Project 5: Vehicle Detection Writeup
Written by: Eddie Ferrufino

Project Objectives:

- Perform a Histogram of Oriented Gradients (HOG) feature extraction on a labeled training set of images and train a classifier Linear SVM classifier
- Optionally, you can also apply a color transform and append binned color features, as well as histograms of color, to your HOG feature vector.
- Note: for those first two steps don't forget to normalize your features and randomize a selection for training and testing.
- Implement a sliding-window technique and use your trained classifier to search for vehicles in images.
- Run your pipeline on a video stream (start with the test_video.mp4 and later implement on full project_video.mp4) and create a heat map of recurring detections frame by frame to reject outliers and follow detected vehicles.
- Estimate a bounding box for vehicles detected

Histogram of Oriented Gradients (HOG)

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

   In code cell 6 from the file *VehicleDetectionProject.ipynb* I extracted HOG features from 6 random images from the provided dataset.(3 from Vehicles, 3 from Non-vehicles). I first read in all of the images from the project dataset using the glob function. Here are example images from the data set. In total there are 8792 images that contain vehicles, and 8968 images that do not contain vehicles.

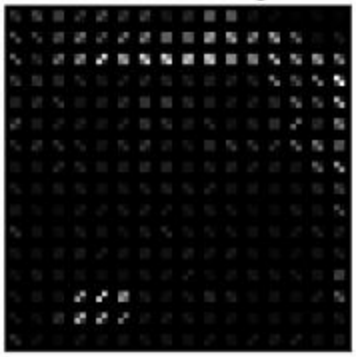
I used the HOG function from the skimage.feature library to extract a histogram of gradients from each image. I experimented with the following parameters - orient, pixel per cell, and cell per block to aid in gaining an understanding for how HOG worked. Reducing the pixel per cell seemed to produce a clearer image and increasing pixels per cell would make the image more abstract and look less like the original image. After experimenting it seems like the HOG feature is a manual way to extract features from an image such as curves and edges or boundaries. I can see how this information is useful in determining the presence or absence of a car. Below are a couple of examples of images and their corresponding HOG image.

