Project 5: Vehicle Detection Report

Vehicle Detection Project

The goals / steps of this project are the following:

- 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.

Rubric Points

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

Writeup / README

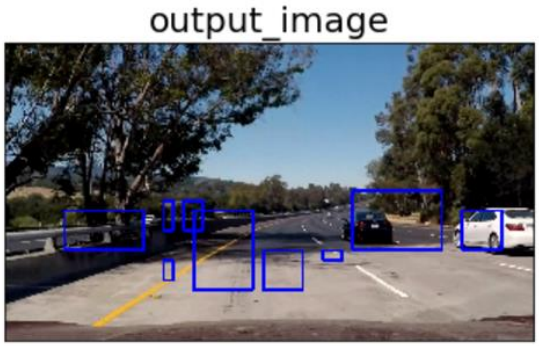
1. **Provide a Writeup / README that includes all the rubric points and how you addressed each one. You can submit your writeup as markdown or pdf. Here is a template writeup for this project you can use as a guide and a starting point.**

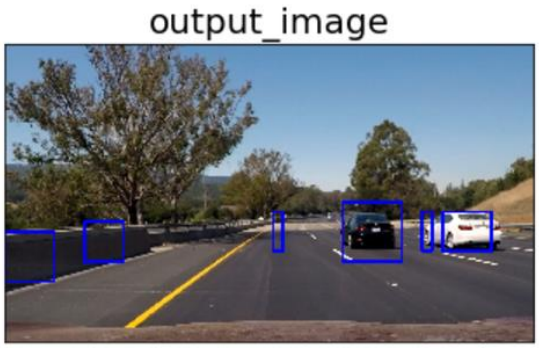
   You are reading it!

Histogram of Oriented Gradients (HOG)

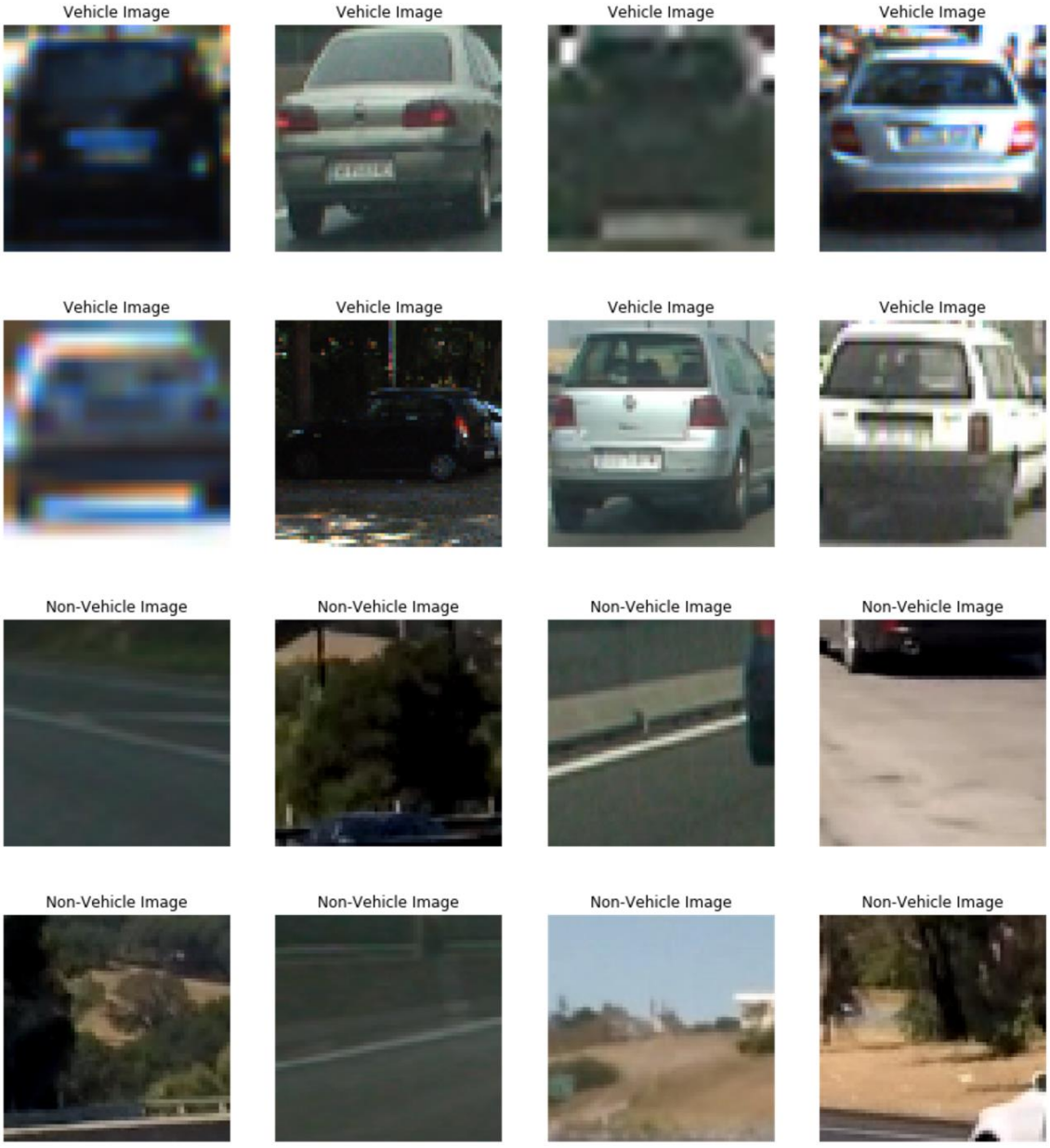
1. **Explain how (and identify where in your code) you extracted HOG features from the training images.**

