Self-Driving Car Project #5

Vehicle Detection and Tracking Project

Complete on April 22nd, 2018

Haowei Zhang

Part 1 Histogram of Oriented Gradients (HOG)

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

Response:
The code for this step can be found in the submitted IPython Notebook at 2nd, 5th and 6th cell. I started by loading the training set for vehicles and non-vehicles images and here is an example of the vehicle and non-vehicle class.

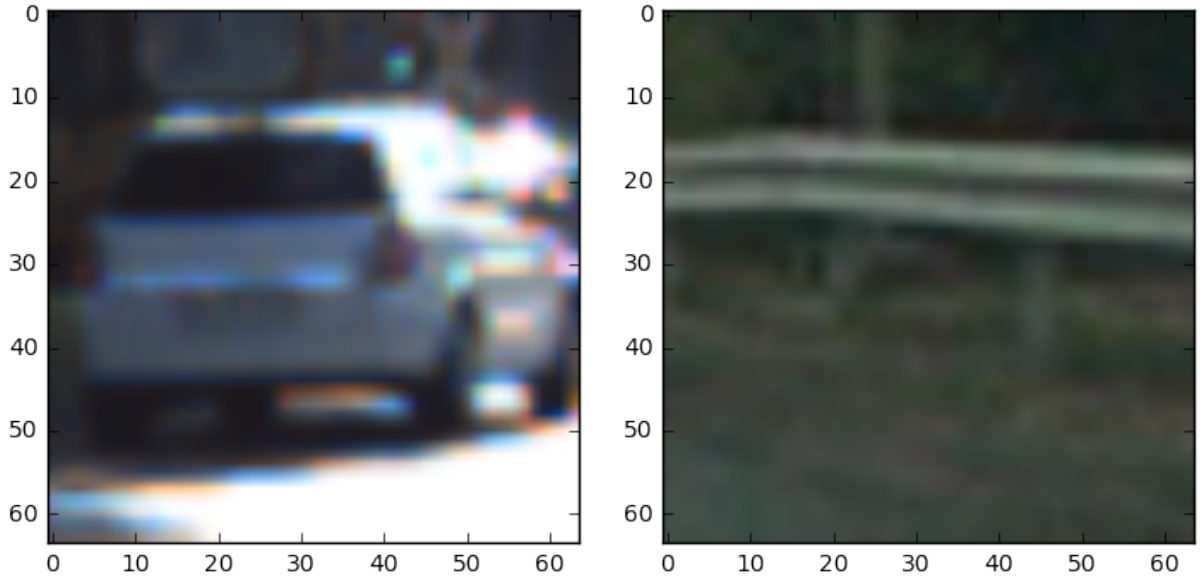

Figure 1: **Example figures in the training set** (left): vehicle image (right): non-vehicle image

After exploring multiple color spaces and different HOG parameters, e.g orientations, $pixels\_per\_cell$, and $cells\_per\_block$, I determined the following parameter setting as the one I used for this project. With an $orientation = 8$, $pixels\_per\_cell = (8, 8)$ and $cells\_per\_block = (2, 2)$, I was able to generate the following figures.
Task 2:
Explain how you settled on your final choice of HOG parameters.

Response:
I played with different sets of HOG parameters, e.g with $pixels_per_cell = (16, 16)$, and the features extracted became unclear and lose characteristics of the vehicle. Also, I tried it throughout the entire pipeline too and the one with $pixels_per_cell = (8, 8)$ ends up with best result.

