Documentation for Udacity SDC Term one, Fifth Project

Objective:

The goal is to create a classifier that can detect other vehicles on the road while the self-driving car is driving on the road with a camera attached into the center of the self-driving car.

Design decision:

The design is base on the Project material provided by the Udacity Project 5 description in the Self driving class term one. I implement three functions to extract the image features to determine rather the image contain a car or not. They are the image spatial features, color histogram, and HOG (Histogram of Oriented Gradients). The implementations are basically identical to how the class description described it. All the features parameters are same as what the class material was used. For example, color histogram bin size is 32, HOG orient is set to 9, etc.

The training data are 64 x 64 pixels color image with three color channels. And the following links contain data that I used for training

https://s3.amazonaws.com/udacity-sdc/Vehicle_Tracking/vehicles.zip

https://s3.amazonaws.com/udacity-sdc/Vehicle_Tracking/non-vehicles.zip

After reading all the training data, they are shuffle before being used my classifier to train.

I finally decide to use only image spatial features, and HOG. No color histogram since I do not think it makes any difference. In addition, I used “YCrCb” coloring for image spatial features and “HLS” color scheme for HOG.

SVM (support vector machine) was used for my classifier. Since the features is from both the HOG and image spatial features, we normalize the feature vector with StandardScalar() function similar to the way how the class material was described.

After training the data, the accuracy rate is slightly above 99%. That mean for our car detection on the road, it will not be perfect but should be very good.

How a car is detected in the actual images or video capture from the camera in the car?

Cars are detected by scanning through the entire capture by the camera. Well, not the entire image since we know there will be no car on the top part of the image. The road is at the bottom part of the image. Now, since I am using SVM classifier, not Convolution Neural Network, the size of the car in the image does affect the predictability of the classifier. And a car that is closer to our self-driving car appear larger to our camera than the car with the same size that is further away from our car. Therefore, we need a few sliding windows for different sizes at different region of our image.

In the function “find_car”, I defined a few windows sizes for our sliding windows. They are 64 x 64, and 128 x128. All windows start to scan our image from “y” axis of 360 to 630. This is assuming that our image size is always 1280 x 720 and anyway below 650 is the dash in our car and above 360 is something that is very far away that we do not have to worry if there is a car that is very far away until we approaching to the far away car closer.

The sliding windows will scan through our image at a specific region and use the classifier that we trained for car prediction. The windows may only capture part of the car and the classifier may not be able to detect a car correctly. So the sliding windows will overlap from the previous windows position. For the smaller windows, I set the overlap value to be 0.75 for both x and y axis. That mean the new windows location will overlap 75% of the previous windows x and y axis. For a larger window, I set the overlap value to be 80% for both x and y axis.

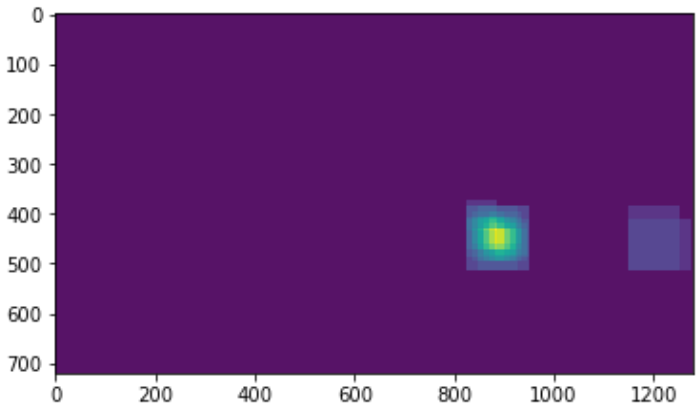
Preventing false positive:

we may detect false positive. But because our classifier accuracy is above 97% doesn’t mean it will be perfect. what we can do is slide the windows on the image, and see if the overlap regions is detected. This give us more confidence that the actual car is detected. The idea is if our current sliding window detected a car and the next sliding windows also detect a car, we have a high confidence that there is a car in the region of the image and the error rate generated by the classifier is reduced.

I used the material described in the class to create a heatmap to determine if an actual car is in a specific region of the image. In addition, I slightly modified the “draw_label_bboxes” function described in the class material by removing a box that is very small. draw the actual box where a car is detected. If the box is too small, in my case, any box that is detected after heatmap filtering that is smaller than 50 x 50, I filter them away because we know that a car cannot be as small as 50 x 50, or if it is actually a very small car, that mean it is very far away from where we are and I do not think I need to worry about car that is too far away.

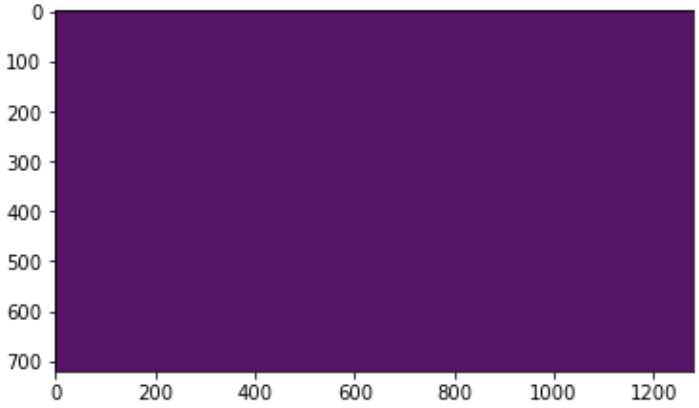
The following are image from test_images directory in the project repository where I apply heatmap and draw a box on a car that I was able to detected.

