```
close all;
clear variables;
clc;
```

Read the training, validation and test partitions from the relevant

text files.
*** Adjust the file path as required. ***

```
folder = "CUB_200_2011_Subset20classes";
trainingImageNames = readtable(fullfile(folder, "train.txt"),'ReadVariableNames', false);
trainingImageNames.Properties.VariableNames = {'index', 'imageName'};

validationImageNames = readtable(fullfile(folder, "validate.txt"),'ReadVariableNames', false);
validationImageNames.Properties.VariableNames = {'index', 'imageName'};

testImageNames = readtable(fullfile(folder, "test.txt"),'ReadVariableNames', false);
testImageNames.Properties.VariableNames = {'index', 'imageName'};
```

Read class info from the relevant text files

```
classNames = readtable(fullfile(folder, "classes.txt"),'ReadVariableNames', false);
classNames.Properties.VariableNames = {'index', 'className'};

imageClassLabels = readtable(fullfile(folder, "image_class_labels.txt"),'ReadVariableNames', fa
imageClassLabels.Properties.VariableNames = {'index', 'classLabel'};
```

```
% bounding box
boundingBox = readtable(fullfile(folder, "bounding_boxes.txt"),'ReadVariableNames', false);
boundingBox.Properties.VariableNames = {'index', 'x', 'y', 'w', 'h'};
```

Create lists of image names for training, validation and test subsets.

To be precise, we create an array of strings containing the full file
path and file names for each data partition.

```
folder = "CUB_200_2011_Subset20classes/";
trainingImageList = strings(height(trainingImageNames), 1);
for iI = 1:height(trainingImageNames)
    trainingImageList(iI) = string(fullfile(folder, "images/", ...
        string(cell2mat(trainingImageNames.imageName(iI)))));
end

validationImageList = strings(height(validationImageNames), 1);
for iI = 1:height(validationImageNames)
    validationImageList(iI) = string(folder + "images/" + ...
        string(cell2mat(validationImageNames.imageName(iI))));
end
```

```
testImageList = strings(height(testImageNames), 1);
for iI = 1:height(testImageNames)
    testImageList(iI) = string(folder + "images/" + ...
        string(cell2mat(testImageNames.imageName(iI))));
end
```

```
% mapping bounding boxes
trainingBox = return_bounding_box_mapping(trainingImageNames, boundingBox);
validationBox = return_bounding_box_mapping(validationImageNames, boundingBox);
testBox = return_bounding_box_mapping(testImageNames, boundingBox);
```

Create image datastores for training, validation and test subsets

```
trainingImageDS = imageDatastore(trainingImageList, 'labelSource', 'foldernames', ...
    'FileExtensions', {'.jpg'});
trainingImageDS.ReadFcn = @readImagesIntoDatastore;

validationImageDS = imageDatastore(validationImageList, 'labelSource', 'foldernames', ...
    'FileExtensions', {'.jpg'});
validationImageDS.ReadFcn = @readImagesIntoDatastore;

testImageDS = imageDatastore(testImageList, 'labelSource', 'foldernames', ...
    'FileExtensions', {'.jpg'});
testImageDS.ReadFcn = @readImagesIntoDatastore;
```

```
% apply bounding box
trainingImageDS.ReadFcn = @(file_name) read_bounding_box_image_to_datastore(file_name, training
validationImageDS.ReadFcn = @(file_name) read_bounding_box_image_to_datastore(file_name, valida
testImageDS.ReadFcn = @(file_name) read_bounding_box_image_to_datastore(file_name, testBox);
```

The images all have different spatial resolutions (width x height), so

need to resize them to the same size. (Experiment with different sizes!)

```
targetSize = [100, 100];
trainingImageDS_Resized = transform(trainingImageDS, @(x) imresize(x,targetSize));
validationImageDS_Resized = transform(validationImageDS, @(x) imresize(x,targetSize));
testImageDS_Resized = transform(testImageDS, @(x) imresize(x,targetSize));

% Combine transformed datastores and labels
labelsTraining = arrayDatastore(trainingImageDS.Labels);
cdsTraining = combine(trainingImageDS_Resized, labelsTraining);
labelsValidation = arrayDatastore(validationImageDS.Labels);
cdsValidation = combine(validationImageDS_Resized, labelsValidation);
labelsTest = arrayDatastore(testImageDS.Labels);
cdsTest = combine(testImageDS_Resized, labelsTest);
```

