Project Report - Vehicle Detection

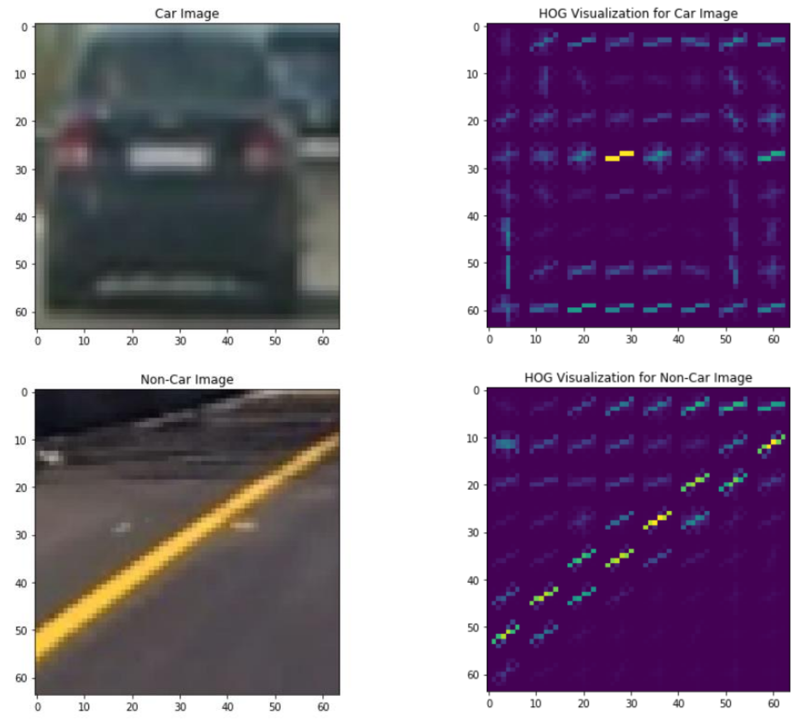
Histogram of Oriented Gradients (HOG)

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

In the code block 6 I have implemented the extract_features function to extract the hog features from the training images (Car and Not Car images).

First, I have read all the training images (Code block #3). The Images below give an example to car and not car images.

The images below show the respective HOG visualizations for the car and not car images. Code block #4.



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

I have tried different combinations of parameters and HOG channels. I have selected my final training parameters based on the accuracy and training time. Based on the above criteria the parameters in the row #7 were applied.

S.NO	Color Space	Hog Channel	Cell / Block	Orient	Spatial Size	Pix per cell	Hist Bins	Accuracy	Train Time
1	RGB	0	2	11	16	8	16	97.13	8.74
2	RGB	0	2	9	16	8	16	97.16	7.51
3	RGB	1	2	9	16	8	16	97.72	7.32
4	RGB	2	2	9	16	8	16	96.93	6.88
5	YCrCb	All	2	11	32	16	32	98.56	8.79
6	HLS	All	2	11	32	8	32	98.85	6.31
7	YCrCb	All	2	11	32	8	32	99.18	7.73
8	YUV	All	2	11	32	8	32	99.01	8.25
9	YUV	All	2	11	32	16	32	98.34	9.95

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 the code block #6 (from line #27 to #55), I have trained the Linear SVM using the parameters above and U have got the test accuracy of 99.18 %.

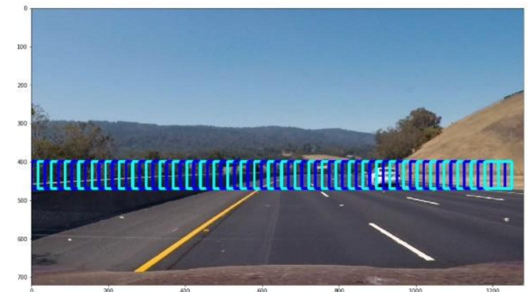
Sliding Window Search

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 have decided upon using the search window from 400th row – Y position(400) as the region above it (from 0 to 399) is not going to have a car. Hence, I have chosen the region (400 – 700) for different window sizes and I have tried different scales and came up with the following sizes based on its performance on the project video. Refer code blocks from #

