Vehicle Detection And Tracking Project

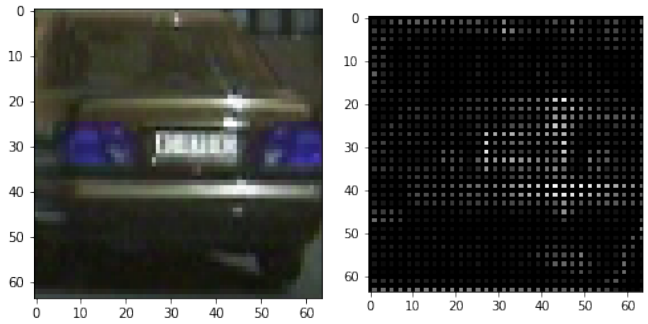
Hog Feature Extraction:

Hog Features are extracted using the hog function from skimage.feature library . Hog Features are extracted in the get_hog_features function in the second block of the jupyter notebook.The Optimal Hog parameters are selected based on these criteria,

1. Accuracy of the classifier.
2. Time taken for Classification.
3. Minimum Number of False Positives.

After experimenting with ‘RGB’ , ‘HSV’, ‘HLS’ , ‘YUV’ , ‘YCrCb’ colorspaces I found out that ‘YCrCb’ gave better accuracy.

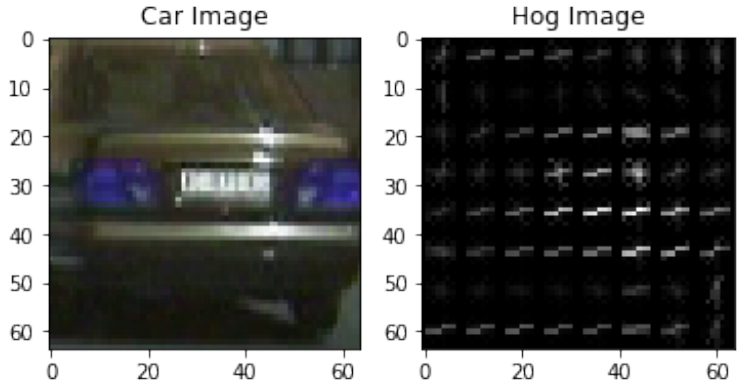
First No of orientations = 11 , Pixels_per_cell = 2 and Cells_per_block = 2 were used, but it is computationally slow and the vehicles were not classified properly in the video.



Car Image and its Hog Visualisation

Then different combinations of no.of.orientations,pixels_per_cell,cells_per_block were tried and finally the one which gave more generalised classification and better results in the video were chosen,

No Of Orientations = 9
Pixels per Cell = 8
Cells per Block = 8



Hog Visualisation with Final Parameters

Tranning the Classifier:

The extract_feature function extracts features by Spacial binning(size = 32) , Color Histogram (nbins = 32) and Hog and the final dataset is labelled as 1 for car images and 0 for non car images. Total no of features extracted per image in the dataset = 4896

Scaling of the Dataset and Classifier Traning is done in the third block of the Jupyter notebook.

The dataset is splitted into X_train and X_test and X_train is scaled using StandardScaler().fit() function. X_test data is cross validated with the classifier to check whether the it is over fitting or not.

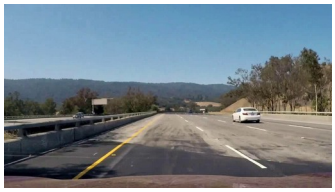
Test accuracy of the classifier is 0.9764292878635907

Sliding Window Implementaion:

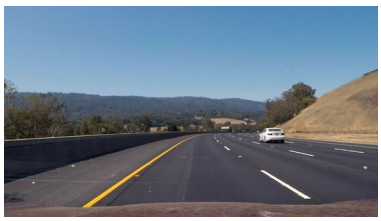
I used different scales of sliding window [1,1.3,1.4,1.47,1.5,2,2.5,3](window size = 64) with 75 percentage of overlapping. Sliding window implementation can be found under the find_car function in the fourth block of the code.

The reason for using 8 different windows is that to detect the vehicles in each frame of the video.

