Image and Vision Computing: Lab 4

In this lab we will explore sliding window-based object detection. Specifically we will re-implement a simplified version of the classic CVPR 2005 pedestrian detector of Dalal & Triggs, one of the most highly cited papers in computer vision, and computer science more generally. We will work with 300x100 as the default size of pedestrian boxes.

Part 1: Training a SVM

Browse the training images (provided in folder `Trainsvm)`, and the testing images (folder `Test`). Since we are doing detection, the train images are person and non-person crops, and the testing images are whole scenes in which we should detect the people. The file names are provided in `matfile/pos_train_names.mat` and `matfile/neg_train_1.mat`. Open the lab script `LAB4_main.m`. In the first cell you see that `pos_data` and `neg_data` are setup to load the images for you. If you run the cell you will see that they show some examples of the extracted training boxes. You will see you can train the SVM like this:

```
svm_model1 = fitcsvm(train_data,train_label,'KernelFunction','gaussian');
```

SVM can be trained with linear and Gaussian among other kernels. Given a trained SVM model, you can apply it to find out if a given 300x100 bounding box contains a pedestrian or not like this:

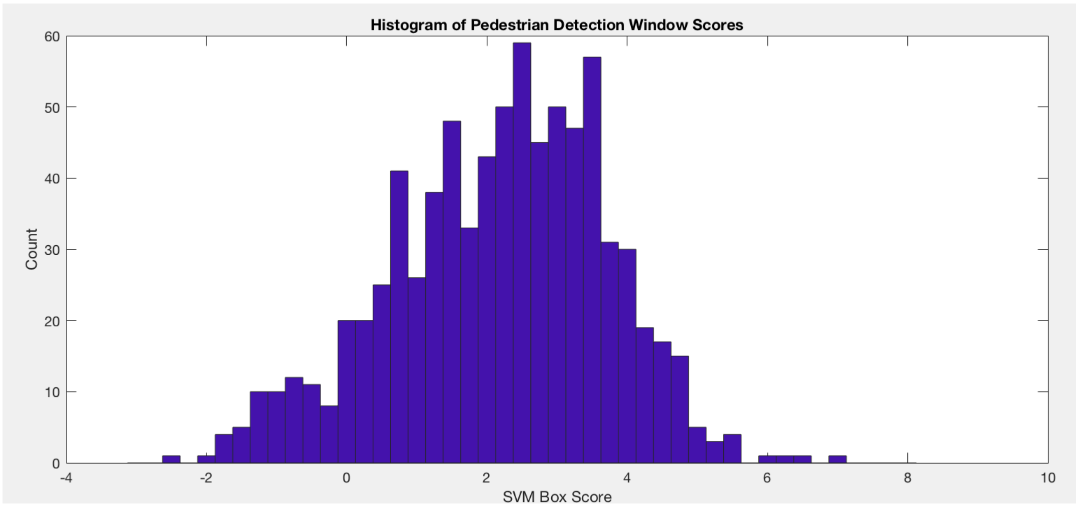
```
 [predict_label,predict_score] = predict(svm_model,test_data);
```

Part 2: Detecting Pedestrians with SVM

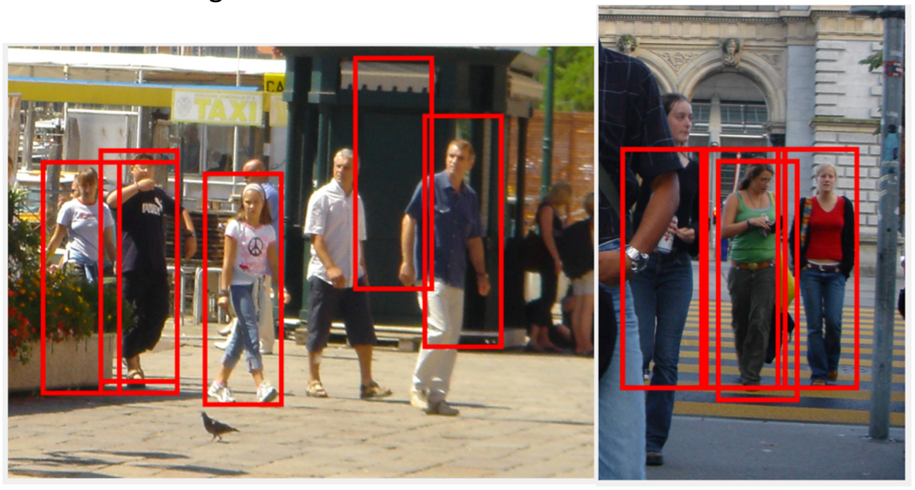
The next cell applies the learned detector to find pedestrians in some test images. Your job is to write the missing function `locations=findWindows`. In this function you should implement a sliding window object detector:

- Make a strided (try increment 16 to start) loop over x and y pixels.
- At each iteration of the loop, extract a pedestrian sized bounding box whose top left corner is the current loop coordinate. Extract HOG features from this bounding box (see `pos_data` and `neg_data`) for examples.
- Send the extracted windows to `[label,score]=predict(svm_model,testHoG)` which will apply the detector to the specified window.

- In a typical image you will extract ~1000 bounding boxes, and these will be discovered with a wide distribution of scores like this:



- Boxes with increasingly negative scores are unlikely to be pedestrians, boxes with increasingly positive scores are likely to be pedestrians.
- `findWindows()` should return a Nx2 sized matrix containing locations of found windows (i,j locations. One per row).
- Make your `findWindows(..)` routine only return bounding boxes with `score>0.1`.
- Finally the routine `merge()` will do some non-maximum suppression and draw the discovered bounding boxes to the screen. It should look like this:



Bonus:

1. Extract the linear SVM weights and implement a fast detector by convolving the image with the weights rather than explicitly calling predict on all bounding boxes.
2. Tune the SVM so it works with high accuracy: Explore the impact of different kernels, different regularization strength, and different HoG extraction parameters. Download extra images from the internet to get more data for training a stronger model.
3. Rather than fixing the detection threshold as in the above, evaluate the detector by implementing a ROC or Precision Recall curve to find out how the hit and miss rate co-vary with a given detection threshold.