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.

Rubric Points

1. Histogram of Oriented Gradients (HOG)

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

The code for this step is contained in the second code cell of the IPython notebook. A function called extract_features() is defined to extract the HOG features as well as the color features. The parameters I used for HOG feature extraction are

```
color_space = 'YCrCb' # Can be RGB, HSV, LUV, HLS, YUV, YCrCb
orient = 9  # HOG orientations
pix_per_cell = 8 # HOG pixels per cell
cell_per_block = 2 # HOG cells per block
```

I extract the HOG features from all the three color channels. I tweak among different parameters. Basically, I don't want the feature dimension to be too large or too small such that the model is not over-fitting or under-fitting.

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

I use the SVM (LinearSVC function) to train the model, and the HOG and color features for the collected data are preprocessed to a zero mean and unit variance before training. The dataset is also randomly split into 20% testing data and 80% training data. The implementation is the fourth cell of the IPython notebook.

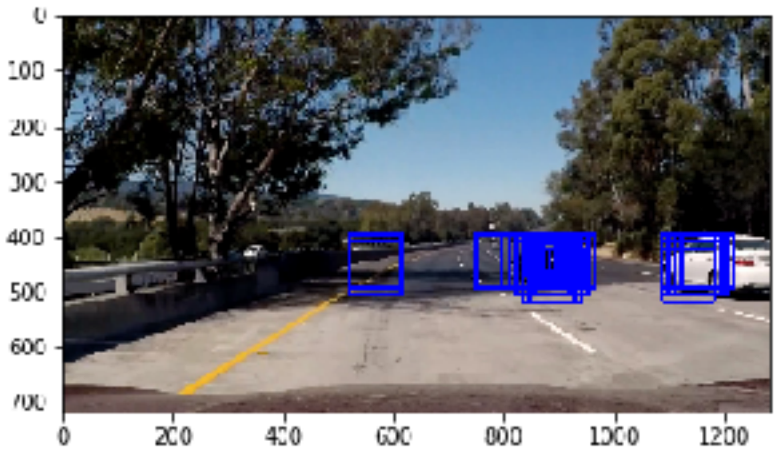
2. Sliding Window Search
<u>Describe how (and identify where in your code) you implemented a sliding window search. How did you decide what scales to search and how much to overlap windows?</u>

The sliding window search is implemented in function find_cars() defined in the fifth cell of the IPython notebook. In this function, I set the parameter cells_per_step = 1 which defines the the number of cells the window search will step each time. The reason why I set cells_per_step = 1 is that I want to collect more windows for vehicle detection.

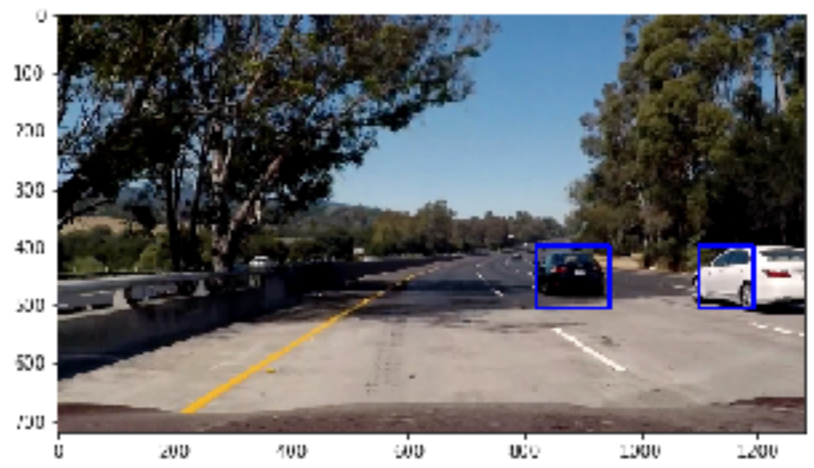
<u>Show some examples of test images to demonstrate how your pipeline is working. How did you optimize the performance of your classifier?</u>

I searched on two scales (scales =[1,1.5]) using YCrCb 3-channel HOG features plus spatially binned color and histograms of color in the feature vector. The image I choose to test is