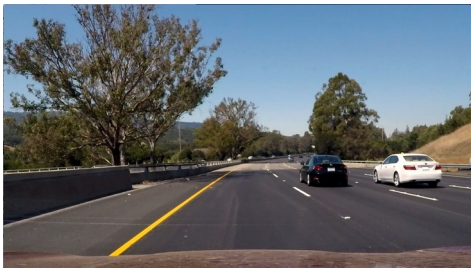
In the image given below the vehicle (just before the horizon) is detected only in the range of 1.4 to 1.5 scaled window

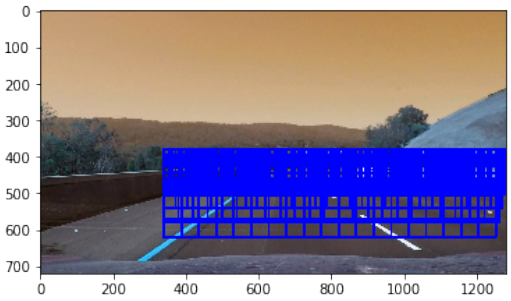
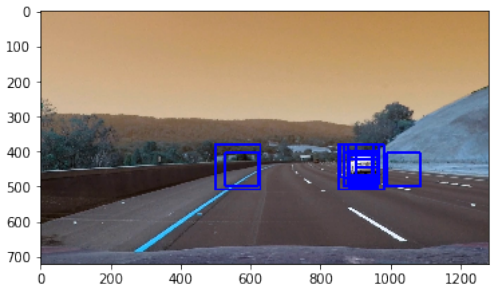


And in this image the vehicle (in the horizon) is detected in the range of 1 to 1.4



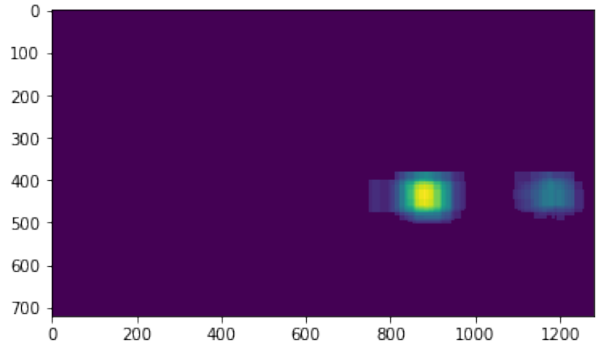
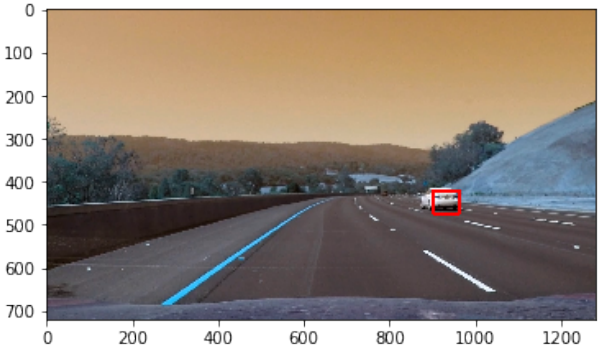
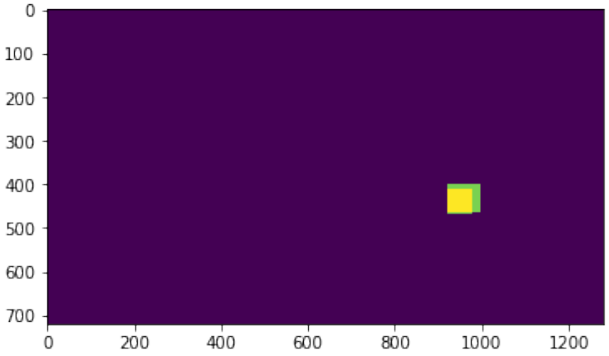
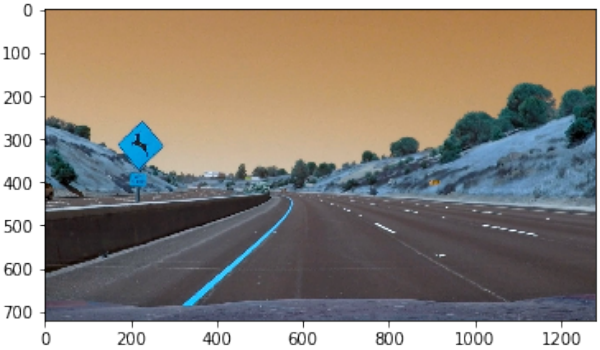
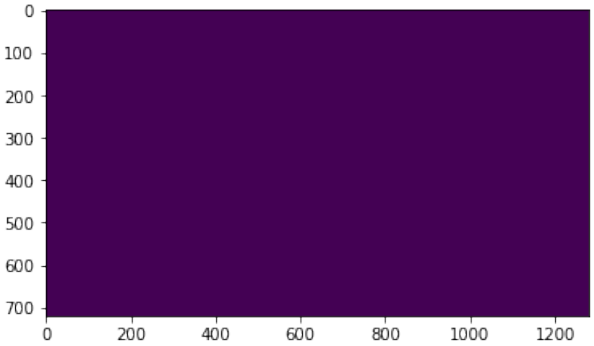
And in this image vehicle (near the camera) is detected in the range of (1.5 to 3)

