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

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

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 this step is contained in the first code cell of the IPython notebook in a method named **Get_Hog_Features**. It is in second code cell.

This method takes image as argument along with orientation, pix per cell, cell per block, features vector and visualise. On the basis of visualise argument it returns features only or both features and hog_image.

Other features gathering methods used:

Bin_spatial is used in code cell 3 and it returns binary faetures of image with given size.

Color_hist is used in code cell 4 to provide histogram features of image.

Finally a master method **Extract_features** is used to call above methods.

Here is an example of one of each of the `vehicle` and `non-vehicle` classes:

Vehicle

Non vehicle

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

I tried various combinations of parameters and finally I got an accuracy of 99.5% with feature vector length of 2052. I used 11 orientation ,16 pix per cell,2 cells per block and 32 histogram bins.

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).

In code cell 7, I trained a non linear SVM using kernel='rbf' for better accuracy. I used udacity training data for vehicles and non vehicles and accordingly labels(1 for vehicle and 0 for non vehicle) were generated. 20 % split of total data was kept for testing the classifier.

I first separated the vehicles and non vehicles and then extracted their features by calling extract features method. Then I created an array stack of features and label vector.

Then I shuffled and split the data using train_test_split method.

After Normalization I trained it.

Sliding Window Search

1. 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?

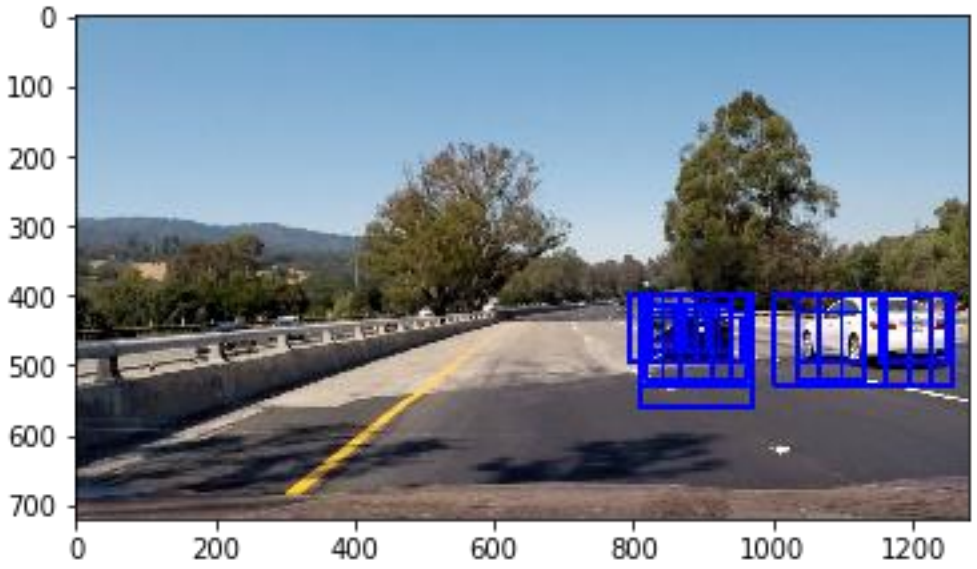
In code cell 5, I decided to search windows with the method **slide_window** which takes X start and stop, Y start and stop, overlap, window size as arguement and it simply plots all the windows with above specification on the image.

Scales are decided on the basis of size of vehicle with respect to distance that is a closed vehicle can be a 160x160 and same vehicle can be a 64x64 at a distance.

Overlap of 0.75 is used to closely capture the boundaries of vehicles.

Here I have used relevant part of image for vehicle detection. So mostly road is covered.

