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.

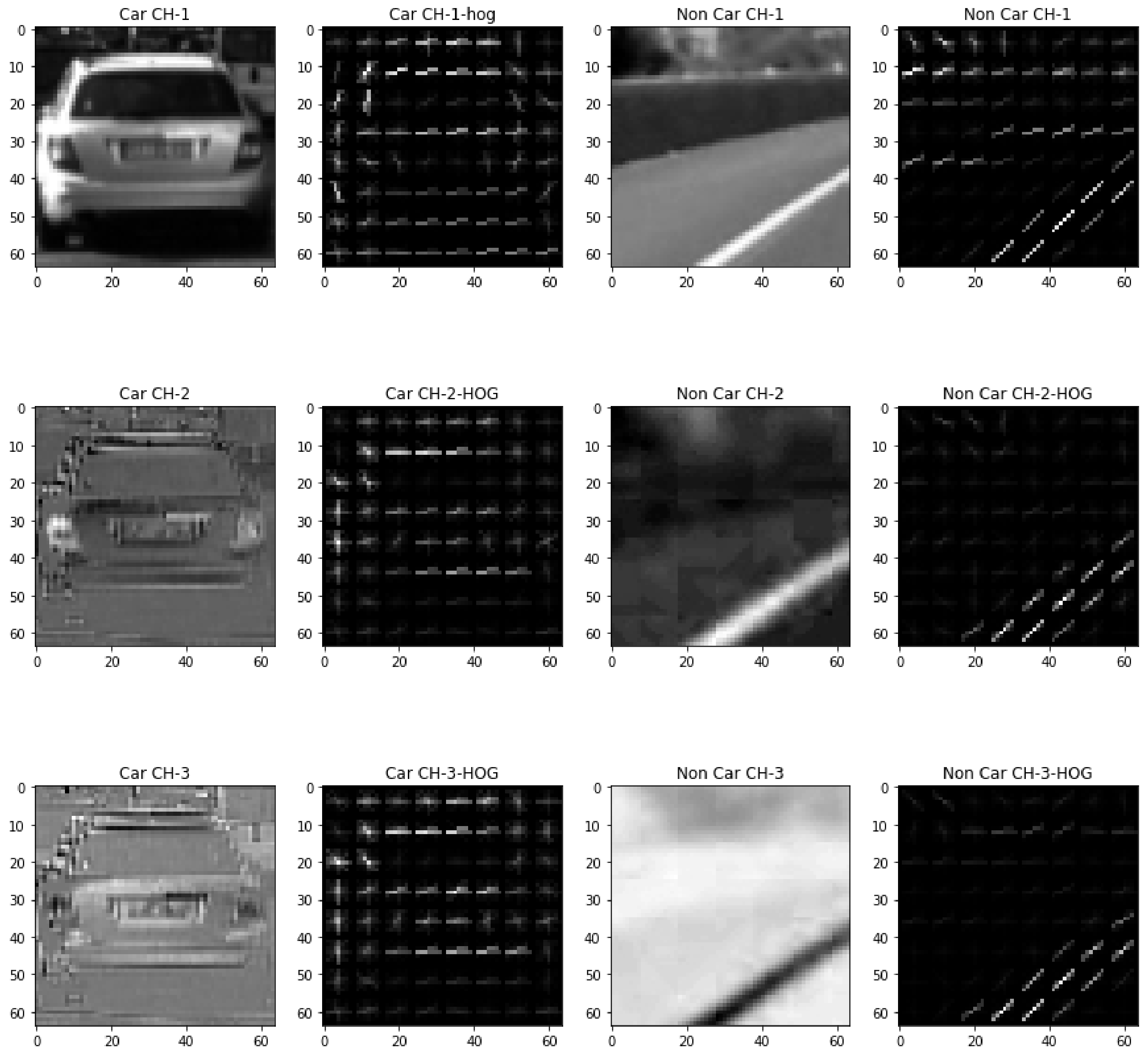
Rubric Points

1. Histogram of Oriented Gradients (HOG)

a. 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 second code cell of the IPython notebook. I started by reading in all the vehicle and non-vehicle images given in the **KITTI** and **GTI** image datasets.
I then explored different color spaces (line HLS, HSV, YUV, YCbCr) and different skimage.feature.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.feature.hog() output looks like.

