Vehicle Detection Project

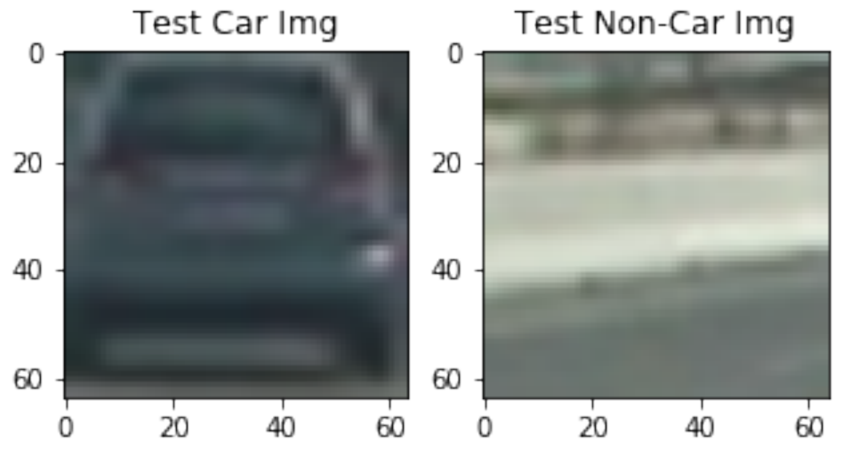
The goals / steps of this project are the following:

- 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.

### 1. Load Training Dataset.

Udacity prepared several images for both cars and non-cars for me to use, after I loaded all the images, it is showing in the project file that the sample size for car images is 8792 and the sample size for non-car images is 8968, which will be quite enough for training. (section 1.1)

I randomly shows two images, one of which is car and the other one is non-car. (section 1.2)



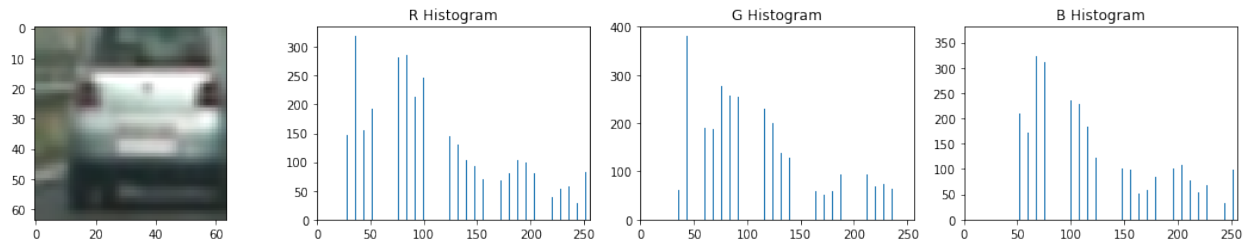
### 2. Classification Construction

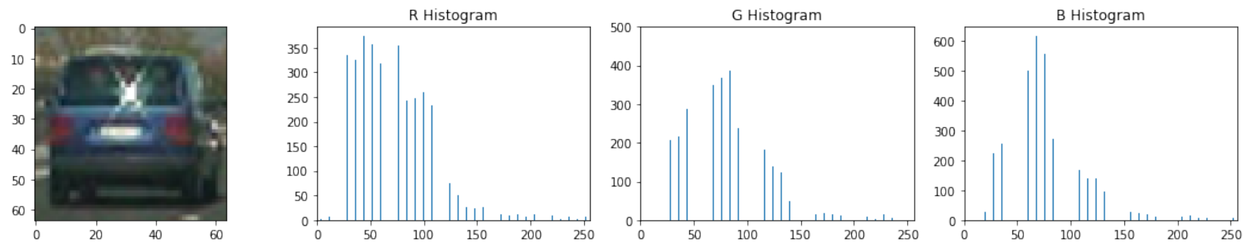
I used two different ways to additionally generate features, one is using the color features, the other one is using the histogram of gradients(HOG) features.

