Vehicle Tracking and Detection

The goal of this project is to develop a pipeline to detect cars in the images and videos. In order to detect and track the cars, it is required to feed images to a classifier that detect if a car is present in the image or not. Usually, an image can contain many cars. To detect every car in the image/frame, it is required to first train binary classifier on fixed image sizes containing cars and non-cars. Then, the window is slid across the parts of image to detect the car in the region masked by window using the trained classifier. The mentioned steps are listed below:

1. Read the labeled dataset provided by Udacity for vehicles and non-vehicles images
2. Divide dataset into train and test set
3. Extract features from dataset – hog, spatial bining and color histograms
4. Train a classifier – Decision tree, SVM etc
5. Check the classifier accuracy. If possible perform cross validation to tune the parameters
6. Create a window sliding function.
7. Test the classifier along with window slider on provided test images
8. Implement filters to minimize false positives and aggregate multiple predictions
9. Create the function to implement the developed pipeline on video

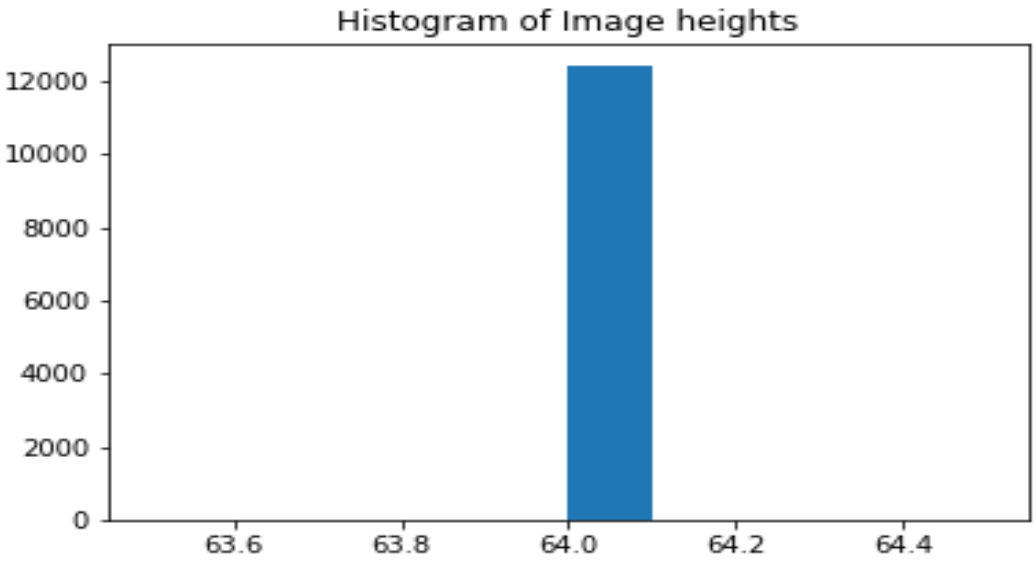
Having developed the pipeline, I would now address the rubrics points required for the project review.

