```
% Load vehicle data set
data = load('fasterRCNNVehicleTrainingData.mat');
vehicleDataset = data.vehicleTrainingData;

% Display first few rows of the data set.
vehicleDataset(1:4,:)

% Add fullpath to the local vehicle data folder.
dataDir = fullfile(toolboxdir('vision'),'visiondata');
vehicleDataset.imageFilename = fullfile(dataDir,
vehicleDataset.imageFilename);

% Read one of the images.
I = imread(vehicleDataset.imageFilename{10});

% Insert the ROI labels.
I = insertShape(I, 'Rectangle', vehicleDataset.vehicle{10});

% Resize and display image.
I = imresize(I,3);
figure
imshow(I)

% Split data into a training and test set.
idx = floor(0.6 * height(vehicleDataset));
trainingData = vehicleDataset(1:idx,:);
testData = vehicleDataset(idx:end,:);

% Create image input layer.
inputLayer = imageInputLayer([32 32 3]);

 % Define the convolutional layer parameters.
filterSize = [3 3];
numFilters = 32;

% Create the middle layers.
middleLayers = [

    convolution2dLayer(filterSize, numFilters, 'Padding', 1)
    reluLayer()
    convolution2dLayer(filterSize, numFilters, 'Padding', 1)
    reluLayer()
    maxPooling2dLayer(3, 'Stride',2)

    ];

finalLayers = [

    % Add a fully connected layer with 64 output neurons. The output size
    % of this layer will be an array with a length of 64.
    fullyConnectedLayer(64)

    % Add a ReLU non-linearity.
    reluLayer()
```

```
    % Add the last fully connected layer. At this point, the network must
    % produce outputs that can be used to measure whether the input image
    % belongs to one of the object classes or background. This
measurement
    % is made using the subsequent loss layers.
    fullyConnectedLayer(width(vehicleDataset))

    % Add the softmax loss layer and classification layer.
    softmaxLayer()
    classificationLayer()
];
layers = [
    inputLayer
    middleLayers
    finalLayers
    ]


% Options for step 1.
optionsStage1 = trainingOptions('sgdm', ...
    'MaxEpochs', 10, ...
    'MiniBatchSize', 256, ...
    'InitialLearnRate', 1e-3, ...
    'CheckpointPath', tempdir);

% Options for step 2.
optionsStage2 = trainingOptions('sgdm', ...
    'MaxEpochs', 10, ...
    'MiniBatchSize', 128, ...
    'InitialLearnRate', 1e-3, ...
    'CheckpointPath', tempdir);

% Options for step 3.
optionsStage3 = trainingOptions('sgdm', ...
    'MaxEpochs', 10, ...
    'MiniBatchSize', 256, ...
    'InitialLearnRate', 1e-3, ...
    'CheckpointPath', tempdir);

% Options for step 4.
optionsStage4 = trainingOptions('sgdm', ...
    'MaxEpochs', 10, ...
    'MiniBatchSize', 128, ...
    'InitialLearnRate', 1e-3, ...
    'CheckpointPath', tempdir);

options = [
    optionsStage1
    optionsStage2
    optionsStage3
    optionsStage4
    ];
```

```
% A trained network is loaded from disk to save time when running the
% example. Set this flag to true to train the network.
doTrainingAndEval = false;

if doTrainingAndEval
    % Set random seed to ensure example training reproducibility.
    rng(0);

    % Train Faster R-CNN detector. Select a BoxPyramidScale of 1.2 to
allow
    % for finer resolution for multiscale object detection.
    detector = trainFasterRCNNObjectDetector(trainingData, layers,
options, ...
        'NegativeOverlapRange', [0 0.3], ...
        'PositiveOverlapRange', [0.6 1], ...
        'BoxPyramidScale', 1.2);
else
    % Load pretrained detector for the example.
    detector = data.detector;
end


 % Read a test image.
I = imread(testData.imageFilename{1});

% Run the detector.
[bboxes,scores] = detect(detector,I);

% Annotate detections in the image.
I = insertObjectAnnotation(I,'rectangle',bboxes,scores);
figure
imshow(I)

if doTrainingAndEval
    % Run detector on each image in the test set and collect results.
    resultsStruct = struct([]);
    for i = 1:height(testData)

        % Read the image.
        I = imread(testData.imageFilename{i});

        % Run the detector.
        [bboxes, scores, labels] = detect(detector, I);

        % Collect the results.
        resultsStruct(i).Boxes = bboxes;
        resultsStruct(i).Scores = scores;
        resultsStruct(i).Labels = labels;
    end

    % Convert the results into a table.
    results = struct2table(resultsStruct);
```

```
else
    % Load results from disk.
    results = data.results;
end

% Extract expected bounding box locations from test data.
expectedResults = testData(:, 2:end);

% Evaluate the object detector using Average Precision metric.
[ap, recall, precision] = evaluateDetectionPrecision(results,
expectedResults);

% Plot precision/recall curve
figure
plot(recall,precision)
xlabel('Recall')
ylabel('Precision')
grid on
title(sprintf('Average Precision = %.2f', ap))
```