Here is an example using the YCrCb color space and HOG parameters of orientations=12, pixels_per_cell=(8, 8) and cells_per_block=(2, 2):



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

My final set of **HOG parameters** were:

ORIENTATIONS	PIXELS PER CELL	CELLS PER BLOCK
12	16	4

First I started with initial values of *Orientations*: 9, PPC (*Pixels per cell*): 8, CPB (*Cells per Block*): 2, these parameters were quite fine but gave a lit bit of false positives. Then I increased the number of orientations to 11. However increase in the orientations didn't help the cause, but in turn increased the training time since the features vector, taking into account the spatial binning and histogram features went in range of 11k.
Next I increased the CPB (*Cells per Block*) from 8 to 16. This reduced the HOG feature vector considerably, and also reduced the training time, keeping the accuracy same. Since the training time reduced so I chose to stick to the current values of *Orientations*: 11 and *Cells per Block*: **16**.
Next I tuned CPB (*Cells per Block*) parameter to **4**, since in sliding windows I needed to detect the vehicles more times in order to make the heat-map correctly. Using this the false positives reduced considerably and also the vehicles were detected correctly. One more advantage was that, using this configuration I trained the model in approx. 2mins. Therefore, as the training time was less, I chose to increase the *Orientations* parameter to **12**, which in turn gave me better results.

c. 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 this project first I started with LinearSVC. But later while fine tuning I moved on to SVC of ***sklearn.svm*** with its default values of 'rbf' kernel and C and gamma. The training of the SVC can be found in 9th code block of IPython Notebook in **train_SVC()** function. This function is the main function that trains the Support Vector Classifier and saves it in a pickle file.

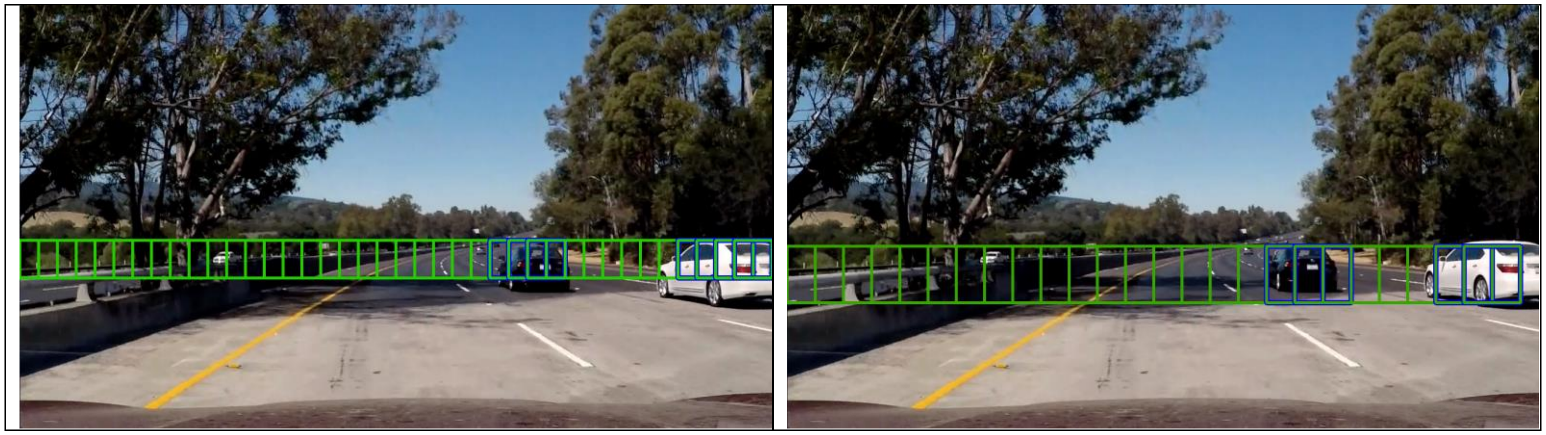
i. First it takes all the car and non-car images from **KITTI** and **GTI** datasets.
ii. Then it extracts the features from the datasets. After that it randomly splits them it test and training datasets. The ratio of this split is 75% as training and 25% as test dataset.
iii. The features are also normalized after splitting, and the scaler is fit to only training data, but the same scaler is used to normalize both training and test dataset.

iv. After that it trains the SVC and also calculates its accuracy using the test dataset.
v. When accuracy calculation is done, it saves the SVC and scaler in 'svc.pickle' file for future use.

