```
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'};
```

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
```

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;
```

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

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

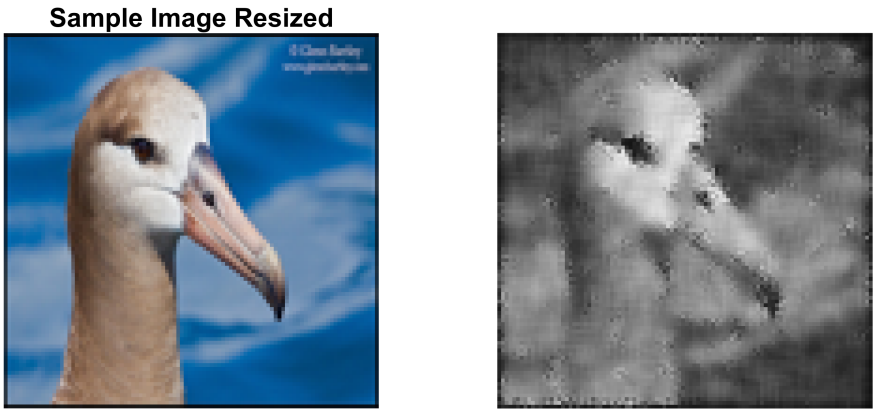
```
targetSize = [100, 100];
trainingImageDS_Resized = transform(trainingImageDS, @(x) imresize(x,targetSize));
validationImageDS_Resized = transform(validationImageDS, @(x) imresize(x,targetSize));
testImageDS_Resized = transform(testImageDS, @(x) imresize(x,targetSize));

% 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
```

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];
```

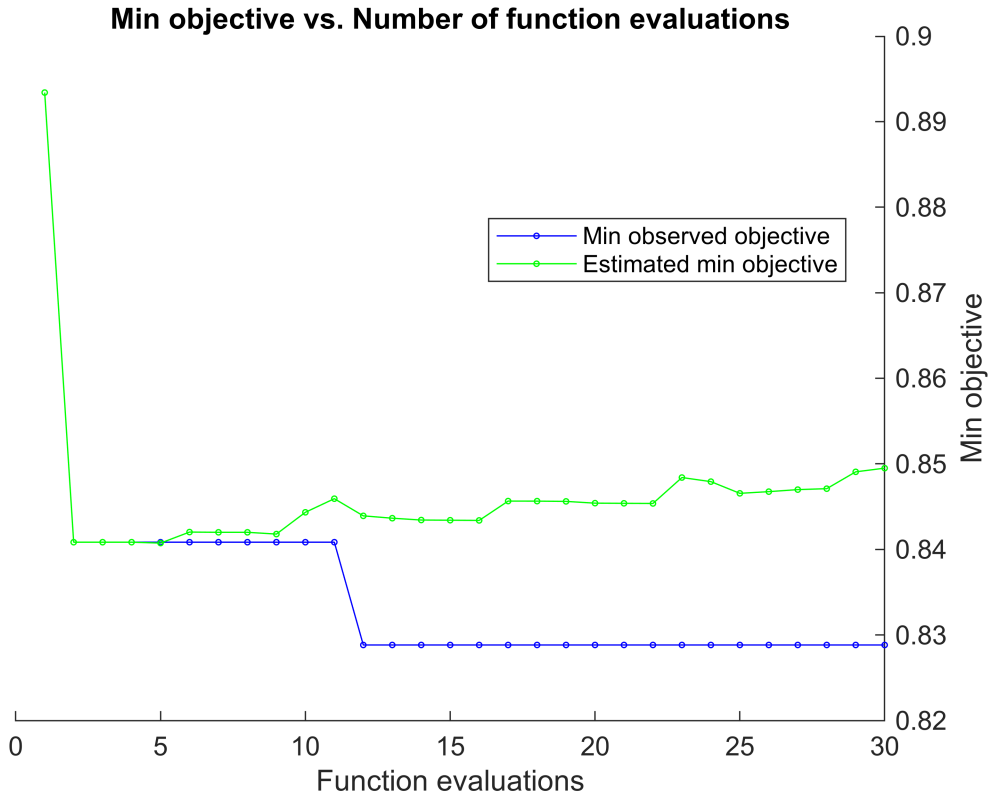
Train a multi-class SVM

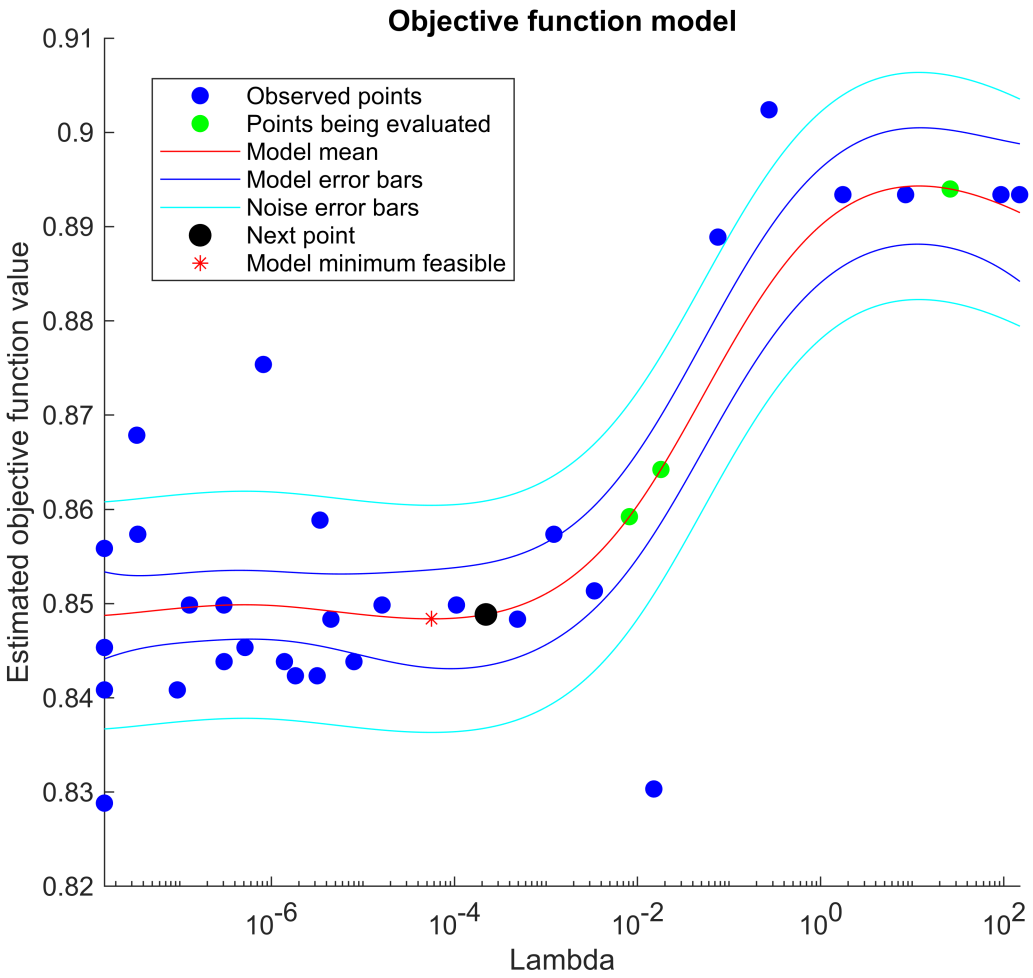
```
t = templateLinear('Solver', 'dual');
options = struct('UseParallel', true);

