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.

###Writeup / README

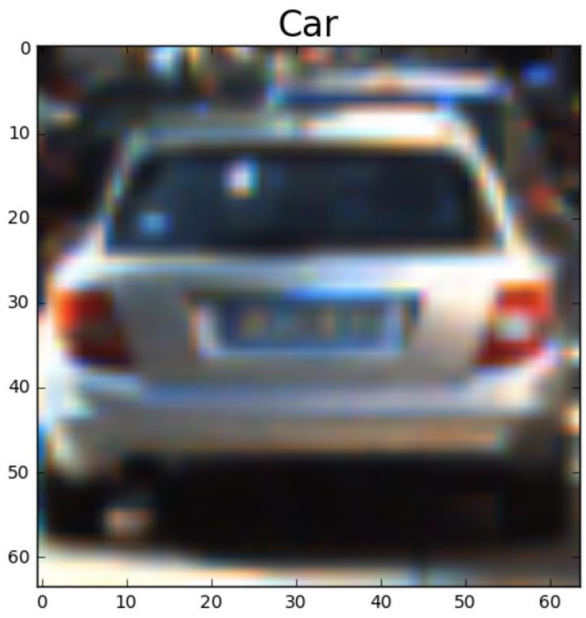
####1. Provide a Writeup / README that includes all the rubric points and how you addressed each one. You can submit your writeup as markdown or pdf. Here is a template writeup for this project you can use as a guide and a starting point.

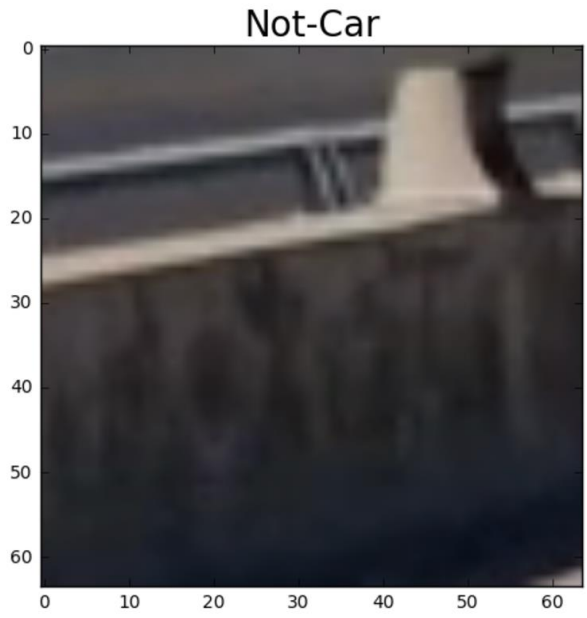
You're reading it!

###Histogram of Oriented Gradients (HOG)

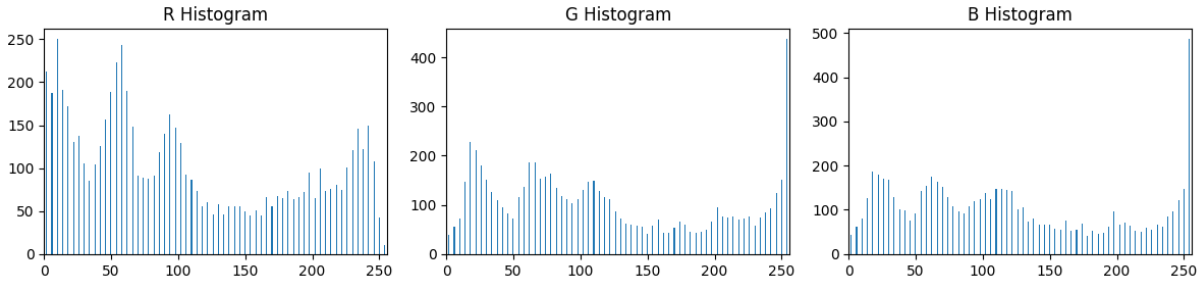
####1. Explain how (and identify where in your code) you extracted HOG features from the training images.

