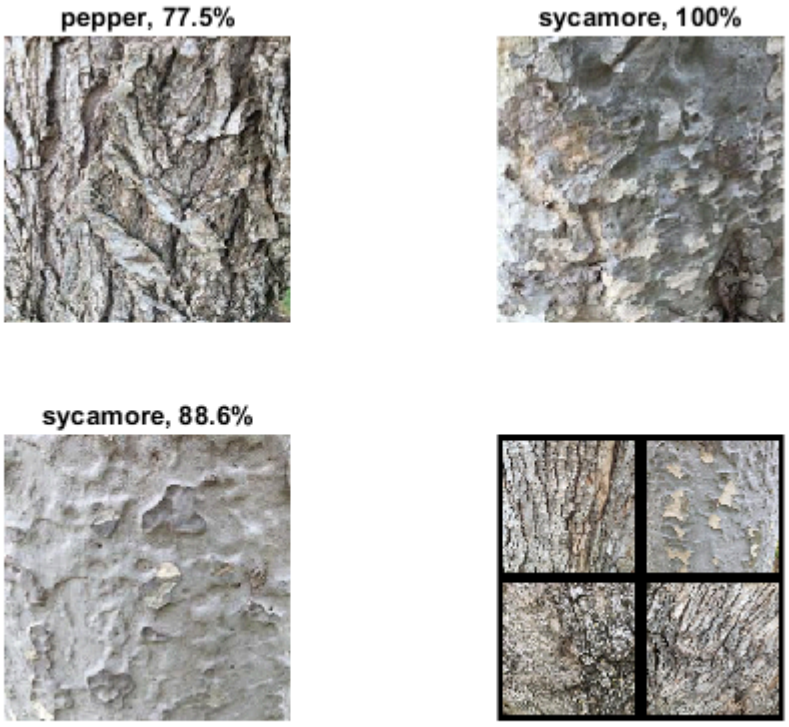
```
imdsTexture = imageDatastore('imagesWithTexture','IncludeSubfolders', true, 'LabelSource', 'fol
```

Verify Texture Images Read In Properly And See How They Look Before and After

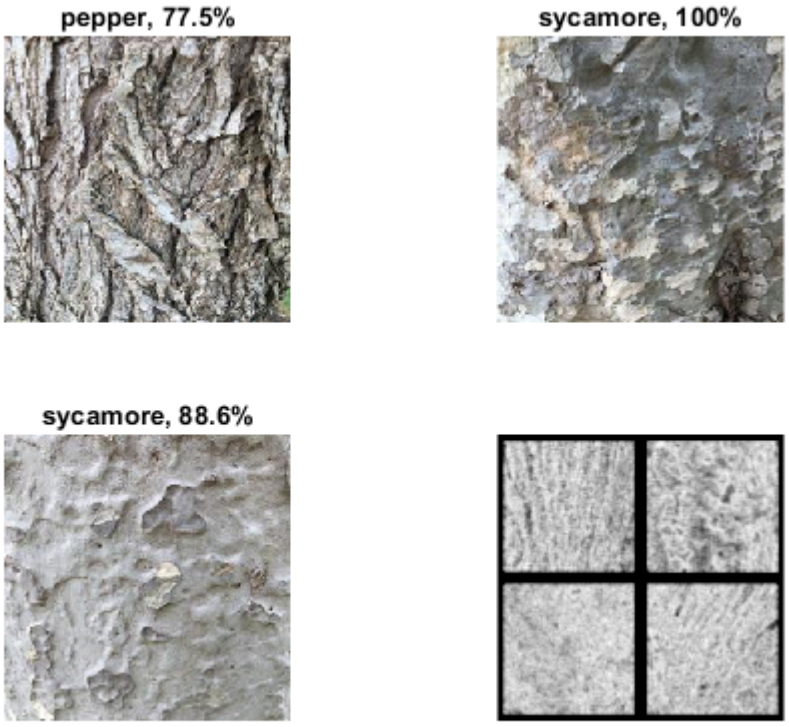
```
idx = randperm(numel(imdsTexture.Labels),4);
after = imtile(imdsTexture,'Frames',idx,'GridSize',[2 2],'BorderSize', 10);
before = imtile(imds,'Frames',idx,'GridSize',[2 2],'BorderSize', 10);
```