Vehicle Image

HOG Vehicle Image

Vehicle Image

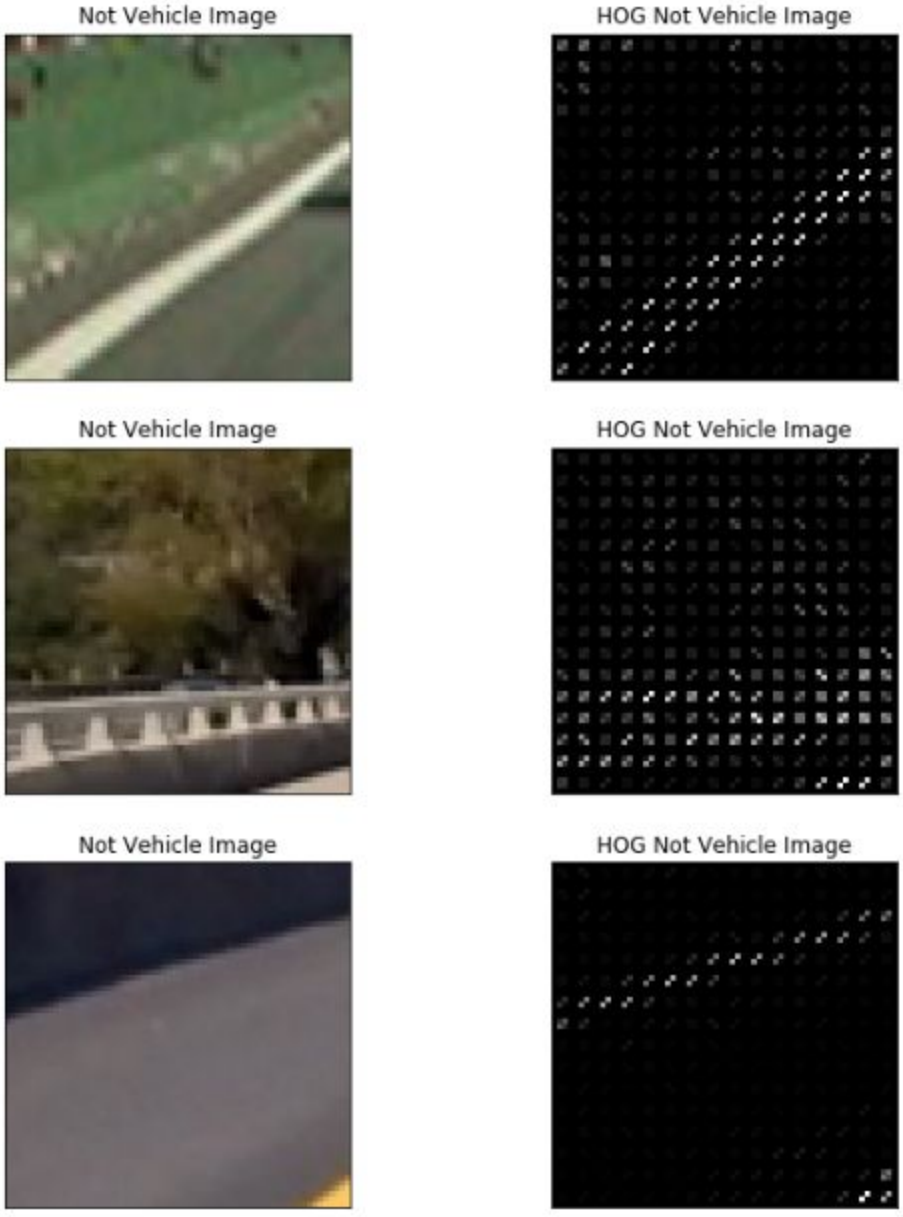
HOG Vehicle Image



Vehicle Image

HOG Vehicle Image

Below are some examples of images that do not contain vehicles. There is some very clear patterns that our classifier should be able to pick up on.



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

I tried various parameter configurations. I noticed that as I increased pixels per cell the image would become clearer or “high resolution”. I thought this may be useful for building a better classification model as it would allow for finer details to be captured. The lower pixel per cell came with a cost of slower classifier performance(<1FPS). In addition lowering pixel per cell

below 4 required more memory than my laptop could handle. For that reason I left pixel per cell at 4.

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 classifier in cell 10 of the VehicleDetectionProject notebook. In order to train the network I first extract all of the HOG features from the images and appended the results to two lists, car_features and notcar_features. I then normalized the data using the StandardScalar() function. I then created a car and notcar label set using np.ones and np.zeros functions. I then performed an 80/20 split of the data with 80% becoming training data and 20% becoming test data. The SVM achieved a test result of 99.07% accuracy. I will say that the notcar data set did contain some questionable samples with cars in the distance. I guess the objective of this particular dataset is to identify cars that are closeby and not far away or driving against traffic. I decided to use all of the channels of the HSV colorspace to provide more information to train the classifier by.

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 code cell 12 of the VehicleDetectionProject.ipynb I implemented a sliding window search algorithm. I selected four different window types to help search for the car. I decided to use four different window sizes to account for the different distances the target car could be from the camera sensor. The larger window size (160,160) is designed to detect cars that are close to the sensor. The smallest window size(100,100) is designed to detect cars that are furthest from the image sensor. The two intermediate window sizes are sized to capture cars that are not close or far from the sensor. The windows are focused on the lower right quadrant of the image where cars are most likely to be in this video. This helps to reduce the occurence of false positives. Window overlap was set to 75% in order to increase the chance of detecting a car. At 0% the amounts of vehicle detected was reduced significantly. At 99% overlap the chance of detecting a vehicle goes up significantly but performance suffers tremendously. The selected overlap percentage provide the best balance between pipeline execution speed and vehicle detection accuracy.

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?**

Below are some examples of my vehicle detection pipeline. In order to optimize the performance of the classifier I used 4 pixels per cell in the HOG feature extractor. This gave the classifier a very feature rich dataset to work with and to learn to classify vehicles. I was able to achieve a test accuracy of 99.07% with my test data.




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.)**

Below is a link to the final video implementation of my vehicle detection pipeline.

https://youtu.be/8oBlao0vNas

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.**

   In code cell 11 from the file VehicleDetectionProject.ipynb I implemented a heatmap function which generates a black image and adds a pixel value of 5 for any pixels within an identified car(hot window). I then applied thresholding by converting any pixel values below 5 to zero. This takes multiple boxes and combines them into a larger box.It also helps to reduce noise by merging boxes together.

   In order to filter out noise I specified a region of interest via the hard coded windows. The windows limit the area to search for cars. By limiting the search area to areas in the lower right quadrant I was able to filter out a lot of noise.

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?**

   Issue 1. There are a lot of false positives that are filtered out by the Windows region of interest. This makes my pipeline less general and work mostly on just this video. To make this more robust I would add more training data to my dataset(Udacity provides additional data) in order to build a more robust classifier.

   Issue 2. Cars on opposite flow of traffic are not detected. This is not good for a real scenario as cars coming from the opposite flow of traffic pose a big safety threat and an autonomous system should be able to perceive them.

   Issue 3. Cars that are far away are not detected. The cars that are near the horizon are not detected. Detecting cars further out will allow for more advanced path planning. In order to resolve this maybe we could take the region of the camera that is the horizon and “enhance” it via deep learning to provide more data that can be used to classify cars. I think the problem now is that cars that are far away are very little pixels. If we can turn a small amount of pixels into a larger amount of pixels then maybe we’d have a better chance of classifying far away objects.

   Here is an example project that has implemented such an algorithm.
   https://github.com/alexjc/neural-enhance