Project 05 - Vehicle Detection

Vehicle Detection Project

The goals / steps of this project are the following:

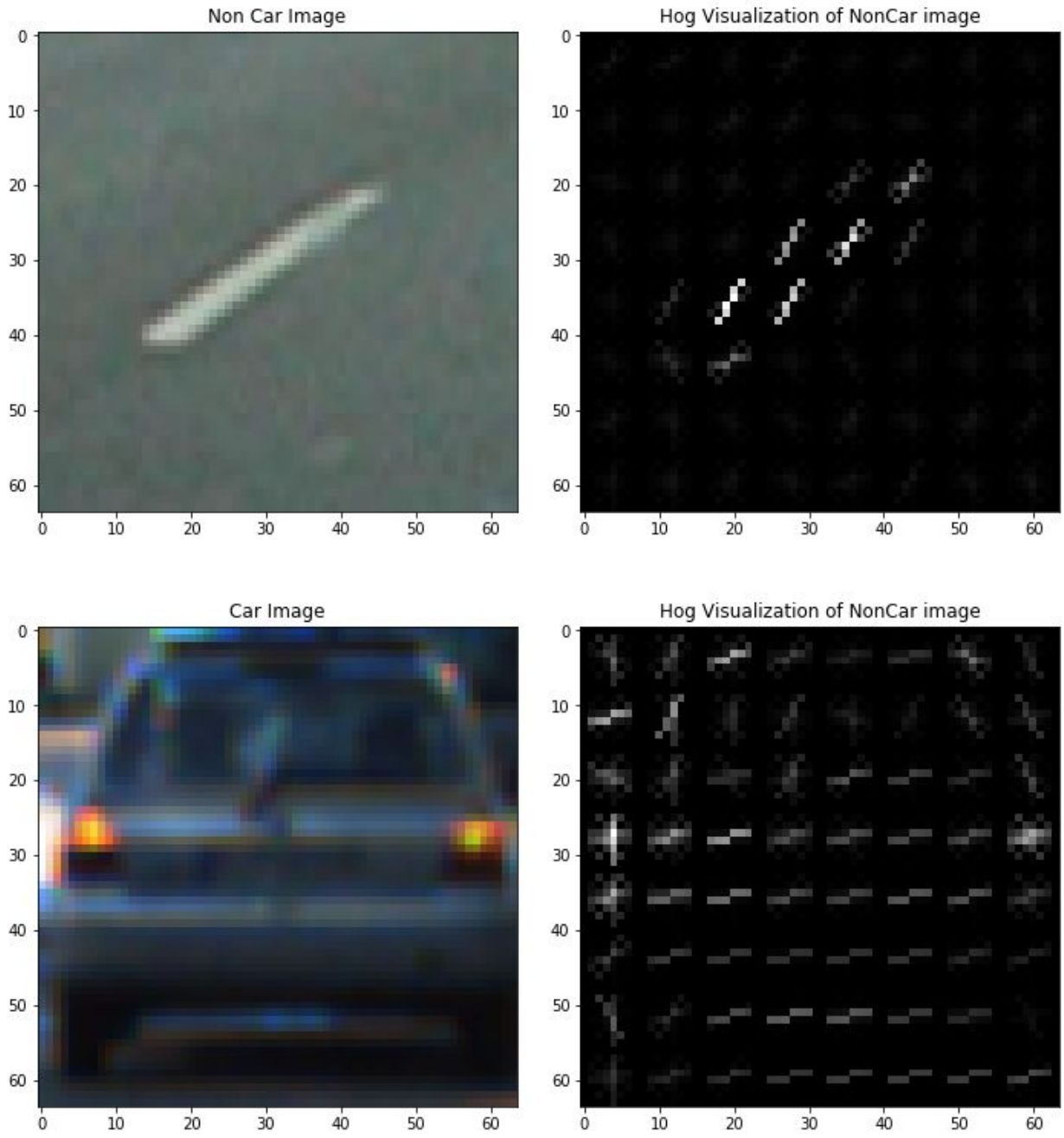
- Perform a Histogram of Oriented Gradients (HOG) feature extraction on a labeled training set of images and train a classifier Linear SVM classifier
- Optionally, you can also apply a color transform and append binned color features, as well as histograms of color, to your HOG feature vector.
- Note: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