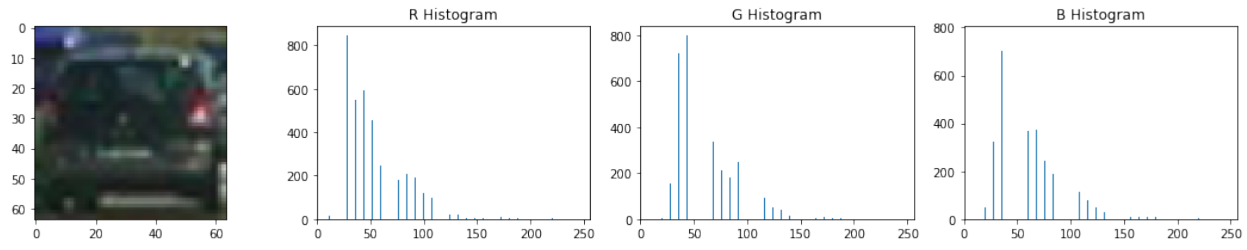
In the end of this section, both color and HOG features are used along with the spatial features to generate the final feature of each image.

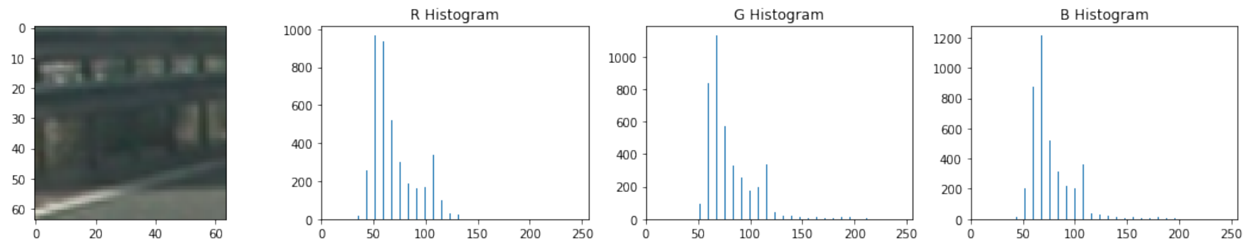
To use the color features, taking RGB color space as an example, we are basically extracting the R, G, B values and use their corresponding histogram as the output color feature. (section 2.1)

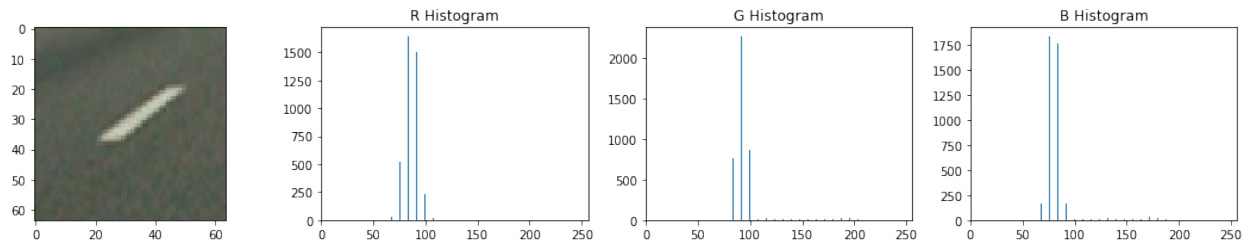
The following images are showing the R, G, B histograms for cars and non-cars.

