Project 5: Vehicle Detection

The code for the vehicle detection project is found on the ipython notebook named test_detection.ipynb found in the submission folder.

Feature extraction is the most important aspect for recognition using traditional machine learning algorithms. They have to be custom designed for particular used cases based on the type of objects we are detecting. We use three types of features: HOG (Histogram of Oriented Gradients) (shape features), binned color (color and shape features) and color histogram features (color only features). We experimented with just using single HOG but the features did not encode enough information and hence the current combination of features can provide enough information for image classification. The parameters used for feature extraction are:

```
# Define parameters for feature extraction
color_space = 'LUV' # Can be RGB, HSV, LUV, HLS, YUV, YCrCb
orient = 8  # HOG orientations
pix_per_cell = 8 # HOG pixels per cell
cell_per_block = 2 # HOG cells per block
hog_channel = 0 # Can be 0, 1, 2, or "ALL"
spatial_size = (16, 16) # Spatial binning dimensions
hist_bins = 32    # Number of histogram bins
spatial_feat = True # Spatial features on or off
hist_feat = True # Histogram features on or off
hog_feat = True # HOG features on or off
```

These parameters were chosen based on a hit and trial scheme after visualization of the HOG features and the training accuracy. The visualized HOG features for the above parameters are

I did not find a big difference in performance between the LUV and YUV image usage for the other features on the feature list. I settled on these parameters after I obtained a training accuracy of close to 99%. The visualization of the HOG features also helped me see the features for cars and not cars using these feature parameters, and convinced me that my selection of the parameters are valid. We can see that car features are structured similarly in orientation and shape compared to not cars examples. We can see the unique orientations around the rear window and the tail lights, which are the most characteristic features of the car.

I increased the number of features by flipping the testing and training images thereby doubling the number of samples. Then I normalize the features. Normalizing ensures that a classifier's

behavior isn't dominated by just a subset of the features, and that the training process is as efficient as possible. That is why, feature list was normalized by the `StandardScaler()` method from `sklearn`. The data is split into training and testing subsets (80% and 20%). The classifier is a linear SVM. It was found that it performs well enough and quite fast for the task. Block 5 in the ipynb notebook defines the process of loading data, extracting features and training the classifier using linear svm. The training accuracy was 98.9%

Localizing the objects to be detected is done using sliding window techniques. I iterate over image area that could contain cars with approximately car sized box and try to classify whether box contain car or not. The distance of the car from the host vehicle determines the scale of the sliding window to be used. I used 4 sliding window sizes of 128, 112, 96 and 80 pixels side size. The overlap criteria used was 75% in horizontal and vertical directions. Examples of sliding windows lattice are shown in the ipynb notebook in block 14. A sample sliding window is drawn in blue. Only areas where cars are expected are covered in the lattice. For ex, we don't consider the area above the origin as cars would not be expected there. Once the pipeline is run on the image at the various scales, there are multiple detections on a single object as shown in figure in block 10 of the ipynb notebook. We need to perform non-maximal suppression to combine the boxes and provide one bounding box.

Non-maximal suppression is done based on obtaining hot boxes as discussed in the course tutorial. After sliding window application we have hot boxes which is acts like a weighing process where classified sliding boxes are used to weigh a matrix to obtain a heat map. The hot boxes have maximum weight near the true car detections along with lesser concentration hot boxes near in false positive detections. We combine the peaks of the detections taking the first box (called average box) from the detected boxes and combined with required area of overlap. After joining two boxes, it is updated as the new average box. The conversion of detections to heat map to average box is shown in block 16 of the ipynb notebook.

The detection and averaging of the boxes perform well enough for me to run frame by frame without having to have a very big false positive filtering scheme. To run the pipeline on video, I keep a history of the hot boxes from the previous few frames. These previous hot boxes are then combined and the in the new frame, target localization is performed and validated with the hot box history. This reduced jitter in the detection of the objects and the bounding boxes were stable. There are false positives but they are few and far and the detections are robust with very few misses. The output video named final_video_resubmit2.mp4 is included in the submission folder.

The pipeline can be improved by detecting around just previously detected objects in subsequent frames. It can also be improved by adding kalman filtering techniques or something similar. The detection of occluded objects is also not very robust with the current pipeline which could be improved by using techniques such as discrete part models etc. SVMs are inherently slow and compute heavy for real time applications. We could have used a boosting classifier (such as adaboost) for the classification and increased speed of execution. I had some trouble trying to tune the parameters for the feature extractor and obtaining the right ROI for the sliding windows.