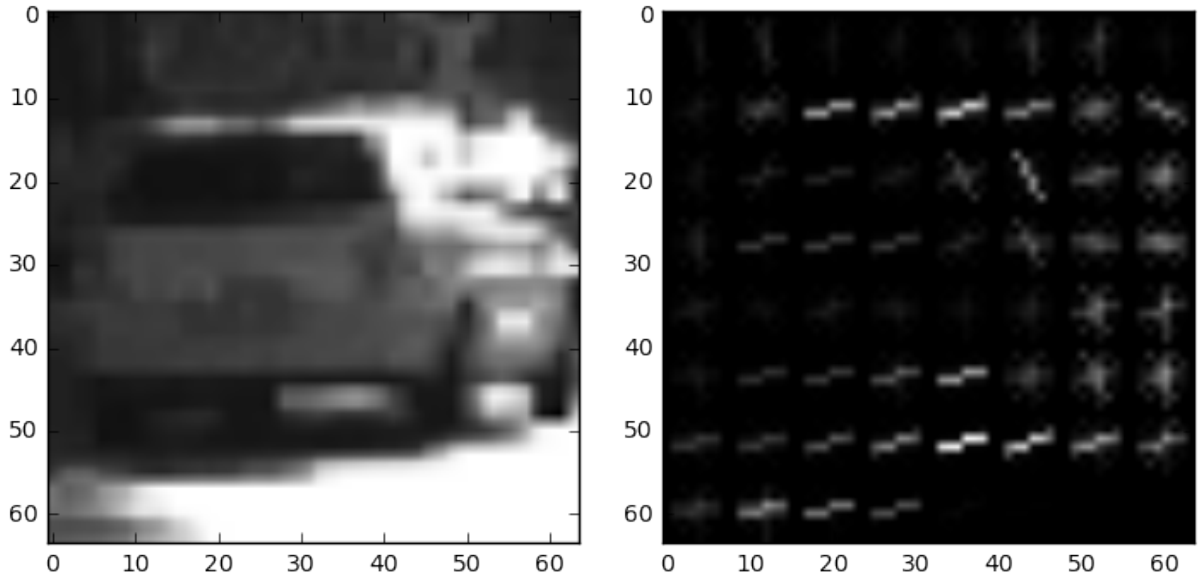

Figure 2: **Example figures before and after HOG** (left): vehicle image (right): vehicle image after HOG

Task 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).

Response:
The code for this step can be found in the submitted IPython Notebook at 2nd and 7th cell. I extracted HOG features and color features in color space $YCrCb$. I used all 3 channels of HOG features and concatenated into a long features vector. Also, I resized the image into $(16, 16)$ and used all 3 color channels to generate a spatial feature vector. What's more, I used histogram stats to form a histogram feature vector after resizing the image. After concatenating these three feature vector, I got a long feature vector for my training classifier.

I used all the data set available from Udacity course, which is GTI and KITTI data set, as positive training samples and all the others as negative samples. I split 20% as test set and used the rest as training set. After performing previous operation, I got a feature vector, i.e more of a matrix, of size $(14208, 6108)$, where each feature vector is 6108 long and there are 14208 training samples.

I trained the classifier with Linear Support Vector Classifier. The training accuracy ends up with 99.21%.

Part 2 Sliding Window Search

Task 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?

Response:
The code for this step can be found in the submitted IPython Notebook at 2nd and 8th cell. Using the provided code from Udacity, I was able to experiment with different scales and following are few examples.

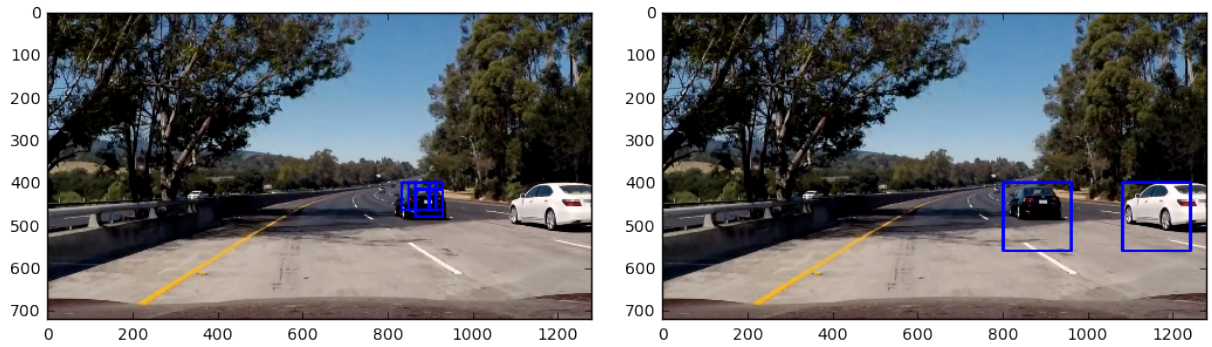

Figure 3: **Test image with different scale factor** (left): scale factor of 1.0 (right): scale factor of 2.5

It is clear from the figures that with a larger scale factor, I am able to search over a larger space. However, for vehicles far away in the picture, this would NOT be able to detect it. Therefore, it is significant to combine multiple scale factors to account for different scales of the vehicle.

Instead of considering how much to overlap an image, I take account into how many cells to shift. This is more intuitive as I shifted 2 cells from this to next window. Since the block has 8 cells in it, the overlap would be 25%.

Task 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?

Response:
At the very beginning, I did NOT train the classifier with all the available data set, but a subset of it. The classifier works badly on the test images and could hardly detect the vehicle. I resolved this issue by taking advantage of the entire training data set.

