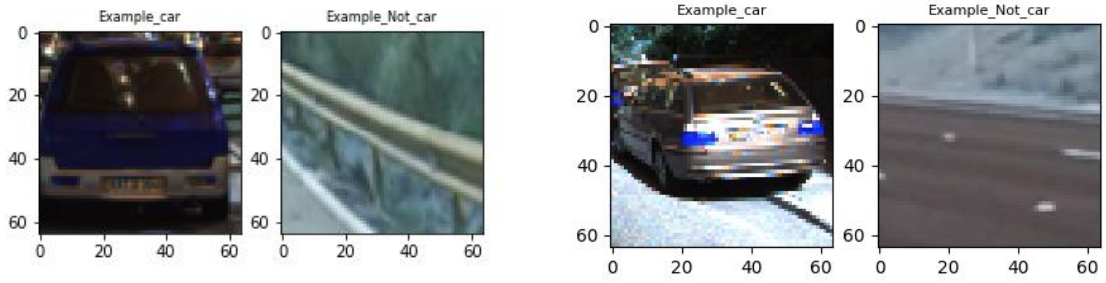
Vehicle Detection Project

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 Linear SVM classifier
- Apply a color transform and append binned color features, as well as histograms of color, to the HOG feature vector.
- Normalize the features and randomize a selection for training and testing.
- Implement a sliding-window technique and use trained classifier to search for vehicles in images.
- Run the pipeline on a video stream 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.

Extraction of Histogram of Oriented Gradients (HOG): -

First, I read in the car and non-car images separately. An example of the car and non-car image is as follows: -