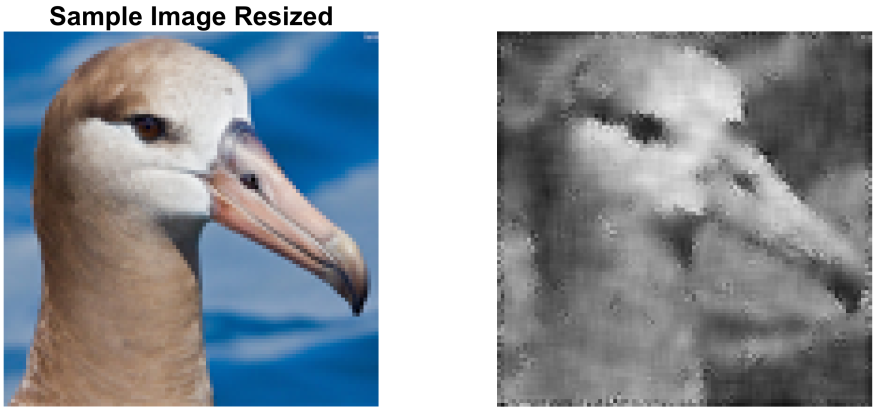
```
% %test
% gray_img = rgb2gray(img);
%
% % Create a SURF object
% surf_obj = detectSURFFeatures(gray_img);
%
% % Extract SURF features
% [ztest_surf_features, ztest_valid_points] = extractFeatures(gray_img, surf_obj);
%
% % Compute HOG features
% ztest_hog_features = extractHOGFeatures(gray_img, 'CellSize', [8 8]);
% [coeff_surf, score_surf, latent_surf] = pca(ztest_surf_features);
% [coeff_hog, score_hog, latent_hog] = pca(ztest_hog_features);
%
% % Use first n principal components to form feature vectors
% n = min(size(score_surf,2), size(score_hog,2));
% combined_features = [score_surf(:,1:n), score_hog(:,1:n)];
```

```
img = cdsTraining.read{1};
```

```
cellSize = [32 32];     % Set cell size to 16x16
[hog_8x8, vis8x8] = extractHOGFeatures(img,'CellSize',cellSize);
numImagesTrain = numel(trainingImageDS_Resized.UnderlyingDatastores{1, 1}.Files);
hogFeatureSize = length(hog_8x8);
```

```

[hog_8x8, vis8x8] = extractHOGFeatures(img,'CellSize',cellSize);
subplot(1, 2, 1);
imshow(img);
title('Sample Image Resized');
subplot(1, 2, 2);
imshow(rgb2gray(img)); hold on;
title('SIFT Feature Points');
plot(vis8x8);
hold off;
% noise reduction
J = imnoise(im2gray(img),'gaussian',0,0.025);
K = wiener2(J,[5 5]);
imshow(K);
```



```
img_equalised = histeq(im2gray(img));
[hog_eq, vis_eq] = extractHOGFeatures(img_equalised,"CellSize", cellSize);
hog_8x8(1, 145:288) = hog_eq;
```