The code for this step is contained in the first code cell of the IPython notebook (or in lines # through # of the file called `some_file.py`).
I started by reading in all the `vehicle` and `non-vehicle` images. Here is an example of one of each of the `vehicle` and `non-vehicle` classes:

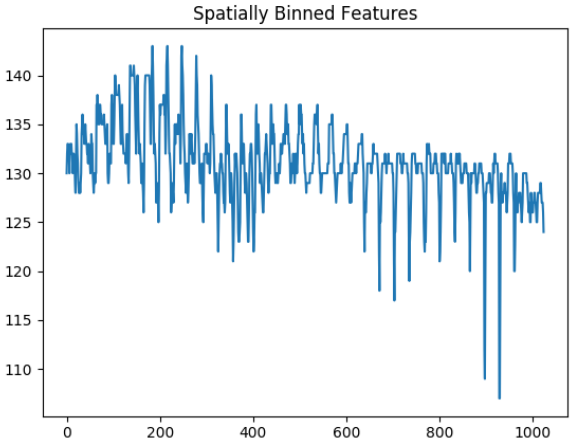




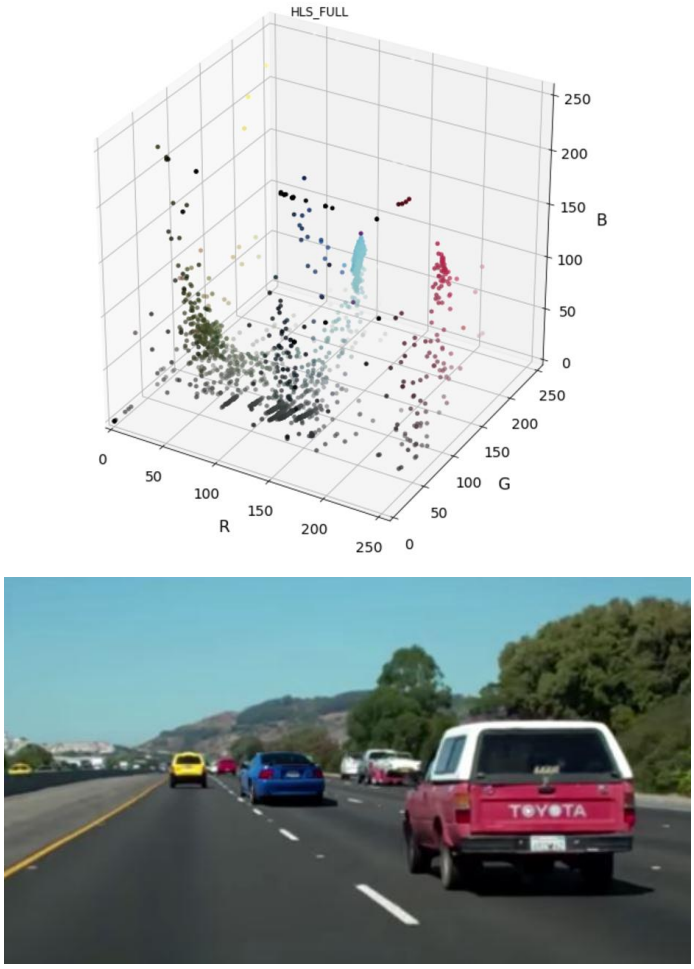
I tried to display some basics histograms of colors...



... some spatially binned too in order to understand what I was about to feed to the classifier:



I also generated all the possible 3D plot color space of an image trying to understand which one could be the best:



An image and one of its 3D plot color space

I then explored different color spaces 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.
I eventually decided to run both color and hog classifier on an important range of combinations

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

BGR	2	4	8	16	32	64	128	256
2	0,8477	0,8477	0,8373	0,8425	0,8358	0,8432	0,8418	0,841
4	0,9108	0,9034	0,916	0,9116	0,919	0,9302	0,9212	0,9116
8	0,9428	0,9428	0,9376	0,9473	0,9487	0,9398	0,9421	0,9428
16	0,9421	0,9406	0,9532	0,948	0,9495	0,9465	0,9443	0,951
32	0,9413	0,9421	0,9421	0,945	0,9465	0,9458	0,9413	0,9331

BGRA								
2	0,8507	0,8611	0,841	0,8484	0,838	0,8336	0,8462	0,8588
4	0,9183	0,916	0,9116	0,9212	0,9094	0,925	0,9108	0,9019
8	0,9473	0,9473	0,945	0,9398	0,9428	0,9376	0,9435	0,9376
16	0,9465	0,9465	0,9354	0,9406	0,9465	0,9435	0,9428	0,9554
32	0,9346	0,9525	0,9465	0,9421	0,9502	0,9465	0,945	0,9443

