Vehicle Detection Project

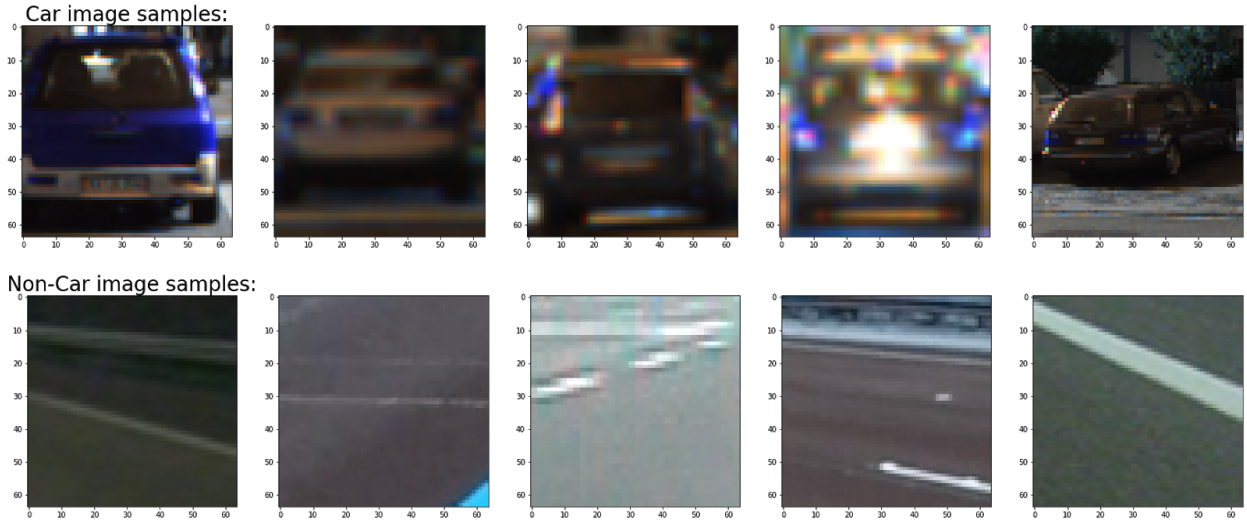
The goals / steps of this project are the following:

- 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)

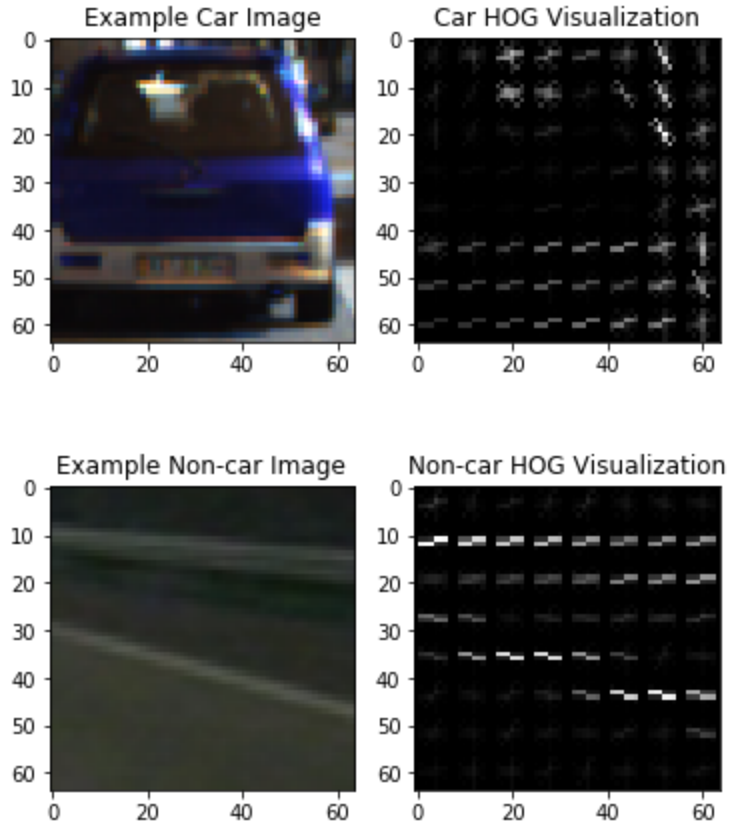
1. Extract HOG features from the training images.

I downloaded training data and test data from web, and put them in non-vehicles and vehicles folder respectively. The following images are the sample images I used.