2. Sliding Window Search

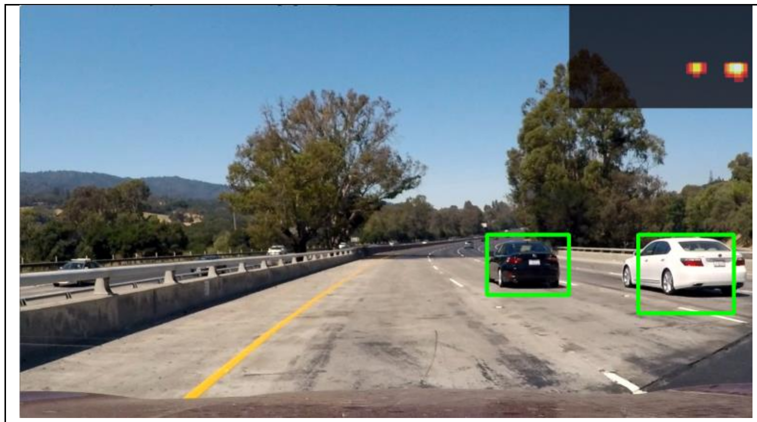
a. Describe how (and identify where in your code) you implemented a sliding window search. How did you decide what scales to search and how much to overlap windows?

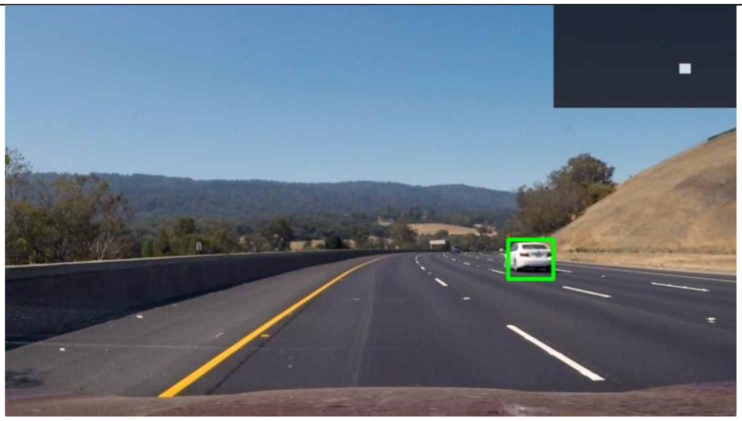
The Sliding window search is implemented inside the function ***find_cars().*** This is the main function of the pipeline. It takes arguments such as *image*, *starting y-coordinate*, *stopping y-coordinate*, *scaling factor*, *svc*, *scaler*, *orientations*, *pixels per cell*, *cells per block* etc. and returns the an image with a rectangle drawn around the detected car, along with the *bounding box coordinates.*
The ystart and ystop are the y-coordinates inside which the sliding window search happens. The height of the window and the scaling factor is calculated such that the height of the search area times the scaling equals the height of the window, and therefore only horizontal sliding is required. Initially I was proceeding with the nested for-loops as given in the classroom, but later found out that if I do 2-3 horizontal window searches with different window sizes, better results were achieved. Given below are some images describing different sliding window searches.

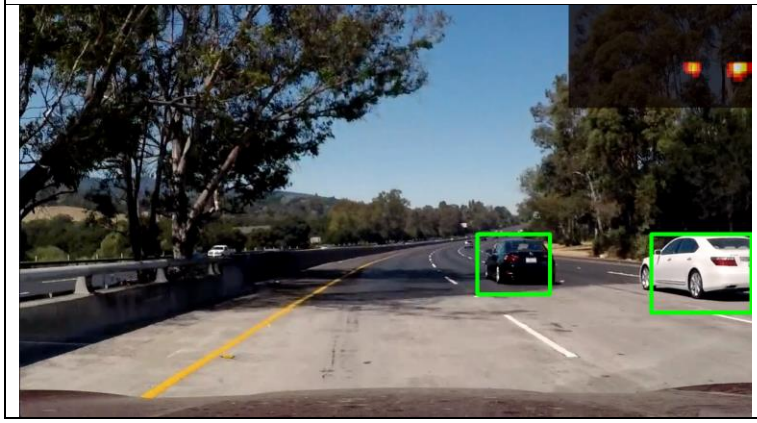

As visible, the blue rectangles overlapping the green ones, have detected cars.