Extract HOG features

```
%fitcsvm
%surf
% trainingFeatures = zeros(numImagesTrain, 2200, 'single');
% reset(cdsTraining);     % Make sure we start with the first image
% for i = 1:numImagesTrain
%     imgFromDS = read(cdsTraining);    % Get item from datastore. Note, this returns a cell ar
%     imgGray = im2gray(imgFromDS{1});      % Ensure images are grayscale
%     points = detectSURFFeatures(imgGray);
%     features = extractFeatures(imgGray, points);
%     sz = size(features);
%     trainingFeatures(i, 1:sz(1)*sz(2)) = reshape(features, 1, []);
% end
```

```
% surf test
%
% numImagesTest = numel(testImageDS_Resized.UnderlyingDatastores{1, 1}.Files);
%
% testFeatures = zeros(numImagesTest, 2200, 'single');
% reset(cdsTraining);     % Make sure we start with the first image
% for i = 1:numImagesTest
%     imgFromDS = read(cdsTraining);    % Get item from datastore. Note, this returns a cell ar
%     imgGray = im2gray(imgFromDS{1});      % Ensure images are grayscale
%     points = detectSURFFeatures(imgGray);
%     features = extractFeatures(imgGray, points);
%     sz = size(features);
%     testFeatures(i, 1:sz(1)*sz(2)) = reshape(features, 1, []);
% end
```

```
% surfPoints = detectSURFFeatures(rgb2gray(img));
% subplot(1, 2, 1);
% imshow(img);
% subplot(1, 2, 2);
% imshow(rgb2gray(img)); hold on;
% plot(surfPoints);
% hold off;
```

```
% HOG
numImagesTrain = numel(trainingImageDS_Resized.UnderlyingDatastores{1, 1}.Files);
trainingFeaturesHOG = zeros(numImagesTrain, hogFeatureSize, 'single');
reset(cdsTraining);     % Make sure we start with the first image
for i = 1:numImagesTrain
    imgFromDS = read(cdsTraining);    % Get item from datastore. Note, this returns a cell arra
    imgGray = im2gray(imgFromDS{1});      % Ensure images are grayscale
    %imgEqul = histeq(imgGray);
    %imgflip = flip(imgGray,2);
    temp = imnoise(imgGray, "gaussian", 0, 0.025);
    reduceNoise = wiener2(temp, [5 5]);
    trainingFeaturesHOG(i, 1:144) = extractHOGFeatures(imgGray, 'CellSize', cellSize);
    %trainingFeatures(i, 145:288) = extractHOGFeatures(imgEqul, 'CellSize', cellSize);
    trainingFeaturesHOG(i, 145:288) = extractHOGFeatures(reduceNoise, 'CellSize', cellSize);
end
```

```
% HOG test
numImagesTest = numel(testImageDS_Resized.UnderlyingDatastores{1, 1}.Files);
testFeaturesHOG = zeros(numImagesTest, hogFeatureSize, 'single');
reset(cdsTest);     % Make sure we start with the first image
for i = 1:numImagesTest
    imgFromDS = read(cdsTest);    % Get item from datastore
    imgGray = im2gray(imgFromDS{1});  % Ensure images are grayscale
    %imgEqul = histeq(imgGray);
    %imgflip = flip(imgGray,2);
    temp = imnoise(imgGray, "gaussian", 0, 0.025);
    reduceNoise = wiener2(temp, [10 10]);
    testFeaturesHOG(i, 1:144) = extractHOGFeatures(imgGray, 'CellSize', cellSize);
    %testFeatures(i, 145:288) = extractHOGFeatures(imgEqul, 'CellSize', cellSize);
    testFeaturesHOG(i, 145:288) = extractHOGFeatures(reduceNoise, 'CellSize', cellSize);
```

```
end
```

```
% %surf
% % imset = imageSet(cdsTraining.UnderlyingDatastores{1, 1}.UnderlyingDatastores{1, 1}.Files);
% imset = imageSet(trainingImageDS.Files);
% surfMetricThreshold = 1000;
% surfNumOctaves = 3;
% surfNumScaleLevels = 4;
% % Extract SURF features from the image set
% [surfFeatures, surfLocations] = helperExtractSURFFeaturesFromImageSet(trainingImageDS,surfMet
```

```
% %sift features
% numFeatures = 50;
% maxFeatures = 100;
%
% % Start by extracting features from the training set via the helper function.
% % These features will be used to train the classifier.
% [trainingFeaturesSIFT, trainingLabelsSIFT] = ...
%     helperExtractSIFTFeaturesFromImageSet(trainingImageDS_Resized.UnderlyingDatastores{1, 1},
```

```
% %sift test
% [testFeaturesSIFT, testLabelsSIFT] = ...
%     helperExtractSIFTFeaturesFromImageSet(testImageDS_Resized.UnderlyingDatastores{1, 1},   nu
```

```
Unable to resolve the name 'testImageDS_Resized.UnderlyingDatastores'.
```

Check if we have a GPU available and clear any old data from it

```
if (gpuDeviceCount() > 0)
    disp('Found GPU:');
    disp(gpuDeviceTable);
    device = gpuDevice(1);
    reset(device);  % Clear previous values that might still be on the GPU
end
```

```
Found GPU:
    Index        Name        ComputeCapability    DeviceAvailable    DeviceSelected
    _____    ___________    _________________    _______________    ______________

      1      "GRID T4-8Q"          "7.5"               true               true
```

```
% combinedFeatures = [trainingFeatures trainingFeaturesHOG];
% combinedFeatures = [trainingFeatures trainingFeaturesHOG trainingFeaturesSIFT];
% combinedFeatures = [trainingFeatures trainingFeaturesSIFT];
```