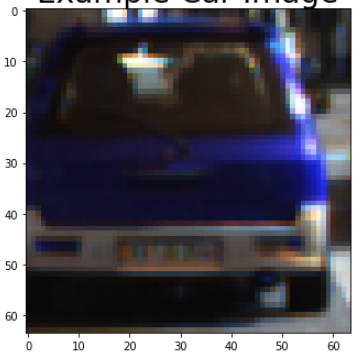
I 2nd cell, I defined get_hog_features function, which is the same offered in the class. It will call hog function in skimage.feature.

Here are the examples of hog image:

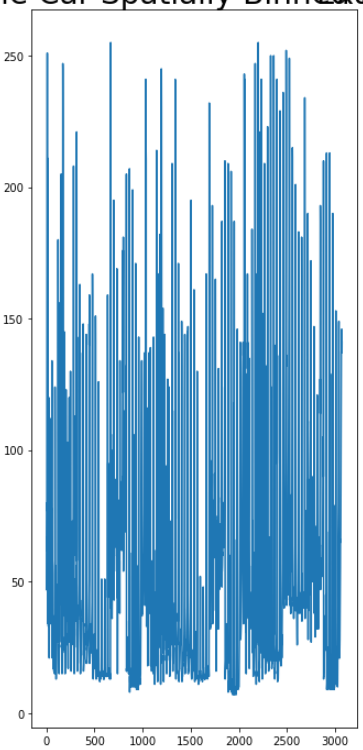


I also extracted images' bin spatial and color hist info as part of model input.