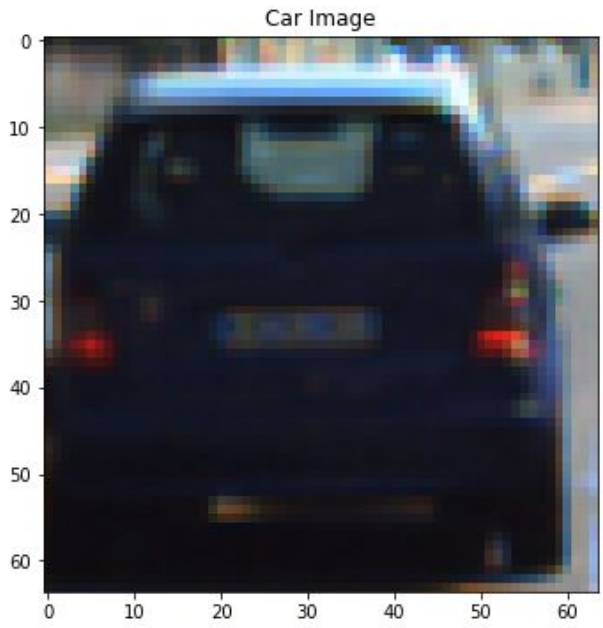
1. Explain how (and identify where in your code) you extracted HOG features from the training images.

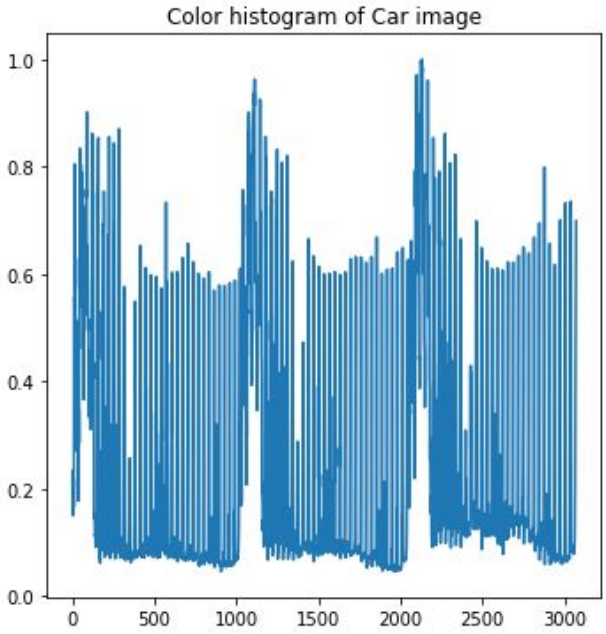
The code for this step is contained in the third code cell of the IPython notebook.

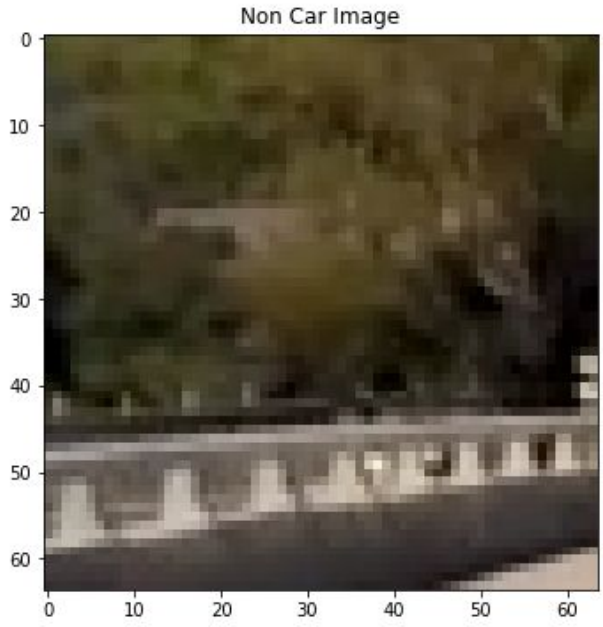
I started by reading in all the `vehicle` and `non-vehicle` images. Here is an example of one of each of the `vehicle` and `non-vehicle` classes and its HOG transform.