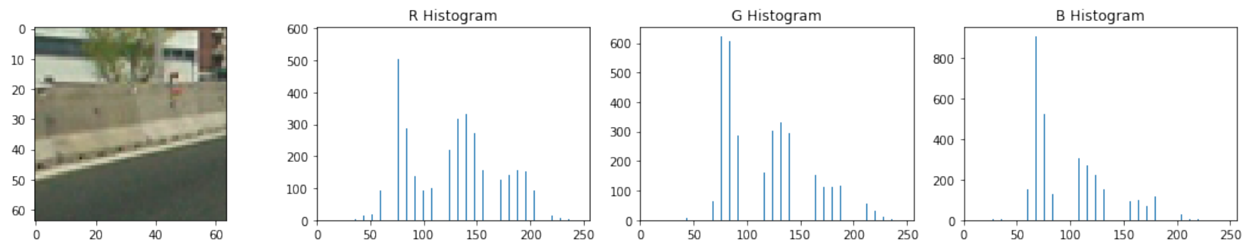






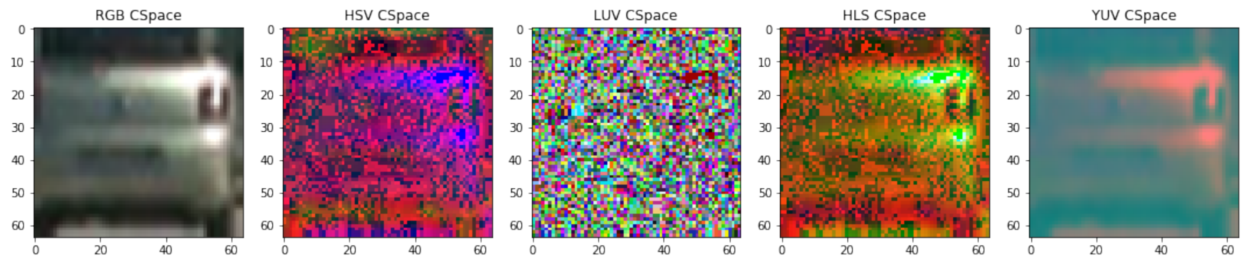






As shown in the above images, the difference are quite apparent between cars and non-cars images.

It is worth noting that, other than RGB color space, in the report, it also mentions HSV, LUV, HLS, YUV and YCrCb color spaces. Difference color spacing gives different perspective of the color features. In the following image, I am demonstrating one image



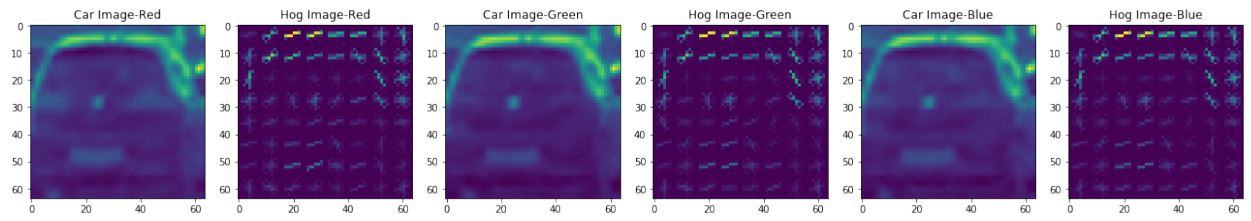
shown in different color spacings.

I also implemented function to extract HOG features. (section 2.2)

The idea of using HOG to classify cars from non-cars is to extract out the variation of gradients of and cell. To realize this we can use the sciki image library functions, which is asking us to define a parameter of cells and blocks.

A cell is consist of m*n pixels points color values. In the pipeline, I made it 8*8 pixels. A block is consist of m*n cells, and In the pipeline, I made it 2*2 cells. It is worth noting that, assuming we have a image separated in to 8*8 cells, and if we define one block is mode of 2*2 cells. Actually, we will be have 7*7 blocks instead of 4*4 blocks. It means, block will be having overlapping to ensure better coverage.

Also we need to define a parameter of gradient changing orientations, usually the range is 9-12. In the pipeline, I chose 9.
The following image shows the HOG of an image in its R, G and B channels.



The next step is to choose the algorithm to train our machine learning system. I simply choose learn support vector classifier as the starting point for both color and HOG extracted features.