HLS								
2	0,899	0,8967	0,9235	0,9019	0,9086	0,9138	0,9309	0,9428
4	0,9383	0,9316	0,9264	0,9331	0,9398	0,9346	0,9487	0,9391
8	0,9458	0,9406	0,9495	0,948	0,9465	0,9287	0,951	0,9413
16	0,9302	0,9435	0,9368	0,945	0,9458	0,9458	0,9584	0,9517
32	0,9562	0,948	0,9435	0,9428	0,9421	0,9525	0,9473	0,9621

HLS_FULL								
2	0,89	0,9101	0,9086	0,9042	0,9175	0,916	0,9212	0,9398
4	0,9391	0,9294	0,9346	0,919	0,9406	0,9294	0,9525	0,948
8	0,9413	0,9458	0,9517	0,9421	0,945	0,945	0,948	0,9383
16	0,9264	0,9376	0,9361	0,9413	0,945	0,9406	0,9517	0,9502
32	0,9532	0,9354	0,9473	0,9539	0,9495	0,9569	0,9547	0,9621

HSV								
2	0,8982	0,8997	0,8975	0,9294	0,9027	0,9212	0,9376	0,9354
4	0,9376	0,9428	0,9309	0,9309	0,9354	0,9376	0,9391	0,9413
8	0,951	0,9532	0,9435	0,9495	0,9421	0,9577	0,9421	0,9443
16	0,9279	0,9361	0,9361	0,9421	0,9443	0,945	0,9406	0,9629
32	0,9495	0,9495	0,9473	0,9435	0,9599	0,9562	0,9621	0,9643

HSV_FULL	2	4	8	16	32	64	128	256
2	0,896	0,9027	0,9086	0,9064	0,916	0,9131	0,9302	0,9383
4	0,9354	0,9376	0,9376	0,9421	0,9294	0,9465	0,945	0,9495
8	0,9517	0,9458	0,951	0,9421	0,9458	0,9525	0,945	0,9517
16	0,9361	0,9361	0,948	0,948	0,9458	0,9465	0,9532	0,9606
32	0,945	0,9443	0,9495	0,9398	0,9629	0,9525	0,9584	0,9584

LAB								
2	0,8648	0,8871	0,8975	0,8997	0,9012	0,8982	0,9264	0,9101
4	0,9183	0,9264	0,9331	0,9294	0,9458	0,9368	0,9368	0,9406
8	0,9435	0,9391	0,9458	0,9562	0,9577	0,948	0,9569	0,9443
16	0,9435	0,9346	0,9398	0,9398	0,9532	0,9584	0,9487	0,9495
32	0,9525	0,951	0,9577	0,9569	0,9584	0,9673	0,9643	0,9666

LUV								
2	0,8685	0,8588	0,8611	0,8707	0,89	0,9071	0,9123	0,9019
4	0,9108	0,9264	0,9086	0,9235	0,9331	0,9242	0,9361	0,925
8	0,9361	0,9547	0,9517	0,9435	0,9376	0,951	0,9368	0,9339
16	0,9391	0,9361	0,9495	0,9487	0,9502	0,9368	0,9502	0,9621
32	0,9443	0,9599	0,9629	0,9502	0,9591	0,9554	0,9643	0,9621

Lab								
2	0,8796	0,8752	0,899	0,8952	0,9086	0,9079	0,9086	0,9168
4	0,9235	0,9294	0,9272	0,9324	0,9235	0,9413	0,9339	0,9316
8	0,9502	0,9473	0,9368	0,9517	0,9591	0,9614	0,9577	0,9383
16	0,9368	0,9517	0,9443	0,951	0,9421	0,9525	0,9562	0,9651
32	0,9517	0,9554	0,9539	0,9591	0,9621	0,9695	0,9591	0,9591

Luv								
2	0,8544	0,8603	0,867	0,8774	0,9027	0,9034	0,9056	0,9116
4	0,9049	0,9101	0,925	0,9175	0,9205	0,922	0,9272	0,9212
8	0,9495	0,9435	0,9331	0,9443	0,951	0,945	0,9473	0,9435
16	0,9465	0,9383	0,948	0,9421	0,9421	0,9487	0,951	0,9532
32	0,9443	0,9473	0,9547	0,9539	0,9688	0,9569	0,9621	0,9643