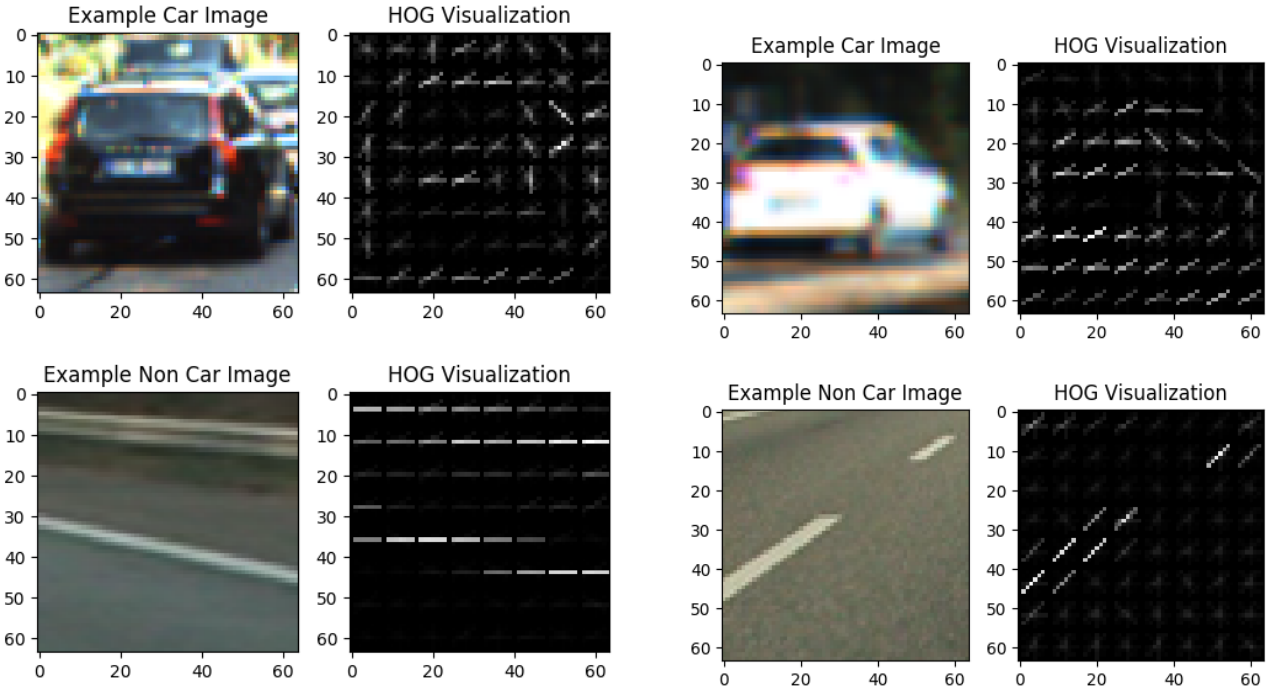


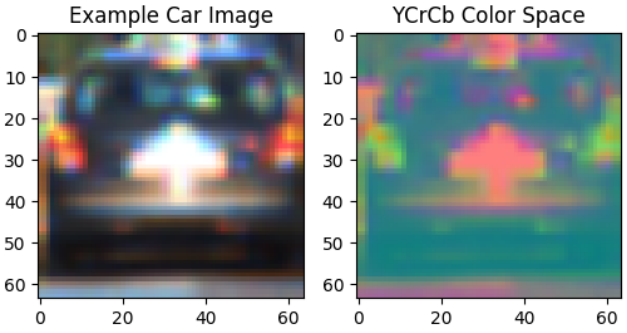
A function (get_hog_features) was created for extracting the HOG features. The code for the same is included in the required_functions.py file from line 8 to 26. The function returns two different outputs depending on the arguments given to it. If the visualization argument is set to false only the HOG features will be returned. If visualization is set to true than an image on which HOG is applied will be returned along with the features.

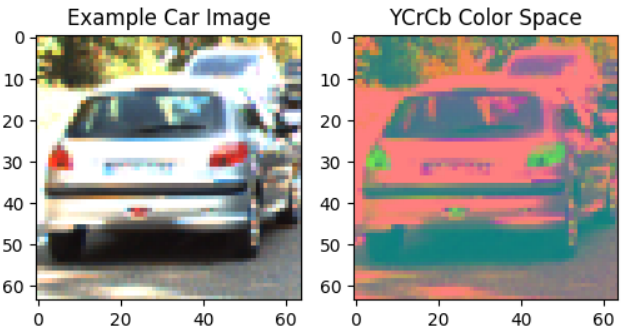
This function in turn is an input to the extract_features function. The code for the same is included in the required_functions.py file from line 65 to 118. This function can also be utilized for extracting bin spatial features and color histogram features from the given images. The selected features Viz. spatial, histogram and HOG will be extracted and appended to the list returned by the function.

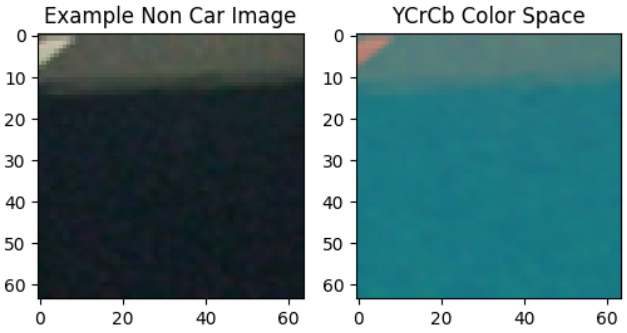
Different color spaces along with orientations, pix_per_cell, cell_per_block, and hog_channel were tried before deciding the final selection. Initially I selected

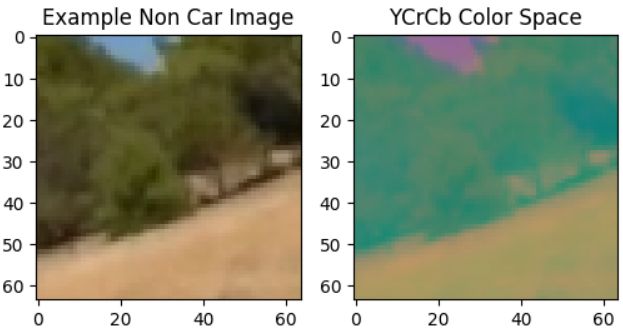
Histogram, spatial and HOG features and was getting a very low accuracy on the svm classifier (around 20%). The classifier took more than 20 hours for producing the output. I thought, selecting all the options like color, Histogram and HOG was the reason for delay in training the classifier. Next I went ahead with only HOG features. The accuracy was still very low (around 20%) and there was no improvement in the training time. I went through the code once again and found a bug in the extract_features function. I did a mistake in feeding the list of images to the function. This was causing the low accuracy and long training times. I selected all (spatial, color and HOG) options for training the classifier. This increased the overall training time but was giving a good accuracy of 99%. Initially I selected channels like RGB, HLS and HSV. Accuracy was low. YCrCb was giving high accuracy. When I trained on only single YCrCb channel, accuracy was low. Including all the channels was giving good accuracy. I experimented with various orientations and 8 was giving better results. Like other attributes, I used hit and trail method for pix_per_cell and cell_per_block also. A value of 8 and 2 for them respectively was giving good accuracy. The moment I achieved 99.16% accuracy on the classifier, I freezed the values and dint change them further.

