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.

I'm reading it! The project code could be found on the “VehicleDetection.ipynb”

Histogram of Oriented Gradients (HOG)

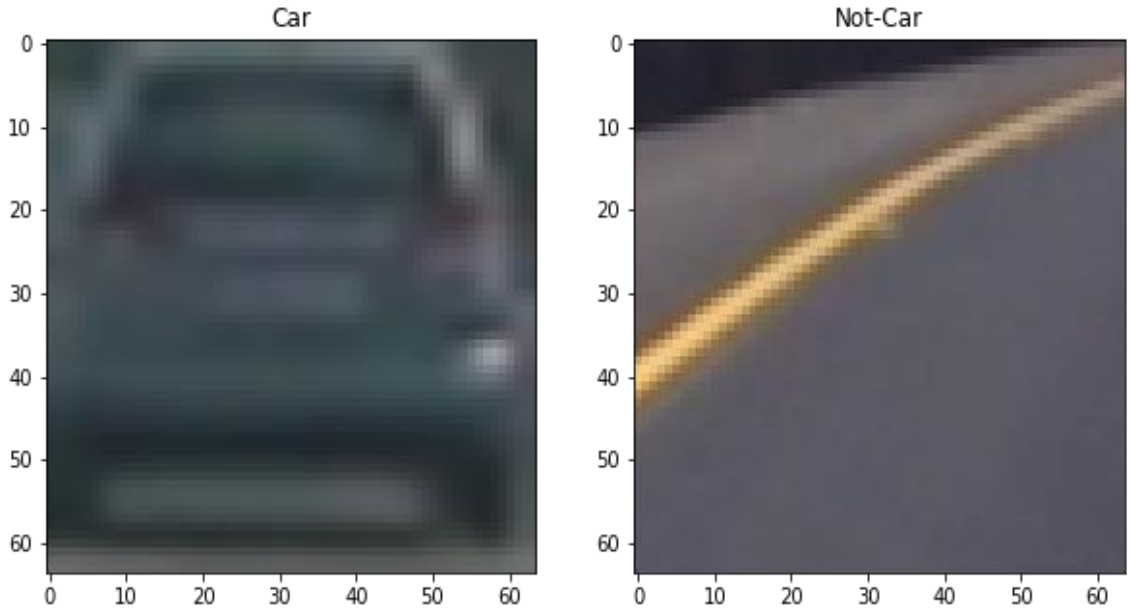
1. Explain how (and identify where in your code) you extracted HOG features from the training images.

The training images are extracted by code cell of In [2].

I started by reading in all the vehicle and non-vehicle images.