Once this is done then I predicted the region where car is detected using the classifier we trained above. Only the windows where we have a detection that is drawn. Example:



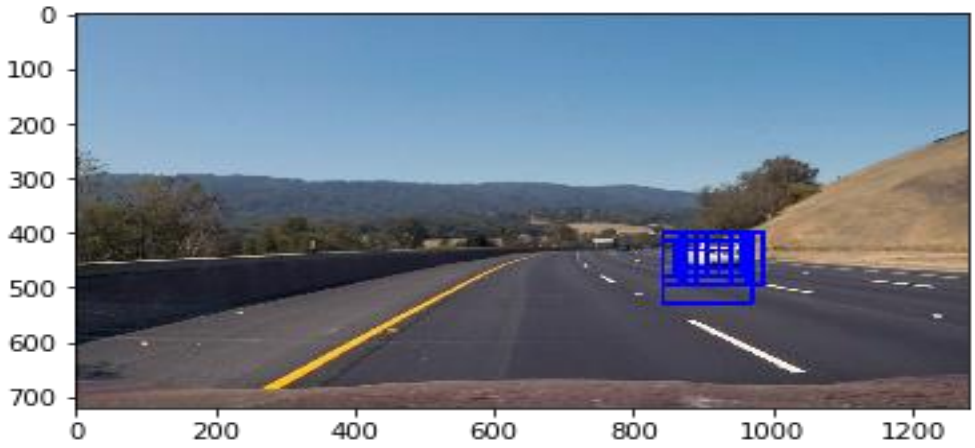
2. Show some examples of test images to demonstrate how your pipeline is working. What did you do to optimize the performance of your classifier?

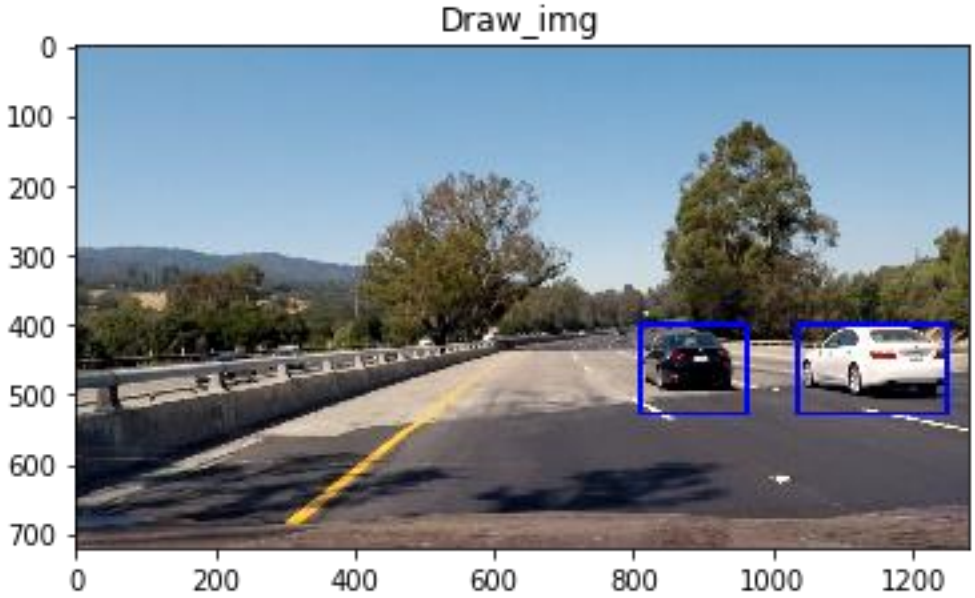
Ultimately I searched on two scales using RGB 3-channel HOG features plus spatially binned color and histograms of color in the feature vector, which provided a nice result.