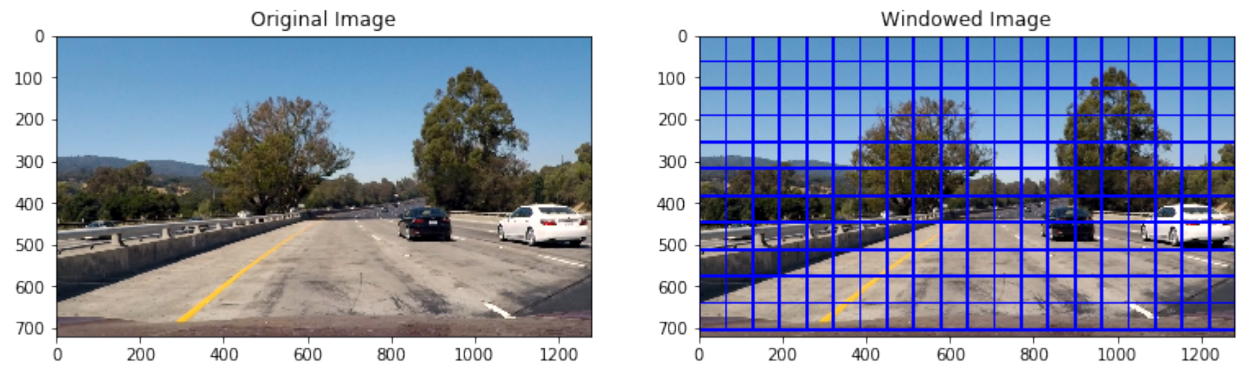
Actually, I tested the performance for spatial & color features vs. spatial & HOG features separately. As shown is the Jupyter Notebook project file, the partial & color features give the accuracy 95% and HOG features gives 92% accuracy, they are both doing a fairly good job. But in the final pipeline, I will use all spatial & color & HOG features to get the best training results.

And the combined features give 99% of prediction accuracy.

### 3. Sliding Windows.

The next challenge on our way to find cars is to slide within the gave image into different windows. Because we don't know where will cars be showing in the camera captured image, we have to kind of scan the whole image, in other words, sliding windows. (section 4)

The first step is just to traditionally define a size of window and slide from left to right, top to bottom. This process is shown in the following image.



The next step is to use HOG subsampling to slightly shift the window of an found car with some overlapping area, to do another prediction, which will generate overlapping windows but it will consolidate the prediction.

Given the following three test images, the found car position are also shown. (section 5)

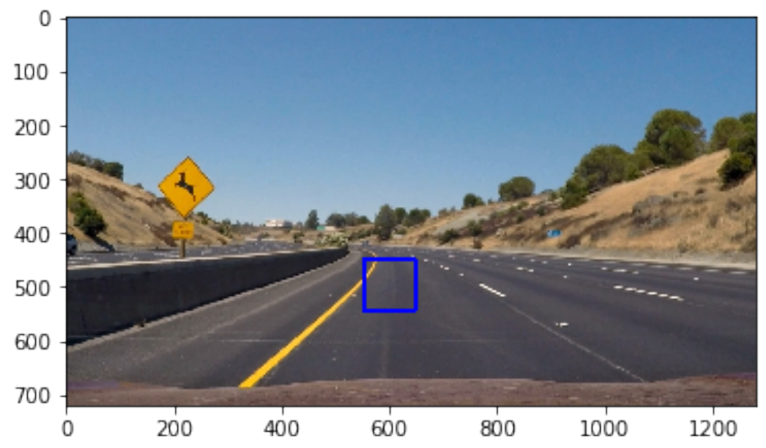


Image1 with false positives.

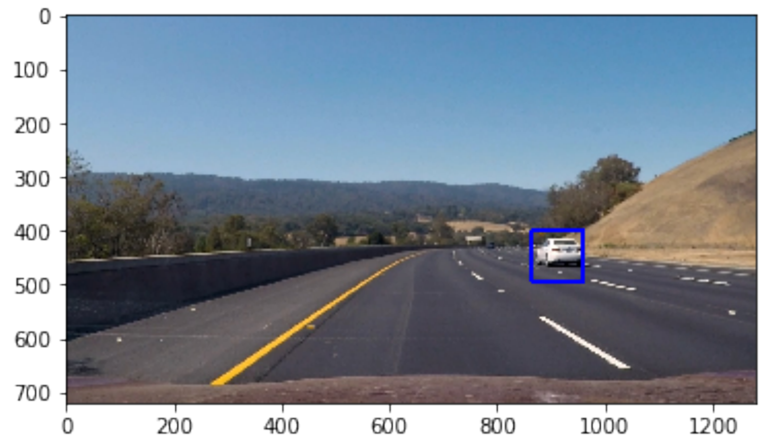


Image2 with correct but weak predictions.