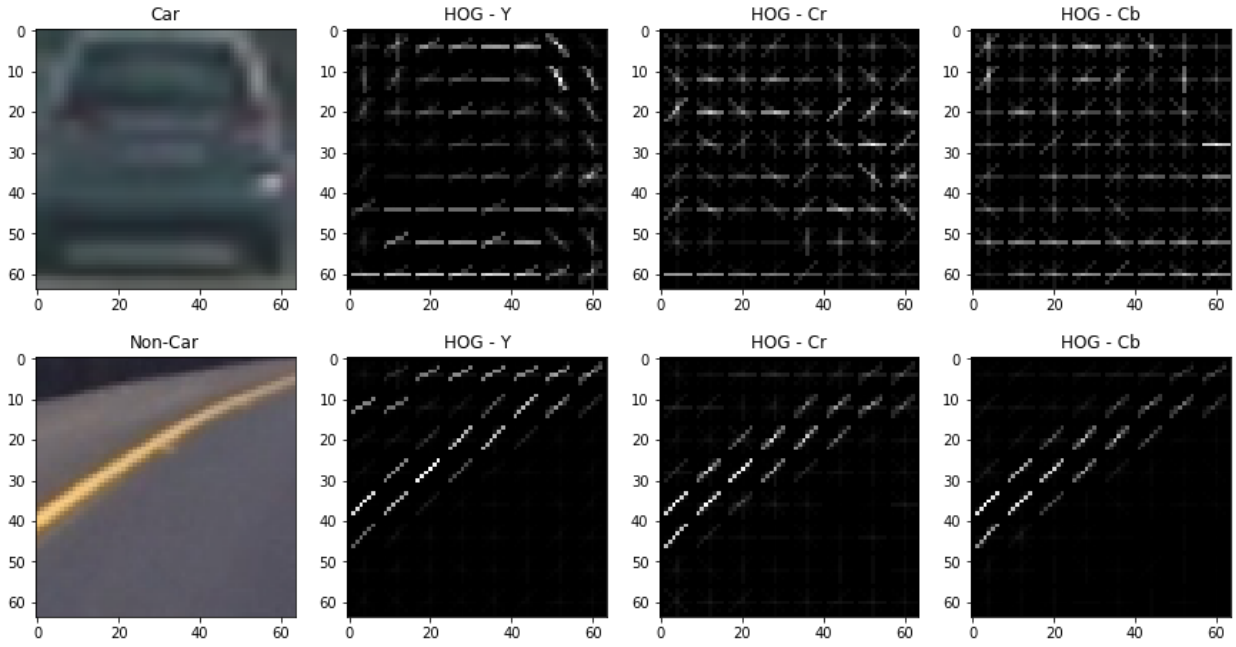
- Vehicle train images count: 8792
- Non-vehicle train image count: 8968

Here is an example of one of each of the vehicle and non-vehicle classes:



I then explored different color spaces and different skimage.hog() parameters (orientations, pixels_per_cell, and cells_per_block) at In[3]. This cell contains a set of functions provided by Udacity's lectures to extract the features from an image. The function extract_features combine the other function and use the class FeatureParameters to hold all the parameters in a single place. I grabbed random images from each of the two classes and 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) for a Cat and a Non-Car calculated at In [5]:



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

I tried various combinations of parameters by manually changing them and experimenting to try to maximize the model accuracy and minimize the fitting time. The final parameters are the following:

Parameter	Value
Color Space	YCrCb
HOG Orient	8
HOG Pixels per cell	8
HOG Cell per block	2
HOG Channels	All
Spatial bin size	(16,16)
Histogram bins	32
Histogram range	(0,256)
Classifier	LinearSVC
Scaler	StandardScaler

