Vehicle Detection Project

The goals / steps of this project are the following:

- Perform a Histogram of Oriented Gradients (HOG) feature extraction on a labeled training set of images and train a classifier Linear SVM classifier
- Optionally, you can also apply a color transform and append binned color features, as well as histograms of color, to your HOG feature vector.
- Note: for those first two steps don't forget to normalize your features and randomize a selection for training and testing.
- Implement a sliding-window technique and use your trained classifier to search for vehicles in images.
- Run your pipeline on a video stream (start with the test_video.mp4 and later implement on full project_video.mp4) and create a heat map of recurring detections frame by frame to reject outliers and follow detected vehicles.
- Estimate a bounding box for vehicles detected.

Histogram of Oriented Gradients (HOG)

1. Explain how (and identify where in your code) you extracted HOG features from the training images.

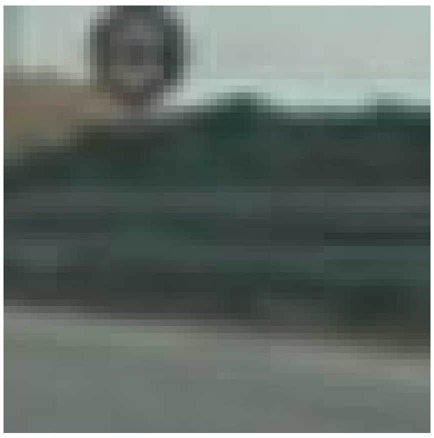
The code for all the svm training is contained in TrainClassifier.py.

HOG features are extracted in the function “get_hog_features” located at line 15. This has been taken from the udacity lecture material.

I started by reading in all the `vehicle` and `non-vehicle` images (line 83,84 in TrainClassifier.py). Here is an example of one of each of the `vehicle` and `non-vehicle` classes:

Vehicle

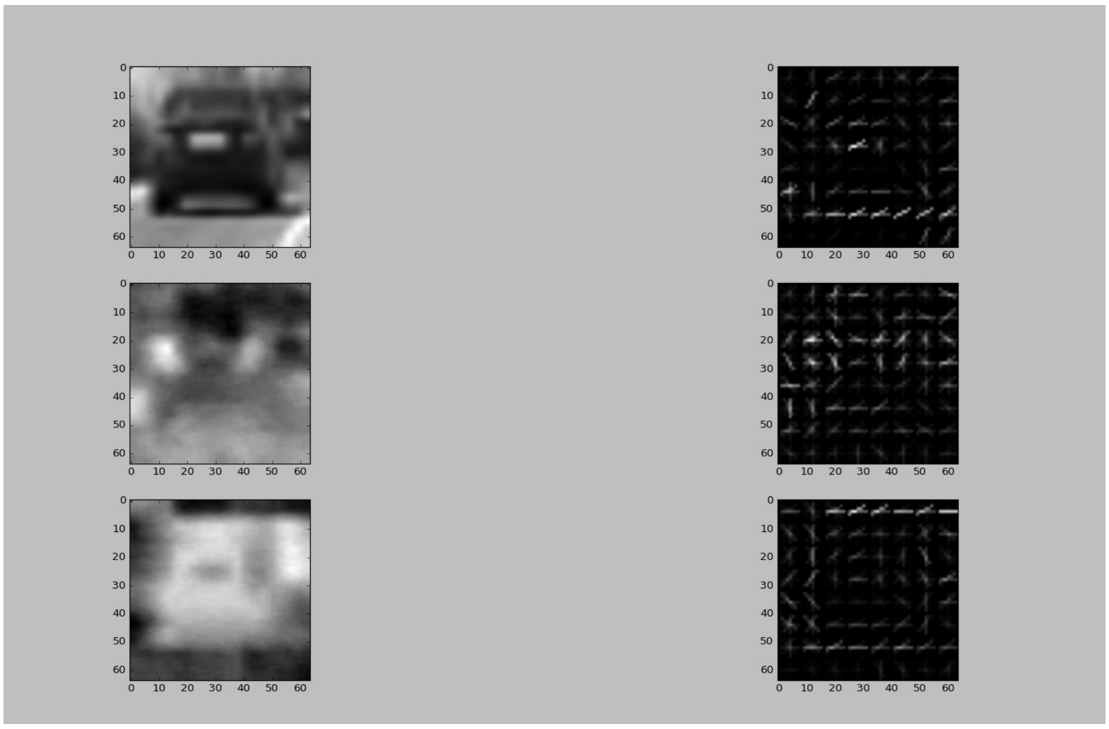
Non Vehicle



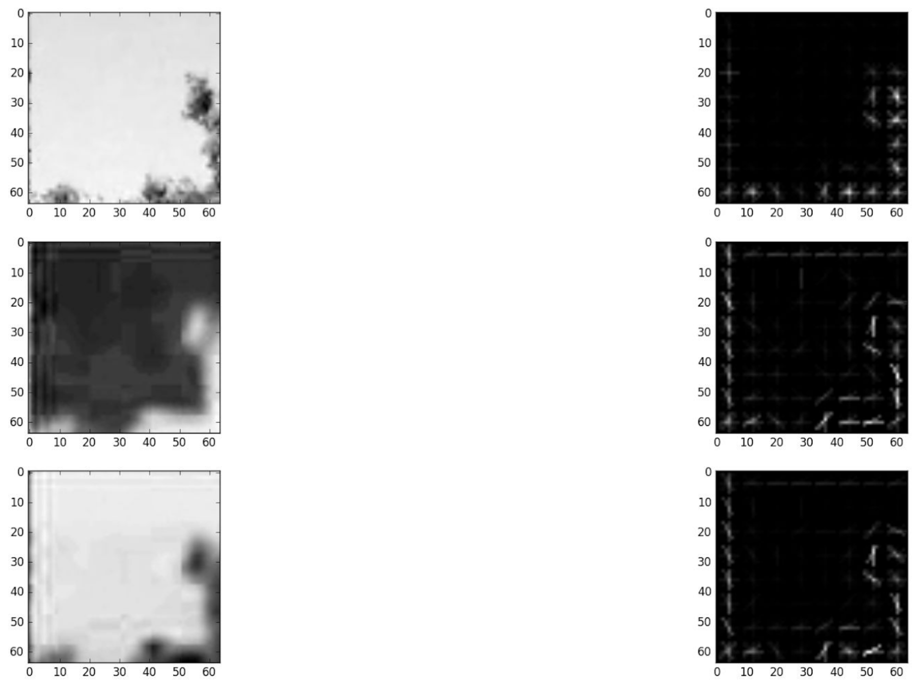
I then explored different color spaces like RGB,HSV,YCrCb and different `skimage.hog()` parameters (`orientations`, `pixels_per_cell`, and `cells_per_block`). I grabbed random images from each of the two classes and displayed them to get a feel for what the `skimage.hog()` output looks like. My final parameters chosen are in lines 88-90 in TrainClassifer.py

Here is an example showing the `YCrCb` color space individual channels along with HOG images with HOG parameters of `orientations=8`, `pixels_per_cell=(8, 8)` and `cells_per_block=(1, 1)`:

Individual YCrCb color channels for car and their corresponding HOG images



Individual YCrCb color channels for non car and their corresponding HOG images



2. Explain how you settled on your final choice of HOG parameters.

Taking the initial code given in the lesson I experimented with different cells per block and pixel per cell. I finally settled with the parameters of `orientations=8`, `pixels_per_cell=(8, 8)` and `cells_per_block=(1, 1)` as this yielded the best validation accuracy for my SVM model. My final parameters chosen are in lines 88-90 in TrainClassifer.py

3. Describe how (and identify where in your code) you trained a classifier using your selected HOG features (and color features if you used them).

For the features, I used the same features used in udacity lessons where I create a single feature vector concatenating binned color features, color histogram features and HOG features from each training image. Hog feature extraction is in lines 15-27 , binned color feature extraction is in lines 63-69, color histogram feature extraction is in lines 72-78 of TrainClassifier.py. The complete feature extraction is in the function “extract_features” located at line 32 of TrainClassifier.py. The features were scaled using `sklearn.preprocessing.StandardScaler().(Lines 104-106 in TrainClassifier.py)`

I trained a linear support vector machine and its accuracy was tested on the test dataset. The classifier achieved an accuracy of 98.975%. (Lines 120-132 in TrainClassifer.py)
The trained svm classifier and scaler are stored in a pickle file later to be used for detection of cars in the video. (Lines 137 & 138 in TrainClassifer.py)

I initially used just HOG features and compared accuracy with HOG + Color BInning + Color Histogram features. The validation accuracy was 96.7% for just HOG and about 98-99% for the other. Hence I went ahead with the other combination.