There is currently a bug in 2019b that doesn't display the next two imshows() correctly. The correct output appears in the 4th tiles, but it should only appear as one picture instead of this 4x4 grid. Most likely a memory purge issue from using imtime() further up in the code.

```
imshow(before);
```




```
imshow(after);
```



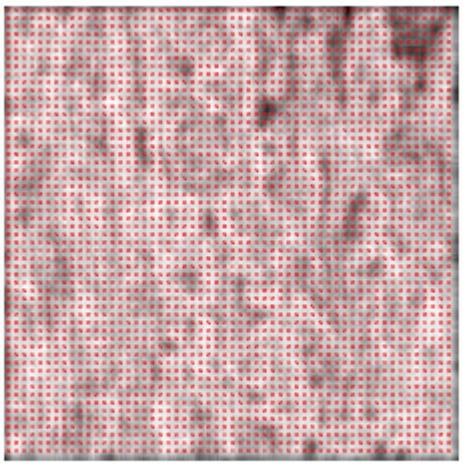
Split Texturized Images Into Training and Validation

```
[barkTrainTexture,barkTestTexture] = splitEachLabel(imdsTexture,0.8, 'randomized');
trainingLabelsTexture = barkTrainTexture.Labels;
testingLabels = barkTestTexture.Labels;
```

So now we have our same pictures but with only texture information encoded in them. We can now use these images to train our HOG and Bag of Words classifier again to see if we get better results.

Verify Appropriate Cell Size To Use When Extracting HOG Features on Texture Images

```
imgTexture = readimage(imdsTexture,15);
[hog_4x4Texture, vis4x4Texture] = extractHOGFeatures(imgTexture,'CellSize',[4 4]);
cellSize = [4 4];
hogFeatureSizeTexture = length(hog_4x4Texture);
figure; imshow(imgTexture); hold on; plot(vis4x4Texture, 'Color', 'Red');
```



Extract Hog Features From Training Texture Image Set

```
numImagesTexture = numel(barkTrainTexture.Files);
HOGtrainingFeaturesTexture = zeros(numImagesTexture, hogFeatureSizeTexture, 'single');
for i = 1:numImagesTexture
    imgTexture = readimage(barkTrainTexture, i);
    imgTexture = imbinarize(imgTexture);
    HOGtrainingFeaturesTexture(i, :) = extractHOGFeatures(imgTexture, 'CellSize', cellSize);
end
```

Train Multi-Class SVM Classifier Using Extracted HOG Features on Texture Training Set

```
SVM_HOG_ClassifierTexture = fitcecoc(HOGtrainingFeaturesTexture, barkTrainTexture.Labels,'FitPo
```

Extract Hog Features From Texturized Test Image Set

```
numImagesTexture = numel(barkTestTexture.Files);
HOGtestFeaturesTexture = zeros(numImagesTexture, hogFeatureSizeTexture, 'single');
for i = 1:numImagesTexture
    imgTexture = readimage(barkTestTexture, i);
    imgTexture = imbinarize(imgTexture);
    HOGtestFeaturesTexture(i, :) = extractHOGFeatures(imgTexture, 'CellSize', cellSize);
end
```