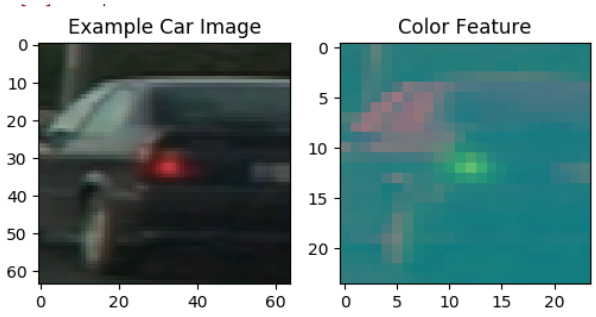


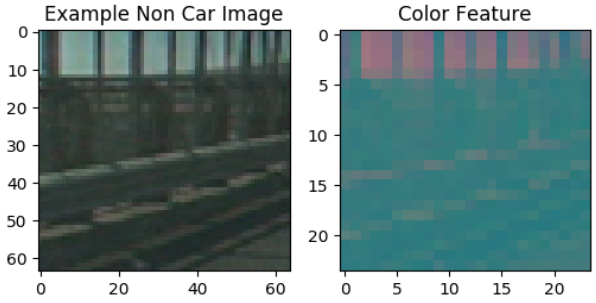

Example Car Image
YCrCb Color Space

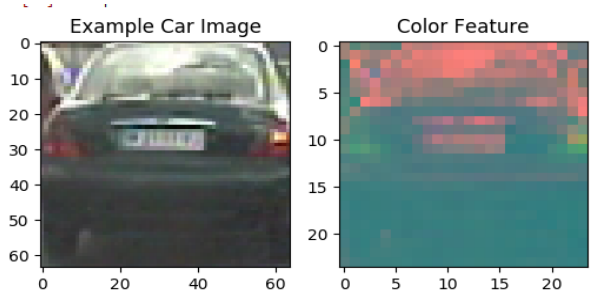

Example Car Image
YCrCb Color Space

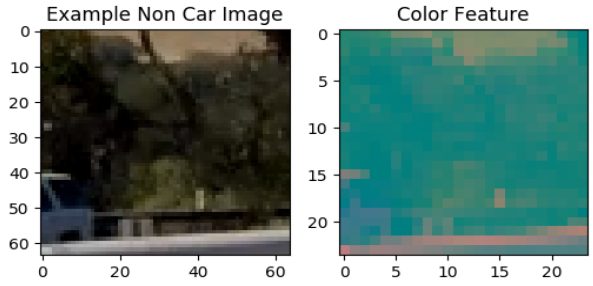

Example Non Car Image
YCrCb Color Space


Example Non Car Image
YCrCb Color Space


Example Car Image
Color Feature


Example Non Car Image
Color Feature


Example Car Image
Color Feature

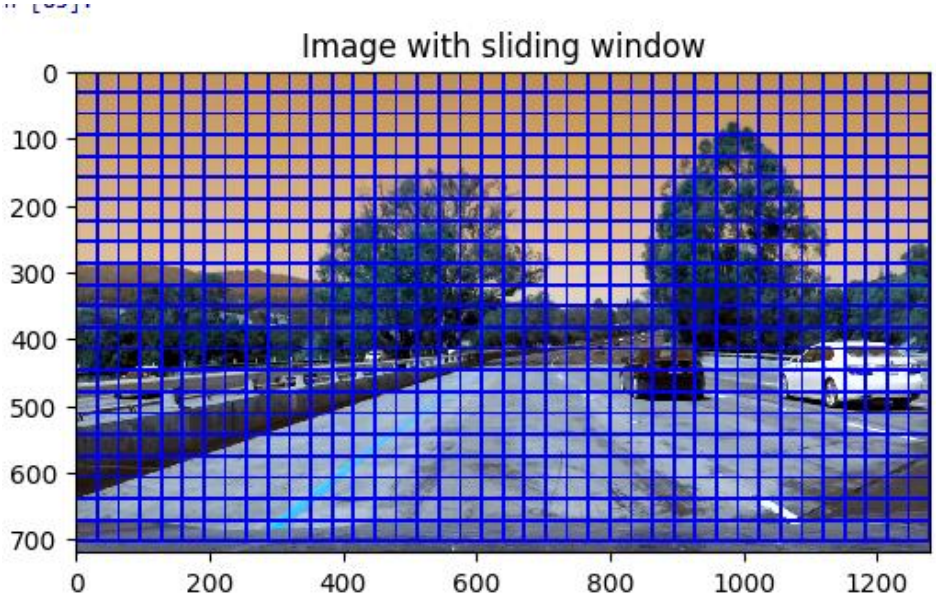

Example Non Car Image
Color Feature

Training the classifier

I trained a linear SVM using the spatial, Histogram and HOG features extracted from the car and non-car images. Before training, the data was scaled using standardscaler function. The function was fit to the features extracted from the image for standardizing it, so that any single feature will not have any undue importance while classifying the images. The code for the same is included in the main_vehicle_detection.py file from lines 123 to 168. A vector of labels for car and non-car images was created and stacked. A seed of 7 was selected for randomly splitting the data and reproducing the same results while tuning the