I started by reading in all the vehicle and non-vehicle data provided in the template project folder, in addition to that, I have taken the extra step to obtain more training data from CrowdAI as instructed in the project instructions. However, I had to abandoned my plan of incorporating these extra data from CrowdAI due to undesirable performances such as these:

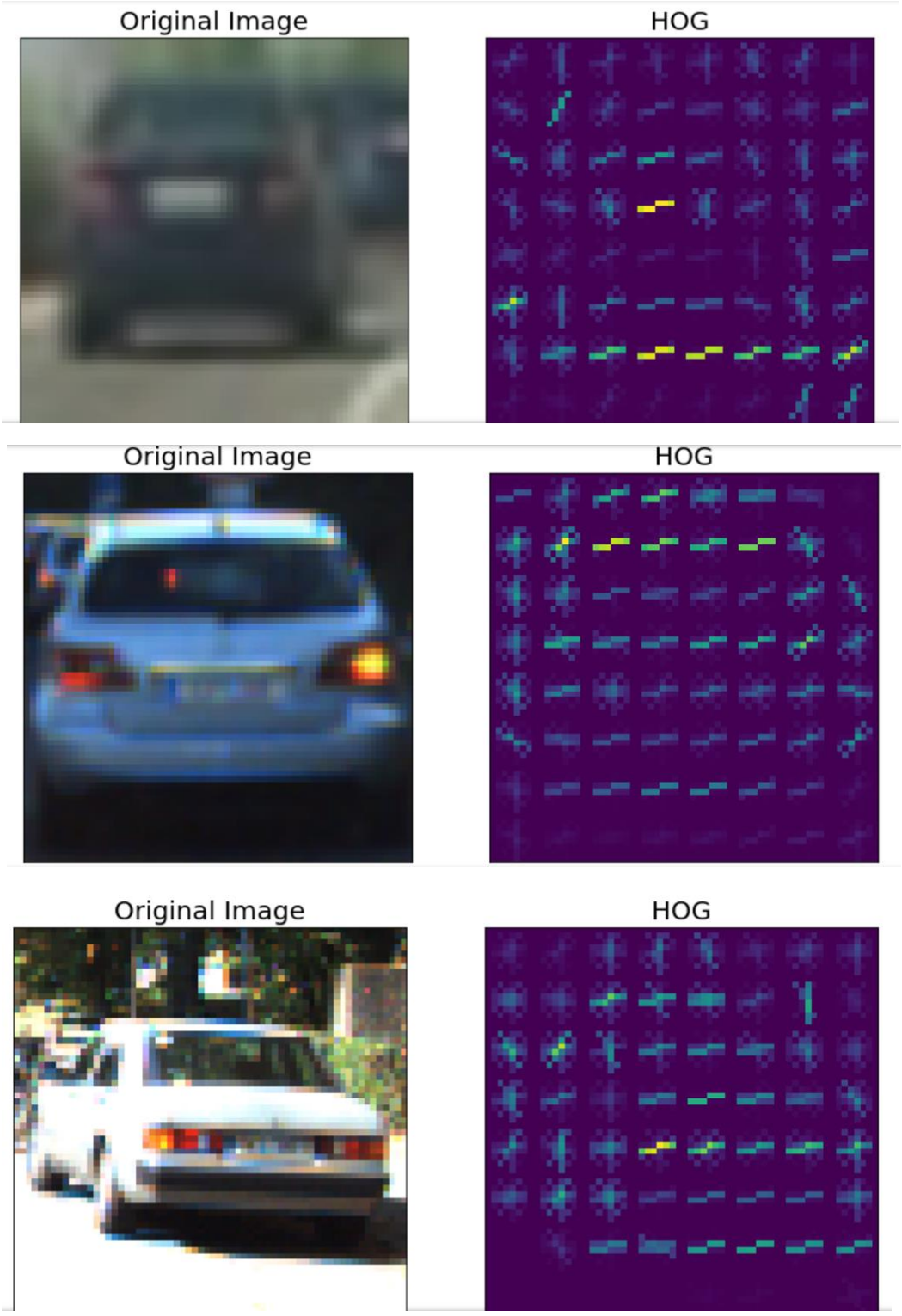




A screenshot of a group of sample images:



A few examples of using HOG(Histogram of Oriented Gradients)
to extract features from images:



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

These are three of my final choices for HOG parameters, C is rejected due to poor output quality with plenty of false positives, thus I am left with a head-scratching dilemma between option A and B, namely a classic trade-off between performance and quality. I opted for the later in the end.