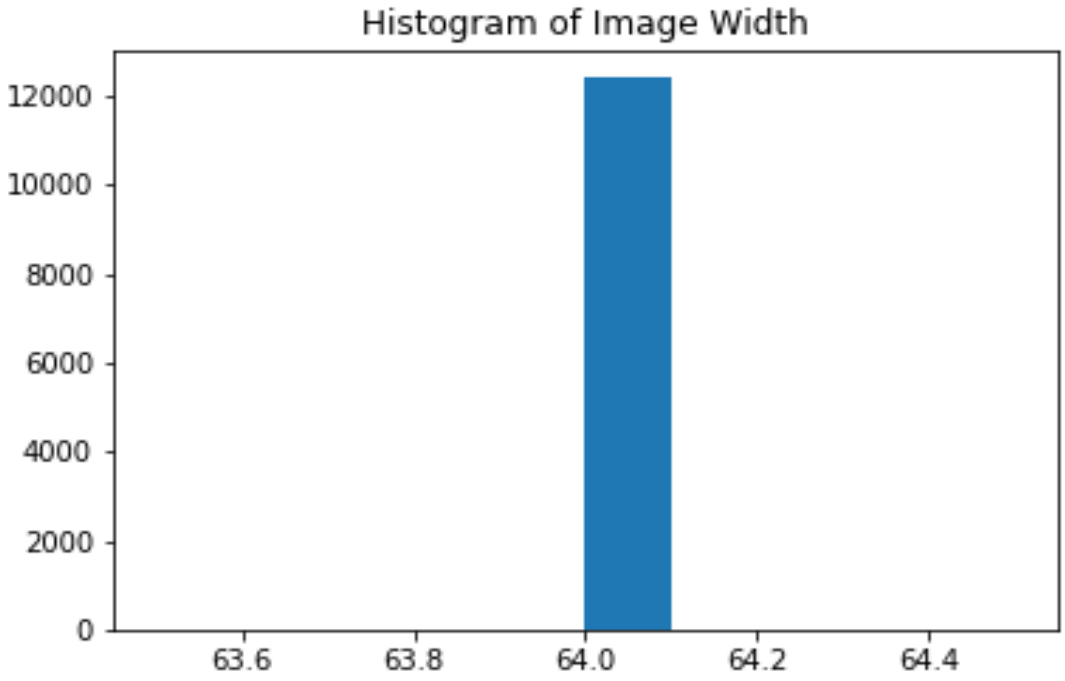
Histogram of Oriented Gradients (HOG)

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

   To begin with, I created two arrays containing the names of vehicle and non-vehicles images. Then, I split the data into train and test dataset. This part of the code containing the data ingestion and split is contained between the cells 1-11 in jupyter notebook. Then, I plotted two random images of car and two of non-car.

Having collated the data into training and test set, I started looking at the distribution of image shapes in the dataset. For this, I plotted the histogram of width and height of images.