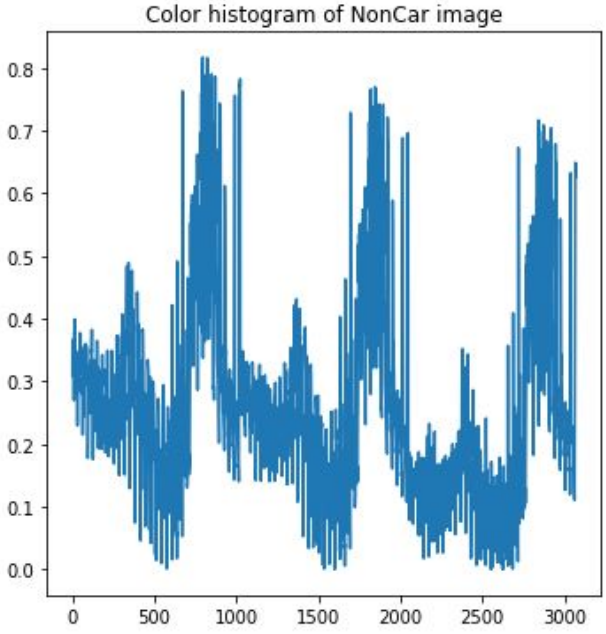


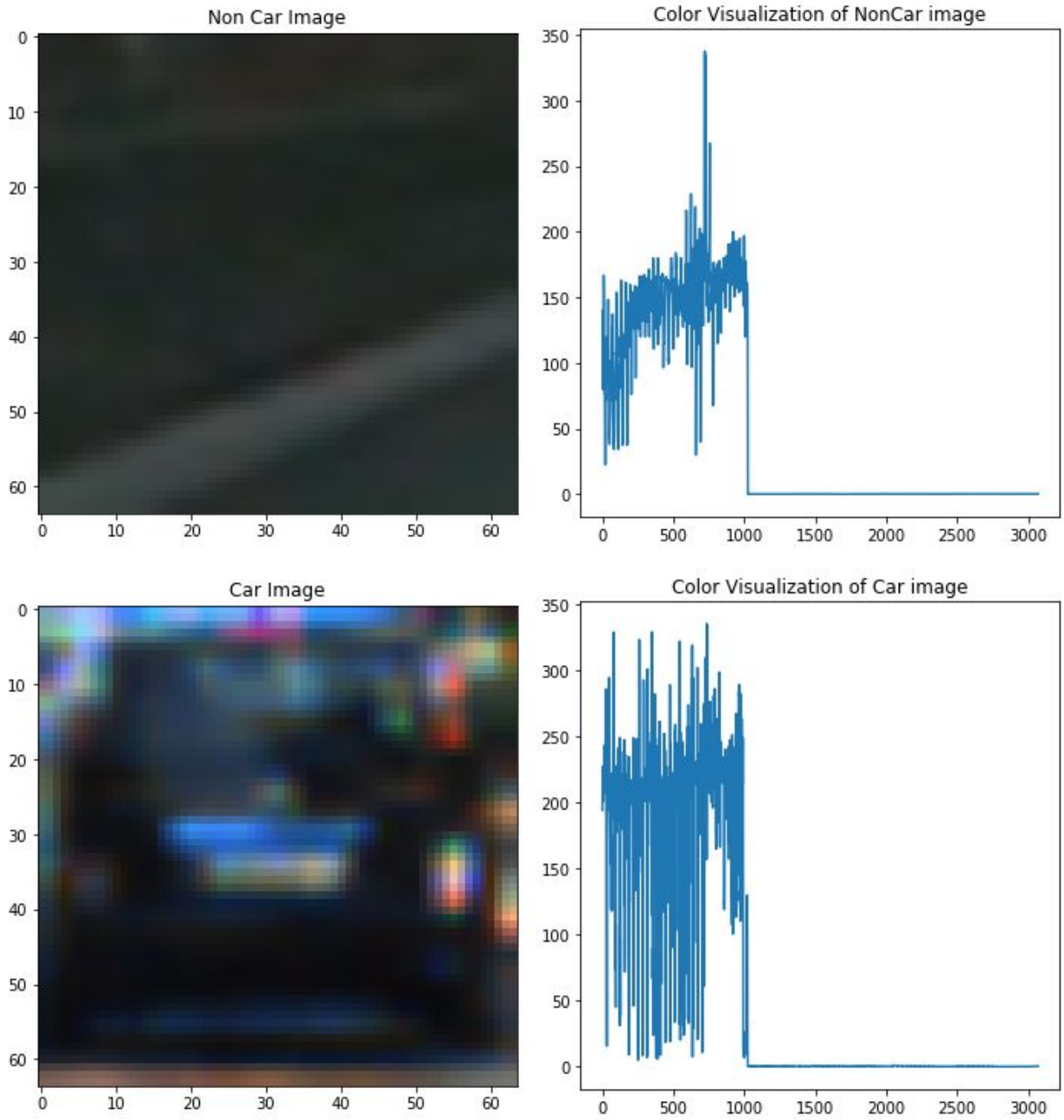
Using recommended parameters from class assignment (orient=9, pix_per_cell=8, cell_per_block=2), I obtained HOG transform using scikit function hog from(skimage.feature). There are 8792 car and 8968 non-car images. Randomly picking car and non-car image and running hog transform shows different signature.
Also, explored other features like color distribution and color histogram. Here how color distribution and color histogram looks for random car and non-car image.