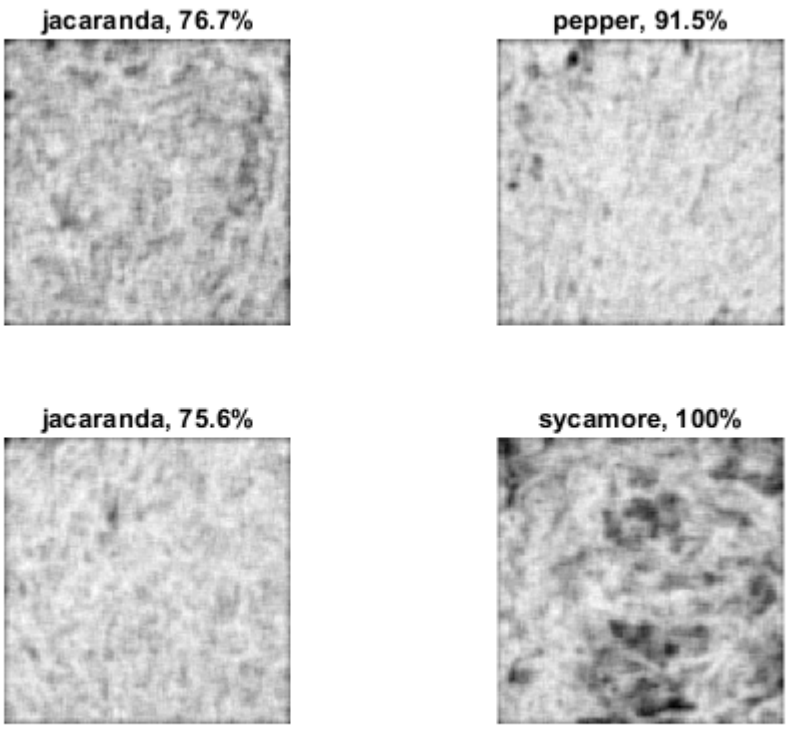
Classify/Predict the Texturized Test Images

```
[predicted_HOG_LabelsTexture, probs, cost] = predict(SVM_HOG_ClassifierTexture, HOGtestFeatures
```

View Some Predicted Texture Labels

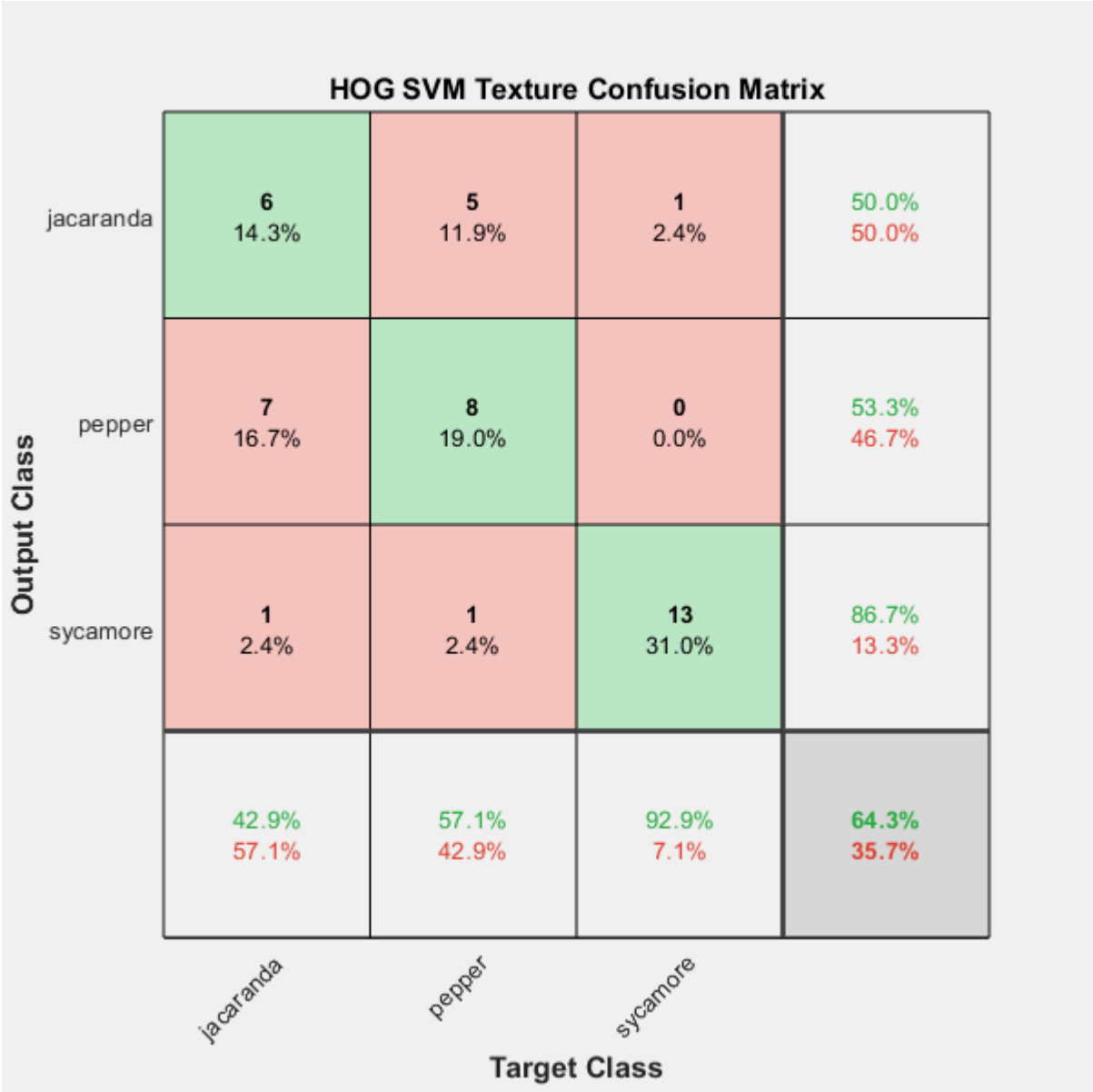
```
idx = randperm(numel(barkTestTexture.Files),4);
```

```
figure
for i = 1:4
    subplot(2,2,i)
    I = readimage(barkTestTexture,idx(i));
    imshow(I)
    label = predicted_HOG_LabelsTexture(idx(i));
    title(string(label) + ", " + num2str(100+(100*max(probs(idx(i),:))),3) + "%");
end
```