4. Describe how (and identify where in your code) you implemented a sliding window search. How did you decide what scales to search and how much to overlap windows?

The code for all the discussions below to detect cars is in **detectCars.py**.

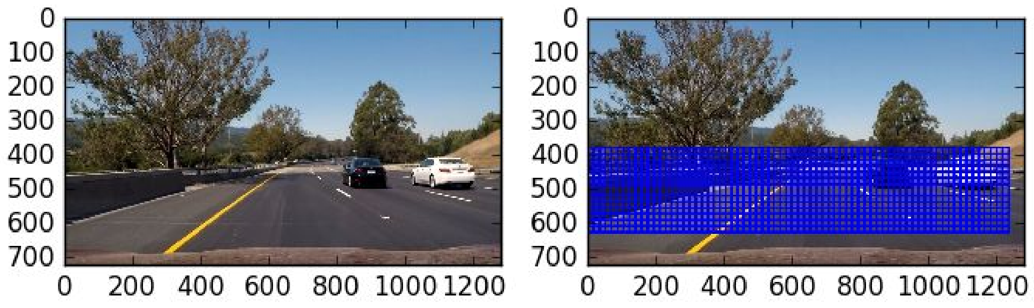
The sliding window code is in “find_cars” function (lines 105- 175) and “find_cars_multiscale” function (lines 178 -189). The sliding window code was taken from udacity lectures and adapted for multiple scales.

The find_cars function takes in required parameters and a fixed scale to search. It then returns all the bounding boxes with positive detections at that fixed scale.

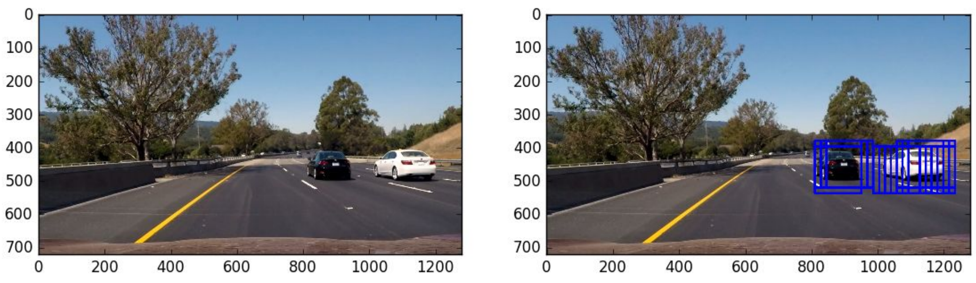
The overlap or the cells to step is chosen as one. (Line 130)

Find_cars_multiscale function calls (line 185) “find_cars” function with different scaling parameters (line 180). It then accumulates all the bounding boxes found at several scales. (line 187)

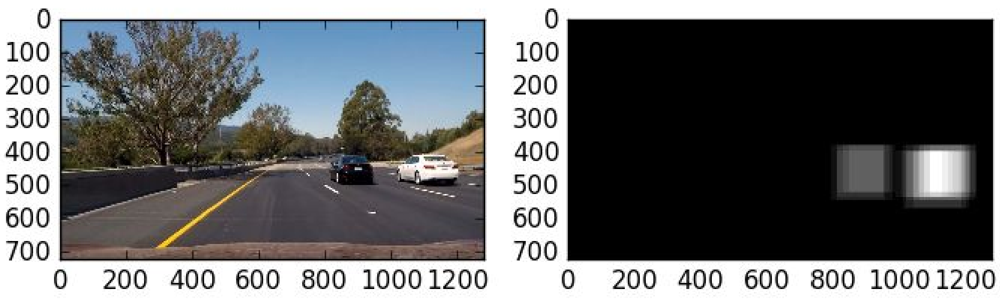
Shown below is an original image and sliding window search area for a given scale (2.2)



