Vehicle detection project

The goals and steps of this project are the following:

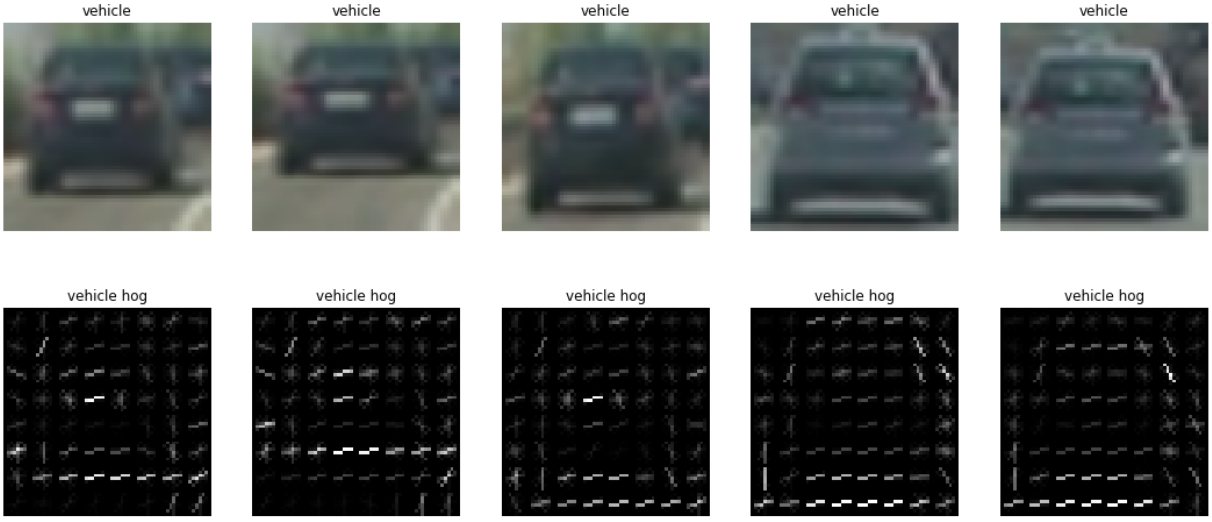
1. Perform a histogram of oriented Gradients (HOG) feature extraction on training data set and train a linear SVC classifier
2. Apply color transform and append binned color features to the HOG feature vector
3. Implement a sliding window technique and use linear SVC classifier to detect vehicles
4. Implement a complete vehicle detection pipeline and run it on a video stream
5. Draw bounding boxes for the vehicles detected

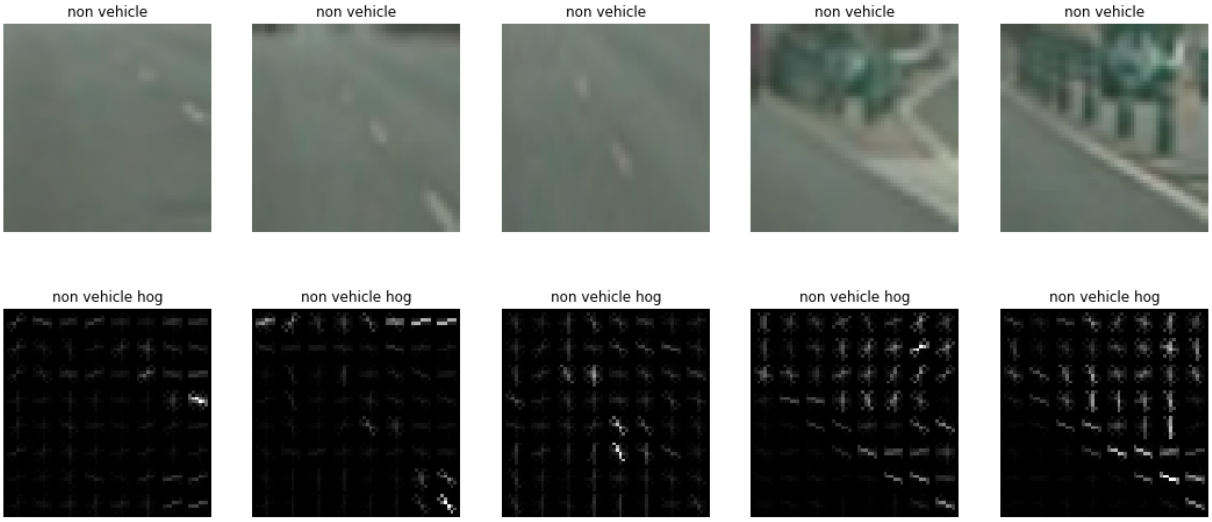
Rubric Points

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

Histogram of Oriented Gradients (HOG)