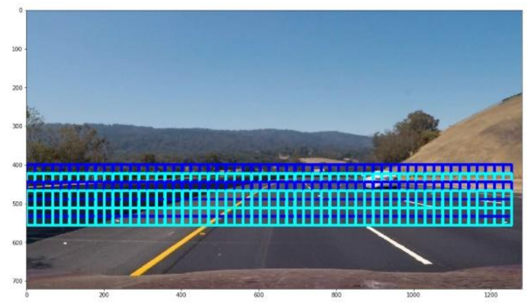
Scale: 1.1

Ystart = 400; Ystop =500



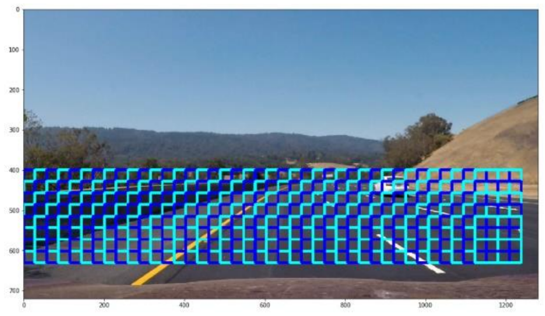
Scale: 1.4

Ystart = 400; Ystop =580



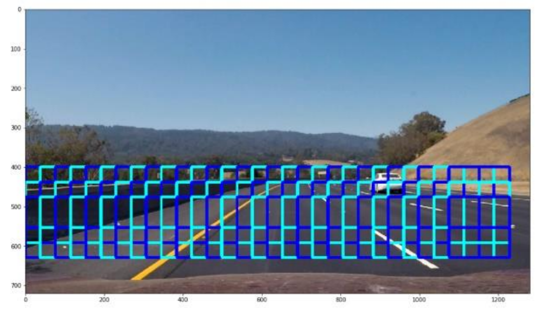
Scale: 1.8

Ystart = 400; Ystop =680



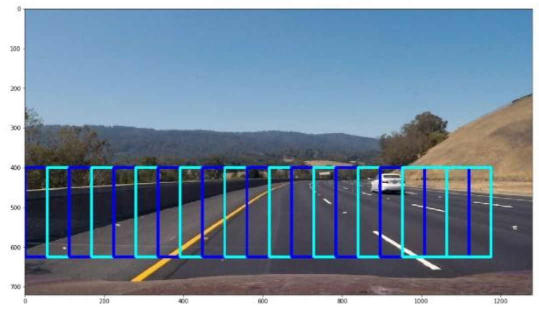
Scale: 2.4

Ystart = 400; Ystop =700



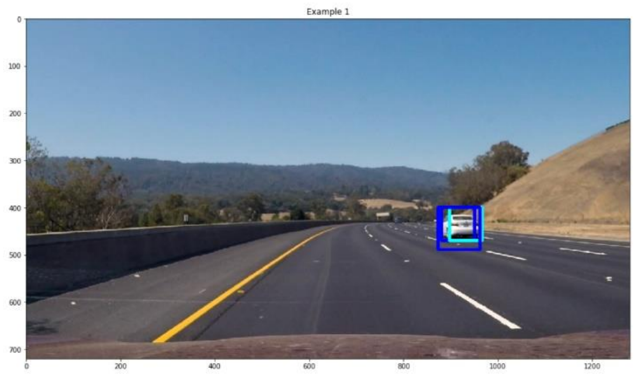
Scale: 3.5

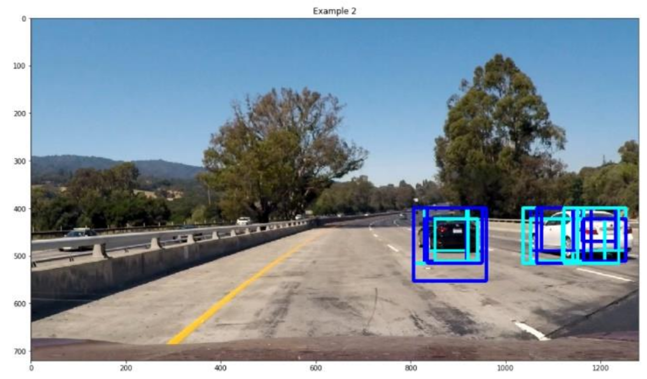
Ystart = 400; Ystop = 720

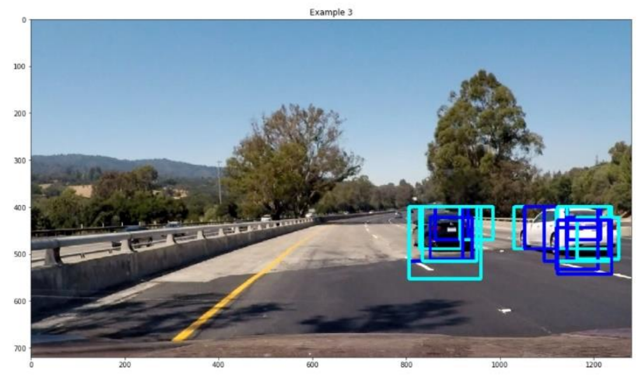


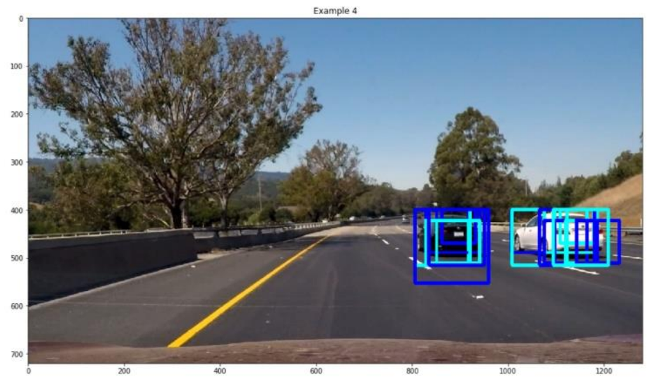
2. Show some examples of test images to demonstrate how your pipeline is working. What did you do to optimize the performance of your classifier?