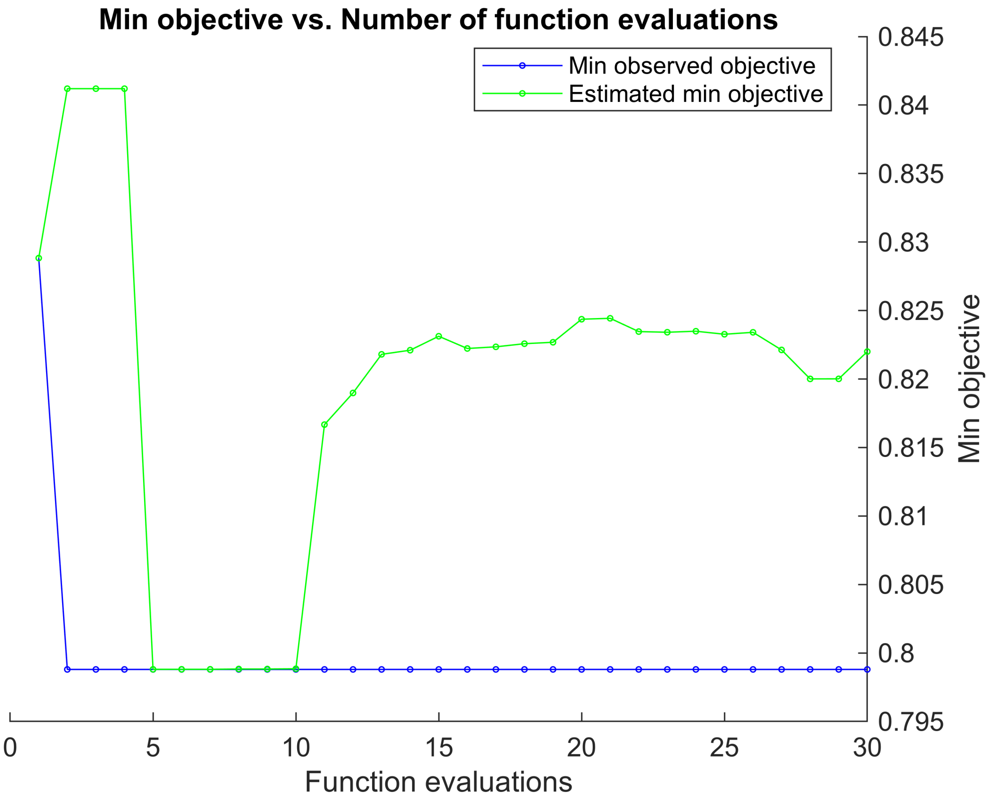
Train a multi-class SVM

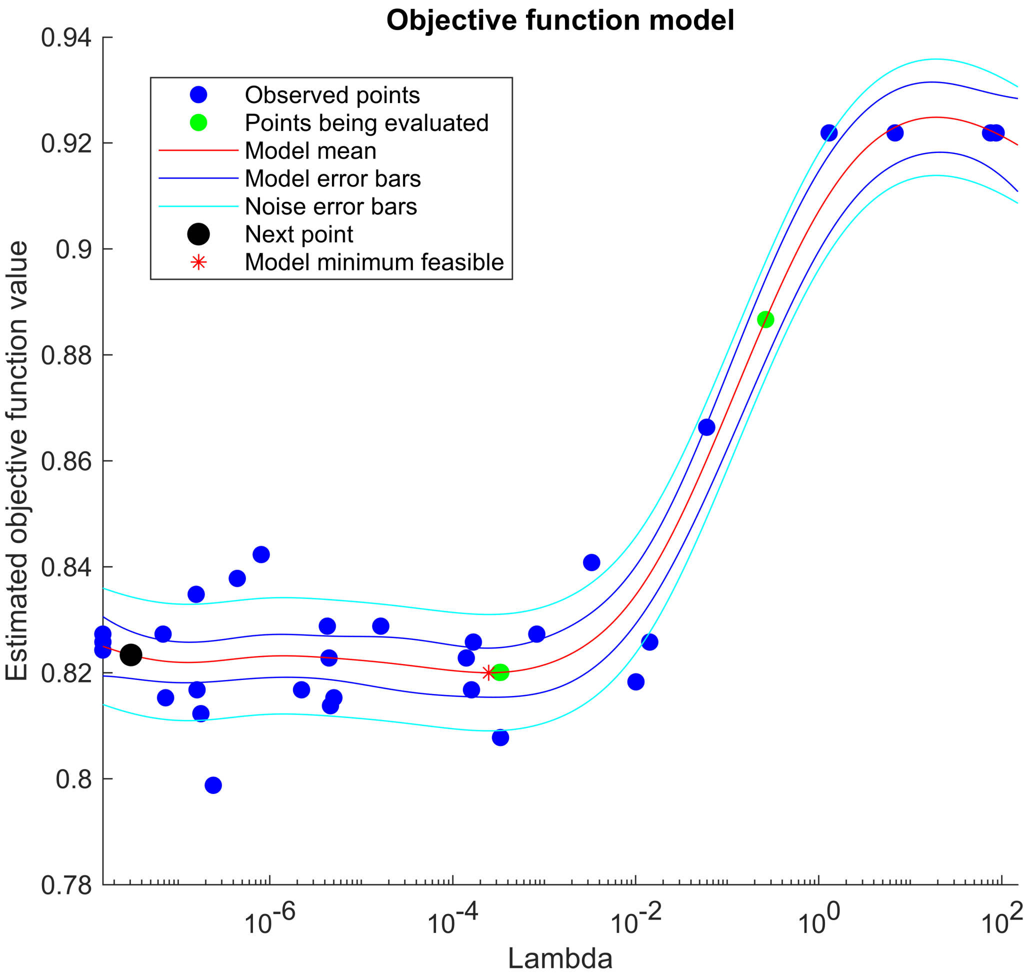
```
t = templateLinear('Solver', 'dual');
```

```
options = struct('UseParallel', true);

% fitcecoc uses multiple SVM learners and a 'One-vs-One' encoding scheme.
% Classifier = fitcecoc(trainingFeatures, trainingImageDS.Labels, 'Learners', t, 'Coding', 'one
%     'OptimizeHyperparameters', {'Lambda'}, ...
%     'HyperparameterOptimizationOptions', options);
Classifier = fitcecoc(trainingFeaturesHOG, trainingImageDS.Labels, 'Learners', t,...
    'OptimizeHyperparameters', {'Lambda'}, ...
    'HyperparameterOptimizationOptions', options);
```

```
Starting parallel pool (parpool) using the 'Processes' profile ...
Connected to the parallel pool (number of workers: 4).
Copying objective function to workers...
Done copying objective function to workers.
|================================================================================================|
| Iter | Active  | Eval   | Objective   | Objective   | BestSoFar   | BestSoFar   |       Lambda |
|      | workers | result |             | runtime     | (observed)  | (estim.)    |              |
|================================================================================================|
|    1 |       4 | Best   |     0.82883 |      24.859 |     0.82883 |     0.82883 |   1.6361e-05 |
|    2 |       2 | Accept |     0.92192 |      25.102 |      0.7988 |      0.8412 |       1.2971 |
|    3 |       2 | Best   |      0.7988 |      25.089 |      0.7988 |      0.8412 |   2.4188e-07 |
