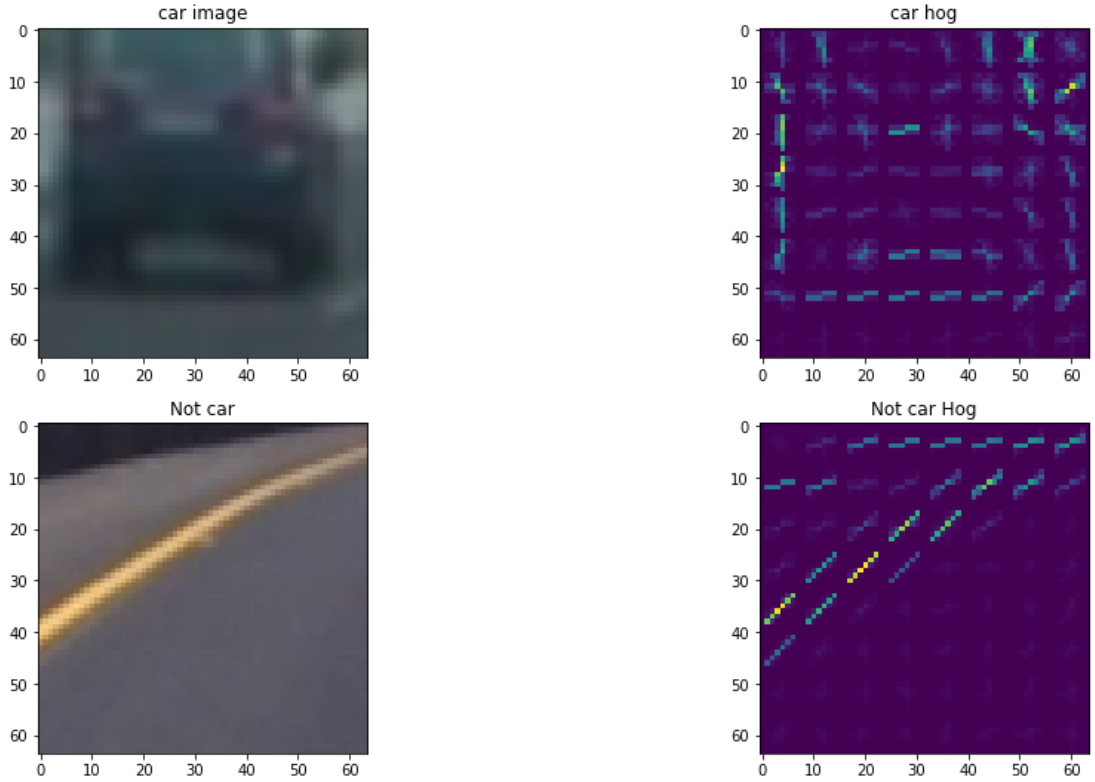
Vehicle Detection and Tracking

The project is done in a Jupiter notebook. This write_up is a summary to show that all requirements in the rubic are met. All output images can be found in the Jupiter notebook "main.ipynb" or in the folder "output_images." An html file exported by jupyter notebook is also provided, named "main.html." Final video output is in the root folder of this repo called "project_out.mp4" and "test_out.mp4."

Histogram of Oriented Gradients (HOG)

Explain how (and identify where in your code) you extracted HOG features from the training images. Explain how you settled on your final choice of HOG parameters.

1. Define a function get_hog_features() to generate features from one simgle image.
2. Define a function extract_features() to apply get_hog_features() to a list of images.
3. In extract_features, I can try different combinations of parametters, such as color_space, spatial_size, etc.
4. I tested some different combinations some parameters, and found out YUV color space works best, as I fed the features to my SVC classifier it gave me a test accuracy of 98.9%.
5. It took 84 seconds to prepare the training data (extract and normalization all features).
6. Feature vector length: 11988, training data: 8792 cars and 8968 non-cars
7. Here is the visualization of hog images. Code is in cell 2-6.



Describe how (and identify where in your code) you trained a classifier using your selected HOG features (and color features if you used them).

1. I used SVC(svm) as my classifier. A default SVC classifier works better than I expected so that I didn't tune different SVM parameters.

2. I used StandardScaler().fit() to normalization the training data, after that, I used tain_test_split() to separate the data into 80% of training set and 20% of testing set.
3. I used pickle and joblib to save the X_scaler and the model in file "scaler_data" and "SVC_model." Which can be find in the folder so that when I implemented the sliding window search, I didn't have to extract features and train the model repeatedly later.
4. The model was trained for 7.24 seconds with a test accuracy 98.9% on 14208 training data and 3552 testing data.
5. Code is in cell 7-9.