RGBA	2	4	8	16	32	64	128	2
2	0,8477	0,8328	0,8492	0,8403	0,841	0,8559	0,8477	0,84
4	0,919	0,9146	0,9146	0,9264	0,9175	0,9034	0,9235	0,90
8	0,9443	0,9354	0,9339	0,9428	0,9435	0,9547	0,9487	0,93
16	0,9473	0,9435	0,9428	0,9443	0,9495	0,9495	0,9346	0,95
32	0,9413	0,9473	0,9458	0,9465	0,9361	0,9465	0,9421	0,94

XYZ								
2	0,8366	0,844	0,847	0,8663	0,8403	0,8522	0,8484	0,8
4	0,9108	0,9175	0,9056	0,9086	0,9198	0,9205	0,9175	0,92
8	0,9406	0,9406	0,9458	0,945	0,9473	0,9435	0,9413	0,95
16	0,9532	0,9495	0,9495	0,9443	0,9577	0,9495	0,9599	0,95
32	0,9354	0,9331	0,9346	0,9487	0,9368	0,9339	0,9391	0,94

YCR_CB								
2	0,8388	0,8536	0,8462	0,8425	0,8522	0,8358	0,8514	0,85
4	0,925	0,9094	0,919	0,9175	0,9108	0,9146	0,916	0,91
8	0,9428	0,9406	0,9354	0,9435	0,9413	0,9316	0,9316	0,95
16	0,9287	0,9361	0,9413	0,9302	0,9346	0,9383	0,9339	0,94
32	0,9406	0,9502	0,9562	0,9569	0,9465	0,951	0,9495	0,95

YCrCb								
2	0,8529	0,841	0,8462	0,8373	0,8284	0,8336	0,8373	0,84
4	0,919	0,9101	0,9235	0,9049	0,9168	0,9108	0,9101	0,9
8	0,9495	0,9443	0,9458	0,9361	0,9391	0,9525	0,9361	0,93
16	0,9391	0,9272	0,9361	0,9339	0,945	0,9406	0,9368	0,93
32	0,9562	0,9435	0,9629	0,9614	0,9525	0,9562	0,951	0,94

YUV								
2	0,8477	0,8373	0,847	0,8522	0,8477	0,8499	0,8395	0,86
4	0,9086	0,9257	0,9264	0,925	0,8997	0,9108	0,9138	0,91
8	0,9421	0,9331	0,9376	0,9368	0,9383	0,9487	0,9339	0,94
16	0,9413	0,9376	0,9294	0,9443	0,9391	0,9272	0,9302	0,93
32	0,9525	0,9495	0,9487	0,9525	0,9547	0,9517	0,9532	0,95

Color classifier ran over 15 color space, from 2 to 32 space size and from 2 to 256 bins. The feature obtained is the accuracy of the classifier. It ran on the vehicule/non vehicule GTI dataset.

BGR	0	1	2	ALL
6	0,9525	0,9569	0,9539	0,9651
8	0,9547	0,9599	0,9525	0,9584
10	0,9621	0,9614	0,9651	0,9695
12	0,9681	0,9747	0,9569	0,9747

HSV				
6	0,8603	0,8811	0,9547	0,9718
8	0,8767	0,8789	0,9577	0,9725
10	0,9086	0,8908	0,9666	0,9747
12	0,8908	0,89	0,9673	0,9785

Lab				
6	0,9502			
8				
10				
12				

YCR_CB				
6	0,9562	0,8878	0,8611	0,9629
8	0,9577	0,8841	0,8536	0,9718
10	0,9606	0,8848	0,8432	0,9733
12	0,9629	0,8834	0,8611	0,974

BGRA	0	1	2	ALL
6	0,9525	0,9643	0,9487	0,9547
8	0,9643	0,9636	0,9525	0,9643
10	0,9629	0,9658	0,9502	0,9688
12	0,9688	0,9725	0,9584	0,9762

HSV_FULL				
6	0,867	0,8923	0,9554	0,9718
8	0,8633	0,8945	0,9643	0,971
10	0,8893	0,8893	0,9606	0,9762
12	0,8967	0,893	0,971	0,9837

Luv				
6	0,951			
8				
10				
12				

YCrCb				
6	0,9577	0,9004	0,8603	0,9651
8	0,9517	0,8715	0,8633	0,9681
10	0,9554	0,8945	0,8692	0,9747
12	0,9562	0,8767	0,847	0,9792

