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.

I then explored different color spaces and different skimage.hog() parameters (orientations, pixels_per_cell, and cells_per_block).

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

I took an approach of trial and error and the came to the conclusion that you see in the third cell.

My goal was to get a result as good as possible on test images and the video, based on my own eyes

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 used SVM to train the features. Before that I normalized the data using StandardScaler

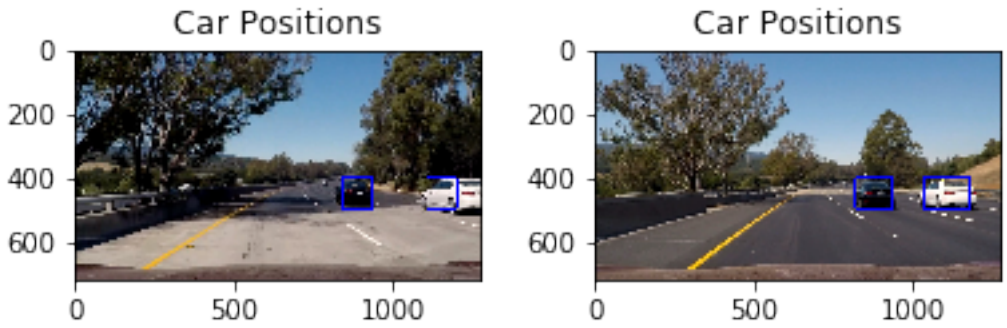
Sliding Window Search

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?

In the 4th cell I created my sliding windows.

First I tried with only one size for the windows but I figured out that this would not be the best idea. The cars that are far away are much smaller than those that are closer. Having a size of (64,64) would take a lot to find the cars, so I created other windows with a bigger size(96,96) that would search for cars that are closer. I ended up with 4 sizes searching in different zones

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?



Ultimately I searched on two scales using YCrCb 3-channel HOG features plus spatially binned color and histograms of color in the feature vector, which provided a nice result. Above are some example images:

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 put the video in the attachment.

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 recorded the positions of positive detections in each frame of the video. From the positive detections I created a heatmap and then thresholded that map to identify vehicle positions. I then
used scipy.ndimage.measurements.label()to identify individual blobs in the heatmap. I then assumed each blob corresponded to a vehicle. I constructed bounding boxes to cover the area of each blob detected.
In order to avoid false positives, I calculated the average of the last 5 heatmaps.

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?

When I started the project, I was expecting to have to implement a lot of filters against false positives, but in the the project_video worked just fine. I assume

that for other videos I will have to do more work on that, and also that I should predict the future position of the car to be able to track the cars faster(in real time)