Vehicle Detection Project

The goals / steps of this project are the following:

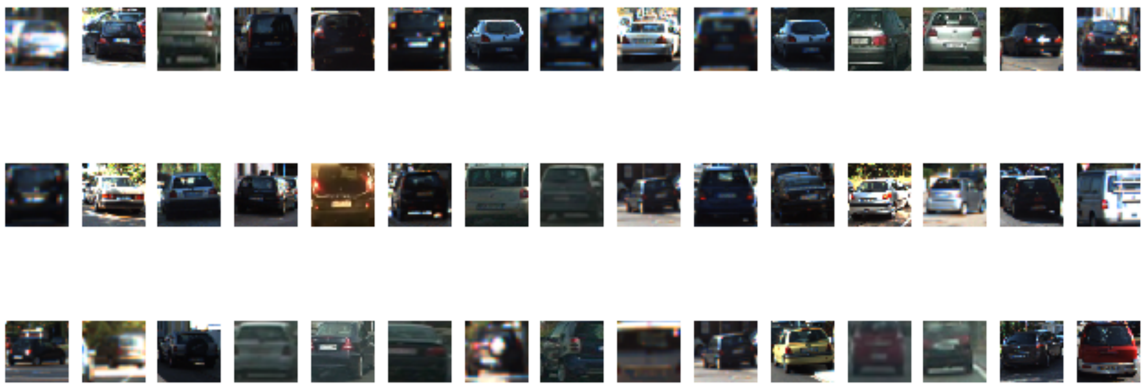
* Perform a Histogram of Oriented Gradients (HOG) feature extraction on a labeled training set of images and train a classifier Linear SVM classifier
* Optionally, you can also apply a color transform and append binned color features, as well as histograms of color, to your HOG feature vector.
* Note: 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.

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

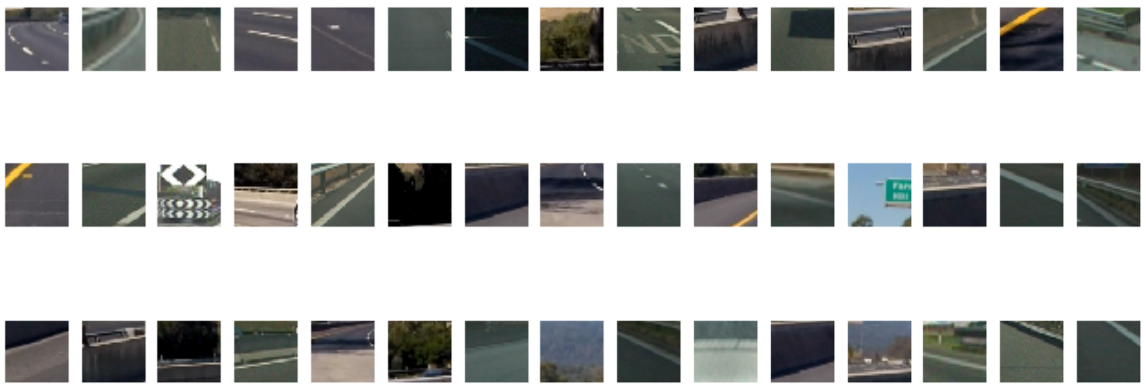
Histogram of Oriented Gradients (HOG)

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

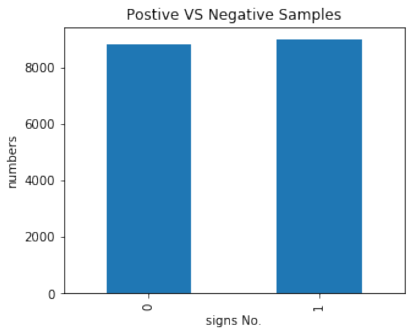
The code for this step is contained in the 19th code cell of the IPython notebook, before I allied the HOG function yo extract the features, I first load all the positive and negative datasets to pos_data and neg_data, then I visualize some samples from each datasets, here is some examples of positive data and negative data:



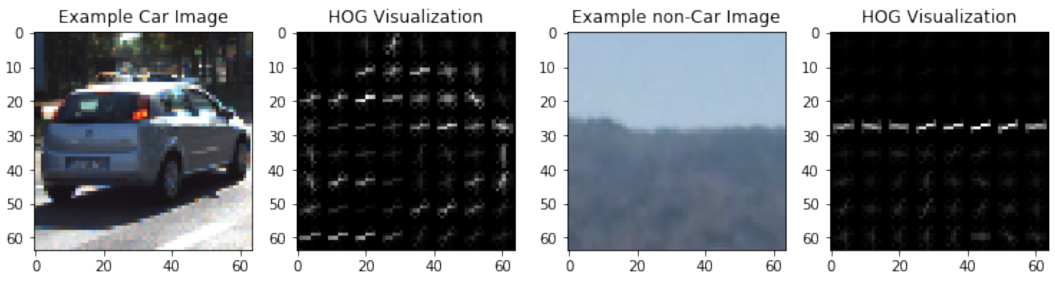
Positive data, vehicles


negative dataset, non-vehicles

I also count the number of positive images and negative images, they are very close, so I think we have enough data to training and need not to balance the dataset.



Then I try to extract the hog features from different channels, and using them to train different classifier models, displayed below is hog features in gray-level image.



The parameters of those two examples, I choose orient = 9 pix_per_cell = 8 cell_per_block = 2

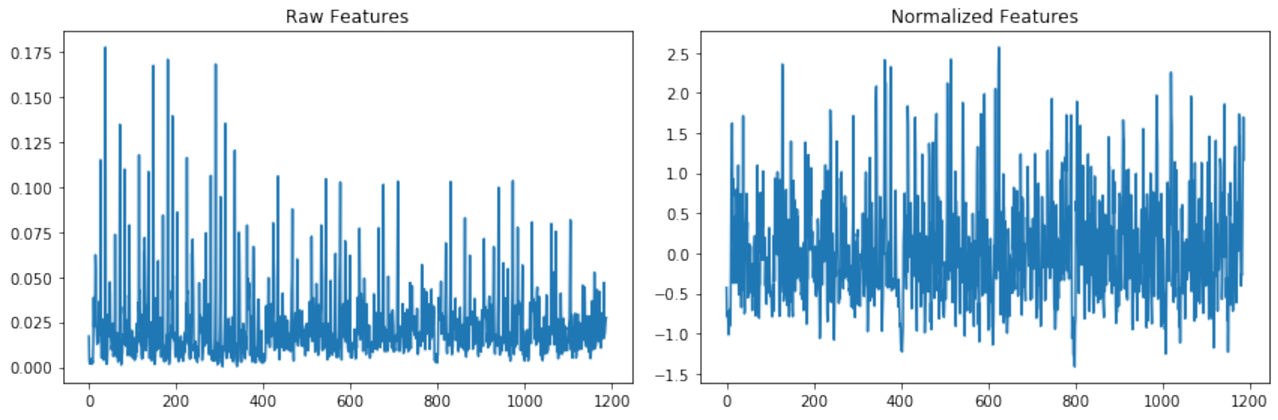
2.   Explain how you settled on your final choice of HOG parameters.

I tried many different hog parameters, and using them to train the svm classifier, according to the feature extract time costs, and prediction time cost, I choose orient = 7 pix_per_cell = 16 cell_per_block = 2 and I decide the YUV channel instead of RGB channel to extract the whole 3 channel hog 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)

I train the linear sum classification model under YUV hog features, I also try many other features combined with YUV hog features, such as color histogram , bin spatial features, and combined them with HOG features, But my final model are use YUV hog features combined with histogram features, because its stable than others and more faster than others. before I than the sum model, I also normalize the data and using sklearn api to do train,test split to random split the dataset to training data and test data, the code is in the 27th code cell of the IPython notebook. My final SVM models are got 0.9761 on test dataset.

the image show as below is the training data before normalize and after normalize(YUV HOG features).