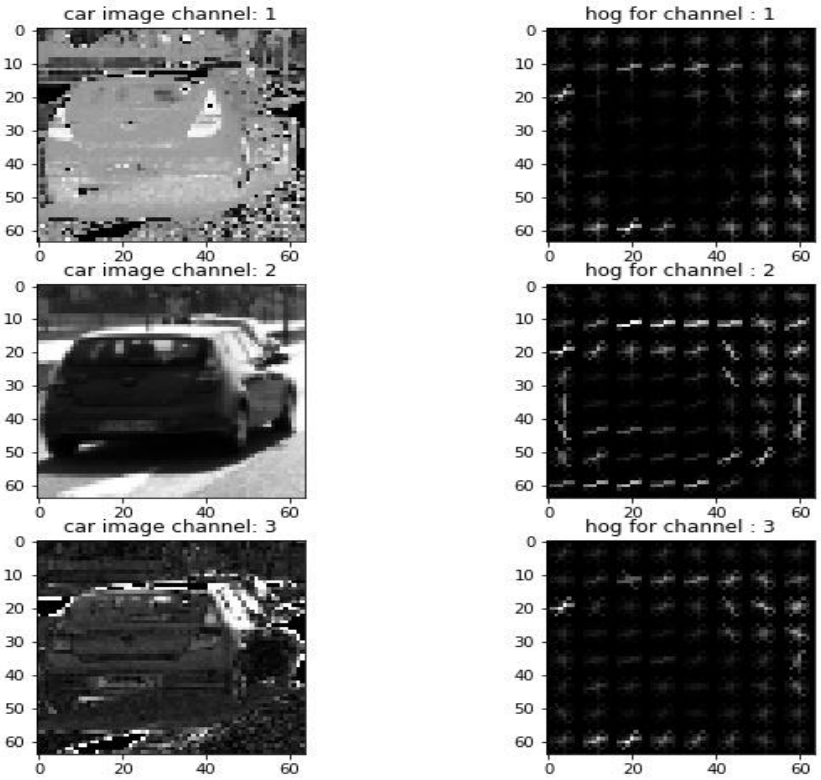


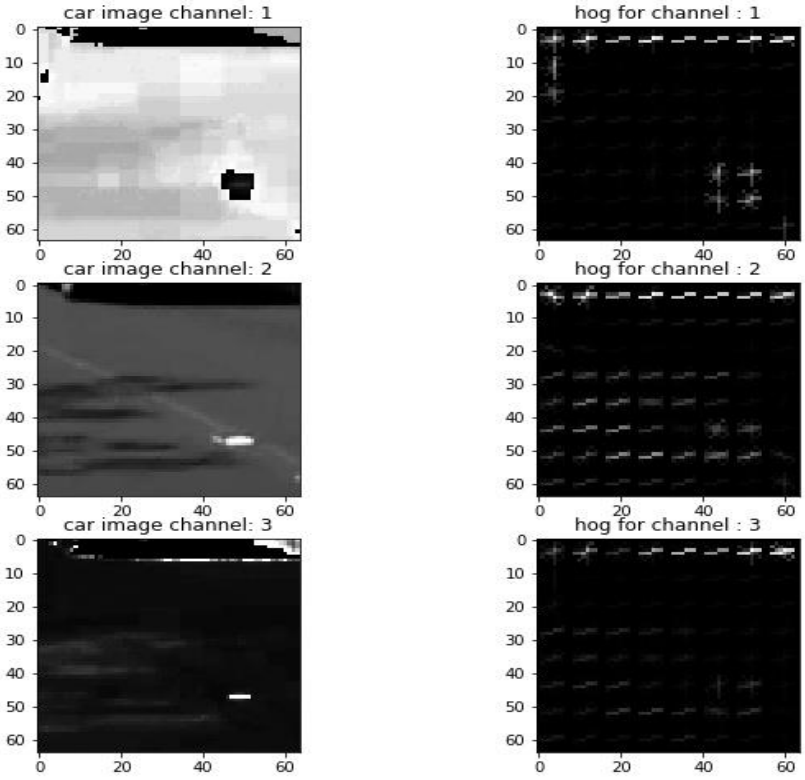
For the histograms, it was evident that all the images were of size 64x64. This helped me to realize that the pixels per cell used for Hog features should not be too high; else, It would be really difficult to extract the meaningful features for separating cars from non-cars. So, I chose pixels_per_cell to be 8x8 and cells_per_block to be 2x2. In addition, I chose the orientation to

be 9. This combination of parameters for hog features yielded 1764 hog features for each image. The part of the code that computes hog features is contained in the ***get_hog_features()*** function. Inside the function, I used skimage.hog() function.

For choosing the color space to work with, I looked at car and non-car images in different color spaces using 3d scatter plots for each channel in that color space. I found out that HLS color space provided enough variance in the each channel for car images versus non car images that it would be possible to use that color space. So, I finalized the HLS color space.

Following are the few examples of images and their corresponding hog features generated:





2. Explain how you settled on final choice of hog parameters?

   As explained before, I chose pixels_per_cell and cells_per_block based on the size of the images that were available. I finalized the pixels_per_cell as (8,8) and cells_per_block as (2,2). In addition, I also selected the orientations to be 9. But later on, I validated my assumptions by running the pipeline using pixels_per_cell to be (16,16) and orientations as 9 and 12. Still, I found the initial estimate of parameters to best for the project video. When I used 16 as pixels_per_cell, I found that pipeline missed cars more often than when 8 was used. For orientations, I didn't find difference with 9 and 12 in the pipeline and hence, I stuck with 9 to limit the number of features.

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).

   For this dataset, I trained a random forest classifier as it took less time to train and experiment, and lesser time to predict with almost similar accuracy as SVM. I used hog features along with spatial bining of size (32,32) and color histograms. There was one

more benefit of using RandomForest – it provides an oob_score that can be easily used to tune the hyperparameters, instead of using cross validation.