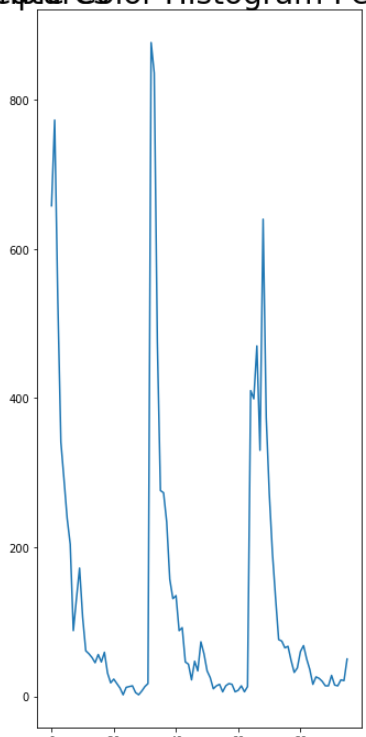
Example Car Image



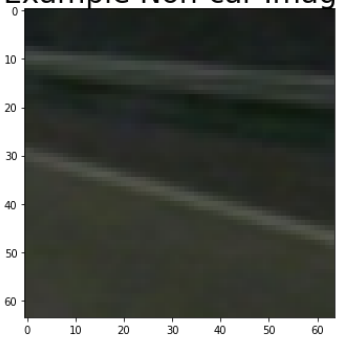
Example Car Spatially Binned Features



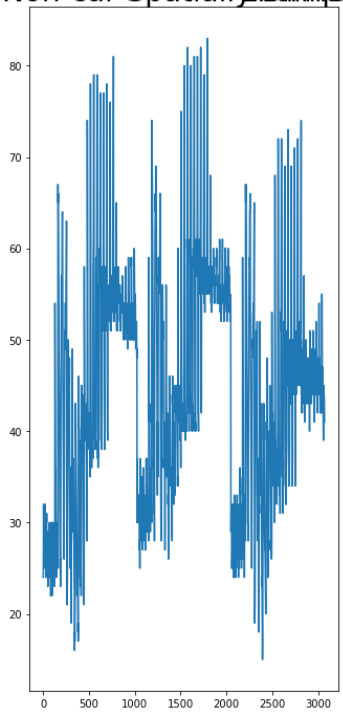
Example Car Color Histogram Features



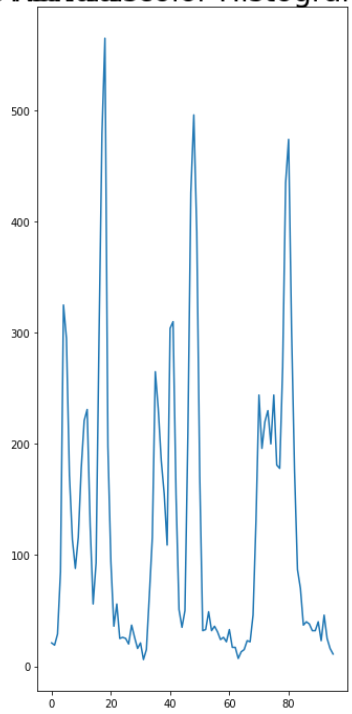
Example Non-car Image



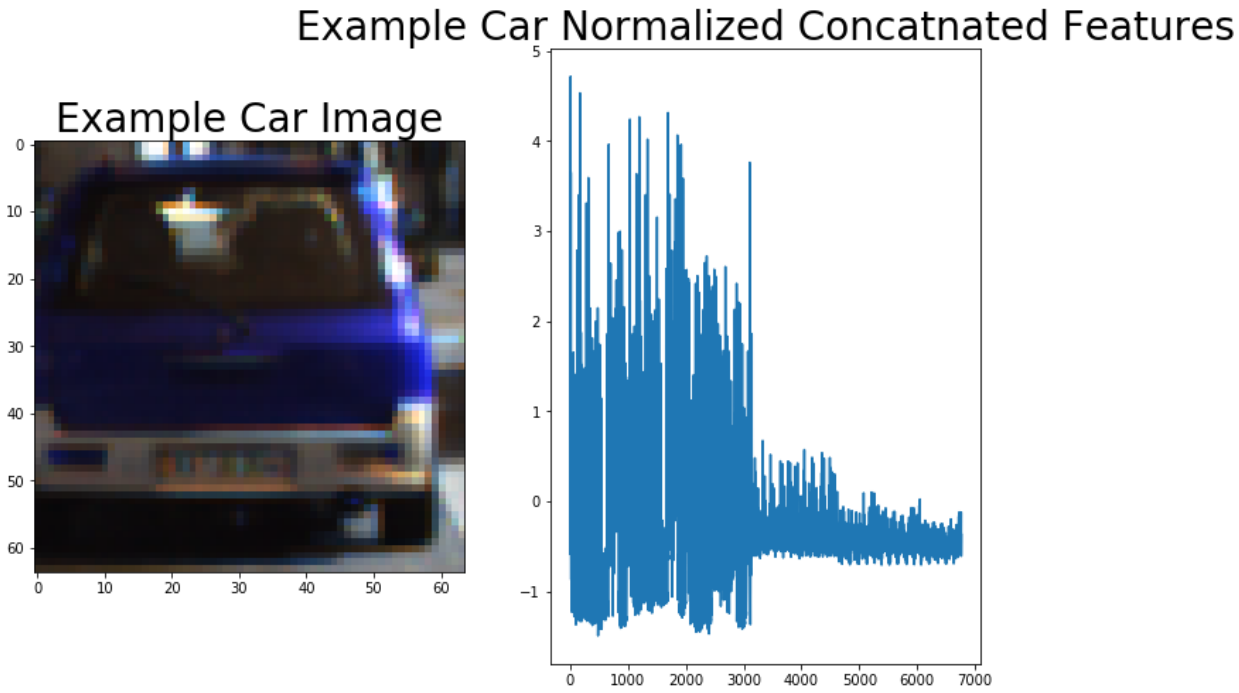
Example Non-car Spatially Binned Features