Ultimately after testing with different settings, I ended up searching over 7 scales, i.e from 1 to 4 per 0.5 step, using YCrCb 3-channel HOG features plus spatially binned color and histograms of color in the feature vector. The setting provided a nice result and here are some example images.

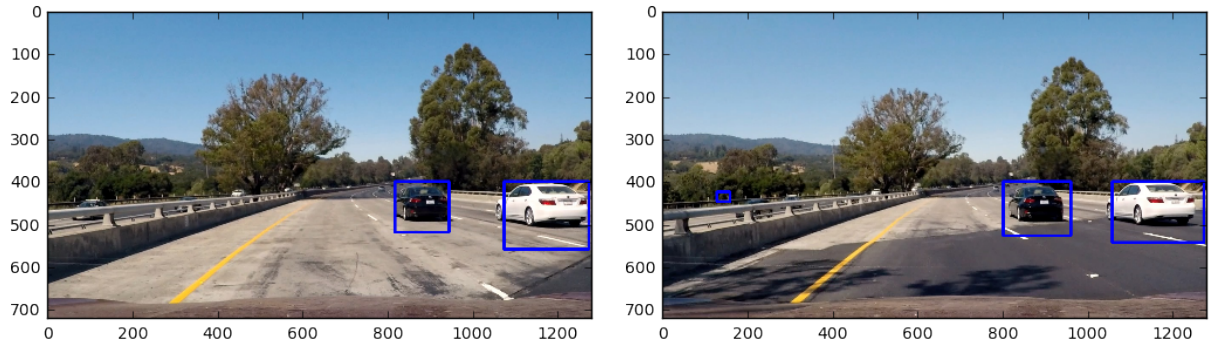

Figure 4: **Test images with vehicles detected in the bounding box** (left): test image No.1 (right): test image No.2

Part 3 Video Implementation

Task 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.)

Response:
The video can be found at the same directory within the Github repository.

Task 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.

Response:
I recorded the positions of positive detections in each frame of the video. From the positive detections, I created a heat map and thresholded that map to identify vehicle positions. I then used *scipy.ndimage.measurements.label*() to identify individual blobs in the heatmap. I assumed each blob corresponded to a vehicle. I constructed bounding boxes to cover the area of each blob detected.

Here's an example result showing the heat map from a series of frames of video, the result of *scipy.ndimage.measurements.label*() and the bounding boxes then overlaid on the last frame of video:

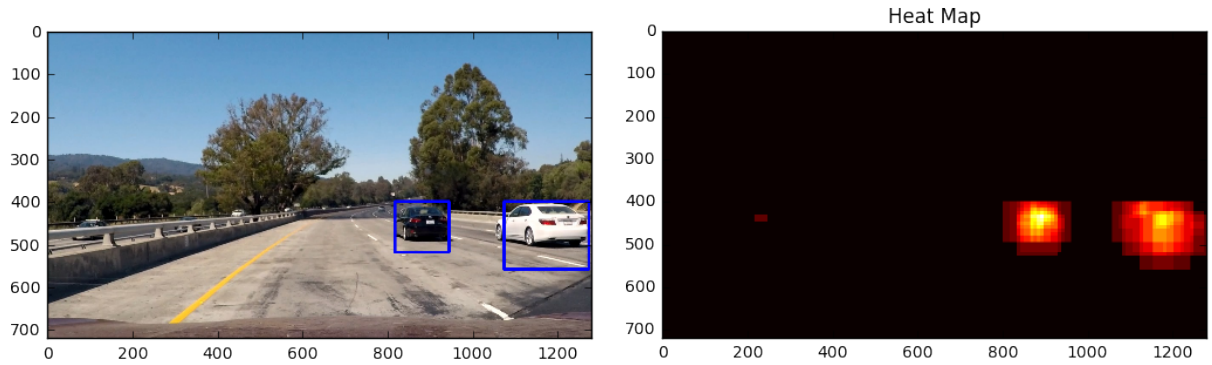

Figure 5: **Vehicles detected with heat map** (left): test image with bounding boxes (right): test image filtered with heat map

And this is the final result drawn from the heat map.

Figure 6: Vehicles detected using heat map

Part 4 Discussion

Task 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?

Response:
There are multiple tasks I had faced when trying to implement this algorithm.

1) I experimented with different settings of HOG, colors and histograms. The experimentation was fun in a way that I began to understand the importance of each parameter.

2) There are outliers in the frame that I was unable to get rid of. Those are probably failures from the classifiers and I did NOT figure out the best way to get rid of it using the algorithm. This should be something that I should work on to make it more robust.

3) It would be great if I could incorporate past frames together to make the algorithm more robust.