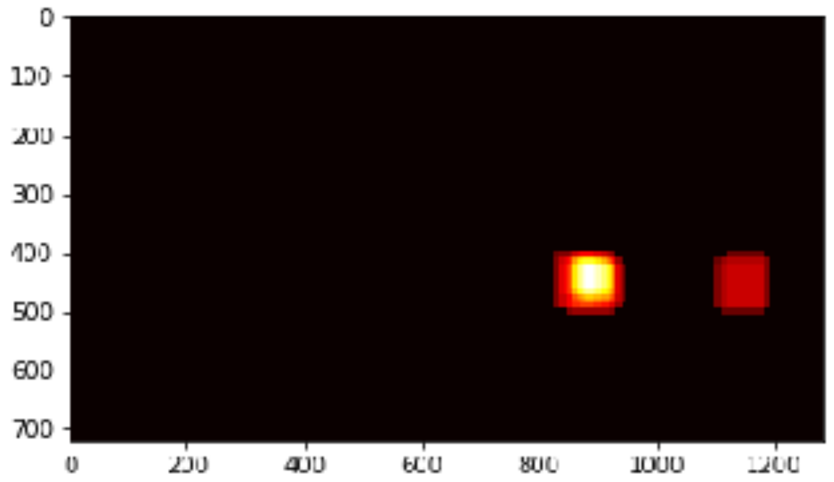
The boxes I obtained through sliding windows search are shown below



After applying a threshold to remove some false positives, I obtain



The heat map for above figure is



To optimize the performance of classifier, first I try different scales of search window for a better detection result. Then I add a threshold filter to filter out the false positives of car detection. I also define a specific interested search region, which can effectively increase the running speed and detection accuracy.

3. Video Implementation

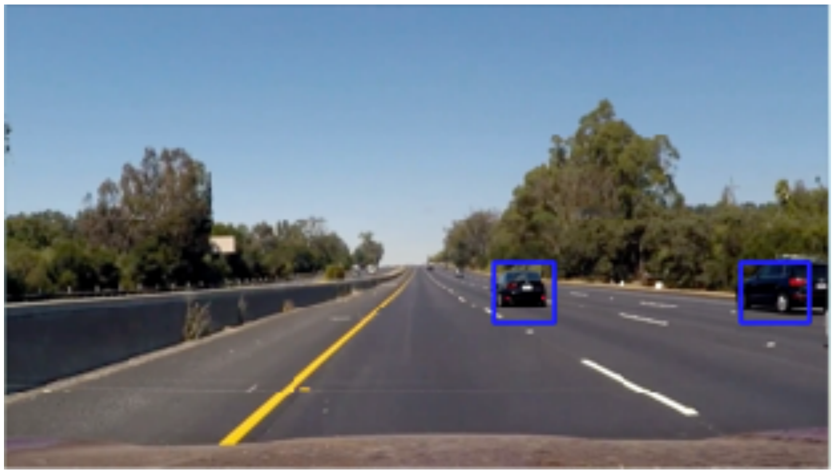
Provide a link to your final video output. Your pipeline should perform reasonably well on the entire project video (somewhat wobbly or unstable bounding boxes are ok as long as you are identifying the vehicles most of the time with minimal false positives.)

The output video is included in the project submission. The pipeline works reasonably well on entire project video.

Describe how (and identify where in your code) you implemented some kind of filter for false positives and some method for combining overlapping bounding boxes.

To further reject the false positives, I define a deque type data called history to store the recent five detections of windows, then I obtain the heat map for all the windows in history data. By doing so, I can strength the heat intensity for detected cars. And I apply a threshold to filter out the possible false positives. This implementation is in the eighth cell of the IPython notebook.

The following is the bounding boxes drawn on the last frame of output video:



4. Discussion

Briefly discuss any problems / issues you faced in your implementation of this project. Where will your pipeline likely fail? What could you do to make it more robust?

In this project, I detect two type of false positives. One is the incoming cars on the other side of road. To filter out this kind of false positives, in this case, I restrict my starting position for window search in x axis from x=400. For a more robust implementation, we should properly define the search region based on the position of the car on the road.

Another false positive is the environment, to filter out this, I actually increase the threshold, but the problem is if you set the threshold too high, it will even comprise the car detection performance. Thus, to make it more robust, we should improve the learning model to reduce the false positive rate on the non-car environment.