parameters. The data was split into train and test in the ratio of 80 and 20 respectively. The classifier trained was tested on test set. The default settings were giving an accuracy of 99.16%., I decided not to further tune it, which may result in overfitting.

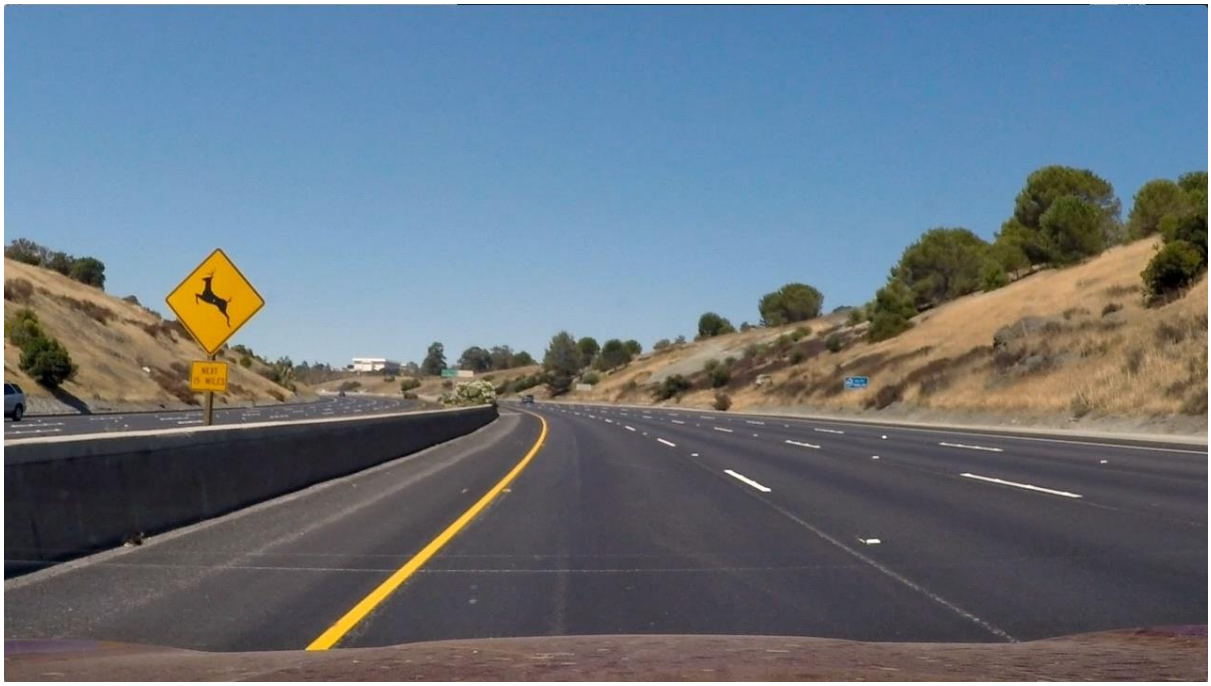
Sliding Window Search

The code for sliding window search with overlapping tiles was included in the required_functions.py file from lines 135 to 177 (slide_windows( )). The function takes an image, x and y start stop position, size of the sliding window and overlap required as inputs. From the given inputs it will compute the span of region to be searched. Depending on overlap it will calculate the number of pixels per step and number of windows in both x and y direction. Looping through the number of windows in both x and y directions it will return a list of windows with x ,y co-ordinates for start and end position. I choosed sliding window method because it was easy to understand and I was able to comprehend how the output will be for x , y start and stop positions, scale and overlap. This made my task of tuning the scale and overlap easy and quick.