Option	A	B	C
cspace	YCrCb	YCrCb	YCrCb
cell_per_block	2	2	2
hist_bins	32	32	32
hist_range	(0, 256)	(0, 256)	(0, 256)
hog_channel	ALL	ALL	ALL
orient	9	9	9
pix_per_cell	8	4	4
spatial_size	(16,16)	(32,32)	(16,16)
Training Time	60+40+2.74+12.7	120+38+61+22.4	120+41+44.6+33.3
Accuracy		98.958%	98.902%
Quality	Great	Good	Poor
Speed	2.5s/it	1.45s/it	N/A

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).

In cell 6, I performed feature extraction on car and no-car images, which are then stacked into feature vectors before being scaled using StandardScaler () function in cell 7. The label vectors are also created to combine with feature vectors to form a complete training set from which a test set of size 20% was split.

In cell 8, a linear Support Vector Machine algorithm is fitted with a C parameter set to 1.0, the accuracy of the predicted result in comparison with the test set was also generated using accuracy_score function from sklearn.metrics.

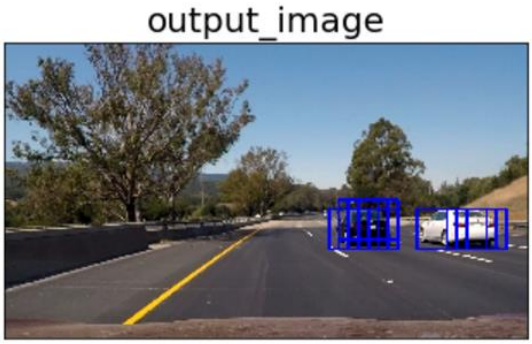
Sliding Window Search

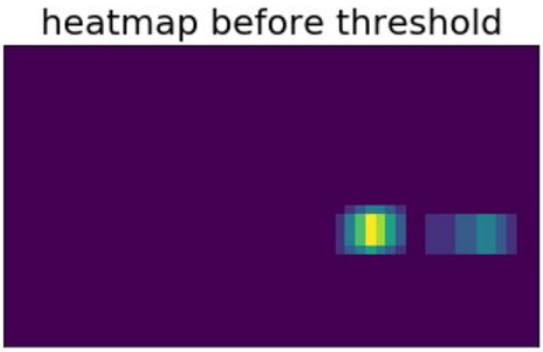
1. Describe how (and identify where in your code) you implemented a sliding window search. How did you decide what scales to search and how much to overlap windows?

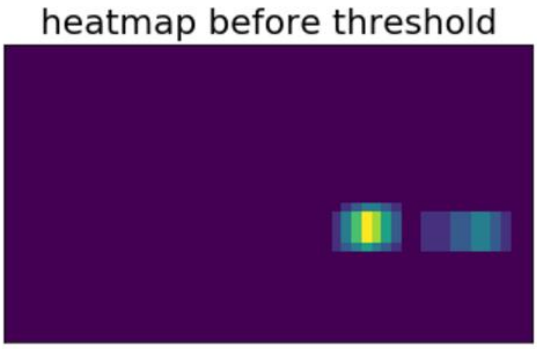
A more efficient technique called Hog Sub-sampling Windows search was employed instead of a classic sliding windows search technique, thus I only had to extract the Hog features once. In cell 9 and 13, you can see Hog Sub-sampling in action!

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.







In addition, I have experimented with a number of viable machine learning algorithm including decision trees and Naïve Bayes, polynomial and linear regression etc., however, none of them can offer the same level of accuracy, performance and accessibility afforded by LinearSVC algorithm.

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.)

Please find all my output video clips within a folder called 'output_videos'.

2. Describe how (and identify where in your code) you implemented filter for false positives and some method for combining overlapping bounding boxes.

Please see cell 11 for details on how I implement a threshold mechanism for rejecting false positive

```
# Setting threshold for rejecting false positives
if smooth_count > smooth_window:
    for A in range(smooth_window):
        heat = add_heat(heat, boxes[-A])
    heat = apply_threshold(heat, smooth_window)
else:
    heat = add_heat(heat, bbox_list)
    heat = apply_threshold(heat, 1)
```

This is bit of codes are important because we only want to classify vehicles that are within a certain distance of the self-driving car, we certainly do not need to take into account the cars travelling in opposite direction on the other side of the highway!

Discussion

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?

In my honest opinion, HOG is a terribly outdated and inefficient real-time object detection techniques for applications such as those in the realm of self-driving cars.

Please take a look at the video clips recorded by myself under different ambient lighting conditions where shortcomings of HOG are blatantly clear. HOG technique failed miserably when the camera faces the sun…

I have searched online for more advanced techniques that involve various elements of deep learning neural networks such as YOLO(You only look once), which is significantly faster and offer the same if not better output quality.

References:

- https://www.github.com
- http://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html
- http://opencv.org/