I created a function extract_features() that takes image data as input and returns extracted features for each image in the data.

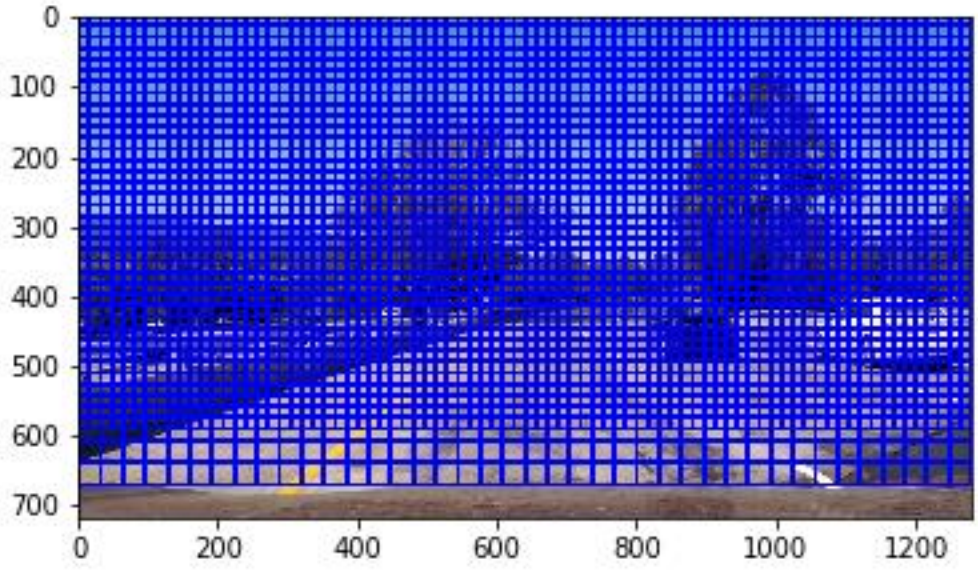
The code for extracting and training the classifier is provided under the section Classifier(Tuning and training) in the jupyter notebook.

The tuned classifier provided the test accuracy of 0.9878, which was excellent.

Sliding Window Search

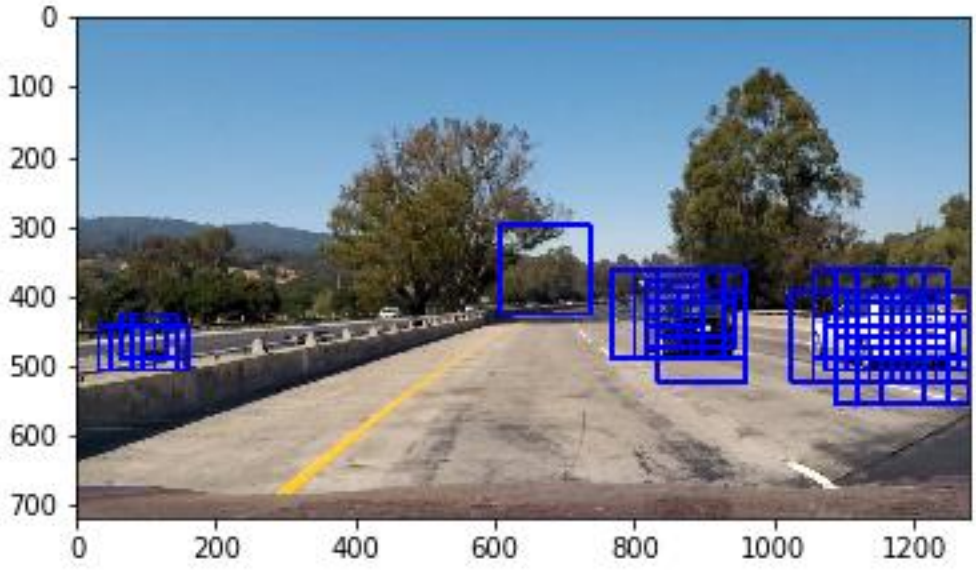
4. Describe how (and identify where in your code) you implemented a sliding window search. How did you decide what scales to search and how much to overlap windows?