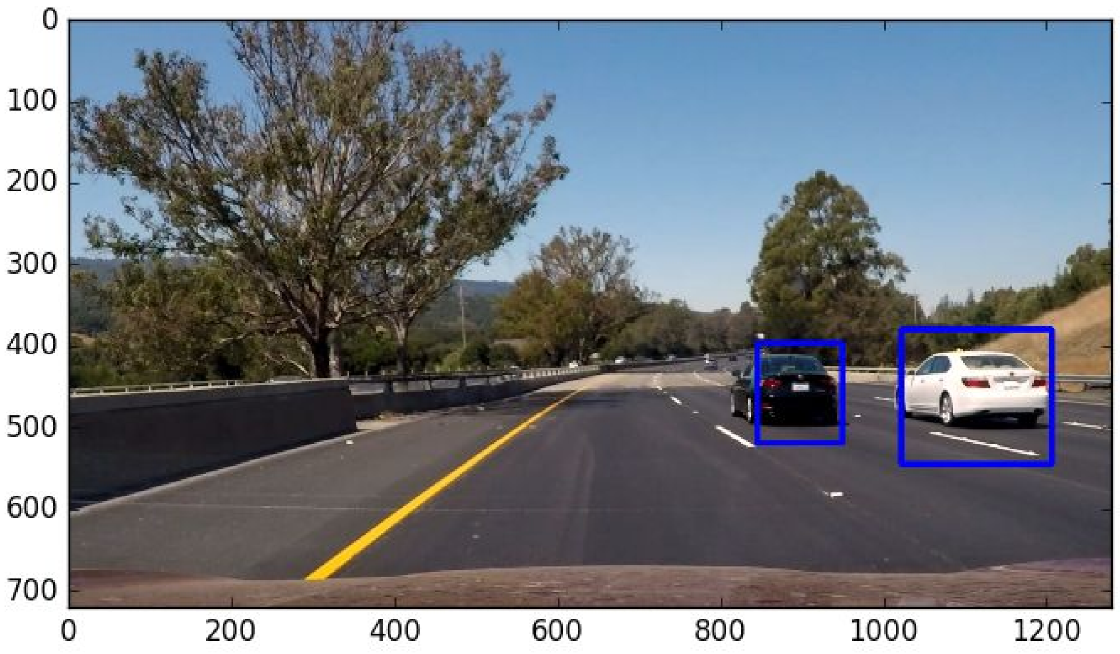
For the same image above at the given scales (2.2) the positive detection with SVM classifier are shown :



For the same image above after positive detections a heatmap and a threshold is applied. (Lines 226 and 229).



For the same image above after the heatmap is obtained a label function is applied (`from sklearn.preprocessing import StandardScaler`) and multiple detections are unified into single detections as shown below. (Lines 244 and 246)