I experimented with various scale and overlap positions and tested them on the some images before selection. Selecting a scale of 32 is giving good results but it is increasing the processing time on the final pipeline. Selecting a scale of 256 is reducing the processing time by 4 times but the results are not encouraging. So, I decided to go ahead with three scales. 64,128 and 256 which was giving good results with average processing time. Using a single scale on the final pipeline was also not giving good results. So, I included three of them to achieve better performance of the pipeline on the video stream and to avoid false positives by thresholding. Different overlap values were also tried out. A overlap of 0.8 was giving good results.

Ultimately, I searched on three scales using YCrCb 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

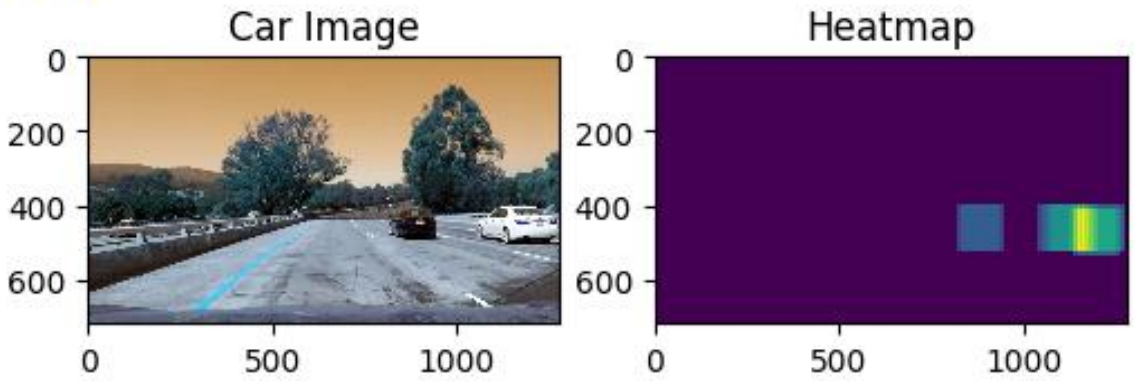
The sliding window search plus classifier has been used to search and identify the vehicles in the videos provided. The code for the same is included in the main_vehicle_detection.py file from lines 178 to 249.

The code for detecting the window positions (lines 243 to 275) , creating a heatmap (lines 278 to 289) and thresholding the heatmap (lines 292 to 297) is included in the required_functions.py file and implemented in the final_pipeline function in the main_vehicle_detection.py file.

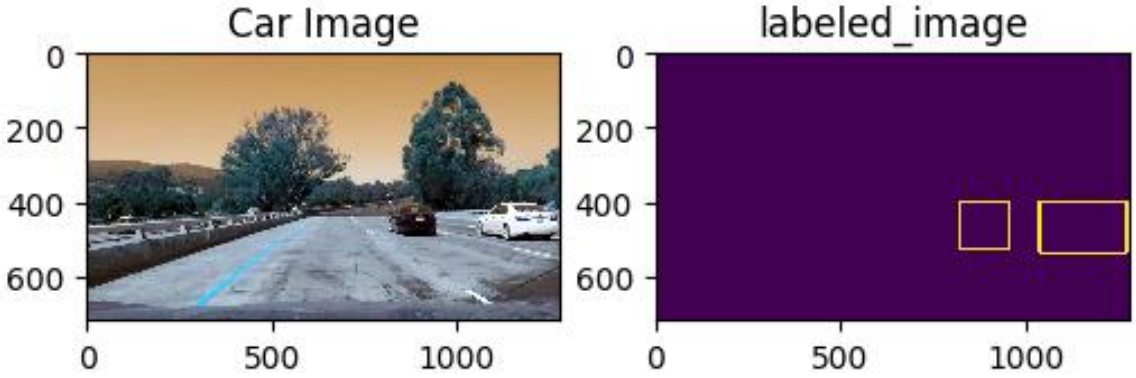
I searched for the window position where the classifier predicted the presence of a vehicle and saved its co-ordinates. A heatmap was created with positive detections and a threshold was applied to avoid the false positives. Then scipy.ndimage.measurements.label( ) was used to identify the detected portions in the heatmap in the final pipeline. Function to draw the labeled boxes on the detected portion was included in the required_functions.py file from lines 300 to 314. Drawing of bounding boxes on the detected portions was implemented in the final pipeline for processing the video stream. Thresholded heatmap and labeled images are applied to screenshots of the project_video and are saved in the output images folder.

