Vehicles Detection and Tracking Project: Krishna

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.

Histogram of Oriented Gradients (HOG)

1. 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 code cells titled “Functions for training Features Extraction” and “Training Features Extraction” of the IPython notebook.

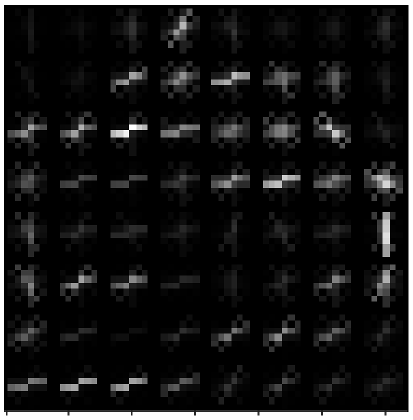
Initially, I started reading the vehicle and non-vehicle images provided by Udacity. I read the files using glob library. Then I read the images using matplotlib which reads in RGB format and in range of (0,1). I then converted the images into YCrCb format. I explored different image formats like HLS, HUV, LUV, YCrCb, etc. Out of these, YCrCb and YUV are the only formats which worked for me. So, I used the YCrCb format. Then for the HOG features, I tried on different values of the HOG parameters like pix_per_cell, orientations and cell_per_block. After trying with a lot of combination of different values for these parameters, I settled with these values:
orientations = 9
pixel_per_cel = (14,14)
cells_per_block = 3
Number of Channels = 3
Feature Vector Length obtained = 1164
Below are the demonstrated images:

Car

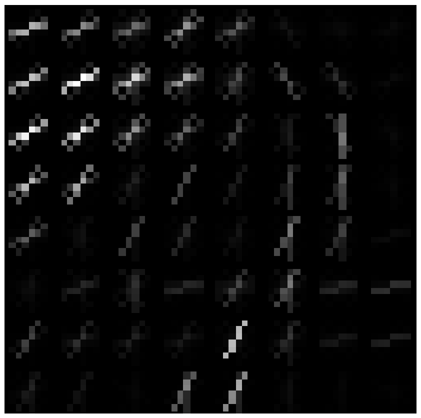
Not-Car

YCrCb format Car

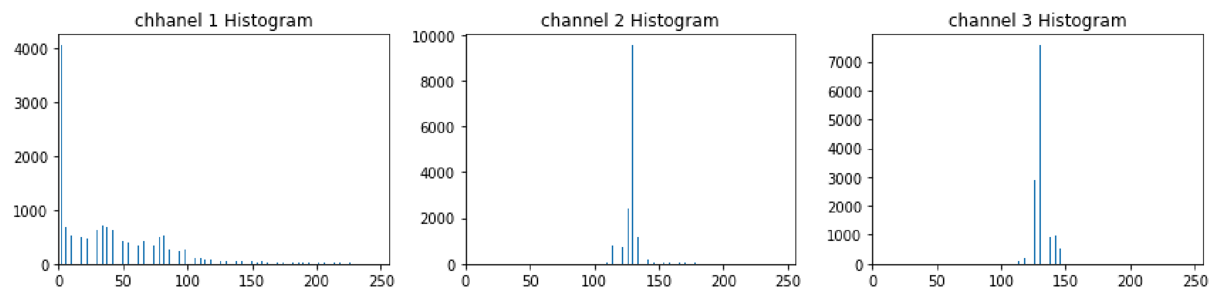
YCrCb format not-car



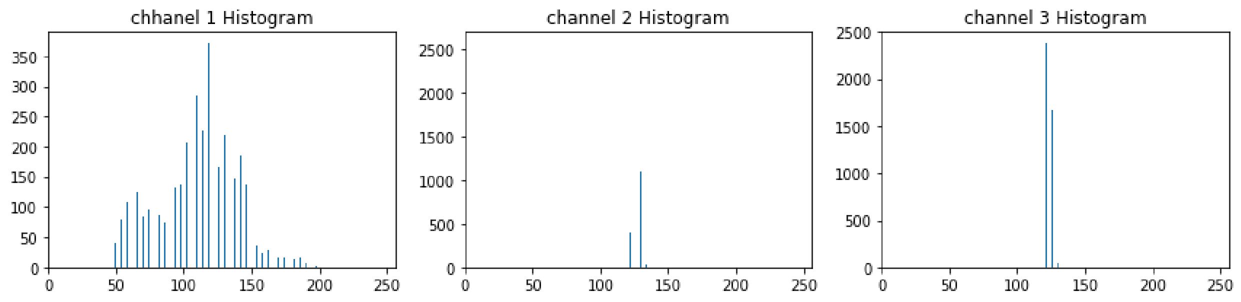
HOG Feature channel 1 for car



non-car images



Color Histogram for Car image



Color Histogram for Non-car image

Besides Hog features, I also used the Color Histogram features in my project on all the channels.
Number of histogram bins = 64

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

The final choice for the selection of my HOG parameters depended upon the following factors:

- Accuracy of the trained model
- Performance of the SVM classifier
- Performance of the classifier on the project video
- Speed of training and testing process on the project video

The details for the approach for the final choice was done on hit and trial basis as explained previously.

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).

Using the HOG and histogram features which I extracted above, I used SVM classifier with radial basis function as kernel and C parameter equal to 10. Before applying the SVM classifier I also normalized the features by scaling them to zero mean and unit variance.
Kernel = 'rbf'

C = 10.0

Initially I began with the LinearSVM, but it did not prove to be fruitful for me because, there were many false predictions. So, I moved on to non-linear SVM with rbf kernel.

This part is shown in the block titled "Train Classifier" of my IPYNB file.

By using this classifier, I achieved the following results:

Accuracy = 99.32%.
Time to train = 39.33s

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?

The section titled "Method to detect cars" contains the implementation of Sliding Window Search. I used the method named "Hog Sub Sampling" which was described in the Udacity Classroom materials. In this method, the features of the complete image are extracted only once and then these features of the complete image is then sub sampled according to the size of search window. Upon performing search on the window, the function returns a list of bounding box coordinates for rectangles for those regions where cars are detected. The example relevant images are given below.

Again, for deciding the scales and overlap factor, I used hit and trial method.

Combination of scales used = 1, 1.5, 2

For, my project, I kept the value of cells_per_step = 2.