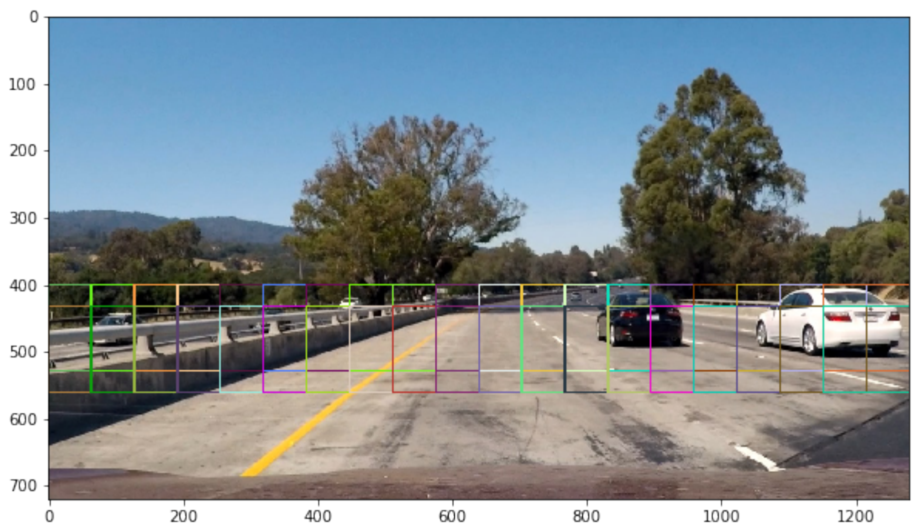


Sliding Window Search

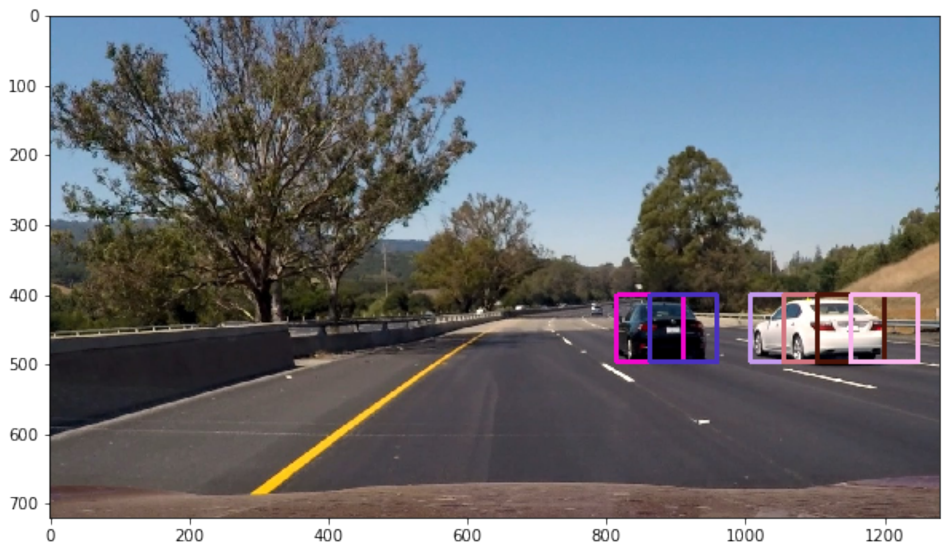
1. Describe how (and identify where in your code) you implemented a sliding window search. How did you decide what scales to search and how much to overlap windows?

I implemented the sliding window search followed by the classroom materials, the scales and other parameters are manually set according to many test. and I extract features using hog sub-sampling and make predictions in the same times, however, those methods still have many negative predict, the results is not stable and well to reach the goal.

the image show as below is some search window displayed on the original images.