Then I have searched all the four scales mentioned above using YCrCb 3-Scales HOG features along with Spatial and Color Histogram features to find the cars. Please see the example images below.









Video Implementation

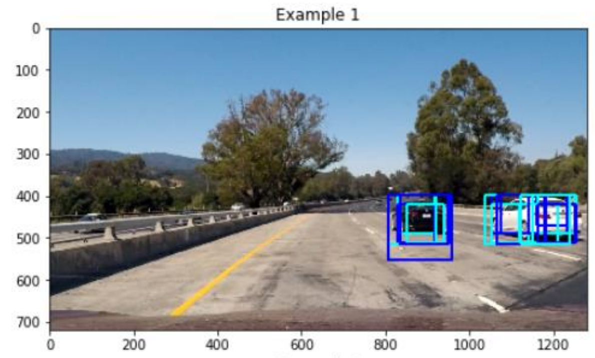
1. Provide a link to your final video output. Your pipeline should perform reasonably well on the entire project video (somewhat wobbly or unstable bounding boxes are ok as long as you are identifying the vehicles most of the time with minimal false positives.)

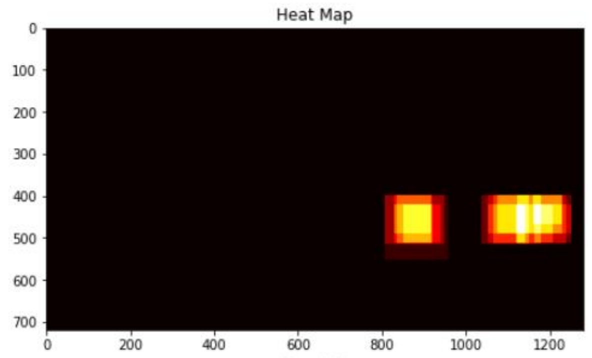
Please find the final video output named as “Final Result Video” in the uploaded zip folder.