|    4 |       2 | Accept |     0.81832 |      24.962 |      0.7988 |      0.8412 |     0.010054 |
|    5 |       4 | Accept |     0.92192 |      23.865 |      0.7988 |     0.79881 |       86.731 |
|    6 |       3 | Accept |     0.92192 |      24.847 |      0.7988 |     0.79881 |       75.379 |
|    7 |       3 | Accept |     0.92192 |      24.844 |      0.7988 |     0.79881 |       6.8226 |
|    8 |       3 | Accept |     0.81532 |      25.423 |      0.7988 |     0.79883 |   7.2855e-08 |
|    9 |       4 | Accept |     0.82733 |      14.416 |      0.7988 |     0.79883 |    0.0008289 |
|   10 |       4 | Accept |     0.82432 |      12.384 |      0.7988 |     0.79885 |    1.502e-08 |
|   11 |       4 | Accept |     0.84234 |      12.585 |      0.7988 |     0.81667 |     8.08e-07 |
|   12 |       4 | Accept |     0.83784 |      13.255 |      0.7988 |     0.81898 |   4.4245e-07 |
|   13 |       4 | Accept |     0.81682 |       14.06 |      0.7988 |      0.8218 |   2.2275e-06 |
|   14 |       4 | Accept |     0.84084 |       14.73 |      0.7988 |      0.8221 |    0.0032987 |
|   15 |       4 | Accept |     0.82733 |       14.03 |      0.7988 |     0.82312 |   6.8382e-08 |
|   16 |       4 | Accept |     0.82583 |      14.951 |      0.7988 |     0.82223 |   1.5041e-08 |
|   17 |       4 | Accept |     0.86637 |      15.918 |      0.7988 |     0.82235 |     0.059643 |
|   18 |       4 | Accept |     0.82733 |      15.817 |      0.7988 |     0.82258 |   1.5024e-08 |