Car Image
0
10
20
30
40
50
60
0
10
20
30
40
50
60


Color histogram of Car image
1.0
0.8
0.6
0.4
0.2
0.0
0
500
1000
1500
2000
2500
3000


Non Car Image
0
10
20
30
40
50
60
0
10
20
30
40
50
60


Color histogram of NonCar image
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
500
1000
1500
2000
2500
3000



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

Now to identify optimal parameters and features, I build a classifier and compare the score with different parameters.

Just with color features the classifier accuracy was 90%(In block 17 in Ipython notebook) .

Using Hog classification with default Hog parameters (hog_channels =ALL, 'orient' = 9 'pixels_per_cell' = 8 and 'cells_per_block' = 2) accuracy was 95.38 %

As we can see combining all the features (color and hog) provides better classification. Now I iterated all possible parameter one at a time to get best value for each parameter.

1. Color List - I tried 'HLS', 'HSV', 'RGB2YCrCb', 'BGR2YCrCb' colorspace found RGB2YCrCb best.
   Accuracies:
   Color = HLS, Accuracy=0.9479
   Color = HSV, Accuracy=0.9502
   Color = RGB2YCrCb, Accuracy=0.9614
   Color = BGR2YCrCb, Accuracy=0.9552

2. Hog Channel : Using RGB2YCrCb colorspace, tried all channel 0,1,2 and 'ALL'. 'ALL' gave the best accuracy.
   Accuracies
   HOG Channel = 0, Accuracy=0.9209
   HOG Channel = 1, Accuracy=0.9195
   HOG Channel = 2, Accuracy=0.9029
   HOG Channel = ALL, Accuracy=0.9651

3. Hog Orientation: Using RGB2YCrCb colorspace and 'ALL' hog channel, tried orientation value of 5,9,11,15. Orientation 11 gave good accuracy.

4. Pixels_per_cell: Using RGB2YCrCb colorspace, 'ALL' hog channel and Orientation of 11, tried 4,6,8,12 pixel lists. Value of 8 gives best accuracy.
5. Cells_per_block: Using RGB2YCrCb colorspace, 'ALL' hog channel , Orientation of 11,pixels per cell of 8 tried cells per block of 2, 4, 8. Value of 2 provided best accuracy.

Finally using best parameters

- Colorspace = RGB2YCrCb
- Hog_channel = 'ALL'
- Orientation = 8
- Pixels per cell = 8
- Cells per block = 2

The classifier provided accuracy of 98.11%. For the rest of the project, I will be using these parameters and classifier.

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

Build_classifier_with_feature_extraction method implements classifier. First extract all features for car and non-car and store them in a car and non-car list. Normalize this data using standardScaler (from Sklearn). Using train_test_split from sklearn.model_selection, split the data into normalized data into 80% training and 20% test. As suggest in the class, I build a Linear SVC from sklearn.svm and fitted with training data and accuracy was obtained using test data. Since scaler, classifier and accuracy are going to used extensively, this function returns scaler, classifier and accuracy.

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?