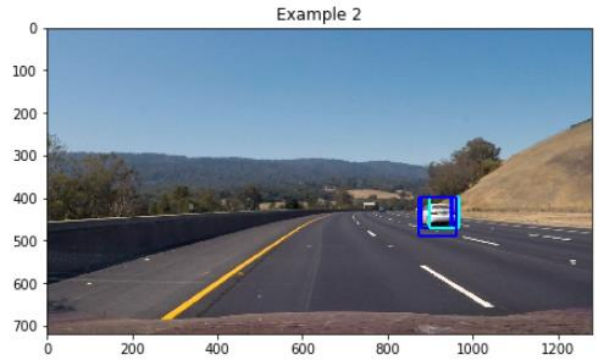
2. Describe how (and identify where in your code) you implemented some kind of filter for false positives and some method for combining overlapping bounding boxes.

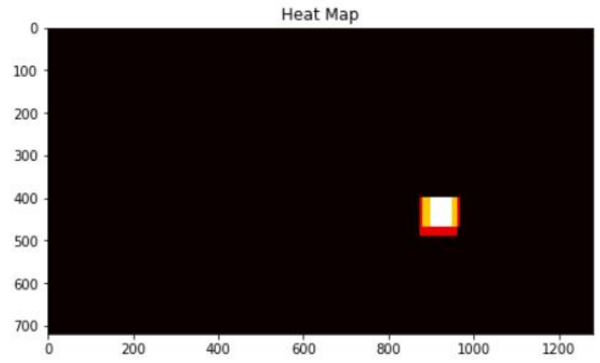
I have recorded the positive detection boxes on each frame and used it to create the heatmap (Code Block #14). I have used threshold on this heatmap to remove the false positives. Then I have used the scipy.ndimage.measurements.label() to label each bounding boxes assuming it represents each car and then I have used this box to represent the car in each frame of the video. Following images explain the steps I have followed.

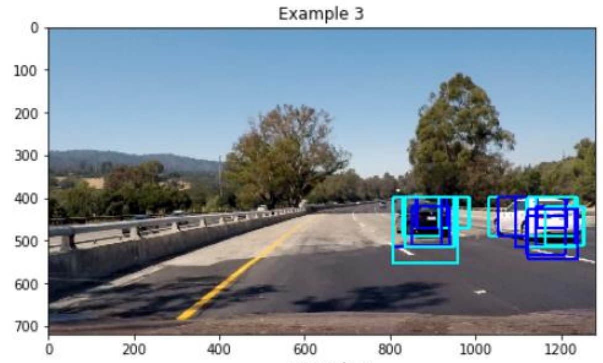
Heatmap generation:

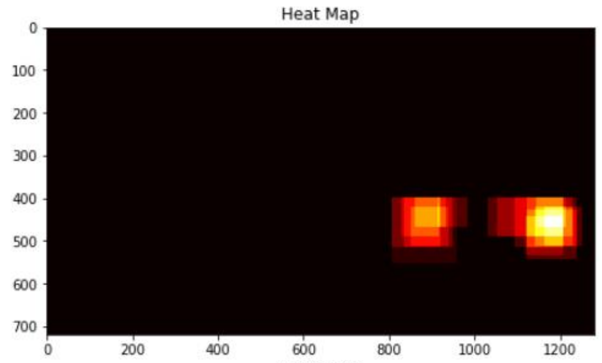


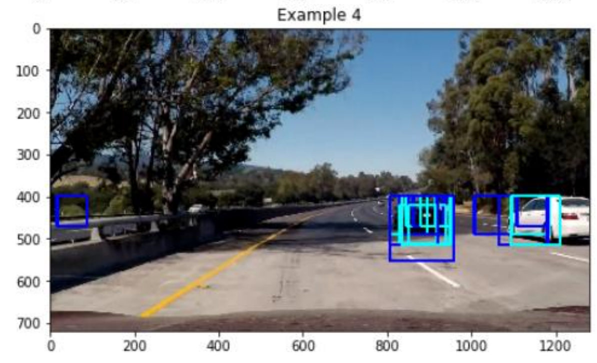


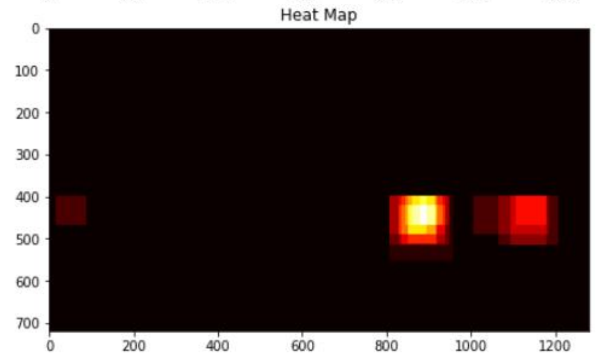




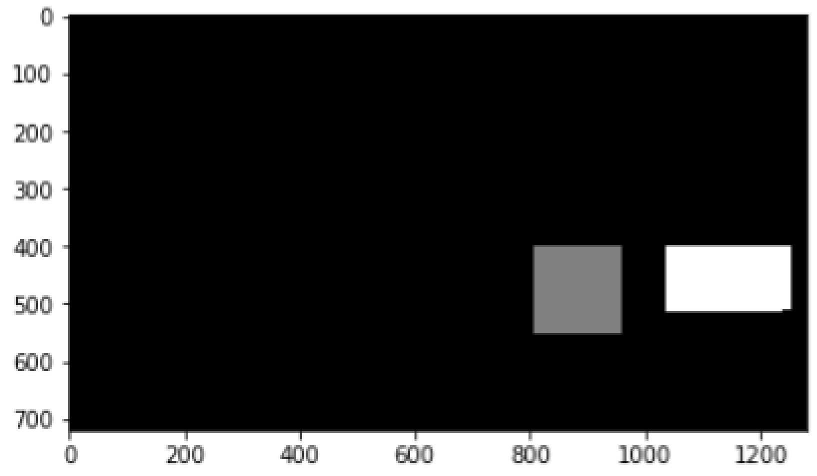




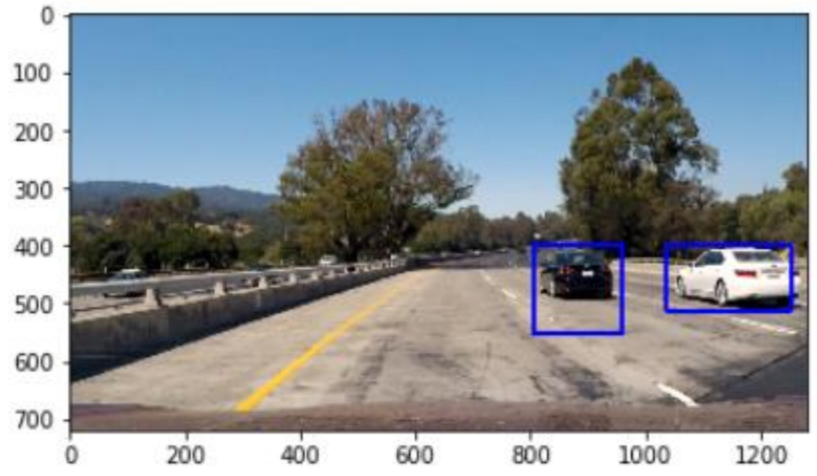




<u>The output of scipy.ndimage.measurements.label():</u> Code Block #23 & #17



<u>The resulting bounding boxes:</u> Code Block #23 & #17



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?

The problems I have faced in this project is mainly associated with the accuracy and detection speed. To to have good accuracy (99%) in detecting the cars, the detection speed is low and otherwise to process the frames like real-time, high processing power is required.

The lightning and environmental conditions will play a crucial role in the pipeline and they will be some of the major factors for the failure of the pipeline. On-coming traffic also provides disturbance in this case. It can be eliminated by using detecting the left and right most yellow lanes in the image frame.

Some of the methods that can be used to make our algorithm more robust are discussed below. Predicting the vehicle position frame in the subsequent frames by previous position and measured speed will be one good approach to track the vehicle. Another approach is to train a highly accurate classifier and use it along with high processing power to accurately identify the cars in the frame.