For car search across entire image, I deployed a siding window search with window size of 64 at two different scales of 1 and 2. Instead of scaling up the window, I scaled down the image to be searched over and then, later scaled up the size of the bounding box identified. This was accomplished in the search_multi() function in the notebook. The function takes a list of dictionary as input along with other inputs like classifier , scaler etc. An item of the mentioned dictionary contained: {‘scale’,’y_start’,’y_stop’} parameters for each window.

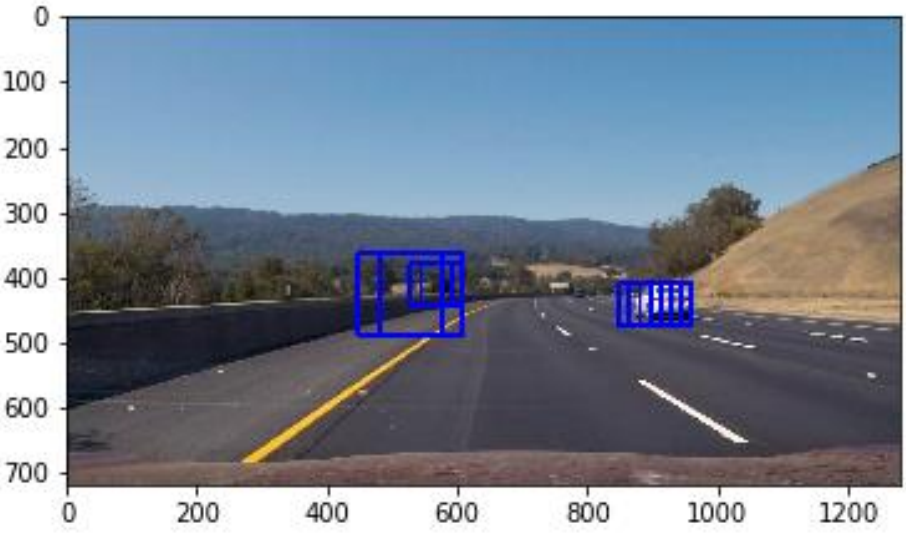


5. Show some examples of test images to demonstrate how your pipeline is working. What did you do to optimize the performance of your classifier?

As mentioned above, I used two scales for the window and the color channel of HLS for the images. The feature vector for image consisted of hog features, spatial bining and color histograms. This pipeline created good results as shown in the images below:



Example 1



Example 2

Video Implementation

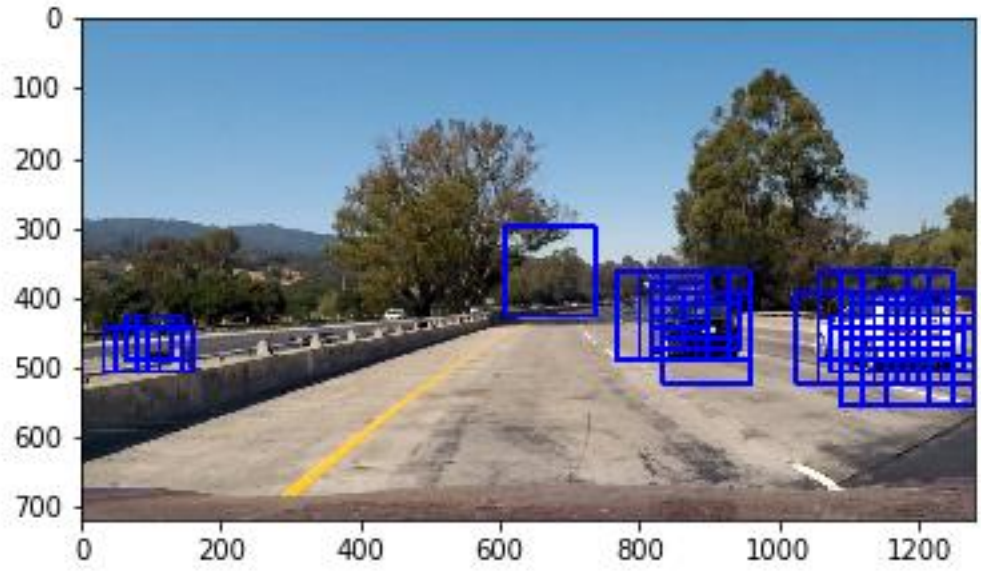
6. Provide a link to your final video output. Your pipeline should perform reasonably well on the entire project video (somewhat wobbly or unstable bounding boxes are ok as long as you are identifying the vehicles most of the time with minimal false positives.)