HOG Texture SVM Confusion Matrix

```
plotconfusion(barkTestTexture.Labels,predicted_HOG_LabelsTexture,'HOG SVM Texture');
```



Train SVM Classifier on Bag of Words Features on Texturized Images

```
bagOfWordsTexture = bagOfFeatures(barkTrainTexture, 'PointSelection','Detector');
```

```
Creating Bag-Of-Features.
-------------------------
* Image category 1: jacaranda
* Image category 2: pepper
* Image category 3: sycamore
* Selecting feature point locations using the Detector method.
* Extracting SURF features from the selected feature point locations.
** detectSURFFeatures is used to detect key points for feature extraction.

* Extracting features from 162 images...done. Extracted 25845 features.

* Keeping 80 percent of the strongest features from each category.

* Balancing the number of features across all image categories to improve clustering.
** Image category 2 has the least number of strongest features: 5274.
** Using the strongest 5274 features from each of the other image categories.
```

```
* Using K-Means clustering to create a 500 word visual vocabulary.
* Number of features          : 15822
* Number of clusters (K)      : 500

* Initializing cluster centers...100.00%.
* Clustering...completed 23/100 iterations (~0.07 seconds/iteration)...converged in 23 iterations.

* Finished creating Bag-Of-Features
```

```
bagOfWordsTextureClassifier = trainImageCategoryClassifier(barkTrainTexture, bagOfWordsTexture)
```

```
Training an image category classifier for 3 categories.
--------------------------------------------------------
* Category 1: jacaranda
* Category 2: pepper
* Category 3: sycamore

* Encoding features for 162 images...done.

* Finished training the category classifier. Use evaluate to test the classifier on a test set.
```

Evalute Bag of Words Classifier on Testing Texture Image Set

```
[BoWTextureConfMatrix, ~,predLabelTexture, ~] = evaluate(bagOfWordsTextureClassifier, barkTestT
```

```
Evaluating image category classifier for 3 categories.
-------------------------------------------------------

* Category 1: jacaranda
* Category 2: pepper
* Category 3: sycamore

* Evaluating 42 images...done.

* Finished evaluating all the test sets.

* The confusion matrix for this test set is:


                          PREDICTED
KNOWN        | jacaranda   pepper   sycamore
-----------------------------------------
jacaranda    | 0.64        0.29     0.07
pepper       | 0.36        0.57     0.07
sycamore     | 0.07        0.14     0.79

* Average Accuracy is 0.67.
```

