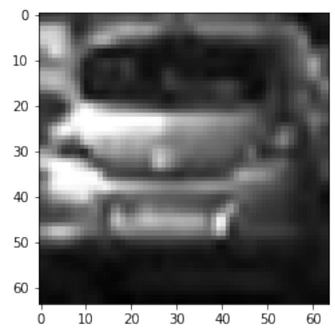
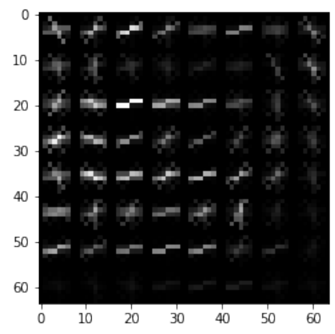
Self Driving Car Nanodegree Project 5

Michael Person

May 13, 2017

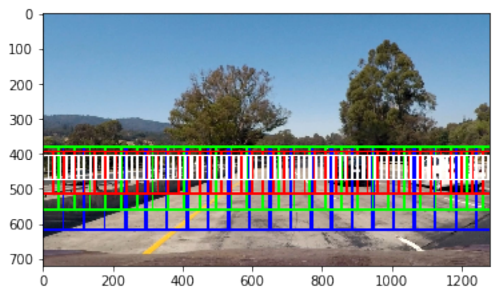
1 Histogram of Gradients

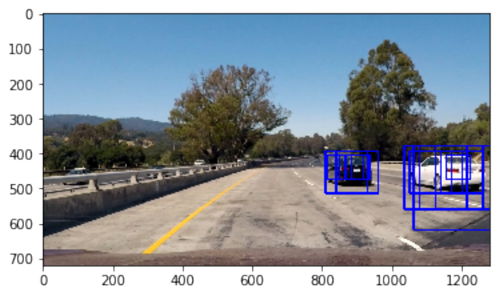


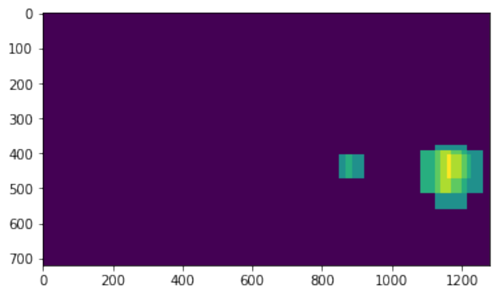


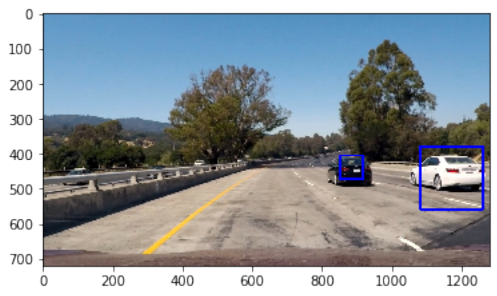
In the feature_classification ipython notebook I extracted hog, color, and spacial features from the training set in order to learn what is a vehicle and what is not. Above is an example of what the hog features look like of a vehicle. The colorspace I used was the YCrCb colorspace and I choose this because I found it gave me best classification rate over the other colorspaces granted it was minimal. To extract my hog features I used a spacial size of 16 by 16 pixles because I was only interested in local pixel intensities and did not need to search any larger window than that. I also used only 9 orientation bins since I found that much more bins than that gave me little improved descriptive power of my image and hindered classification due to increased dimensionality. I used 8 pixels per cell and 2 cells per block. I decided to use all channels of my image in order to capture all the possible information I could about the image. I also scaled all the features using the StandardScalar and transform function from the sklearn library. Finally I train a support vector machine (SVM) to classify whether these scaled features belong to a vehicle or not. I achieve 99% validation accuracy.

2 Sliding Window Search









I implemented a sliding window approach that generated windows of progressively smaller size as they appeared higher in the scene. The logic for this was that cars that are higher in the scene are farther away, will appear smaller, and therefore will fit inside a smaller box. I then extract the spacial, color, and hog features from each of these ROIs and use the trained SVM to classify whether it is a vehicle or not. In order to improve the stability of my hot windows, I use a heatmap of my found boxes to combine how they overlap and then threshold the values in order to only have 1 single box per vehicle or at least make it closer to that.

3 Video Implementation

For my video implementation I combined the 4th and 5th projects to find both the lanes and also to find the vehicles. In order to cut down on false positives and keep tighter bounding boxes around the vehciles, I implement a box class that keeps tracks of past found box frames. Rather than only create the heatmap from boxes found in this frame, I store the past 3 found boxes and use those to form the heatmap. This makes sure that the boxes are tight around the vehicles and also cuts down on false positives.

4 Discussion

One of the major problems I faced was speed of my algorithm. Granted I combined the lane finding and vehicle detection into the same pipeline but it took a very long time to complete. I believe that a good solution to this problem would be to use CUDA to perform most of the image processing in order to find the lanes since opencv has a good gpu api and therefore only a few custom cuda kernels would need to be written to complete the lane finding portion. Also for the vehicle detection portion, I would like to use a CNN and deploy it with TensorRt. Since once the network is trained, a single foreward propgation of the ROI is a series of matrix multiplication of Toeplitz matrices to represent the convolutions, the computation will be extremely fast, and it could learn more efficient features to look for rather than just spacial, color, and hog.

One of the places that my current pipeline will fail at is if there are many vehicles in the frame. Since I specified that I only use 3 boxes per frame in order to get very tight bounding boxes around the vehicles, my pipeline would not be able to find more than 3 vehicles which of course is a major flaw that would need to be addressed moving forwards.