HLS	0	1	2	ALL
6	0,8611	0,9517	0,8848	0,971
8	0,8574	0,9614	0,8908	0,9725
10	0,9079	0,9614	0,8796	0,9762
12	0,8967	0,9681	0,8863	0,9814

LAB				
6	0,9517			
8				
10				
12				

RGBA				
6	0,951	0,9577	0,9577	0,9681
8	0,9569	0,9577	0,9599	0,9725
10	0,9591	0,9599	0,9681	0,977
12	0,9539	0,9673	0,9554	0,9666

YUV				
6	0,9495	0,899	0,8499	0,9666
8	0,9577	0,8938	0,8685	0,9792
10	0,9569	0,8804	0,838	0,974
12	0,9562	0,8997	0,8529	0,9785

HLS_FULL	0	1	2	ALL
6	0,867	0,9577	0,8886	0,957
8	0,8581	0,9532	0,9019	0,974
10	0,8967	0,9643	0,8915	0,978
12	0,8908	0,9569	0,8923	0,97

LUV				
6	0,9629			
8				
10				
12				

XYZ				
6	0,9569	0,9525	0,9487	0,958
8	0,9547	0,9606	0,9621	0,961
10	0,9532	0,9643	0,9606	0,961
12	0,9606	0,9502	0,9606	0,97

I then did the same for the hog classifier. I could understand that ALL the channel was in general more accurate than channels alone, and the higher was the orientation the better it was too.

So my final choise was to use the HSV_FULL color space with ALL the channel and an orientation of 14, a space size of 32 and 32 bins.

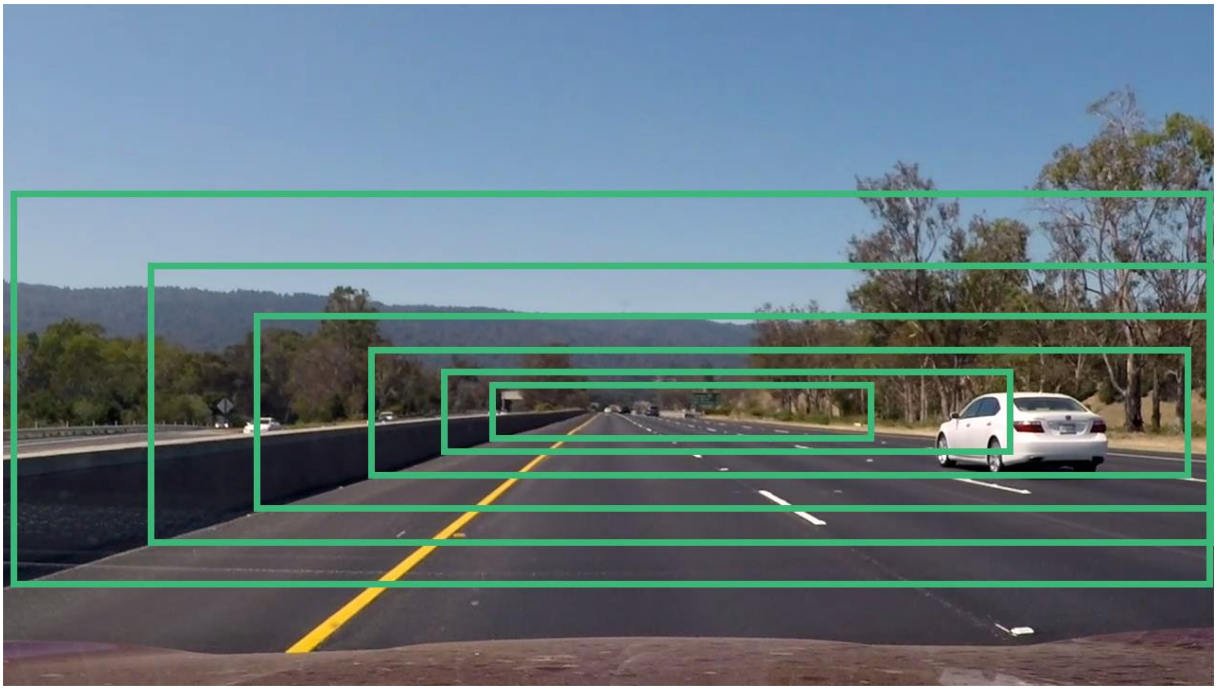
####3. Describe how (and identify where in your code) you trained a classifier using your selected HOG features (and color features if you used them).