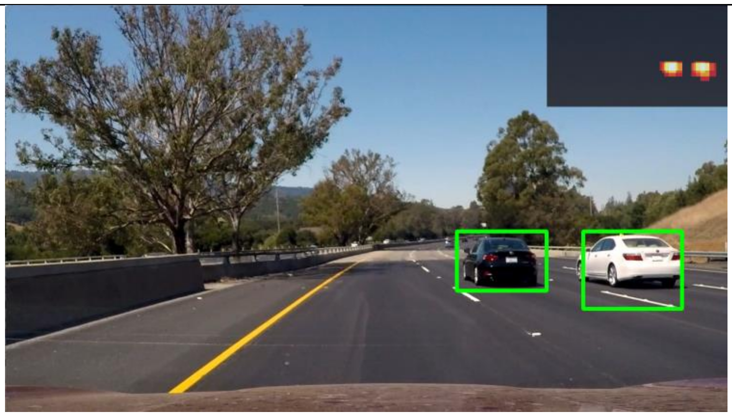
b. Show some examples of test images to demonstrate how your pipeline is working. What did you do to optimize the performance of your classifier?

Initially I started with **RGB** images. But that didn't produce enough HOG features for SVC as the **YCrCb** channel. Additional to this I also used spatial binning and histogram of images to train the classifier. Some examples of the images from pipeline are









3. Video Implementation

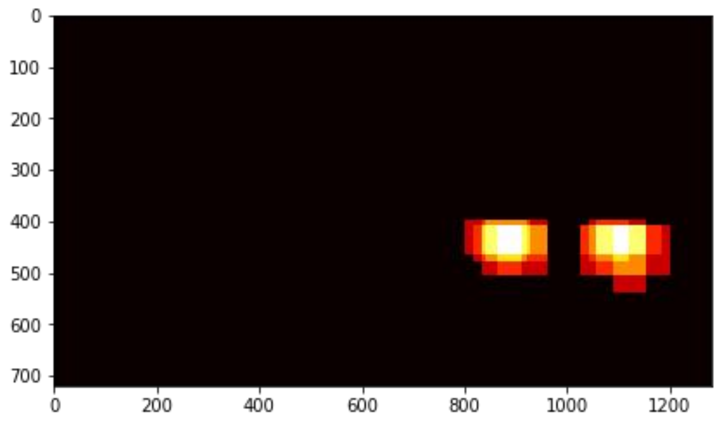
a. **Provide a link to your final video output. Your pipeline should perform reasonably well on the entire project video (somewhat wobbly or unstable bounding boxes are ok as long as you are identifying the vehicles most of the time with minimal false positives.)**

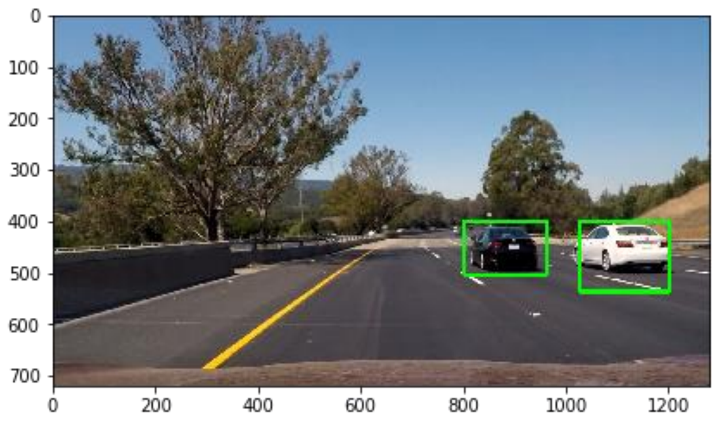
   The output video is located at output_videos/project_video.mp4