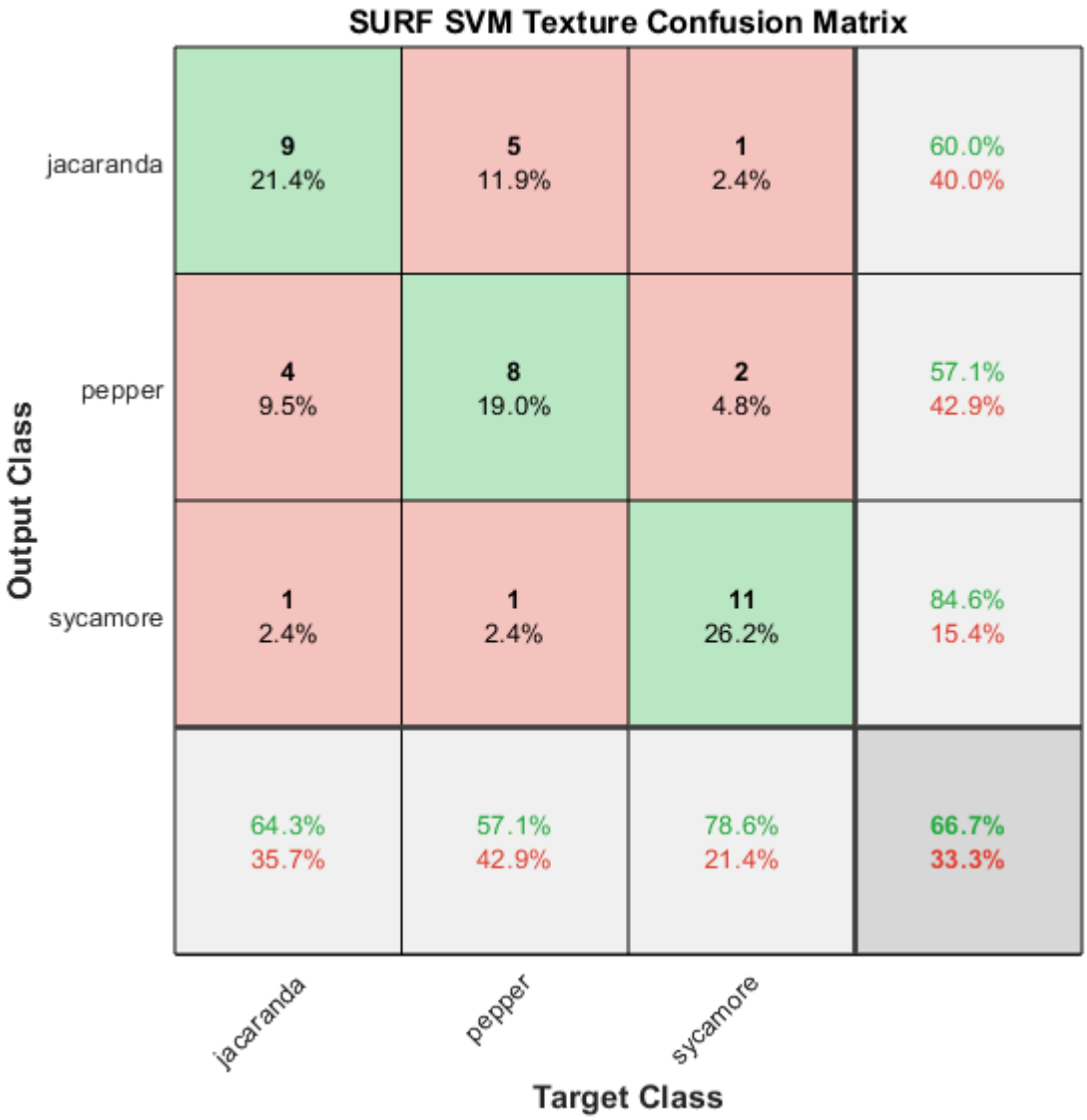
Doing some label cleanup

```
test = cell(size(predLabelTexture,1),1);
for i = 1:size(predLabelTexture,1)
    if predLabelTexture(i) == 1
        test{i} = 'jacaranda';
    elseif predLabelTexture(i) == 2
        test{i} = 'pepper';
    else
        test{i} = 'sycamore';
    end
```

```
end
predBoWTexturelabels = categorical(test);
mean(diag(BoWTextureConfMatrix));
```

