Vinh Nghiem

Project 5 : Writeup / README

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 the IPython notebook P5.ipynb, in “Extract Features”: lines # 40-62 in P5.ipynb.
All the `vehicle` and `non-vehicle` images are first read in. Here is an example of one of each of the `vehicle` and `non-vehicle` classes:

I then explored different color spaces (RGB, BGR, YCrCb, etc.) and different `skimage.hog()` parameters (`orientations`, `pixels_per_cell`, and `cells_per_block`).

Random images from each of the two classes were displayed them to get a feel for what the `skimage.hog()` output looks like.

Here is an example using the `YCrCb` color space and HOG parameters of `orientations=8`, `pixels_per_cell=(8, 8)` and `cells_per_block=(2, 2)`:

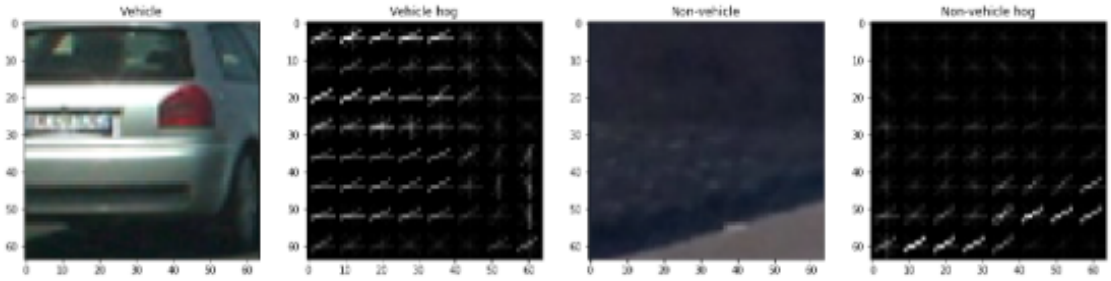
2. Explain how you settled on your final choice of HOG parameters.

I tried various combinations of parameters and chose HOG parameters that resulted in the highest classification test accuracy on an SVM linear classifier.

The final HOG parameters used were :

scale = 1.5, orient = 8, pix_per_cell = 8, cell_per_block = 2

Here is a sample of HOG features extracted from images :



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 trained a linear SVM classifier using HOG parameters extracted from the HOG parameters specified above. These HOG features were concatenated onto spatial features and color space histogram features. The linear SVM classifier was trained by scaling the concatenated features into 0 mean and unit variance, from splitting the data set into a 20% test set, then randomizing the objects in each set. The scaler was learned from only the training features, to avoid exposing the test set to the program. Subsequent training of all data into the classifier was then scaled through the scaler, for numerical stability.

The classifier was trained in “Train Classifier”: lines 1-25 of P5.ipynb.

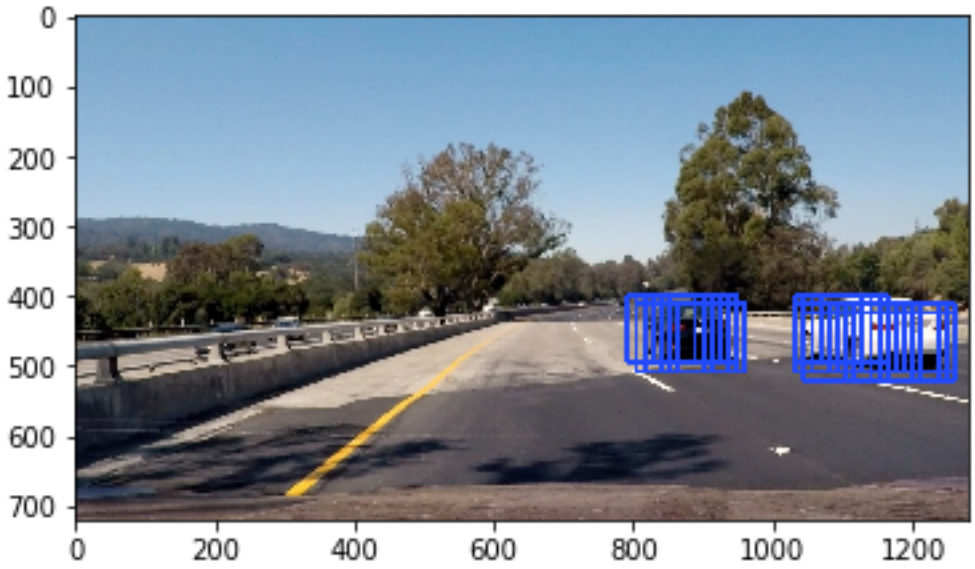
Sliding Window Search

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?

I used HOG Subsampling to efficiently extract HOG features and perform classification. I experimented with a few different parameters but discovered that the parameters in the suggested course code for HOG subsampling gave the best test accuracy for my classifier.

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 using YCrCb 3-channel HOG features plus spatially binned color and histograms of color in the feature vector, which provided a nice result. Here is an example image:



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.)