b. **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.**

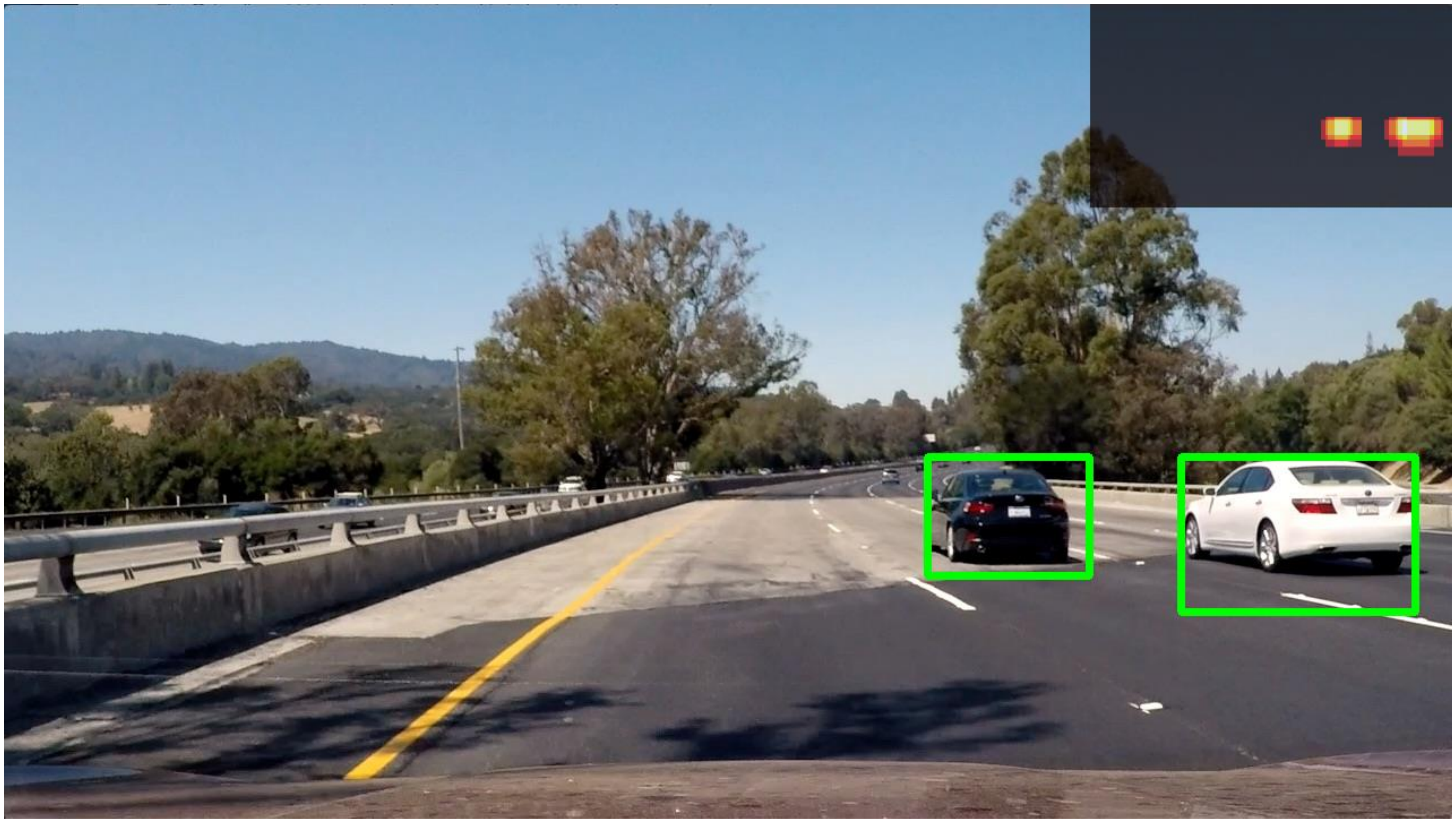
   I recorded the positions of positive detections in each frame of the video. From the positive detections I created a heatmap and then thresholded that map (by a small value of 2) to identify vehicle positions and appended it to a *deque* HEAT_MAP_HISTORY. I then used ***scipy.ndimage.measurements.label()*** to identify the final individual blobs in the heatmap by thresholded it with the length of deque. The deque is a implementation of the history of frames where the car was detected previously. This prevented the flickering/ fluctuating of the boxes around the cars and made the bounding boxes smoothly contain the car(s) This is implemented **pipeline()** function, after receiving the results from **find_cars()** function.