Bag of Words Texture SVM Confusion Matrix

```
plotconfusion(barkTestTexture.Labels,predBoWTexturelabels, 'SURF SVM Texture');
```



Option 4: Transfer Learning Using AlexNet

Augment Only the Size Down to Input Size of AlexNet

```
nonaugdsTrain = augmentedImageDatastore([227 227 3], barkTrain);
nonaugdsTest = augmentedImageDatastore([227 227 3], barkTest);
```

Verify The Augmentations Look Correct

```
minibatch = preview(nonaugdsTrain);
imshow(imtile(minibatch.input));
```

Load AlexNet and Substitute In Necessary Layers

```
net = alexnet;
```

Checkout the layers of the net

```
net.Layers
```

```
ans =
  25x1 Layer array with layers:

     1   'data'     Image Input                 227x227x3 images with 'zerocenter' normalization
     2   'conv1'    Convolution                 96 11x11x3 convolutions with stride [4  4] and padding [0  0  0  0
     3   'relu1'    ReLU                        ReLU
     4   'norm1'    Cross Channel Normalization cross channel normalization with 5 channels per element
     5   'pool1'    Max Pooling                 3x3 max pooling with stride [2  2] and padding [0  0  0  0]
     6   'conv2'    Grouped Convolution         2 groups of 128 5x5x48 convolutions with stride [1  1] and padding
     7   'relu2'    ReLU                        ReLU
     8   'norm2'    Cross Channel Normalization cross channel normalization with 5 channels per element
     9   'pool2'    Max Pooling                 3x3 max pooling with stride [2  2] and padding [0  0  0  0]
    10   'conv3'    Convolution                 384 3x3x256 convolutions with stride [1  1] and padding [1  1  1
    11   'relu3'    ReLU                        ReLU
    12   'conv4'    Grouped Convolution         2 groups of 192 3x3x192 convolutions with stride [1  1] and paddin
    13   'relu4'    ReLU                        ReLU
    14   'conv5'    Grouped Convolution         2 groups of 128 3x3x192 convolutions with stride [1  1] and paddin
    15   'relu5'    ReLU                        ReLU
    16   'pool5'    Max Pooling                 3x3 max pooling with stride [2  2] and padding [0  0  0  0]
    17   'fc6'      Fully Connected             4096 fully connected layer
    18   'relu6'    ReLU                        ReLU
    19   'drop6'    Dropout                     50% dropout
    20   'fc7'      Fully Connected             4096 fully connected layer
    21   'relu7'    ReLU                        ReLU
    22   'drop7'    Dropout                     50% dropout
    23   'fc8'      Fully Connected             1000 fully connected layer
    24   'prob'     Softmax                     softmax
    25   'output'   Classification Output       crossentropyex with 'tench' and 999 other classes
```

Get the required input size for images to the net

```
inputSize = net.Layers(1).InputSize
```

```
inputSize = 1×3
   227   227     3
```

Capture the layers to modify

```
layers = net.Layers;
```

Replace the last fully connected layer with your own based on the number of classes in your project

```
layers(end-2) = fullyConnectedLayer(numClasses);
```

Replace the very last layer with your own

```
layers(end) = classificationLayer;
```