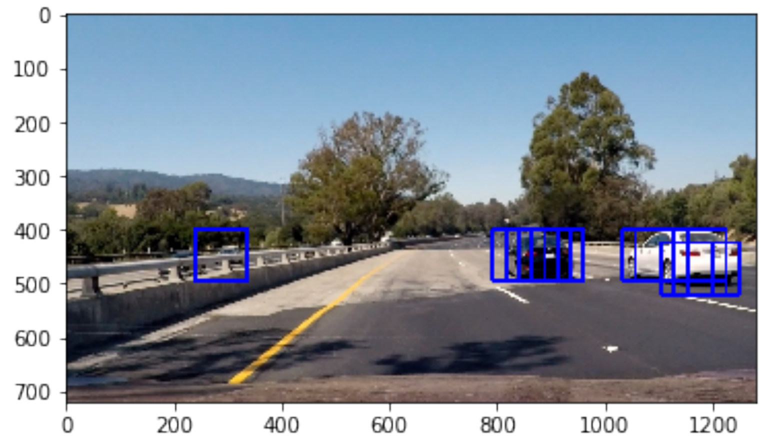


Image3 with strong and correct predictions.

The results of the test images lead to our next section, multi detection & false positive.

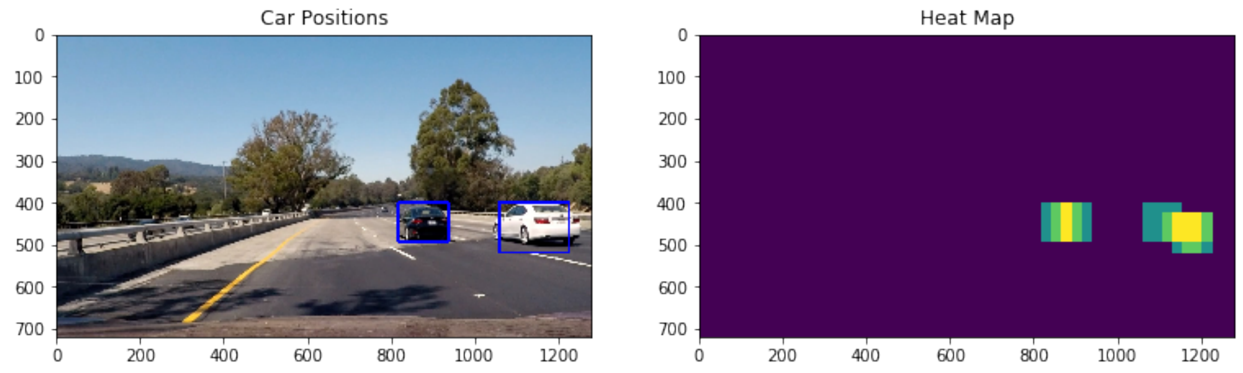
### 4. Multi Detection & False Positive

As in the above images, image1 has false positive predictions, image2 has correct predictions but it is kind of weak, and image3 has the best output. (section 6)

Although image3 has the idea output, the duplicated or overlapped windows are giving confusion as a image output.

To eliminate false positive and gracefully draw strong positive windows the heat map is introduced.