Video Implementation

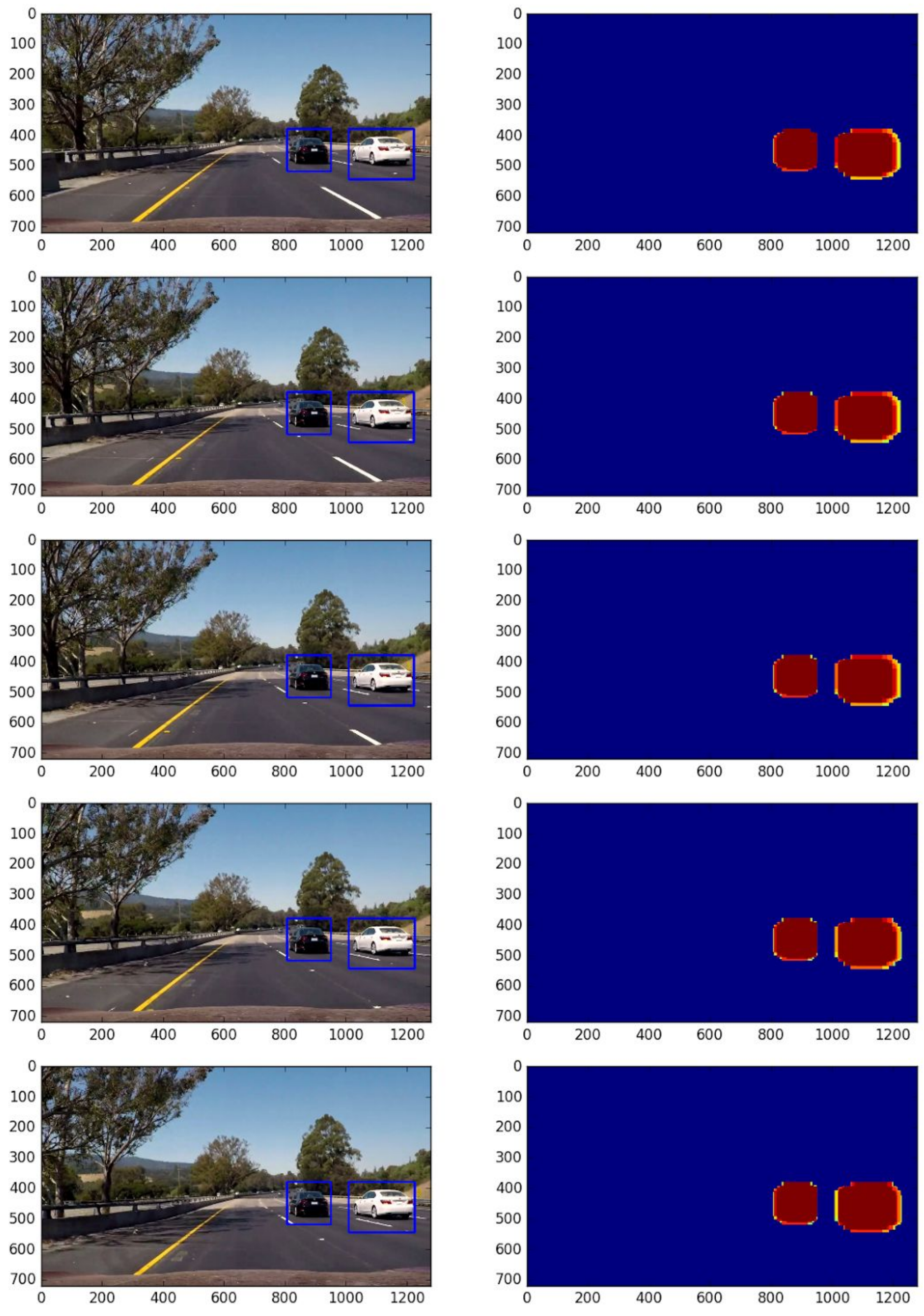
The video with car detections is provided as “output.mp4” with this submission. Also a youtube link for the same is provided here: https://www.youtube.com/watch?v=cp9tqwlm0fs

False positive Reduction and combining overlapping bounding boxes

There were a few false positives that I saw when the video was processed frame by frame without any temporal smoothing. What I realized is that the false positives are in just few frames randomly. I implemented a queue of length 15 frames (Line 208) to store all the previous heat maps. Each heatmap for individual image is made into a binary image and then appended to the queue. (Line 232 - 235)

When a new frame comes in I sum all the 15 heatmaps (Line 237-238) from the previous frames along with the new frame heat map and use a threshold to make a final heatmap on which label and bounding boxes are applied. This reduces any false positives from appearing.

Also once bounding boxes are detected I discard any bounding boxes that are very small in area (Line 46) as they could be false segments because of temporal averaging of heatmaps.
Six consecutive images from video with their detections and average heat maps is shown below:



Discussion:

The current pipeline detects and tracks cars well enough through the given video without any false positives. The thresholding and moving average heat map helps quite a lot in eliminating false positives and the spatial binning of color features along with HOG takes care of terrain changes.

The issue with the current pipeline is that it takes about 2-3 seconds to process a single frame. The computation is because of extraction of three different types features along many sliding windows at several scales.

To make the pipeline fast it could be written to run on GPU as the detections across different sliding windows are completely parallelizable.

Deep learning approaches that only make a single pass through the image like YOLO(You only look once) could also be exploited to increase the speed quite a bit.

Additional improvements could be to have a tighter bounding box around cars such that if two cars are closeby they are individually detected than a big bounding box around both as seen in some of the frames. So multiple cars side by side very close would cause this pipeline to fail so we need tighter bounding boxes.