Set training algorithm options - changing any of those will affect your accuracy and training time

```
miniBatchSize = 10;
valFrequency = floor(numel(nonaugdsTrain.Files)/miniBatchSize);
options = trainingOptions('sgdm', ...
    'MiniBatchSize',miniBatchSize, ...
    'MaxEpochs',6, ...
    'InitialLearnRate',1e-4, ...
    'Shuffle','every-epoch', ...
```

```
    'ValidationData',nonaugdsTest, ...
    'ValidationFrequency',3, ...
    'ExecutionEnvironment','auto',...
    'Verbose',false, ...
    'Plots','training-progress');
```

Perform Training - this can take a while if you're on a slow computer or laptop.

```
%[barknet,info] = trainNetwork(nonaugdsTrain, layers, options); %Uncomment
%if you want to train the network on your computer
```

Classify Testing Images

```
[preds, probs] = classify(barknet,nonaugdsTest);
```

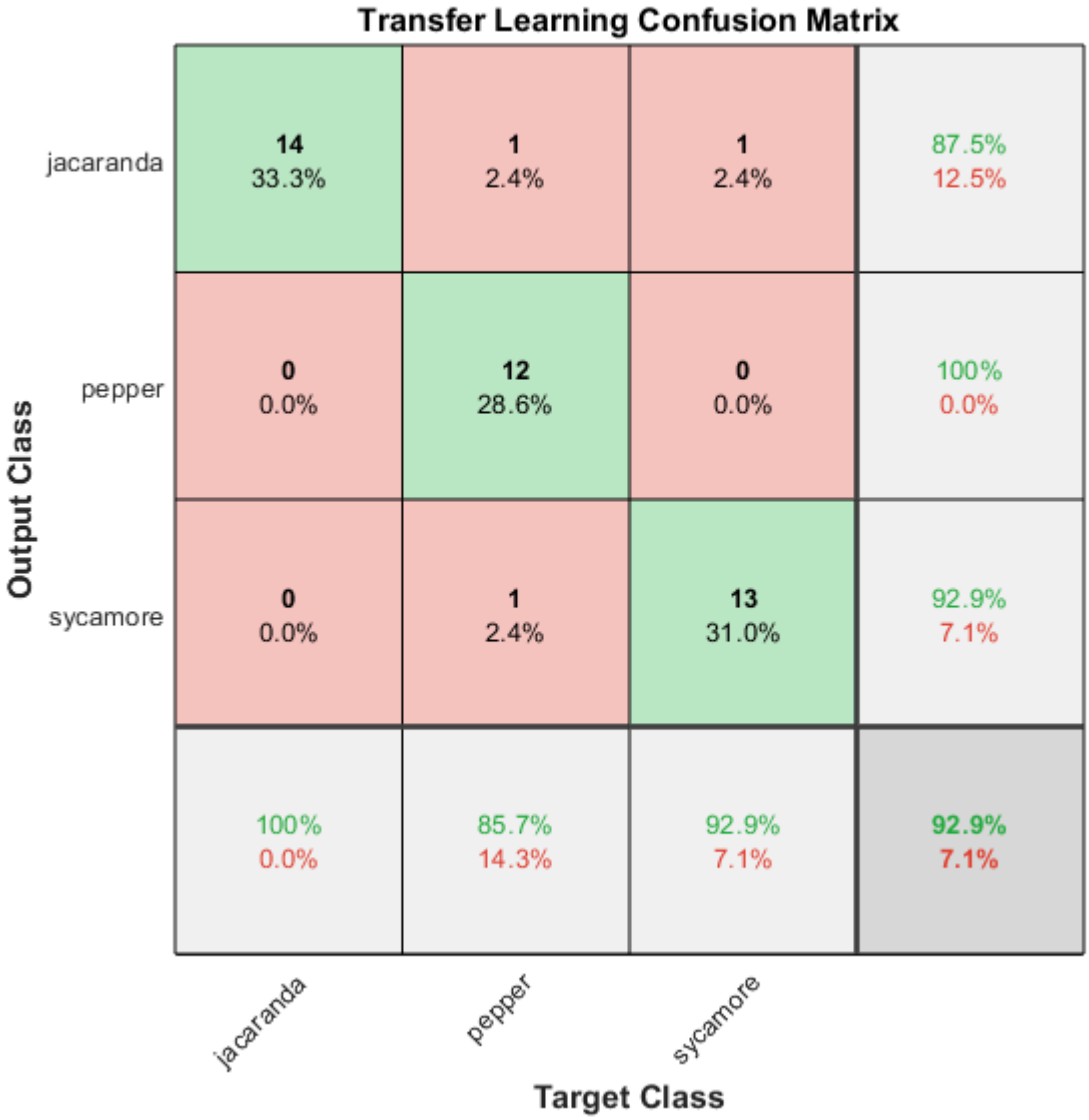
Get Labels, Number Correct, Calculate Proportion Correct

```
numCorrect = nnz(preds == barkTest.Labels);
fracCorrect = numCorrect/numel(barkTest.Labels)
```

```
fracCorrect = 0.9286
```