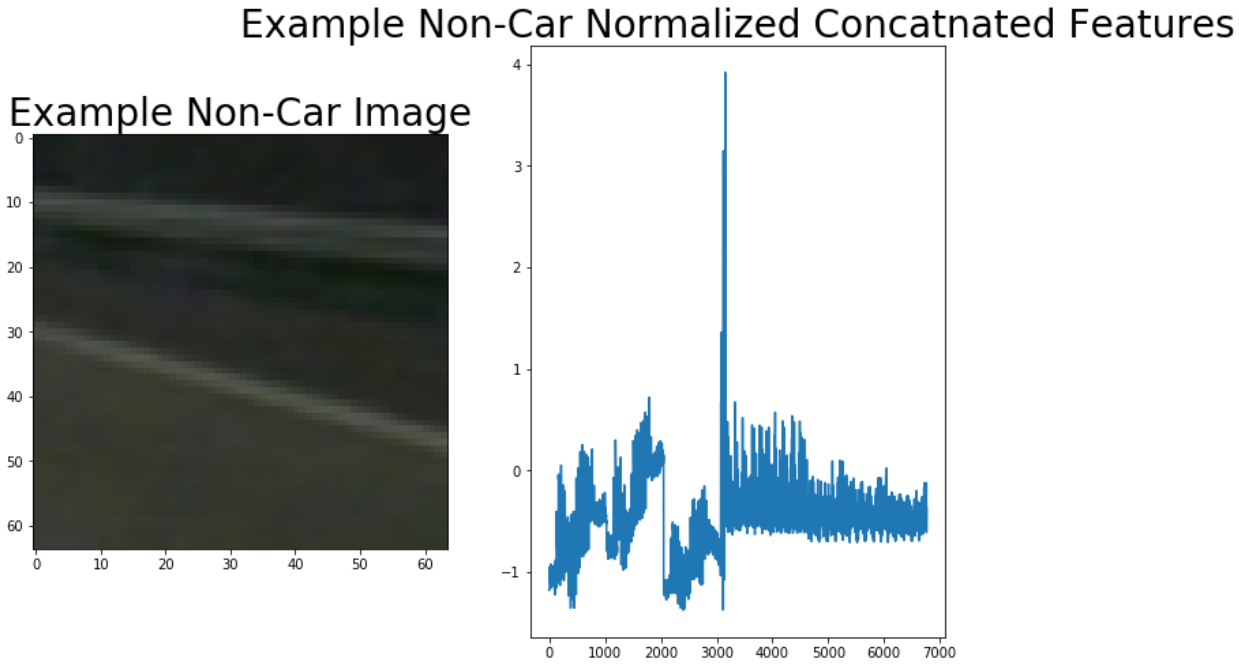


Example Non-car Color Histogram Features



In 4th cell and 6th cell, I concatenate these features and normalized them.





Finally, a linear SVM model is setup and trained in 8th cell. Here is the result.

25.31 Seconds to train SVC...
Test Accuracy of SVC = 0.971

2. Select HOG parameters.

First of all, I tried RGB all and HSV all, they both perform very well. However they take much more time than other models, since they need to extra 2 channel features. Then I tried GRAY color space and HSV color space with H, S, V channel respectively. I sliced out 2 pieces of sample video, 5 secs each to test their result. One is from 0s-5s, another is from 25s to 30s. Based on the final result, I choose to use HSV with S channel, since it performs very well without using too much running time.

3. Train SVM model.

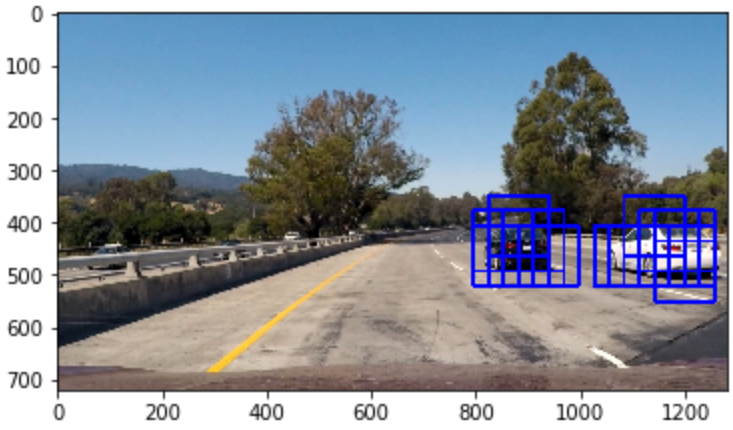
In cell 8th, I trained SVM model. I used LinearSVC from sklearn.svm. It is very straightforward. The input feature are normalized concatenated features with hog, bin spatial and color hist. In 6th cell, I randomly split the training and testing sets. I used 20% data to test.

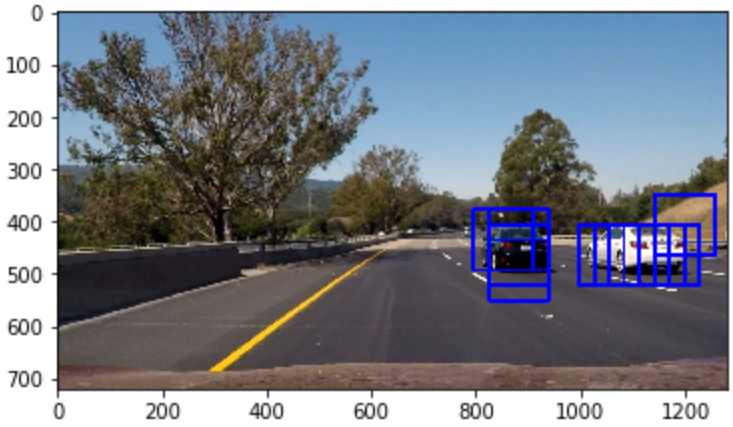
Sliding Window Search

1. Implemented sliding window and search vehicles.

I implemented these codes in 9th cell. A find_cars function will take some input and search vehicles with certain scale in the sub area we pre-defined. Here, I hard coded a x_start as 450, means we only search the area larger than 450 from left. This is because the left side is difficult to detect, the relative speed is much higher than right side. Also, since we are driving on the left most lane of a highway, we don't need to take care of left side too much.