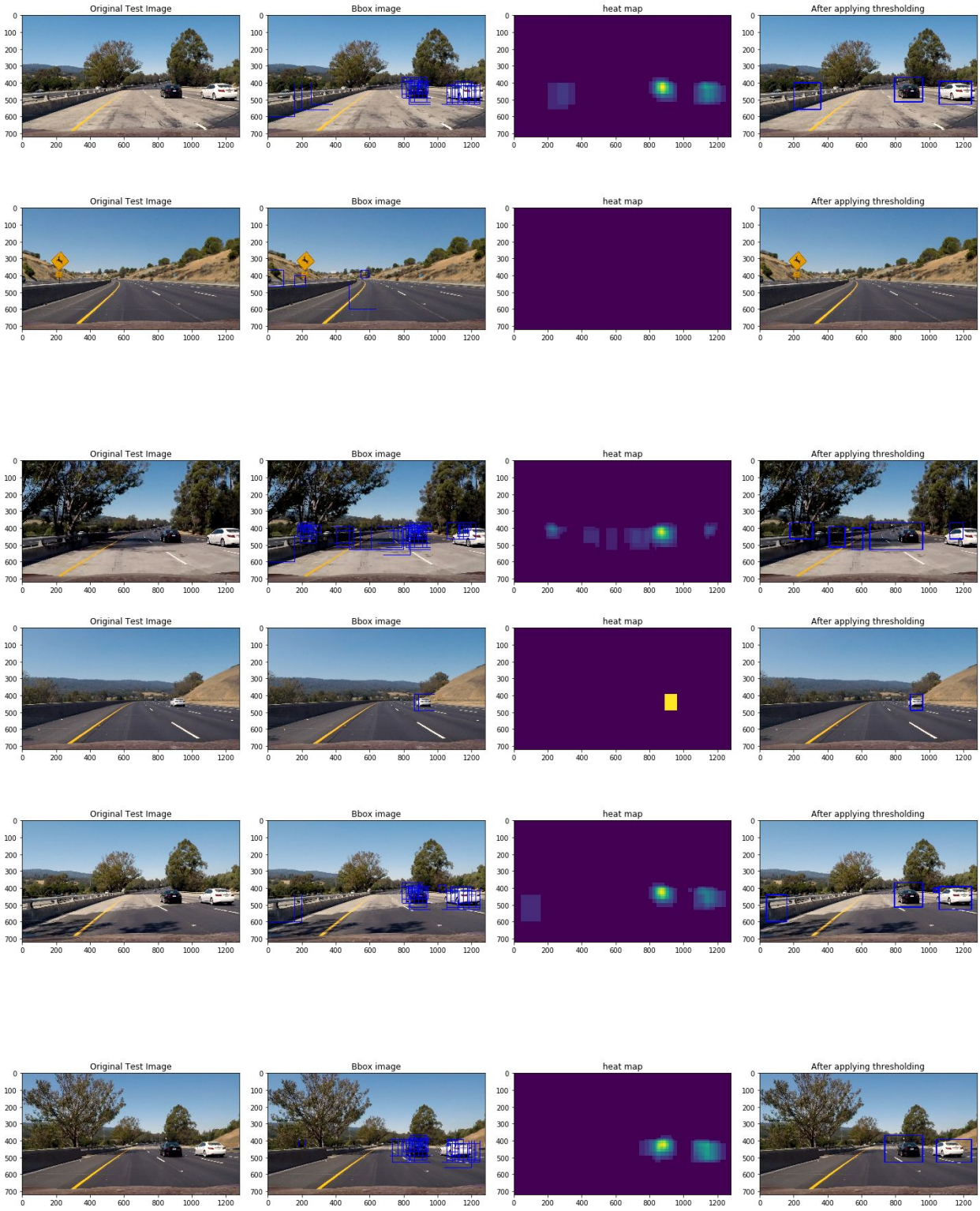
find_cars() in feature_extraction_classifcation.py file implements sliding window search. This is similar to example code provided in the class.
Using scale of 1.5, ystart=400, ystop=656, here output of test images.

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

Ran find_cars() with different sliding window values.