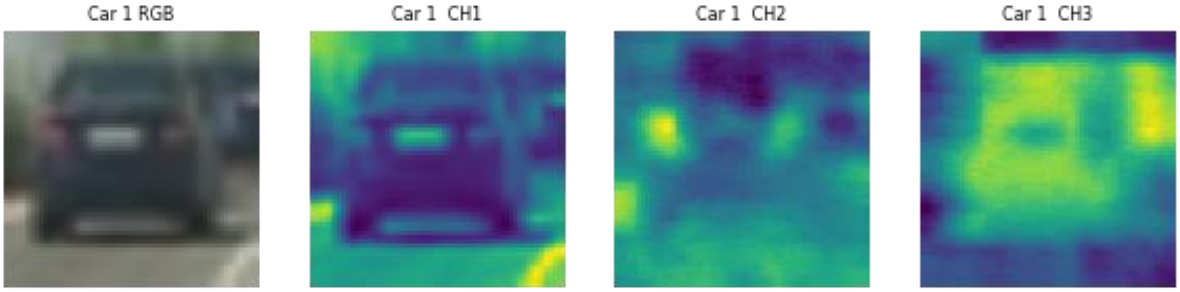
1. Explain how (and identify where in your code) you extracted HOG feature from the training images
   The code for this step is contained in second code cell. I first imported the car images and non-car images and visualized some parts of the car images and non-car images. Then I extracted the HOG features from images using the get_hog_features(). This method accepts images and parameters such as colorspace, orientations, pixels per cell, cells per block and Hog channels and return feature vectors and hog images.
   The figures below show car images and non-car images and its corresponding HOG images.

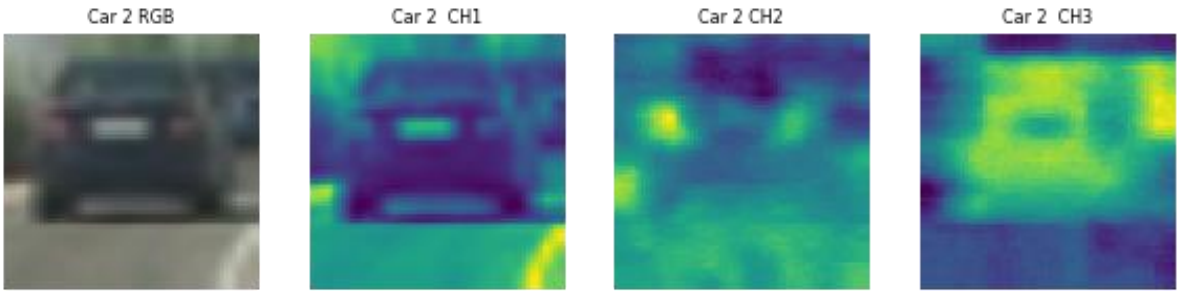




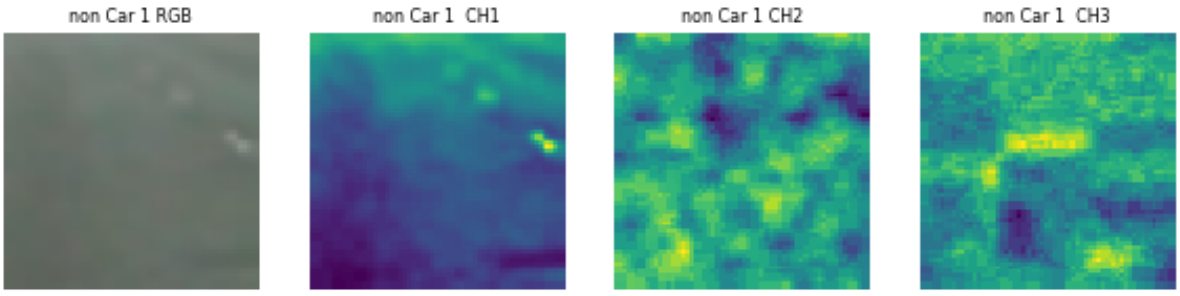
We can see the contours of the cars and non-cars from the hog images which helps to distinguish between cars and non-cars.