Then I defined a find_car_with_multiple_scales function, which will take an additional input nrounds. This parameter is used to help us define how many area we want to divide by y. Since the top side always prefer a small scale than the bottom side, we don't want to use same scale for them. I did test running for my test videos, and decide to use nrounds=1.2, initial scale as 1.2 and 1.5 as scale multiplier coefficient. Also, each interval will have 50% interval overlap with their up and down intervals. The ystart and ystop are 350 and 650 respectively.





Video Implementation

1.Here's the link to my video

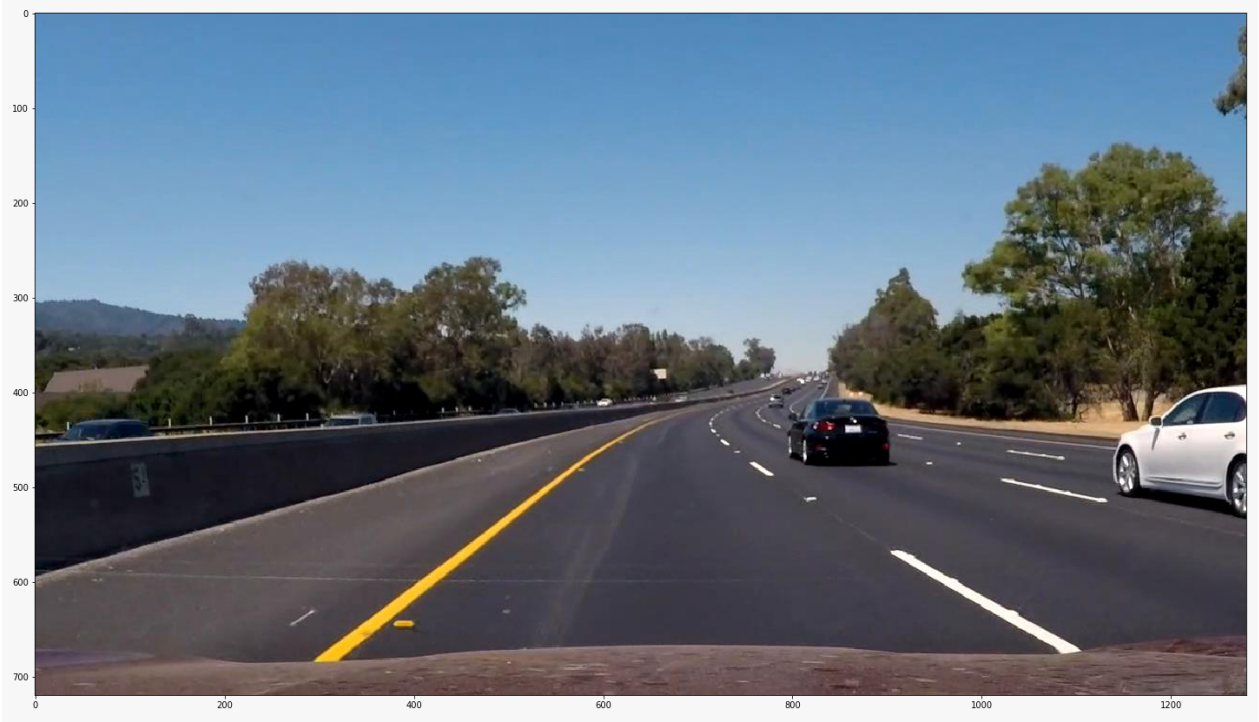
2. Use multiple frame info to remove false positive cased.