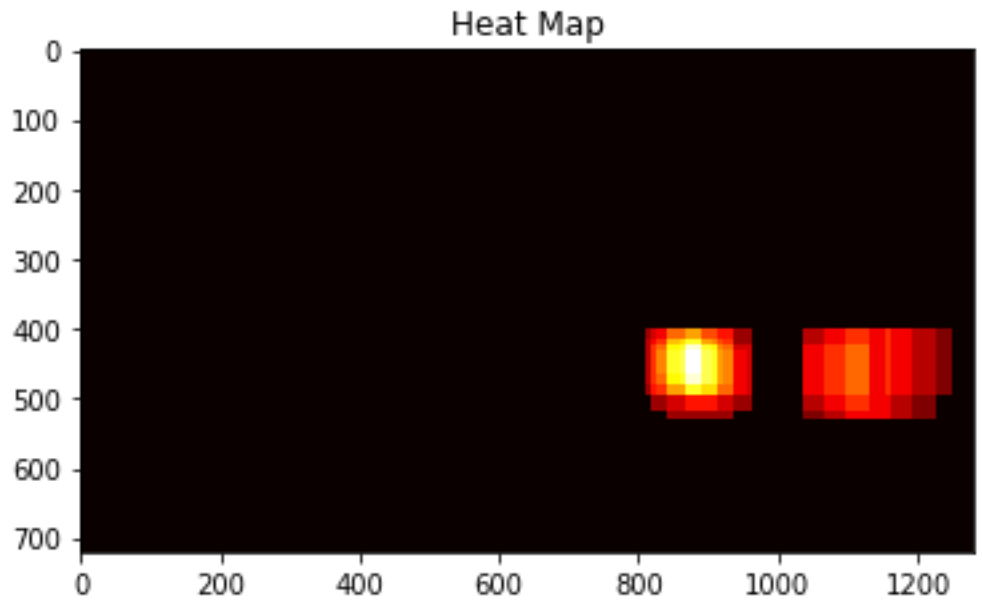
After getting the windows from slide window, I plotted heat maps to determine how many boxes have covered which part. This gives the probability of a vehicle there. This is to optimize the classifier.

I have also used thresholds.

Here are some example images:


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


Draw_img
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200



Video Implementation

1. 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.)

Video has been included as part of submission.

2. 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.

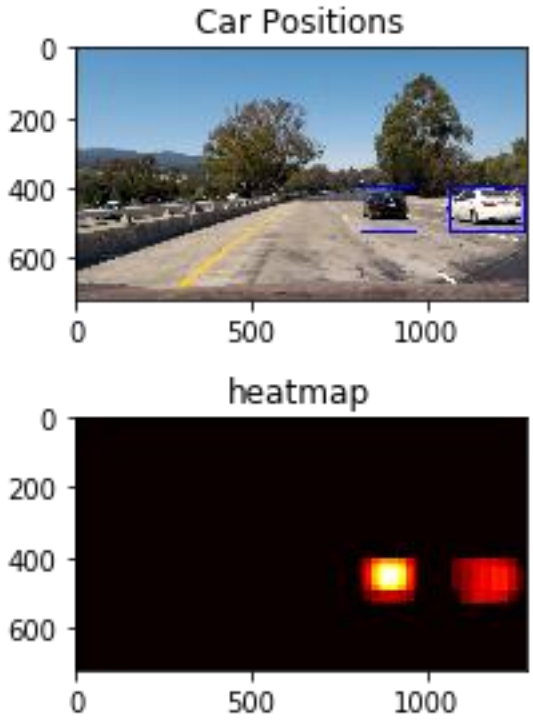
I recorded the positions of positive detections in each frame of the video. From the positive detections I created a heatmap and then thresholded that map to identify vehicle positions. I then used `scipy.ndimage.measurements.label()` to identify individual blobs in the heatmap. I then assumed each blob corresponded to a vehicle. I constructed bounding boxes to cover the area of each blob detected.

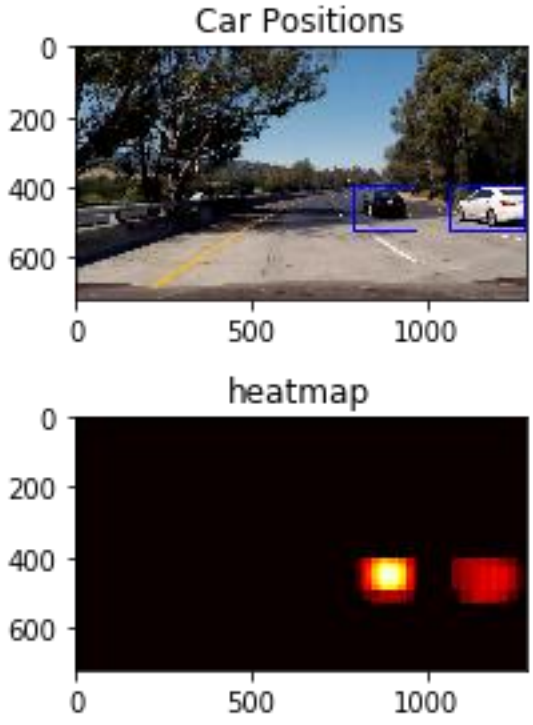
Here Thresholding is done to remove false detections.