Thresholded Heatmap and Labeled Images

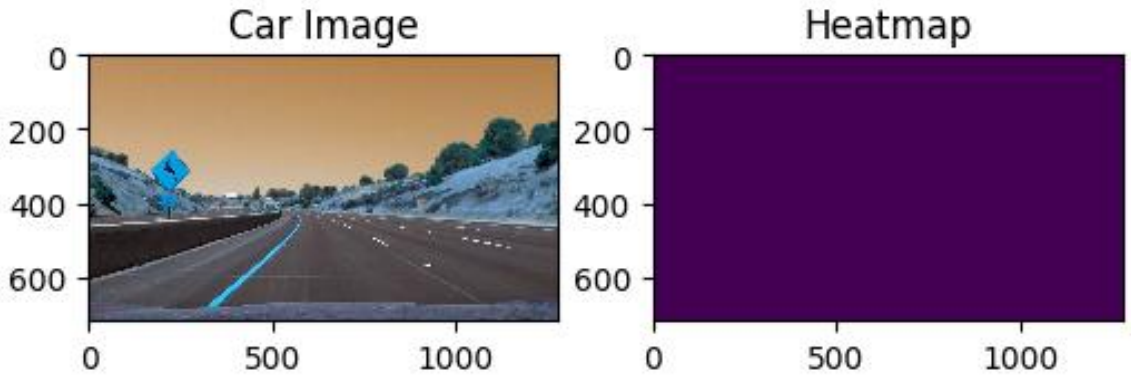
In [86]:



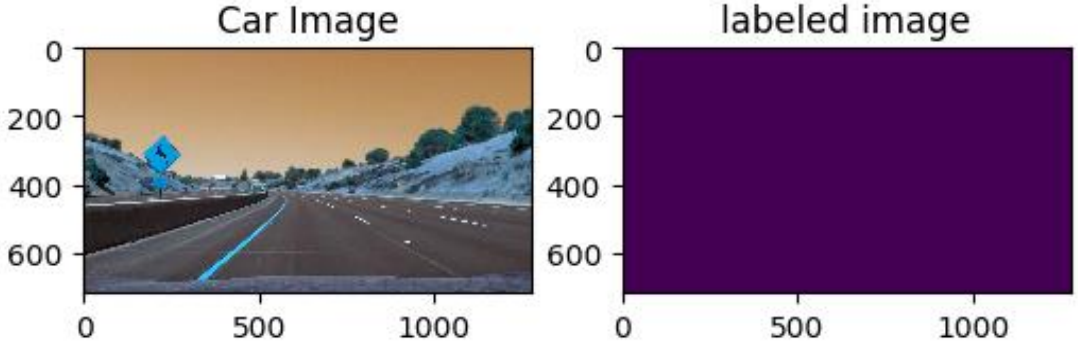
In [105]:



n



In [111]:



Final Output

Discussion

I faced problems in implementing the HOG subsampling method. To avoid false positives on the left side of image, I decided to restrict the search only to the right side of the image. The function I implemented was unable to restrict the search only to the right side. Even when I ignored the left side of the image in my function, false positives were generated in that region. I tried to tune the scale and cells_per_step parameters to avoid the false positives but was not completely successful. There was some bug in my function which I could not eliminate. So, I finally decided to use the sliding window search method which is slow to process but produces better results. Implementing the HOG subsampling method will improve the speed and provide quicker results.

Instead of traditional computer vision techniques, implementation of deep learning might have provided better results. Implementation of deep learning on this project might provide much better insight regarding the advantages and disadvantages of computer vision Vs deep learning.

The other problem I faced is moviepy got crashing repeatedly 'OS error'. I downgraded python from 3.6 to 3.5 which dint solve the problem. Pickling saved me some time here.

This pipeline might fail if implemented on some cars with fancy shapes and colors or images where the surroundings look very different from the non-car images. Including more training images with different car and non-car images might make the model more robust.

If the environment conditions change (lighting, weather) or the surroundings in which car is being driven change (bridge) it may confuse the model. These are some of the challenges which can be tackled in future versions.

The processing time of this pipeline is very high. In fact to increase the processing time I experimented with the color_space as 'YUV', HOG orientations 11, pix_per_cell 16, cell_per_block 2, ALL HOG channels without spatial and histogram features. The accuracy of the model was 98% and the processing time was reduced to 20 minutes. The output was neither very good nor bad. The algorithm is not suitable for real time implementation because of slow processing time which can be further worked upon.