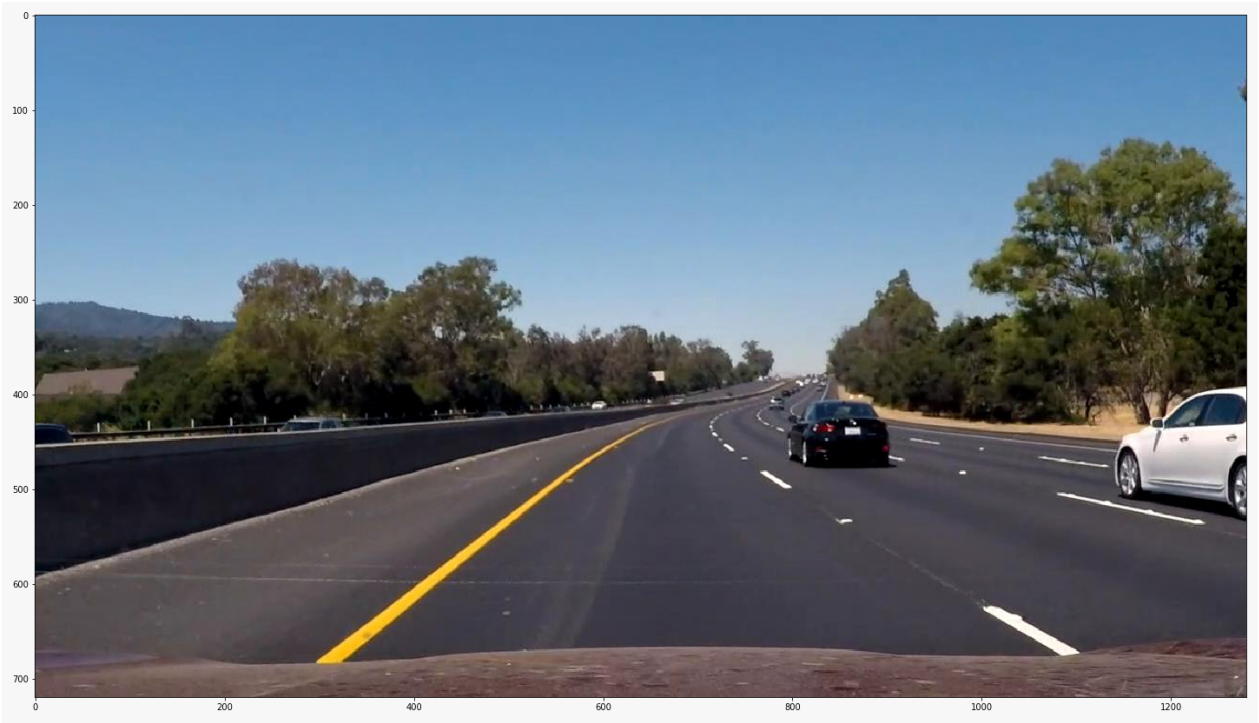
Here I used a heatmap method to remove false positive cases, with a little bit variant.