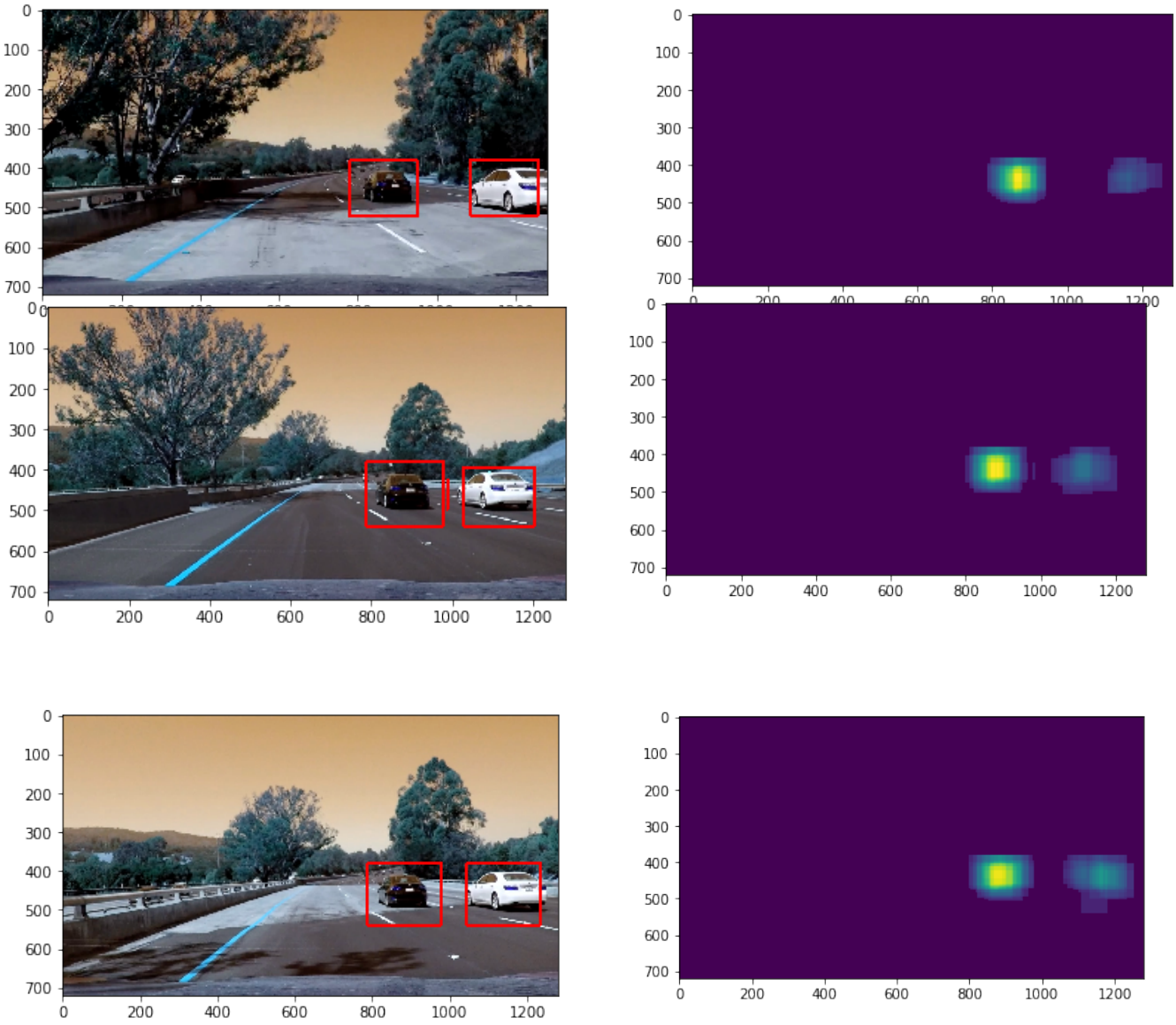


Reduction of False Positives:

Heat maps are used to reduce false positives. For each positive classification the corresponding pixels in the heat maps are incremented by one and then by applying threshold (Threshold = 4) false positives are reduced. Heat maps and threshold implementation can be found under the add_heatmap and apply_threshold functions in fourth block of the code.



Discussion:

Problems Faced:

It was very challenging to select the correct window size that detects the vehicle throughout the entire video.

It took very long time to process the entire video .

Drawbacks in the Pipeline:

Most of the non_car images in the dataset were extracted from the video itself so it is not a generalised classifier. More non_car images must be taken from the real world to generalise the classifier.

Feature extraction is computationally slow so this pipeline is not fit for real time vehicle detection and tracking . Deep learning techniques must be implemented.