The video is available at the link: https://github.com/rishubhkhurana/SelfDrivingCar--VehicleDetection/blob/master/project_video_out.mp4

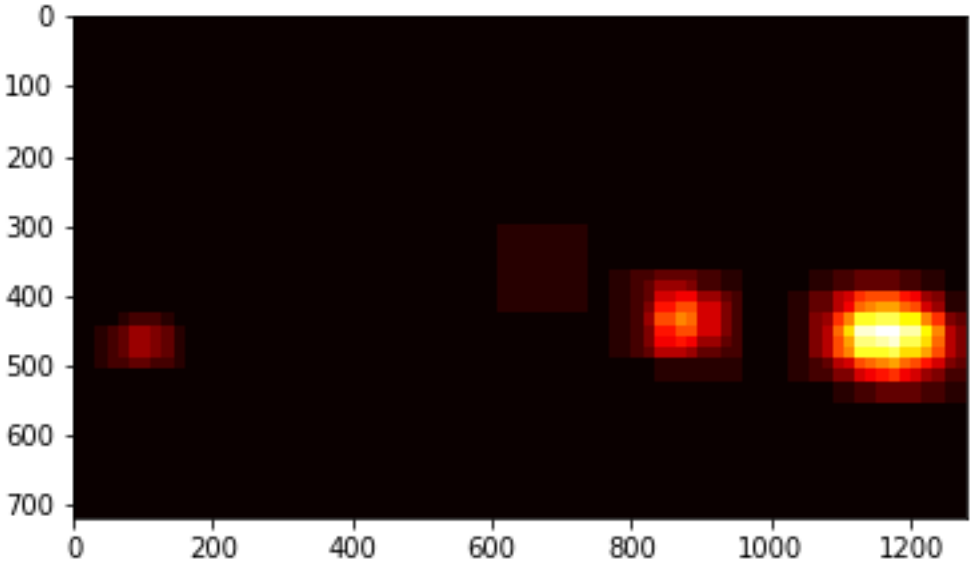
7. Describe how (and identify where in your code) you implemented some kind of filter for false positives and some method for combining overlapping bounding boxes.

For filtering the false positives and the multiple detections, I used heatmap followed by thresholding to remove the false positives. In order to identify the correct bounding boxes, I used the labels function of `scipy.ndimage.measurements.label()` to identify the concentrated area of detections. I then used xmin,xmax,ymin and ymax to generate the boundary box around each identified concentrated area.