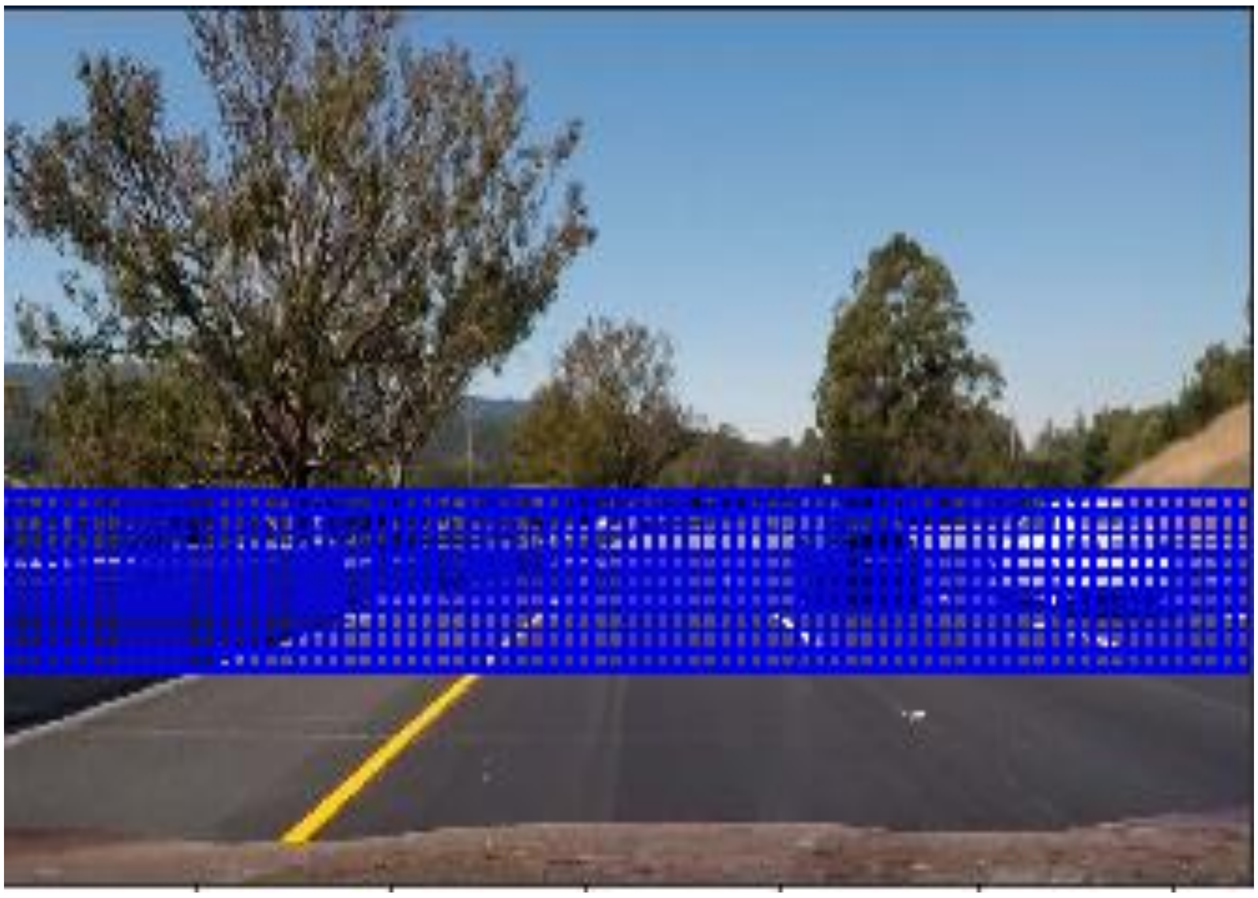
2. Show some examples of test images to demonstrate how your pipeline is working. What did you do to optimize the performance of your classifier?

My project pipeline calls the function find_cars for the image frames of the video. This function does the initial pre-processing of the image by converting it to YCrCb channel and then extracts the part of the image which is limited by the ystart and ystop. Ystart and ystop are the regions where there are chances of cars being there.