|   19 |       4 | Accept |     0.82282 |      15.088 |      0.7988 |     0.82268 |   0.00014077 |
|   20 |       4 | Accept |     0.83483 |      15.468 |      0.7988 |     0.82437 |   1.5729e-07 |
|================================================================================================|
| Iter | Active  | Eval   | Objective   | Objective   | BestSoFar   | BestSoFar   |       Lambda |
|      | workers | result |             | runtime     | (observed)  | (estim.)    |              |
|================================================================================================|
|   21 |       4 | Accept |     0.81532 |      15.623 |      0.7988 |     0.82443 |   5.0305e-06 |
|   22 |       4 | Accept |     0.81682 |       16.06 |      0.7988 |     0.82346 |    1.607e-07 |
|   23 |       4 | Accept |     0.82583 |      15.514 |      0.7988 |     0.82342 |   0.00016813 |
|   24 |       4 | Accept |     0.81682 |       15.06 |      0.7988 |     0.82349 |   0.00016028 |
|   25 |       4 | Accept |     0.81381 |      14.797 |      0.7988 |     0.82327 |   4.6144e-06 |
|   26 |       4 | Accept |     0.82883 |      14.983 |      0.7988 |     0.82341 |   4.2638e-06 |
|   27 |       4 | Accept |     0.81231 |      14.849 |      0.7988 |     0.82212 |   1.7707e-07 |
|   28 |       4 | Accept |     0.80781 |      15.404 |      0.7988 |     0.82001 |   0.00033284 |
|   29 |       4 | Accept |     0.82282 |      15.237 |      0.7988 |     0.82001 |   4.4514e-06 |
|   30 |       4 | Accept |     0.82583 |      15.018 |      0.7988 |       0.822 |     0.014221 |
```