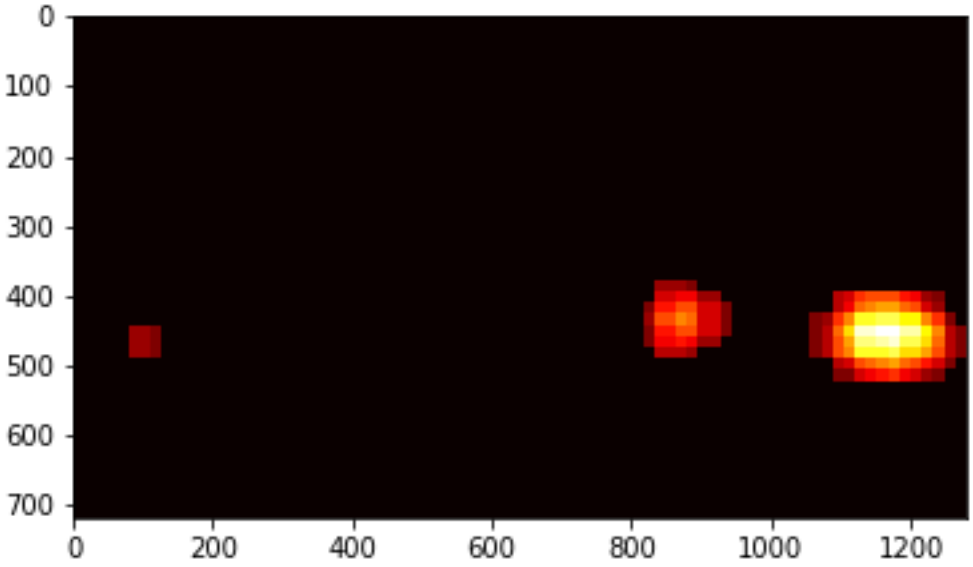
The procedure followed step by step could be seen below:



Raw predictions on test image 1



Heatmap of predictions



Heatmap after thresholding

Final output image

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?

   Now I will briefly discuss the steps I took to complete this project , issues I faced and the potential for improvements.

   Summary--

   In this project, I implemented an object(car) detection pipeline. Following are the list of techniques I used:

   1. Used hog features, color histograms and spatial binning as features for each image to be classified
   2. Used RandomForest classifier for the classification of car vs non car

3. Deployed a sliding window with size of 64x64 pixels. Used two scales for window—1 and 2. Instead of scaling up the window, I scaled down the image and then scaled up the potential bounding box predictions
4. Searched over only partial image for windows. There was no need to search in sky for the cars. So, I instead used (y_start,Y-stop) as (300,600) for scale 1 windows and (300,700) for scale 2 windows.
5. I then utilized heatmap with the threshold of 3 to filter out unwanted detections.
6. Finally, labels function from scipy.ndimage.measurements was used to detect the potential area of concentrated detections. The min, max of x and y of these regions were then used to predict the bounding boxes.

Issues faced:

1. The time detect cars on each image seemed pretty high and prohibited lot of experimentation.
2. Ideal way to detect cars using windows with different scales was to have different classifier for each type of scaling. This was not possible with single classifier that was trained on fixed length of feature set. But feature set size varies with each scaling window. So, in order to utilize the same classifier, I had to scale down the image instead of using windows of different size. Of course, this point is invalid if we extract the hog features for each subset of images selected by window. But the time required by hog extraction was too high to extract hog features again and again. Hence, hog features was extracted once and then subset of those features were selected based on the size of the window. Due to this, the feature vector length kept changing with different size of windows.
3. To generate the output video, I had to wait for 4 hours as the pipeline took almost that much time. This was the major issue and downside with using this pipeline.

Improvements:

1. Although I selected color space of HLS based on visual inspection of few random images, I felt there should have been a more robust method to select the color space by exploring many more images.
2. The problem of detection and bounding box prediction ideally should be tackled using end to end deep learning. SSD and YOLO are some good algorithms that can be used to detect and accurately predict the bounding boxes.
3. In current pipeline, we have no way of accurately predicting the bouding boxes. This could be problem if this pipeline has to be used in real world scenario where not only need to detect cars but also to understand how far are they from our self driven car.
4. The time consuming aspect of this sliding window technique was quiet prohibitive. We could have utilized some selective search on image instead searching for entire image.
5. I used two scales of window for searching cars in image. Under ideal condition, it would be better to use different sizes of window with not just different scaling but with different aspect ratios as well.