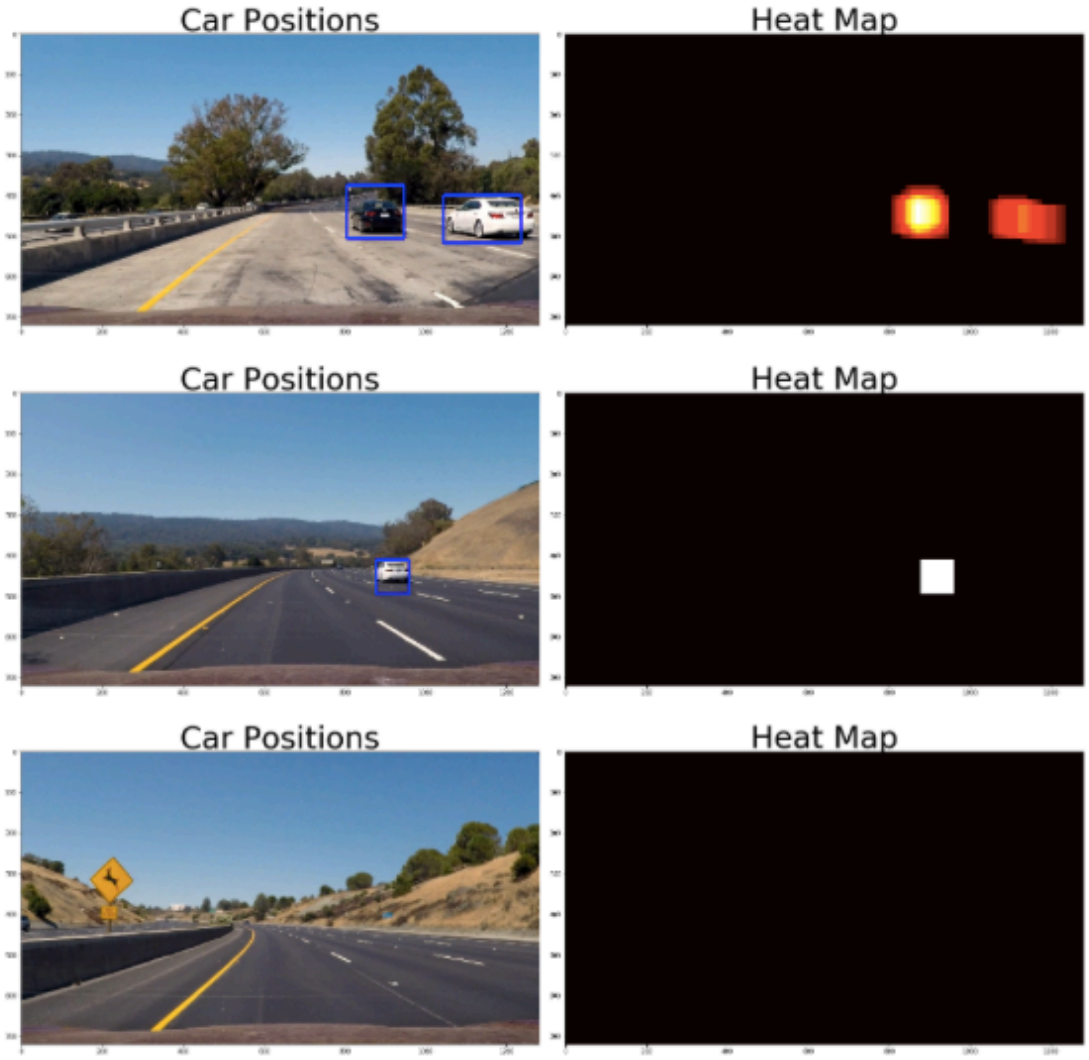
My video for vehicle detection is attached in the file project_video.mp4, while the video for both vehicle detection and lane detection is in challenge_video.mp4.

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.

The positions of all positive detections were made. False positives were handled with a thresholded heatmap. The heatmap was a counter at every pixel indicating the number of detected boxes bounding that pixel. The label() function in the scipy module was then used to identify contiguous heatmap regions, corresponding to individual vehicles**.**

The code for the heatmaps is in “Handling Multiple Detections and False Positives” in P5.ipynb.

Here are the thresholded heatmaps and resulting bounding boxes on three example frames :



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?

With the pipeline I used, I originally had a lot of false positives. There were a lot of regions on the left side of the road, along the yellow lane markings, in the test video, that got incorrectly detected as vehicles. Even very high thresholds in the heat map resulted in false positives in almost every frame. This was ultimately overcome by changing the block normalization parameter in the skimage.feature.hog() function from the preferred 'L2-Hys' in the course code to 'L1'. This was discovered only through trial and error and wholly unexpected, as 'L1' performed the worst in experiments by the original researchers of HOG.

The pipeline would likely fail in extreme lighting conditions or extreme shadows that have low representation in the training set. Also, other objects in the road that should be avoided, such as pedestrians, motorcycles, overhanging branches, shadows, road debris, etc would not have adequate training.

To make the pipeline more robust, more training of other other objects to be avoided, rather than just cars, should be used in the training set.