Min objective vs. Number of function evaluations
Min observed objective
Estimated min objective
0.845
0.84
0.835
0.83
0.825
0.82
0.815
0.81
0.805
0.8
0.795
Min objective
0
5
10
15
20
25
30
Function evaluations



```
Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 157.1875 seconds
Total objective function evaluation time: 524.2382

Best observed feasible point:
      Lambda
    __________

    2.4188e-07

Observed objective function value = 0.7988
Estimated objective function value = 0.82221
Function evaluation time = 25.0892

Best estimated feasible point (according to models):
      Lambda
    __________

    1.5729e-07

Estimated objective function value = 0.822
Estimated function evaluation time = 15.9742
```

```
% combineTestFeature = [testFeatures testFeaturesHOG];
% combineTestFeature = [testFeatures testFeaturesHOG testFeaturesSIFT];
% combineTestFeature = [testFeatures testFeaturesSIFT];
```

Test the accuracy on the test partition

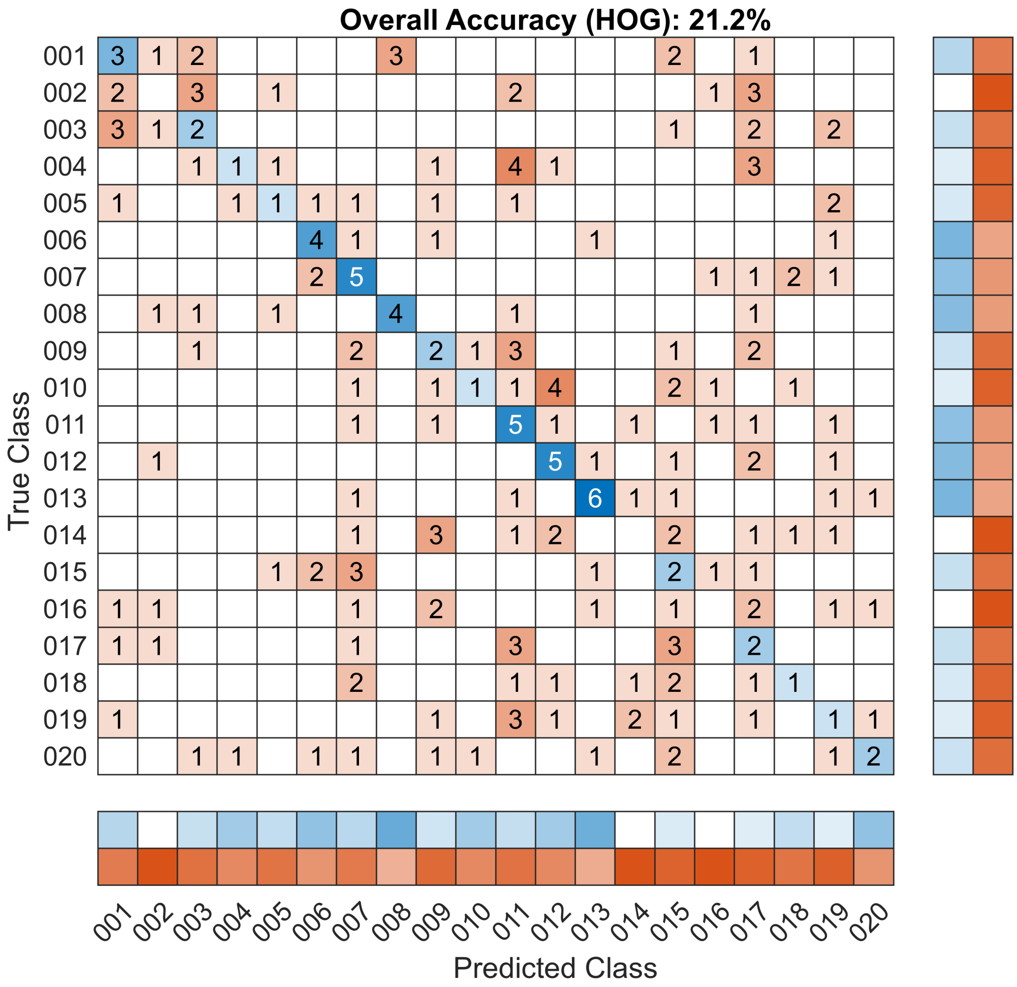
```
YPred = predict(Classifier, testFeaturesHOG);
YTest = testImageDS.Labels;

% Calculate overall accuracy
accuracy = sum(YPred == YTest)/numel(YTest) % Output on command line
```

```
accuracy = 0.2117
```

```

% Show confusion matrix in figure
[m, order] = confusionmat(YTest, YPred);
figure(2);
cm = confusionchart(m, order, ...
    'ColumnSummary','column-normalized', ...
    'RowSummary','row-normalized');
title("Overall Accuracy (HOG): "+ string(round(accuracy*100, 1)) +"%");
```



	001	002	003	004	005	006	007	008	009	010	011	012	013	014	015	016	017	018	019	020
001	3	1	2					3							2		1			
002	2		3		1						2					1	3			
003	3	1	2												1		2		2	
004			1	1	1				1		4	1					3			
005	1			1	1	1	1		1		1								2	
006						4	1		1				1						1	
007						2	5									1	1	2	1	
008		1	1		1			4			1						1			
009			1				2		2	1	3				1		2			
010							1		1	1	1	4			2	1		1		
011							1		1		5	1		1		1	1		1	
012		1										5	1		1		2		1	
013							1				1		6	1	1				1	1
014							1		3		1	2			2		1	1	1	
015					1	2	3						1		2	1	1			
016	1	1					1		2				1		1		2		1	1
017	1	1					1				3				3		2			
018							2				1	1		1	2		1	1		
019	1								1		3	1		2	1		1		1	1
020			1	1		1	1		1	1			1		2				1	2