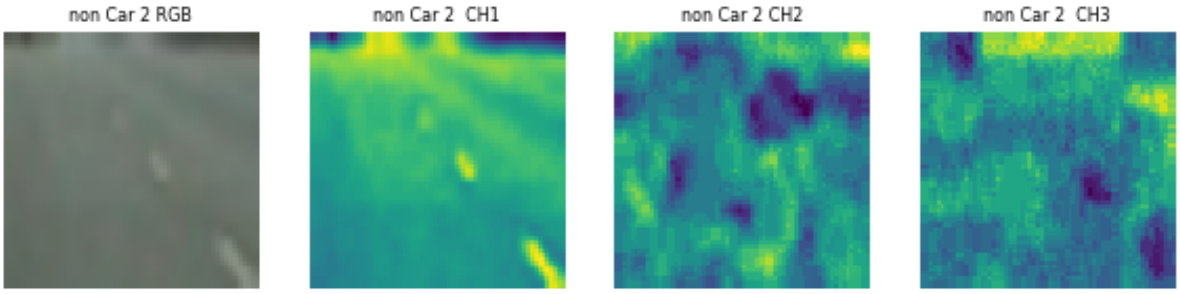
Then we explored various color spaces such as RGB, HSV, LUV, HLS, YUV and YCrCb. HSV and HLS did give us a clear contour of the car image. Compared to LUV and YUV, YCrCb color space gave me the best result.
Here is an example using YCrCb color space.





\





As we can see from the above color space images, the outlines of the cars are very clear.

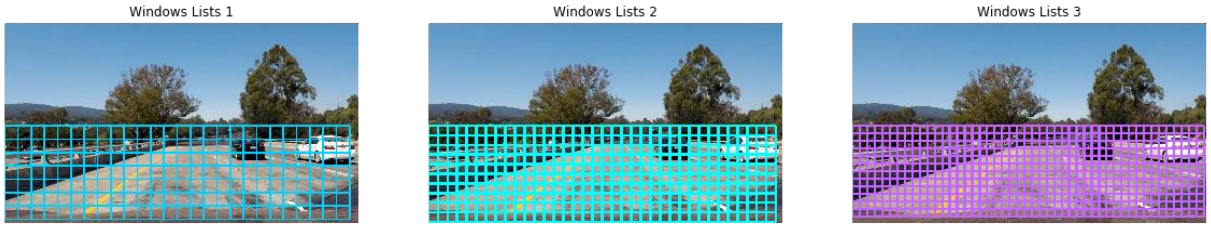
2. Explain how you settled on your final choice of HOG parameters.
   After several experiments, I settled on my final choice of HOG parameters based on the performance of the linear svc classifier. I tried a various combination of the HOG parameters. The final parameters are lists as follows:
   colorspace = 'YCrCb'
   orient = 9
   pix_per_cell = 8
   cell_per_block = 2
   hog_channel = 'ALL'
   hist_range = (0,256)
   spatial_size = (32,32)
   hist_bins = 32

I tried vaious values of orientations and found out that 9 orientations gave me the best result. Next, I tried different combinations for pixels per cell and cells per block and found 8 pixels per cell and 2 cells per block gave me the best result. Then I explored the hog_channel and colorspace parameters. The result shows that if the hog
Channel is 'all', the performance of the classifier is the best. Then I compared the three color spaces: YCrCb, LUV, and YUV. The YCrCb color space is the best. It has both the highest accuracy and least extracting time. Therefore, I choose to use YCrCb color space.