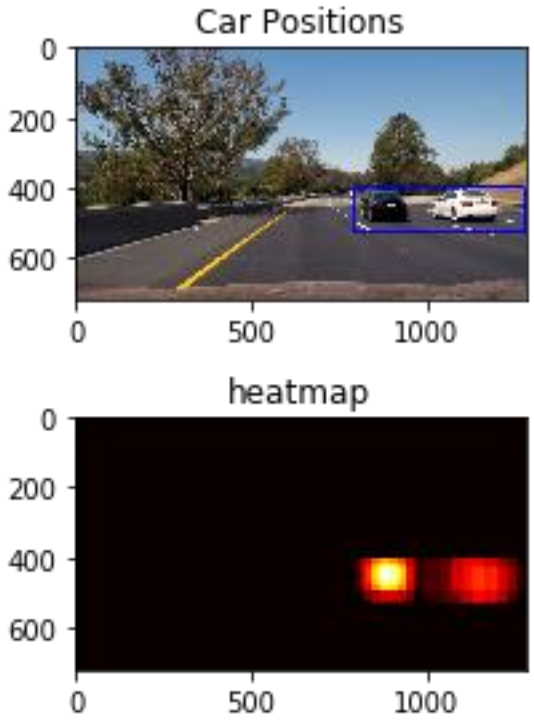
Labels are used to combine overlapping boxes to get a better idea of the region where we have a vehicle.

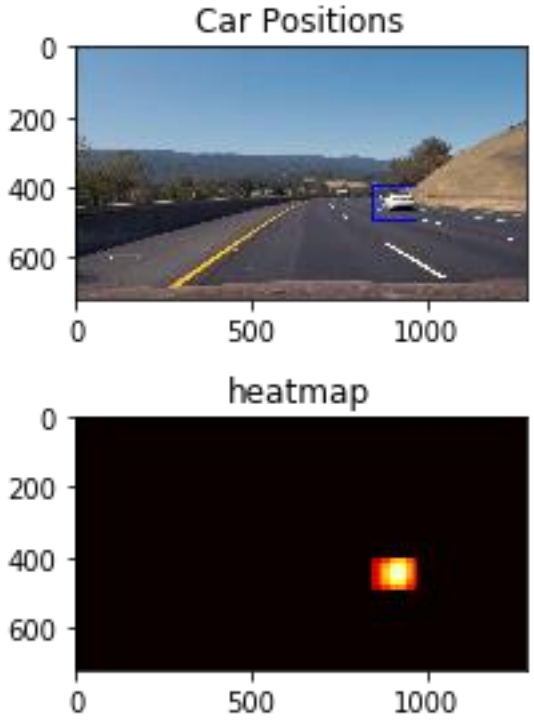
Here's an example result showing the heatmap from a series of frames of video, the result of `scipy.ndimage.measurements.label()` and the bounding boxes then overlaid on the last frame of video:

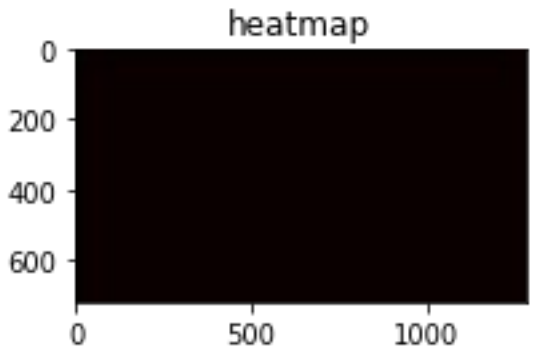
Here are six frames and their corresponding heatmaps:

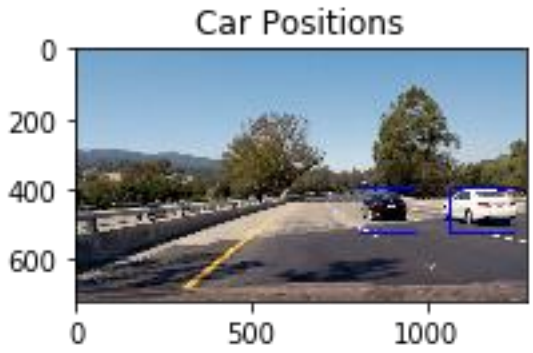

Car Positions
0
200
400
600
0
500
1000
heatmap
0
200
400
600
0
500
1000

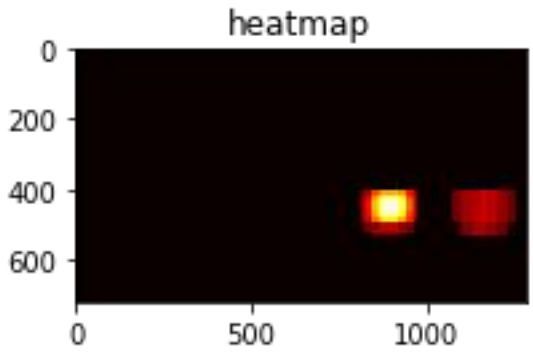

Car Positions
0
200
400
600
0
500
1000
heatmap
0
200
400
600
0
500
1000


Car Positions
0
200
400
600
0
500
1000
heatmap
0
200
400
600
0
500
1000

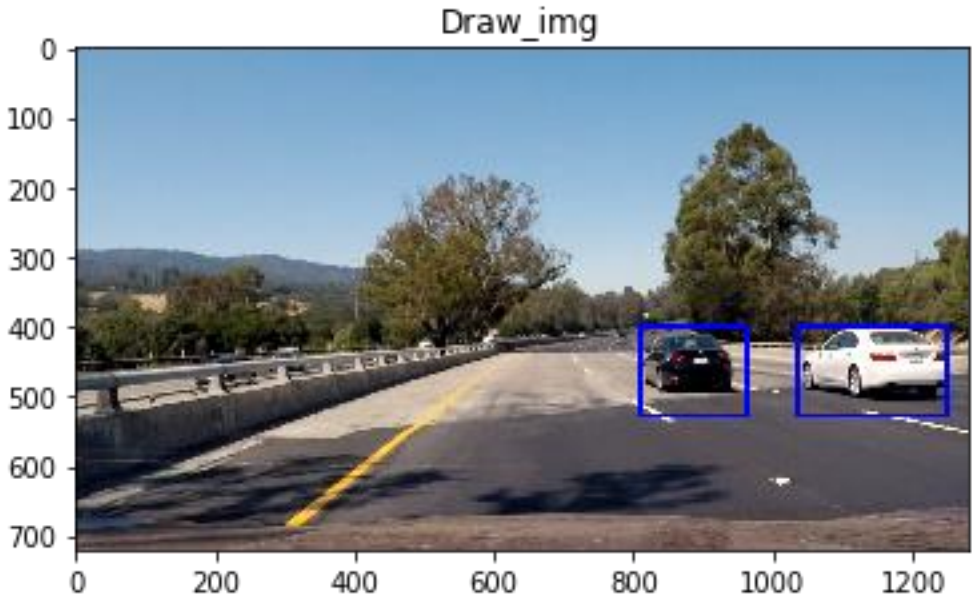

Car Positions
0
200
400
600
0
500
1000
heatmap
0
200
400
600
0
500
1000







Here the resulting bounding boxes are drawn onto the last frame in the series:



Discussion

1. 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?

Here I'll talk about the approach I took, what techniques I used, what worked and why, where the pipeline might fail and how I might improve it if I were going to pursue this project further

- As of now we have focussed on cars but there can be cases when we have motorcycles on road, then there may be a problem as the classifier is not trained for that.
- It would be better to detect pedestrians also to take it forward.
- One challenge that I faced was that of wobbles which got resolved by using more windows.
- Thresholds changes somehow removed false detection.