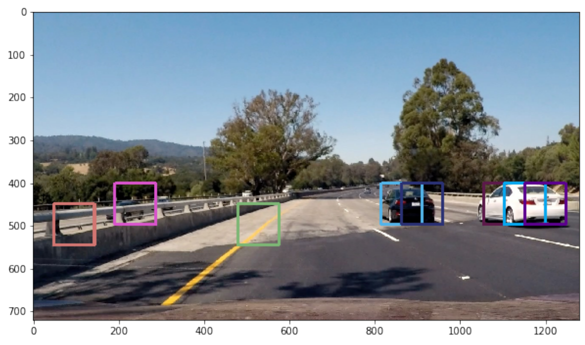


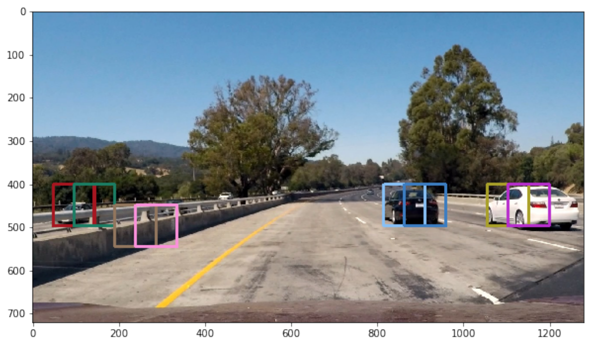
Here, I set more than 4 different window generate methods to sliding the window, and to make the predict. the boxes is more close more bigger, more far more small.



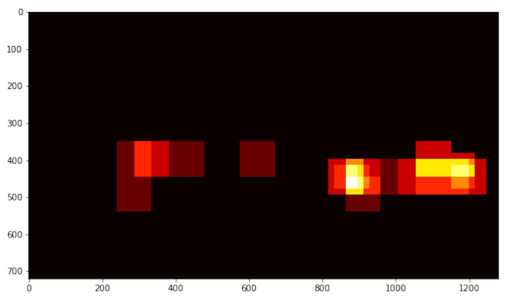
This image shows the predict results, because the sliding window have many different scales, so we may have many predict bounding box for the same object, so I applied the heat map methods to optimize the final output, I will explain it at next section.

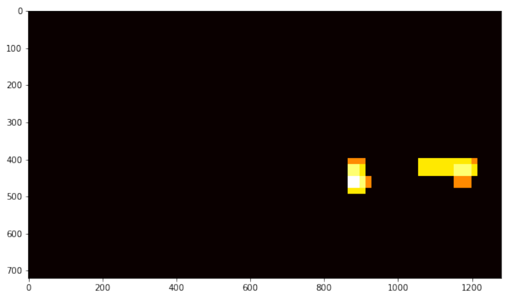
2. Show some examples of test images to demonstrate how your pipeline is working. What did you do to optimize the performance of your classifier?

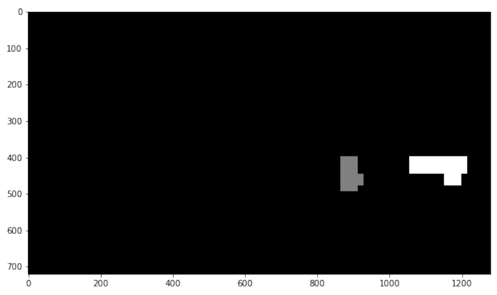


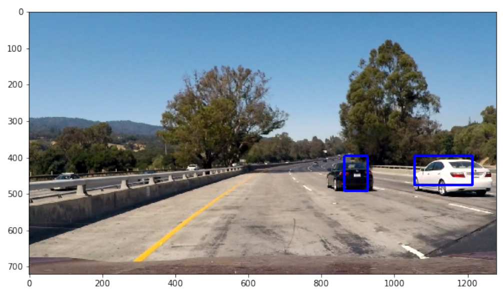


The model will output many wrong predict, and make the performance not well, I implemented the heat map methods followed by the classroom materials, the heat map function is in the 37 and 38th code cell of the IPython notebook. and here I can show you some images to show how the heat map function to optimize the performance of my classifier.