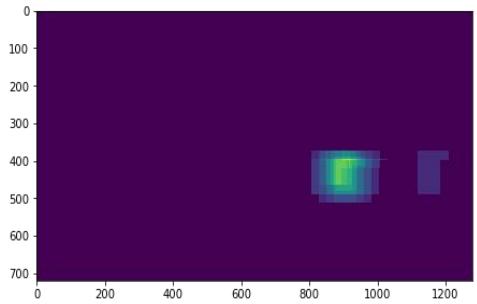
Sliding Window Search

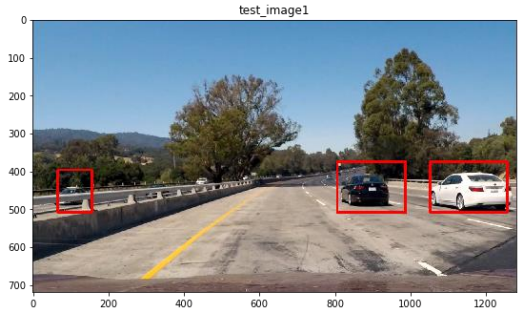
Describe how (and identify where in your code) you implemented a sliding window search. How did you decide what scales to search and how much to overlap windows?

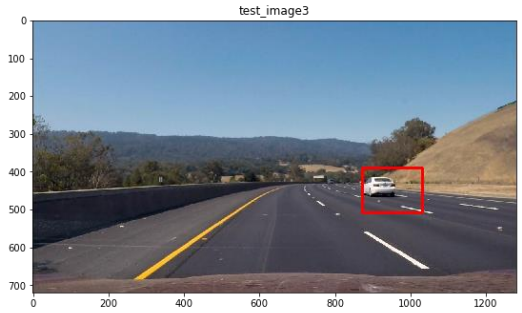
1. First of all, I defined some help functions learned from the tutorial such as conver_color, draw_boxes, slide_window, etc.
2. The main function is find_cars(), which take an image, and the vertical area (ystart, ystop) to slide small window in that area. We don't need to think about the region of sky or the region behind us, so the mainly region was from 400-550 on y direction.
3. I used different window size on different areas to make sure different sizes of cars can be found instead of just one fixed size of window. I may have tried more than 1000 different combinations of sizes and regions and find a "fairly good" combination.
4. The scales I used were based on experience. I image the cases that might happen and adjust some estimate scales first and give rid of the impossible scales. For example, we won't have a large scaler car if the car is far away from us. I used
5. Here are the images after applying the sliding window method. There are some false positive and overlapping but will fix later after adding heatmap and threshold.
6. Code is in cell 10-11.

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

1. After finding a bunch of rectangles, I applied heatmap method on them to give me a final rectangle.
2. One handy method I used was optimize the add_head() and draw_labeled_bboxes() functions a little bit.
   2.1 In add_heat(), I checked that if the X1 and X2 are not at the edge of the image, I would make the heat map a little bigger. In this case, the output heatmap rectangle won't be too small after I apply the threshold.
   2.2 In draw_labeled_bboxes(), not only use threshold to avoid false positive, but also I checked if the rectangle is too small, I will assume that is a false positive. Otherwise, I would assure that each rectangle should have a larger horizontal width. If I found the rectangle has a much larger vertical height, I will also make the rectangle a little bigger in x direction.
3. Code is in cell 12-13
4. Images of heatmap and final output







Video Implementation

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 provided in the repo folder named “project_out.mp4” and “test_out.mp4.”

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.

As discussed before, to avoid false positive, I used heatmap threshold and ignoring rectangles that are too small to be a vehicle or too far away that we don't really have to care. For the heatmap. I only accept the area that overlapped more than 1 time so that "a coincident bad predict" would be rejected.

Discussion

1. The size of rectangle still not showing nicely fit even I tried to adjust them a little bit. Usually they are smaller than what they are supposed to be.

   Try implement the add_heat(), apply)threshold() and draw_labeled_bboxes() to give me a better rectangle. For example, when a heat map has a big value at some small region, radiate out to make the region bigger by adding values, so that the rectangle can be larger. In the old code, region with value<=1 will be omit even thought the region contains part of the vehicle.
2. Tried more than 10 hours totally on the sliding window region and scale chosen, but still got very few false positive on left side of the video.

   By increasing the threshold and apply the technique above in Discussion 1, try more sliding region and scalers combinations may help solve this.
3. **Can implement the advanced line finding project in this project to give a even better, professional output video.**