   An example an image with heat map:





Here is a final image:



4. Discussion

a. 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?

Initial problem that was faced while working on this project was choosing of the wrong **color channel** (**RGB**). Although the histogram features, spatial binning features and HOG features were extracted but it didn't seem to train the SVC enough to classify correctly. Moreover the HOG features from 3 channels of RGB was also making the **feature vector** very large. But as I switched to **YCbCr** color channel, and took HOG feature for only **Y** channel, the performance was better.

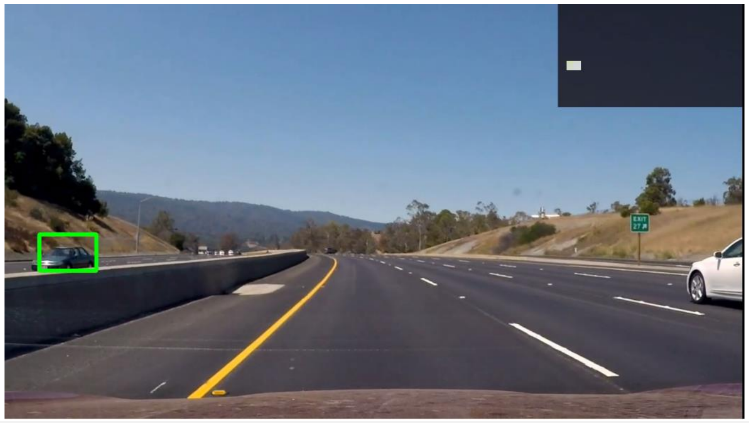
Next problem was faced choosing the **correct parameters for HOG features**. Increasing the orientations, reducing the pixels per cell increased the size of HOG features array. Therefore a correct combination of parameters had to be determined by fine tuning to get the best results along with optimal training time. It was observed that when HOG feature vectors were expanding to range of 11k to 14k, around 2GB of memory was consumed and training time was around 9mins which was not at all required. Therefore a correct combination of parameters was chosen to reduce memory as well as training time.

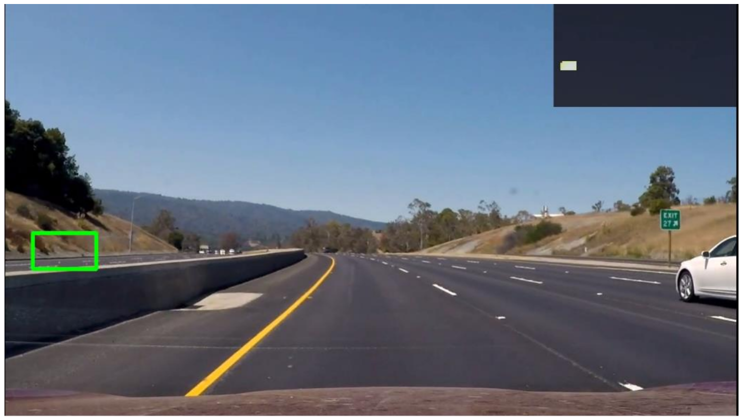
The **choice of kernel** of the Support Vector Classifier was also a small decisive factor. Initially I started with LinearSVC but later moved to SVC from ***sklearn.svm*** because I felt that my pipeline was detecting the cars well in SVC (non-linear) which had default value of kernel as 'rbf'

The **training dataset** also played a major role in improving the performance of the pipeline. Initially I started with the small dataset of vehicles and non-vehicles which Udacity provided in classroom. But it resulted in detection of many false positives. Then upon my mentor's advice I used both **KITTI** and **GTI** datasets for training of vehicles and non-vehicles and that although increased the training time, but improved the pipeline's performance a lot.

The final major challenge was to **stabilize the position of the bounding box** determined by the heatmap plotted from the prediction of classifier. Initially the bounding box was fluctuating/flickering around the cars since prediction of different windows (in sliding windows) varies in between frames. To stabilize the position I chose to maintain a history of the positions where the vehicles were detected for the last 10 frames, and then with the help of thresholding **weed out false positives**. This technique was implement by using **deque** from collections library of python.

One **drawback** is, the pipeline detects cars from the other lane but due to 'maintaining history' techniques the bounding box stays even if the car has vanished from the frame. The following images demonstrate:





Car detected initially

Bounding box stays even if the car has moved out of frame