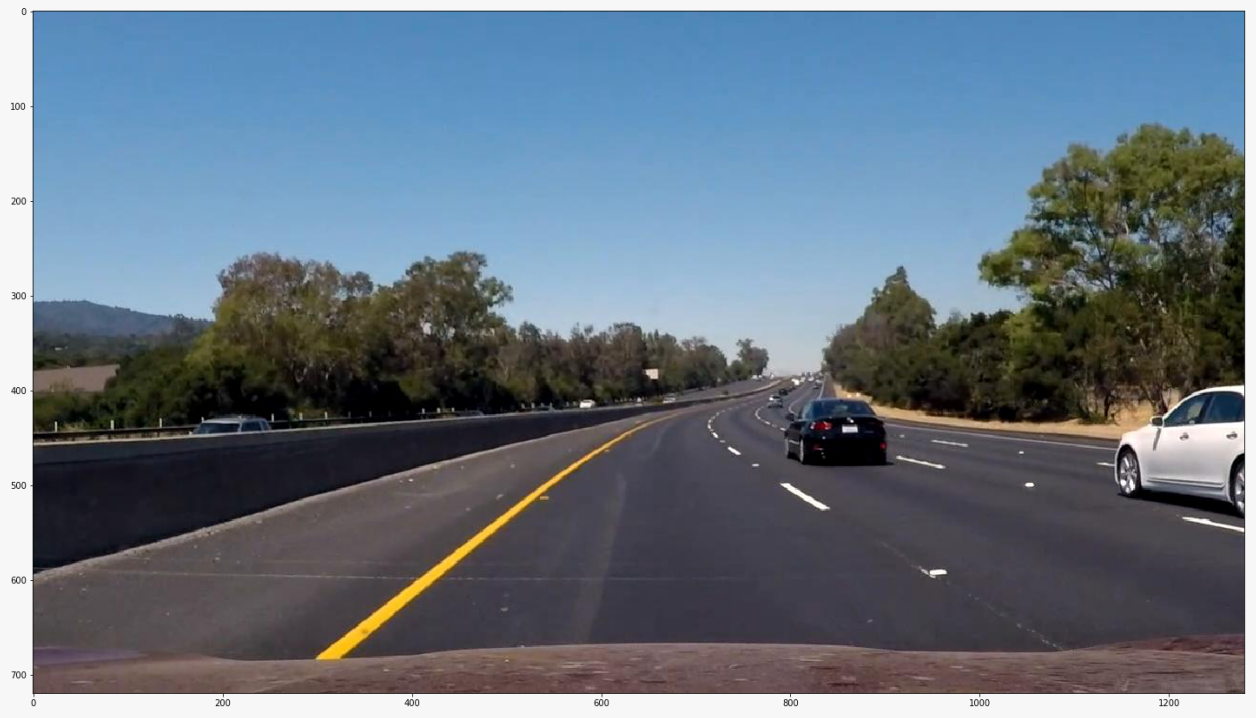
1. I added weight as input to add_heat function. Thus, I can make recent frames contribute more to the heatmap
2. For apply_threshold, I set non-zero positions to 1. Thus, some recent frames will not over contribute to the most recent detection.
3. For draw_labeled_bboxes, I only draw and return bbox with minimum side larger than 60 to remove false positive cases.
4. I use a memory size 15, which means recent 15 frames will contribute to the most recent frame detection.
5. I introduced two steps in the detect_car. The first step using an exponential weight to help filter out some false positive bboxes and keep some new introduced boxes. This is very important, for new introduced box, we can hardly say it is an false positive case or true positive case. We keep their info first and draw in the next several frames, if we still can find these bboxes. An exponential weight, will make the recent fram contribute as much as possible and slightly use previous frame info to filter out some false positive cases.
6. The second step is use a linear decrease weight to decide whether to draw the image. I use this method here, since it can let previous frames contribute more in deciding whether we need to draw the bbox than exponential weight. Even we make a mistake and fail to draw a bbox, the new introduced bboxs will still store in self.recent_bboxes. Also, we sue all the 15 frame info to decide whether draw the bbox, this will let us have a much smaller chance to make the mistake.

Here are the continue frames output for a video:

The first three are initializing frames, thus vehicle is not labeled on it. The remaining ones perform very well.