To train the classifier I used the simple SVM() module with voluntarily no tweak at all. Indeed training the classifier was most of the time really long. I used both color and hog features to train it and didn't had the time to try other combination.

###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?

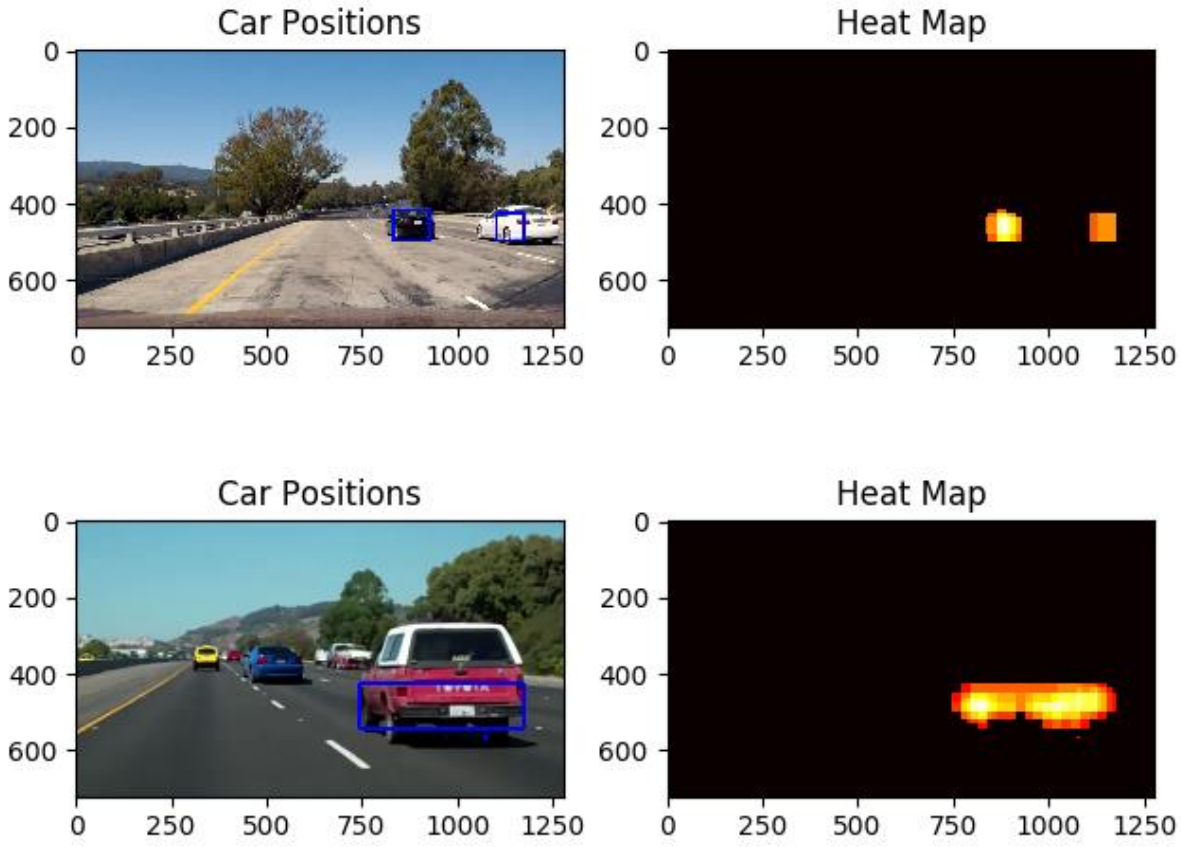
I first tried to scale the window search through the perspective but I was not sure rather it was efficient or not.



Then I came back on something simpler with two windows search, both with ystart = 400px and stop = 650px, but with two different scales (1.2 and 1.8)

####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 HSV_FULL 3-channel HOG features plus spatially binned color and histograms of color in the feature vector, which provided a nice result. Here are some example images:



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.) Here's a link to my video result

####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.

###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?

Unfortunately, with all these parameters to tweak, I didn't had the time to make it properly. My classifier is not satisfying, I think it's underfitting as the training time is low and the test accuracy is high.

```
Using: 14 orientations 8 pixels per cell and 2 cells per block
Feature vector length: 11400
8.06 Seconds to train SVC...
Test Accuracy of SVC =  0.9918
```