% 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);
```

```
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   |     0.89339 |      11.188 |     0.89339 |     0.89339 |      93.217 |
|    2 |       2 | Best   |     0.84084 |       11.41 |     0.84084 |     0.84084 |    9.398e-08 |
|    3 |       2 | Accept |     0.88889 |      11.635 |     0.84084 |     0.84084 |     0.075455 |
|    4 |       2 | Accept |     0.84384 |      11.548 |     0.84084 |     0.84084 |   7.9845e-06 |
|    5 |       4 | Accept |     0.84985 |      11.455 |     0.84084 |     0.84074 |   1.6191e-05 |
|    6 |       4 | Accept |     0.84835 |      11.814 |     0.84084 |     0.84203 |   0.00048873 |
|    7 |       3 | Accept |     0.84535 |      11.885 |     0.84084 |       0.842 |   5.1564e-07 |
|    8 |       3 | Accept |      0.9024 |      12.217 |     0.84084 |       0.842 |      0.27303 |
|    9 |       4 | Accept |     0.84084 |      12.042 |     0.84084 |     0.84179 |   1.5041e-08 |
|   10 |       4 | Accept |     0.84985 |      12.158 |     0.84084 |     0.84434 |   1.2761e-07 |
|   11 |       4 | Accept |     0.85586 |      11.668 |     0.84084 |     0.84593 |   1.5016e-08 |
|   12 |       4 | Best   |     0.82883 |      11.956 |     0.82883 |     0.84392 |   1.5023e-08 |
|   13 |       4 | Accept |     0.84535 |       13.33 |     0.82883 |     0.84365 |   1.5045e-08 |
|   14 |       4 | Accept |     0.85135 |      12.986 |     0.82883 |     0.84343 |    0.0033783 |
|   15 |       4 | Accept |     0.84384 |      13.096 |     0.82883 |     0.84341 |   1.3896e-06 |
|   16 |       4 | Accept |     0.84985 |      13.494 |     0.82883 |     0.84339 |   0.00010517 |
|   17 |       4 | Accept |     0.85736 |      11.644 |     0.82883 |     0.84565 |   3.4766e-08 |
|   18 |       4 | Accept |     0.89339 |      11.689 |     0.82883 |     0.84564 |       8.4752 |
```

19	4	Accept	0.84234	11.463	0.82883	0.84561	3.1662e-06
20	4	Accept	0.86787	11.292	0.82883	0.8454	3.3915e-08

Iter	Active workers	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	Lambda
21	4	Accept	0.85736	11.117	0.82883	0.84539	0.0012199
22	4	Accept	0.89339	11.045	0.82883	0.84537	1.7487
23	4	Accept	0.85886	11.32	0.82883	0.8484	3.3985e-06
24	4	Accept	0.84384	11.514	0.82883	0.84792	3.0475e-07
25	4	Accept	0.84234	11.857	0.82883	0.84655	1.8324e-06
26	4	Accept	0.84835	11.755	0.82883	0.84675	4.472e-06
27	4	Accept	0.83033	11.713	0.82883	0.84699	0.015063
28	4	Accept	0.84985	11.833	0.82883	0.8471	3.0316e-07
29	4	Accept	0.87538	11.833	0.82883	0.84907	8.183e-07
30	4	Accept	0.89339	11.436	0.82883	0.84949	149.12






```
Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 104.0134 seconds
Total objective function evaluation time: 355.393

Best observed feasible point:
      Lambda
    __________

    1.5023e-08

Observed objective function value = 0.82883
Estimated objective function value = 0.84875
Function evaluation time = 11.9557

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

    9.398e-08

Estimated objective function value = 0.84949
Estimated function evaluation time = 11.8312
```

```
% 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 = 0.1667
```

```

% 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)) +"%");
```