Test1.jpg heatmap



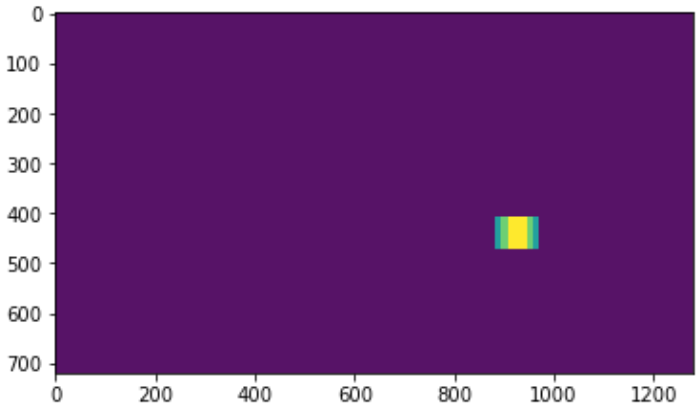
Actual cars detection for test1.jpg

Test2.jpg heatmap



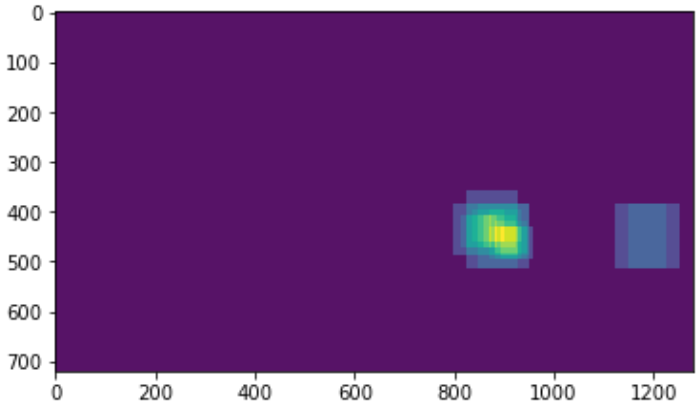
Actual cars detection for test2.jpg

Test3.jpg heatmap



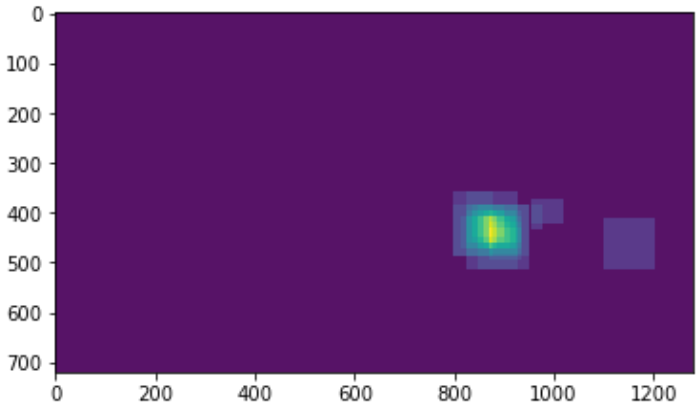
Actual cars detection for test3.jpg

Test4.jpg heatmap



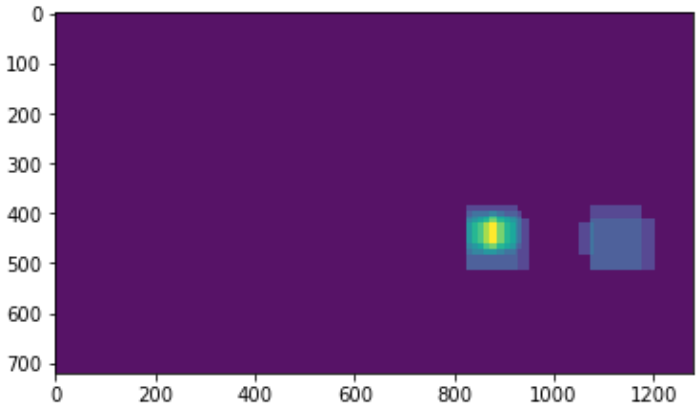
Actual cars detection for test4.jpg

Test5.jpg heatmap



Actual cars detection for test5.jpg

Test6.jpg heatmap



Actual cars detection for test6.jpg

As we can see above, test5.jpg has detected a false positive. But it was not too bad.

How do we further remove false positive in the actual video?

For each video frame, we detected some regions where we believe a car is located. We buffer them together. In this case, the buffer size is 10. And we add up all these 10 frame's boxes like we did in the heapmap above for each individual image and use the same algorithms described above or in the class project material.

Challenge and Improvement:

I found that processing each frame is very slow, the Histogram of Oriented Gradients is slow, sliding windows make it slower since there may be a lot of windows. Especially we need to overlapping those windows to make sure we don't miss any car. We can have this improve by sliding our windows only

within the lanes area, therefore the advantage lane detection can be used here. Using Support vector machine with a lot of windows size for sliding windows also slow it down. Maybe we can have convolutional neural network with a fixed predefined windows size. The training may take longer but it will be much quicker at run time. We want our self-driving car to response as quick as possible. Another way to speed up the computation time during the video processing is to use HOG Sub-sampling technique. However, this only applies to sliding windows of size 64 x 64 only.

I think the quality of training data is also very important. Improving the quality of the data for training can also avoid as much false positive as possible.