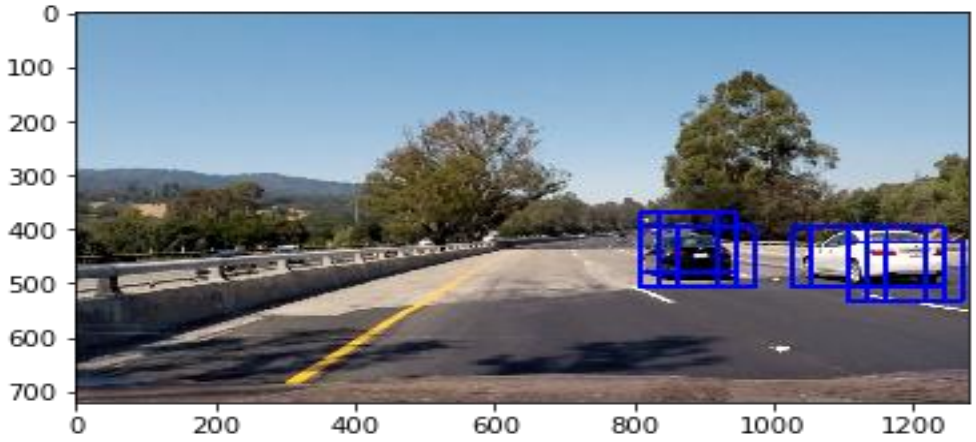
3. Describe how (and identify where in your code) you trained a classifier using your selected HOG features (and color features if you used them)
   The code for training classifier is in 14th code cell. I shuffle and split the data into training and test sets. The extracted features are feed to the linear SVC classifier using default classifier parameters. It takes 37 seconds to train the SVC classifier and the test accuracy is 0.9904 which is pretty good. I then save the classifier svc and X_scaler for later use

Sliding Window Search

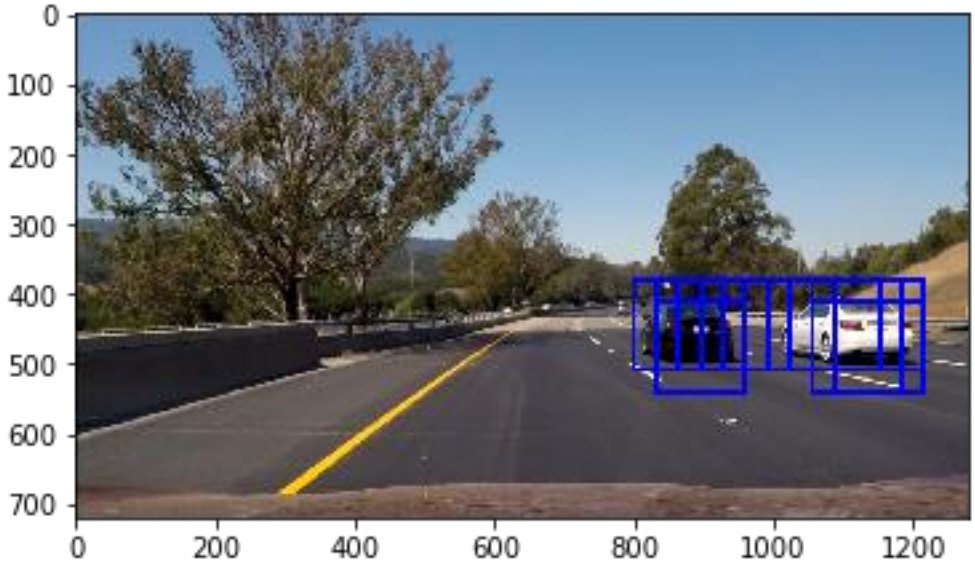
1. Describe how (and identify where in your code you implemented a sliding window search. How did you decide what scales to search and how much to overlap windows?
   The sliding window search method is under the title sliding window search. I first implemented a sliding window search method to scan through the image to identify cars. The figure below demonstrates the sliding window search method.



   The images show three different window sizes and overlap fractions. After some experiments, the overlap fraction of 0.75 and window sizes of (110, 110) gave me the best result.
   The figure below shows the detected cars using sliding window search.