Show Confusion Chart

```
plotconfusion(barkTest.Labels, preds,'Transfer Learning');
```



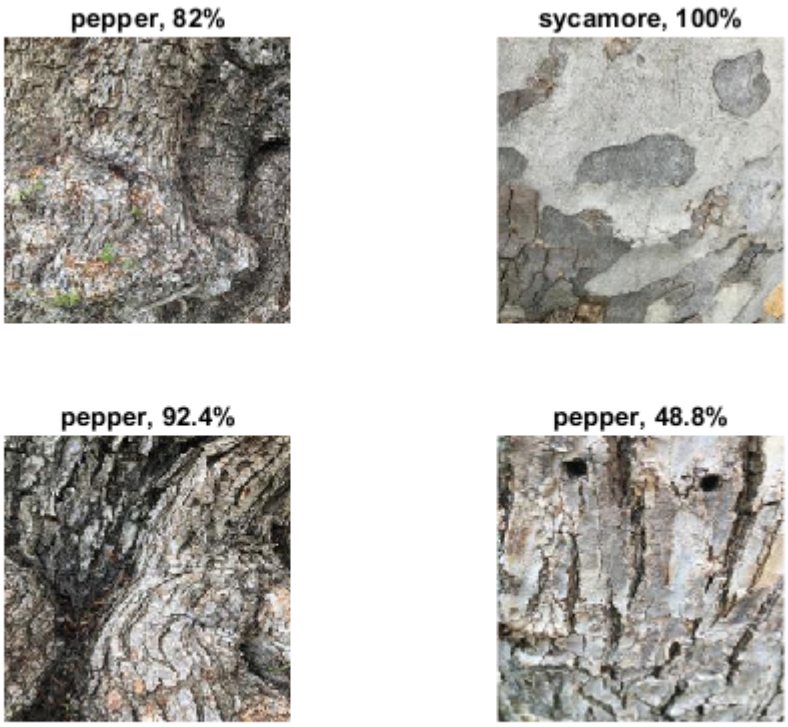
The goal of training is to minimize the loss function, which is a measure of how poorly the network performs of the training data. Loss also accounts for misclassification that incorporates the probability of each class, based on the distrubtion of the data.

One of the first adjustments to training a network is the learning rate. If you pick to big a value, weights can potential blow up or you get bad performance out of your network. When this happens, divide your current learning rate by 10. A good starting learning rate is 0.001.

Be careful with tuning the options too carefully as it can lead to over-fitting. This is when the network performs very well on the training images, but much worse on the test images. To prevent this, use validation images during training of the network. Visualize the validation loss and when it starts to rise, stop the training because letting it rise will mean it will start to over-fit.

Display 4 images with labels and probabilities

```
idx = randperm(numel(barkTest.Files),4);
figure
for i = 1:4
    subplot(2,2,i)
    I = readimage(barkTest,idx(i));
    imshow(I)
    label = preds(idx(i));
    title(string(label) + ", " + num2str(100*max(probs(idx(i),:)),3) + "%");
end
```



Sources

- https://www.mathworks.com/help/vision/ug/local-feature-detection-and-extraction.html
- https://www.mathworks.com/help/vision/ref/extracthogfeatures.html
- https://www.mathworks.com/help/stats/fitcecoc.html