- Scale 0.7 with ystart = 370 and ystop = 450
- Scale 1.0 with ystart = 370 and ystop = 490
- Scale 1.5 with ystart = 370 and ystop = 550
- Scale 2.0 with ystart = 400 and ystop = 600
- Scale 2.5 with ystart = 400 and ystop = 650

After applying heatmap and threshold of 1, got the following results for test images.



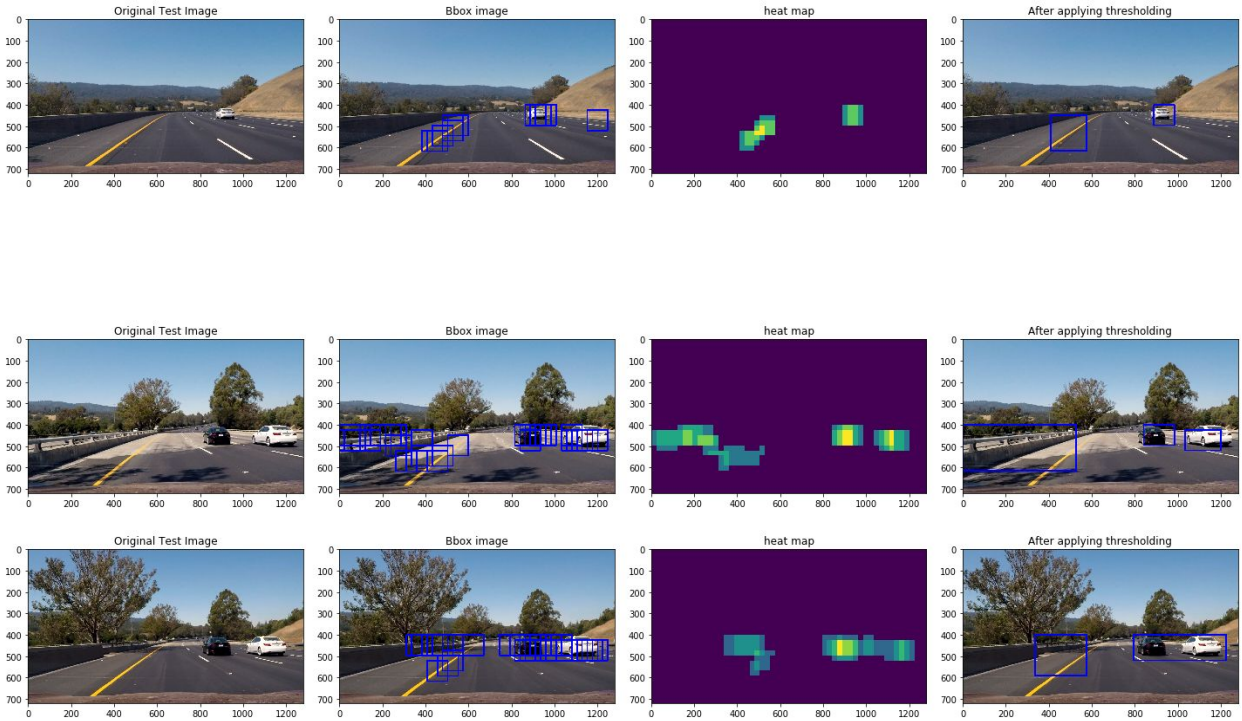
Provide a link to your final video output. Your pipeline should perform reasonably well on the entire project video (somewhat wobbly or unstable bounding boxes

are ok as long as you are identifying the vehicles most of the time with minimal false positives.)

Attached in the submissions (vehicleDetection.mp4). In the pipeline, smoothed the images by doing bounding box calculation every 3 iteration. To suppress false positives at the same time identify car at a distance (test_image3) was a challenge. Based number of bounding boxes identified I applied different threshold. This helped to supress some of false positives and recognize the true positives.

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.

Using classifier build with colorspace of 'BGR2YCrCb', found following false positives on test images.



Modifying the classifier with colorspace of ‘RGB2YCrCb’, fixed some of the false positives. Used different dimension of windowing to get best results.

- Scale 0.7 with ystart = 370 and ystop = 421
- Scale 1.2 with ystart = 370 and ystop = 460
- Scale 1.5 with ystart = 370 and ystop = 550
- Scale 2.0 with ystart = 400 and ystop = 600
- Scale 2.5 with ystart= 4-- and ystop = 650