However, there is a better sliding search method called Hog sub-sampling window search. This method is under the title hog sub-sampling window search. In this method, the hog features are extracted for the entire image and these features are subsampled based on the size of the window. This method performs prediction on extracted features and return a list of bounding boxes. The hog sub-sampling window search method is proven to be more efficient.
Here is an hog sub-sampling window search method example.



I experimented with scale parameter in hog sub-sampling function and found out that if the scale is less than 1, the accuracy of car detection is relatively low. Therefore, the scale parameter should be within the range of (1,2.5). In addition, I found out that combining several scales together makes pipeline work better. Therefore, I apply multi-scale method in my final detection pipeline.

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?

I searched on four scales using YCrCb 3 channel HOG features plus spatially binned color and histograms of color in the feature vector. This pipeline works well.

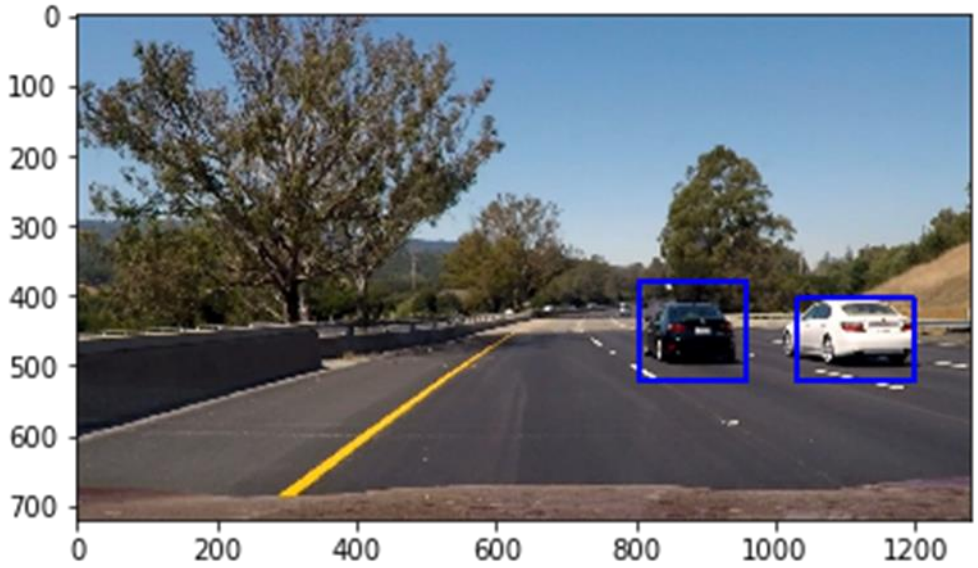
Here are some examples of test images.


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



I smoothed the vehicle detection pipeline by averaging the heat maps over the last 5 frames. As a result, the bounding boxes in the output video become more stable.

Since I found out that using one scale to search for detected cars is not sufficient, I applied multi-scale method in my pipeline. As a result, cars are detected in every video frame.

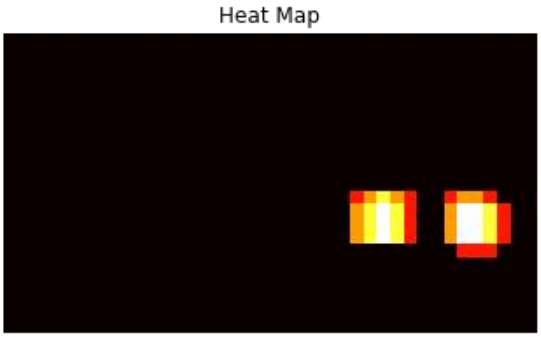
I also applied heatmap and thresholding decision function method to my pipeline. The linear SVC classifier has a method decision_function which return the distance from the decision boundary. I set it equal to 0.3 and this helps to remove false positives. This is the best way to remove false positives. The heatmap is applied to the pipeline by adding heat to a map for a list of bounding boxes. By applying the threshold to the heatmap, the false positives can be removed.