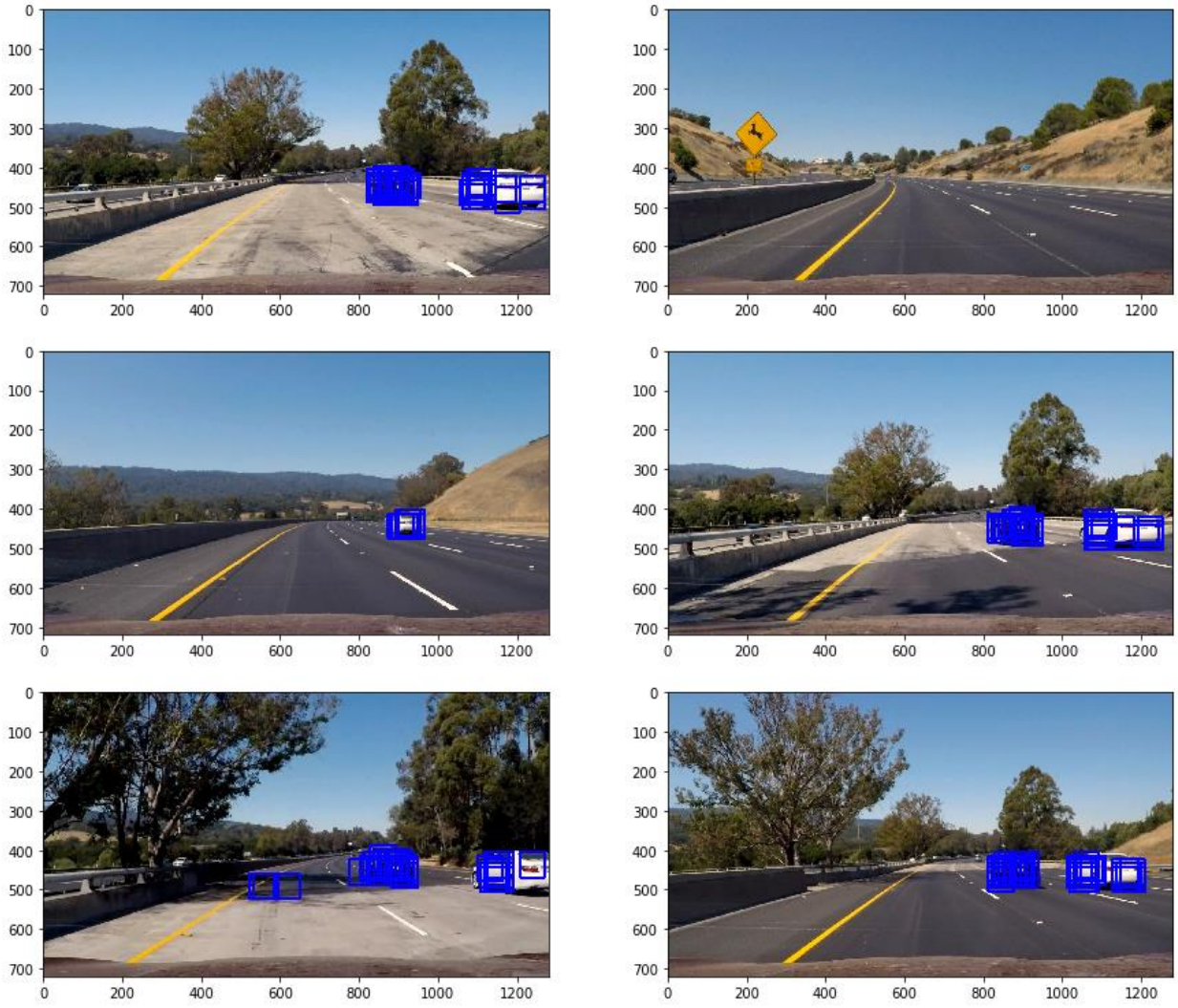
3. Describe how (and identify where in your code) you trained a classifier using your selected HOG features (and color features if you used them).

I trained a linear SVM using the default classifier parameters and using HOG features above. The code is in In [4] with codes LinearSVC(), svc.fit() and svc.score(). And it was able to achieve a test accuracy of 99.1 % and it took 10.37 seconds to train.

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?

My first approach to implement sliding windows was to calculate all the windows and then apply the feature extraction to each one of them to find the one containing a car. It is implemented on In [6]. Cell In [7] contains the code for loading the test images, applying the classifier to the images and drawing boxes. The scales and overlap parameter where found by experimenting on them until a successful result was found. The following image shows the results of this experimentation on the test images:



To combine the boxes found there and eliminate some false positives, a heat map as implemented with a threshold and the function label() from scipy.ndimage.measurements was used to find where the cars were. The code for this implementation could be found on In [8], and the next image shows the results on the test images:

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?

The performance of the method calculating HOG on each particular window was slow. To improve the processing performance, a HOG sub-sampling was implemented as suggested on Udacity's lectures. The implementation of this method could be found on In [9]. The following image shows the results applied to the test images (the same heatmap and threshold procedure was applied as well on In [10]):