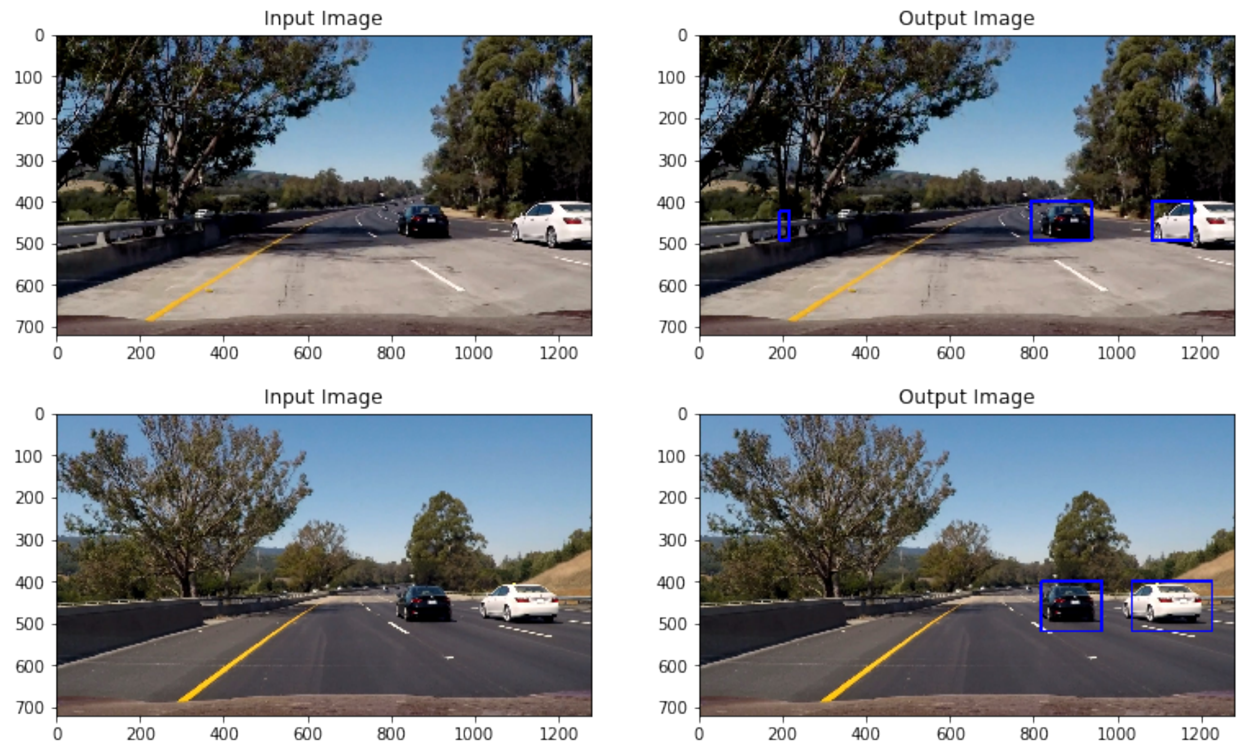
The following image demonstrates the use of heat map.



Still, we are using image3 as the input, but this time we have set a threshold, only if we found two or more overlapped windows we will draw the final window. Also, when we draw the window for the overlapped area, we will get the minimum top left corner and maximum bottom right corner of all overlapped windows and only draw a single window.

### 5. Combine Everything for Video

After combining everything, we finally get the function process_image. I tested the process_image function and get the following image results.



The last but not least step is to use moviepy library to get each frame image out of the input video and process it and put into the result image. Processing time is also logged for both test video and project video.

```
 97%|██████████| 38/39 [00:10<00:00,  3.51it/s]
CPU times: user 10.2 s, sys: 736 ms, total: 10.9 s
Wall time: 12.1 s
```

```
100%|██████████| 1260/1261 [05:52<00:00,  3.64it/s]
CPU times: user 5min 30s, sys: 20.9 s, total: 5min 51s
Wall time: 5min 53s
```

Project output video is available here in Youtube.
https://www.youtube.com/watch?v=RV-QveGuuhM

### 6. Discussions

The learn support vector classifier seems robust as we predict on our training dataset (9000 images). Indeed, it gives as high as 99% prediction accuracy when we combined spatial, color & HOG features. However, there is one frame in the video still gives wrong result, which is actually caused by the shadow generated from a tree.

There are multiple ways to eliminate this kind of sudden failure, the first way to do this is to slide the window more smartly, because, we know, we expect larger windows when other cars are closer to our car. Second solution could be predicting the car movement, if a car is detected to be moving we should be able to trace the corresponding window and predict its next position, in this case the sudden failure can be filtered out as well.

I hope I will be able to implement these features when I keep learning from Udacity.