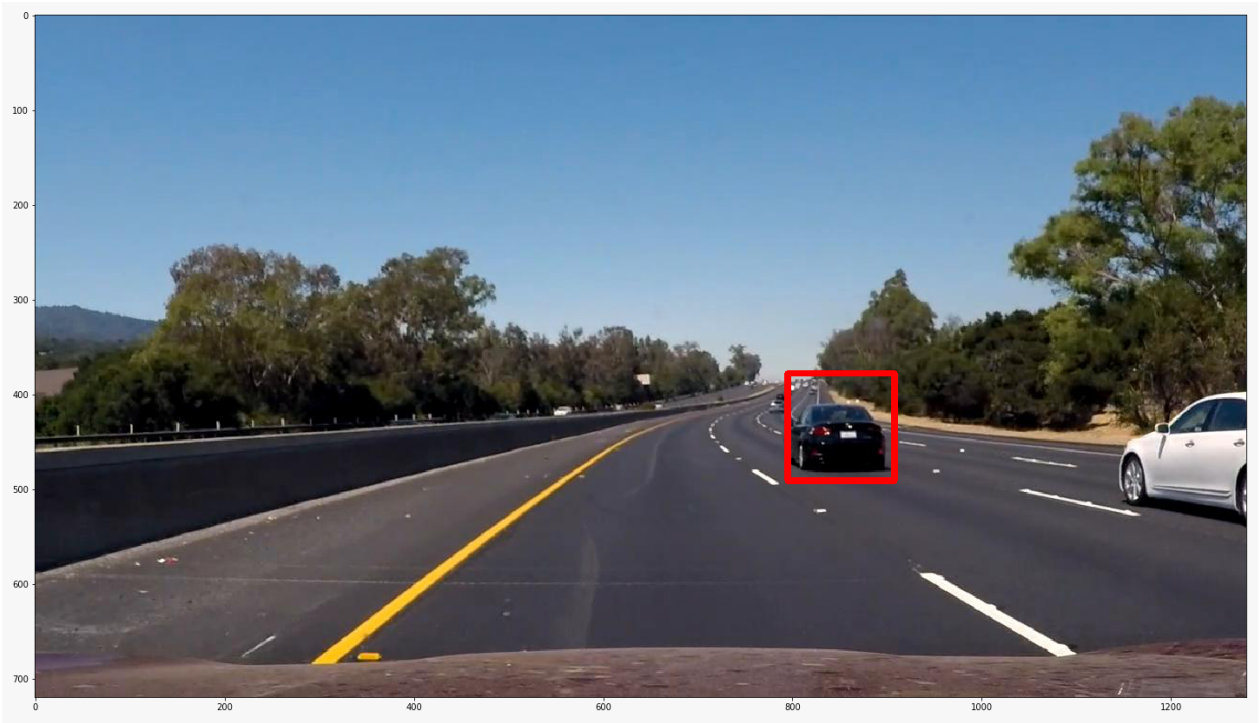

0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

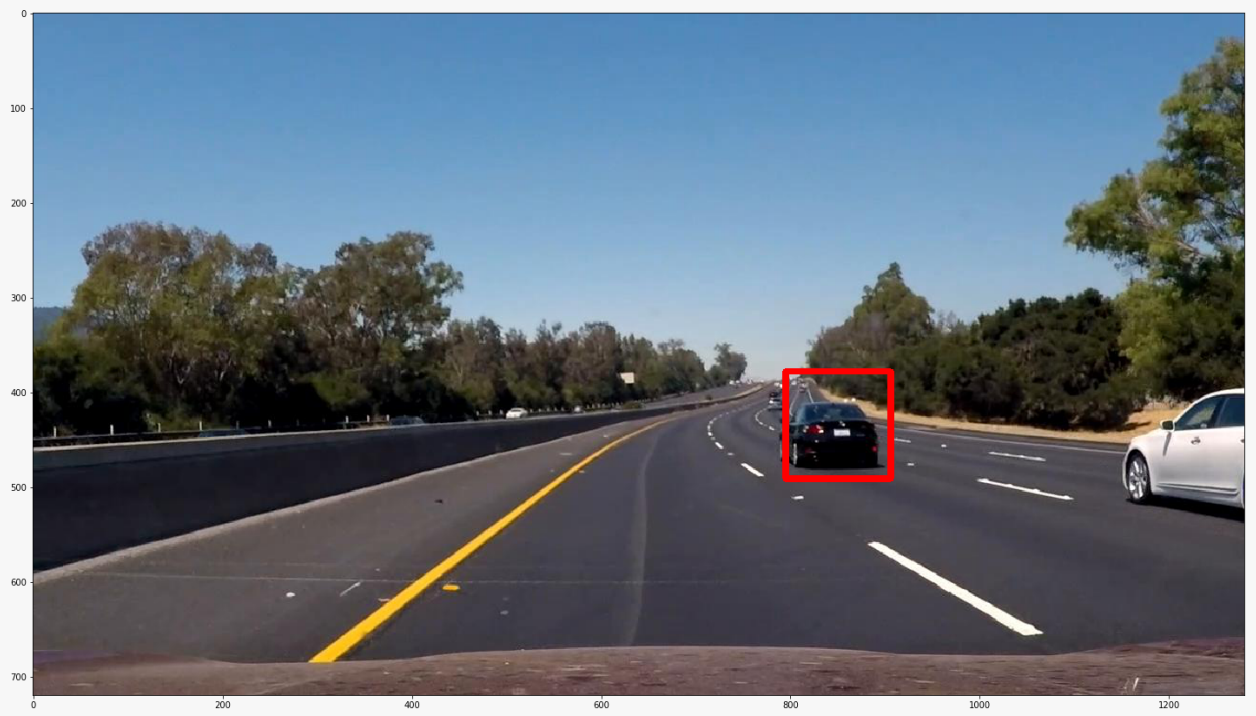

0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

Discussion

The most part of video predicts very well. However we still have several frames introduced false positive bboxes. Also, sometimes we lost our bbox. If two cars stand very close, we cannot identify them from each other.

1. For the first problem, we can use multiple cameras from different angles to take video and do analysis. Also, undistort image may help us to reduce false positive.
2. For a bbox we detected for a long time, we can think it's a car and specially label it. Store its location, size and some other info. After we store these info, we can make the bbox size relatively stable and relabel it when we miss it suddenly.(not miss in the detection boundary case)
3. For two cars intersection cases, The 2nd approach may help us to solve it. Since we are able to track each car's speed and position once we have some related info stored.