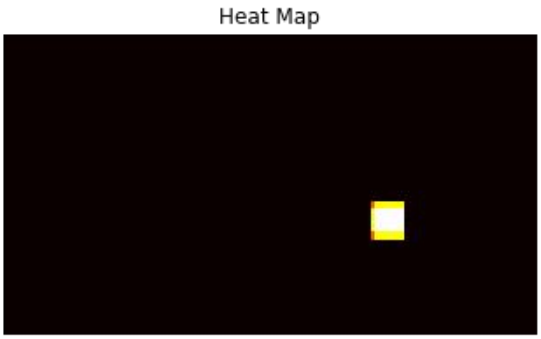
Video implementation

1. Provide a link to your final video output.

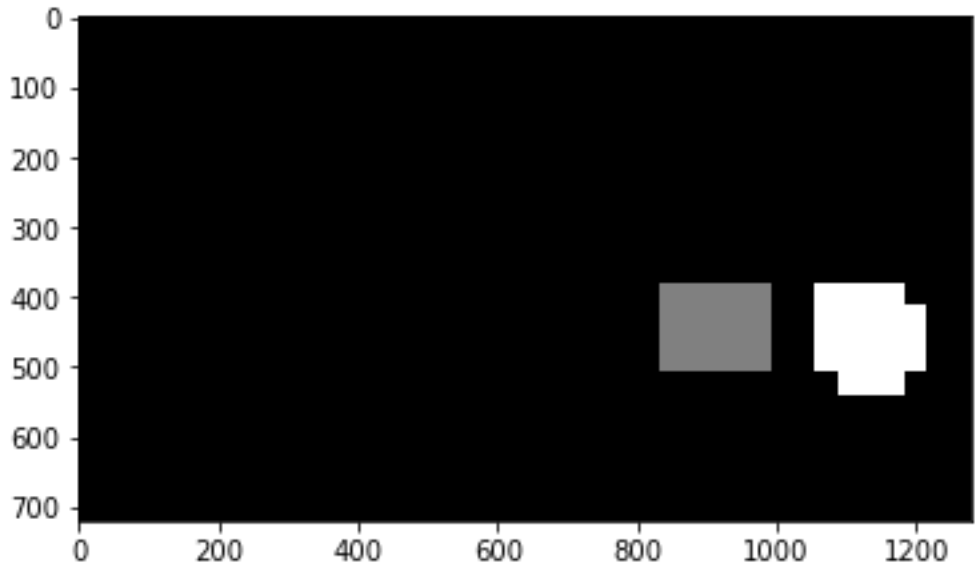
   Final video output is in the zip file

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.
   I applied thresholding decision function method to remove false positives. If the detection item is far away from the decision boundary, then the classifier is more certain that the detection item is a car. This helps to remove false positives and improves pipeline performance.
   I also applied heatmap method which is in 26th code cell. I first applied add_heat() function to add heat to a map for bounding box list. Then I applied threshold to the heatmap and reject false positives. Here I set threshold equal to 2. After thresholded heatmap was generated, I applied label function and put bounding boxes around the label regions.
   Here is an example of heatmap image.





   Here is an example of the output of the label function



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?
    1. Performance: overall, the performance is good. However, there are still two false positive. Ideally, these false positive should be dealt with. But, I have tried my best to minimize false positives. The speed of feature extraction and classification is quite slow.
    2. Decision function: I first tried using heatmap method to deal with false positives. However, there are still many false positives generated in the output video. Finally, I found that the decision function method of the linear SVC classifier can remove most of the false positives. This decision function method is a very powerful tool for improving the accuracy of the vehicle detection pipeline and reject false positives.
    3. Smoothing: Before I take the average of the heatmap over the last n frames, the bounding boxes are not stable. After I take the average, the output video and bounding boxes become more stable.
    4. Potential improvements: The dataset I used to train the classifier is not large enough. If there is more training data in the dataset, I believe that the vehicle detection pipeline will perform better. Another improvement is that using a convolutional neural network will help to identify important features and improve the performance of vehicle detection pipeline.
    5. Problems: When I trained my linear SVC classifier, sometimes the training process just stop. In the future, we need to find a way to solve this problem.