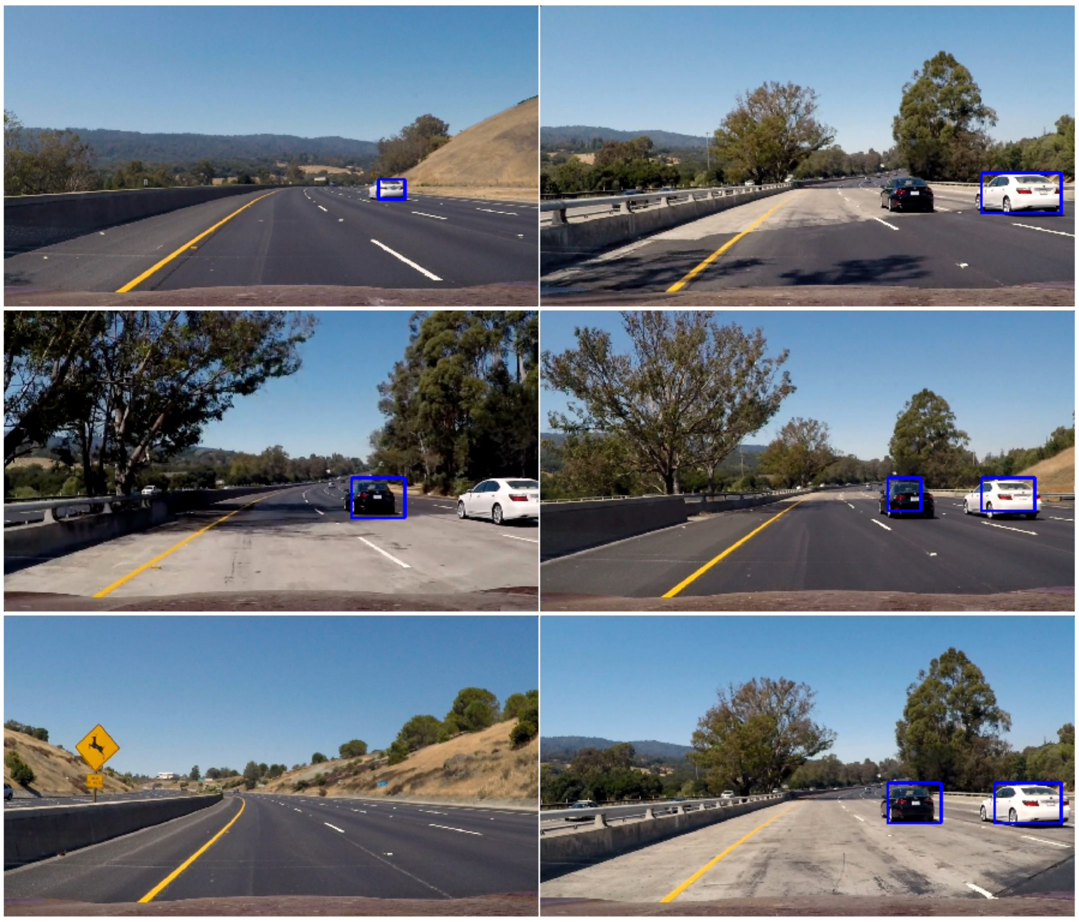








we can see the the heat map function works well to reduce many wrong bounding box, in this example, I set the threshold to 3, and the final output is well and stable here. because we generate large number of sliding window in the same time, and some of them may have wrong predict, and only the real positive predict may can generate losts window in the same time, so it means the heat map can works for this project.



Video Implementation

1. 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.)

I generate the final output video in output_images, and you can see the video, in this video, my pipeline can make the predict more stable, and identifying the vehicles most of the time with minimal false positives.

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.

here is my pipeline for vehicle detection project:
1.generate many sub-windows from original image, and crop them and resize them to target size
2.transform those image to YUV colorspace.
3.extract the 3 channel features and concat them together.
4.normalize the dataset
5.svm model to predict the results
6.for postive predict, append the window rectangles to lists
7.apply heat map function to all detected rectangles with threshold
8.draw the final rectangles to the original image

here I use the rectangles from the previous frame to make the current results more stable. this function works well, and make the bounding box more stable before not using this function. This function is in the 43 th code cell of the IPython notebook.

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?

This pipeline have many issues, because the hog features is manually labels by the human, don't like the CNN(CNN can auto learn the features from different datasets). and I found when I applied some data augmentation tech to generate more data, the classifier will confused and make many wrong predict, I think the reason maybe because for some data, the hog features is very similar for both negative sample and positive sample, I need change the hog parameters again to make them different, and then to train a new classifier, this is more difficult. and for some far distance vehicle, my pipeline will failed, I think because the window become more small, and less then 64*64 pixels, so the classifier may fails. To make the model more robust, we can use some advanced machine learning algorithm such as random forest, ensemble trees or CNN to train the data. or just add more different samples.