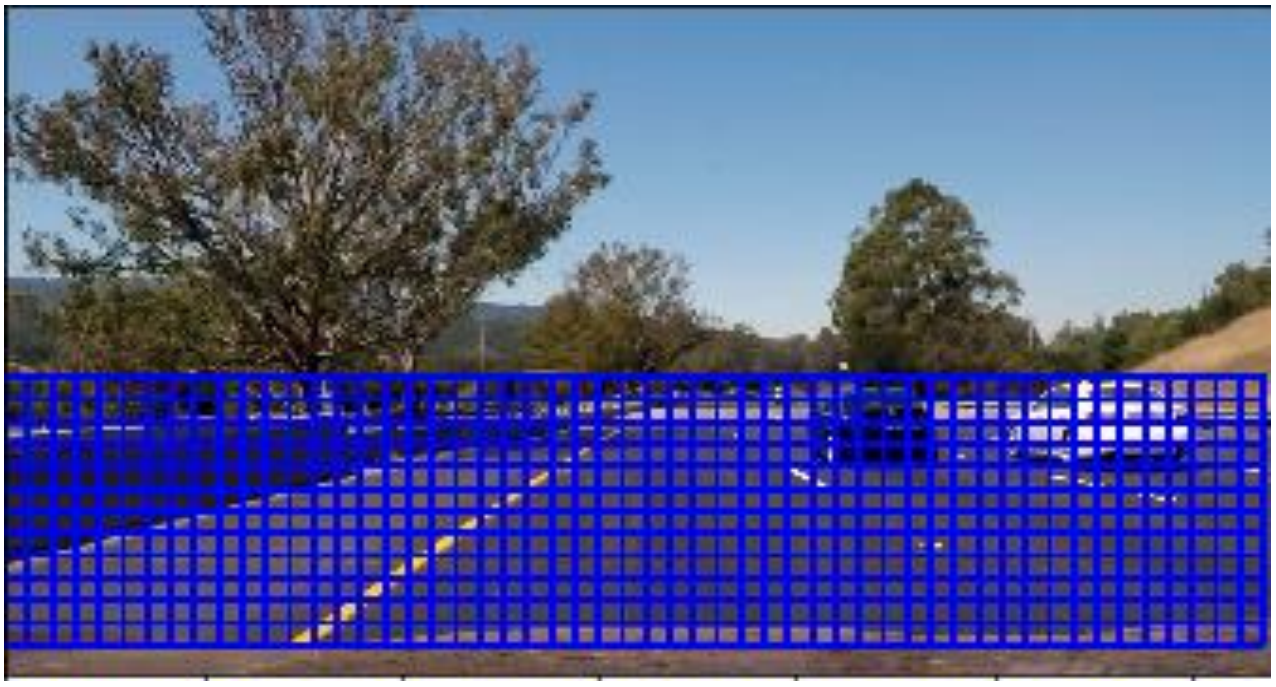
Now, according to the scaling size and cells_per_step, different images are extracted from the image. Then the function extracts the Hog and Histogram features. It then uses the standard scaler to normalize the features of the image. Finally, using the SVM classifier a prediction is made on these window images whether they contain vehicle or not. Finally, if the sub images are classified to contain cars then they are appended into the final list of the bounding boxes. Finally, the find_cars function returns the bounding_box coordinates. As mentioned before, in order to optimize my performance, I used the YCrCb channel and used histogram features along with Hog Features. My main focus was to keep the speed of processing along with accuracy. It's the main reason, I chose a high value for pixel_per_cell i.e., 14

Also, in order to optimize my speed, I have only considered the right half portion of the image for window search, i.e., x>640. Since, we are driving in the right lane, so there is need to detect cars in the right lane only.

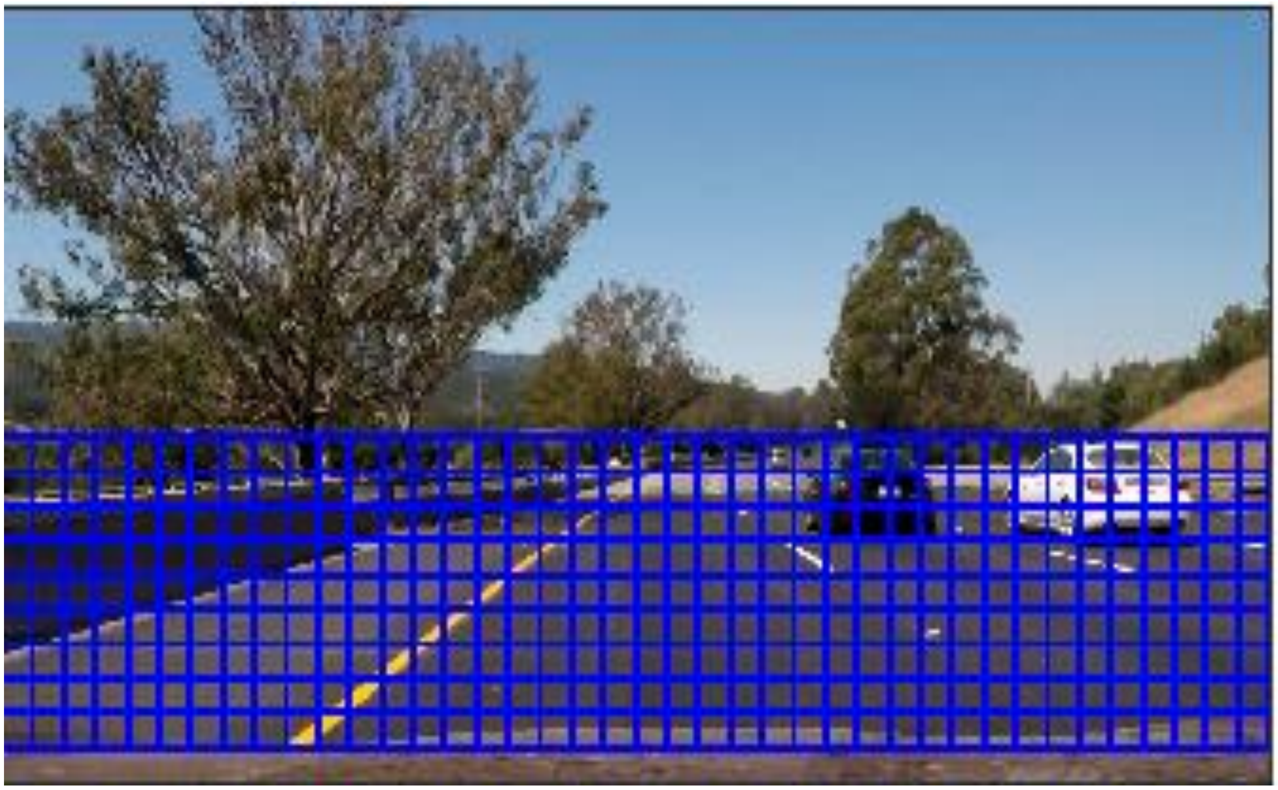
For scale factor = 1.0, the search windows: for this scale factor, I have only considered the top portion of frames



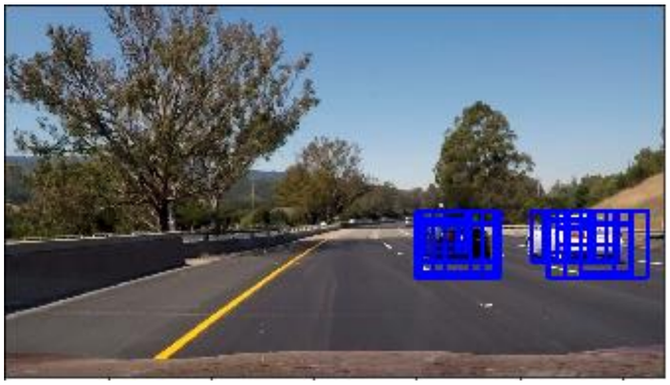
For scale factor = 1.5, the search windows:

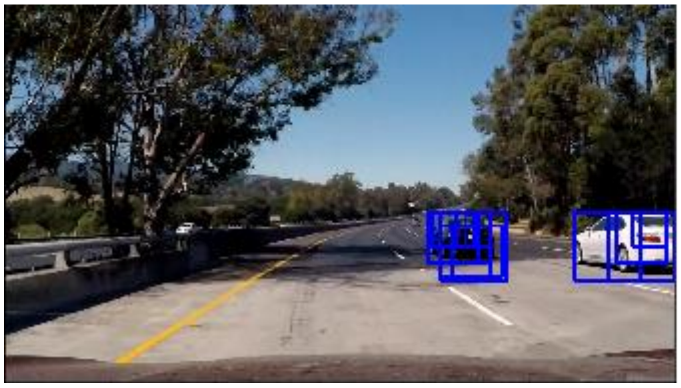


For scale factor = 2.0, the search windows:



The images below show the final configuration rectangles found after searching the cars in test images:





Video Implementation

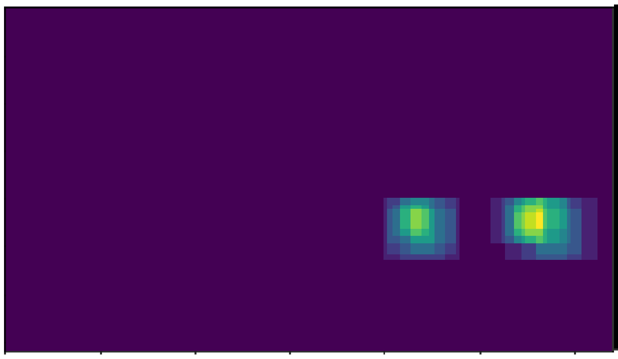
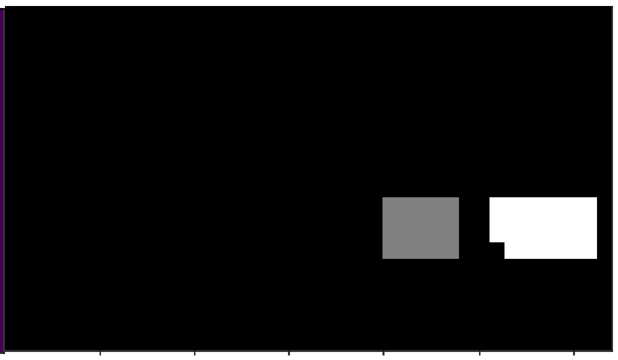
2. Describe how (and identify where in your code) you implemented some kind of filter for false positives and some method for combining overlapping bounding boxes.

I 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 to identify vehicle positions. I then used `scipy.ndimage.measurements.label()` to identify individual blobs in the heatmap. Assuming each blob corresponds to a vehicle, I constructed bounding boxes to cover the area of each blob detected.

Besides, I also used averaging method to filter out the false positives in my images. I am taking a combination of consecutive 6 images and adding the bounding boxes to the heat maps. After that I threshold the resulting combination of the images to identify the vehicle positions.

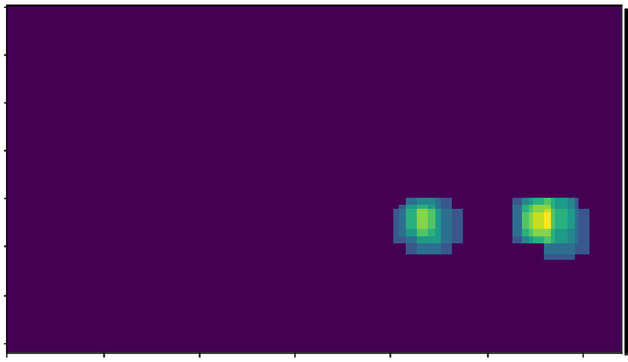
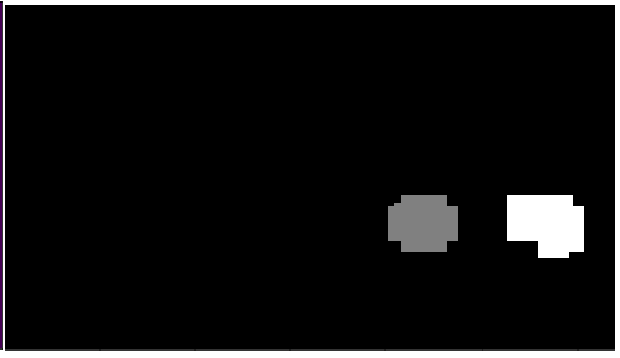
No. of consecutive frames for averaging = 6

Threshold value = 3

Heatmap without thresholding

Label image without thresholding

Heatmap with thresholding

Label image with thresholding

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 main issue which I faced while doing this project was to maintain the performance and accuracy with respect to speed of the detection. I had to compromise one of the two things. The speed of the detection process really went low as I increased the number of features of the images.

Also, though the pipeline works for most part of video, but there are parts where the drawn rectangles flickers a little bit. It is the part when there is resemblance in the car as well as surroundings.

My pipeline is likely to fail when the images of the cars don't fit to the profile of the dataset on which I have trained my model. Also, in conditions when there is much resemblance between the surroundings and the cars, the implementation may not give best results. Also, my pipeline works specifically for this video only, because to increase the performance speed, I have performed search window on only right half of the image frames.

I believe that, given more time for the project, a strong classifier along with a good dataset to work upon might do the trick. Also, if we use CNN, it might be far better approach than the currently used approach in this project.