Video implementation

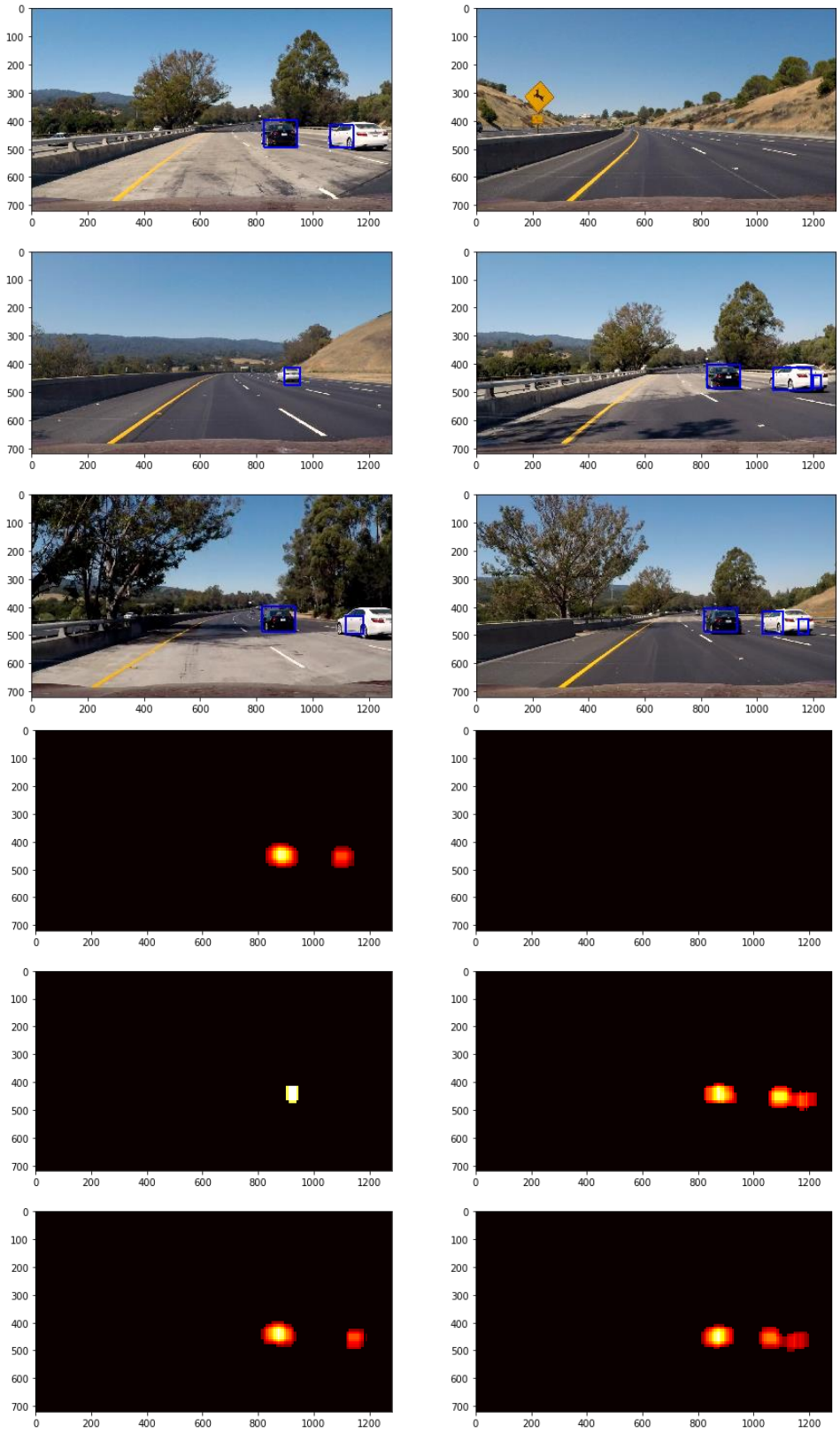
1. Provide a link to your final video output. Your pipeline should perform reasonably well on the entire project video (somewhat wobbly or unstable bounding boxes are ok as long as you are identifying the vehicles most of the time with minimal false positives.)

Two video outputs could be found from my submission as: test_video_out.mp4 and project_video_out.mp4

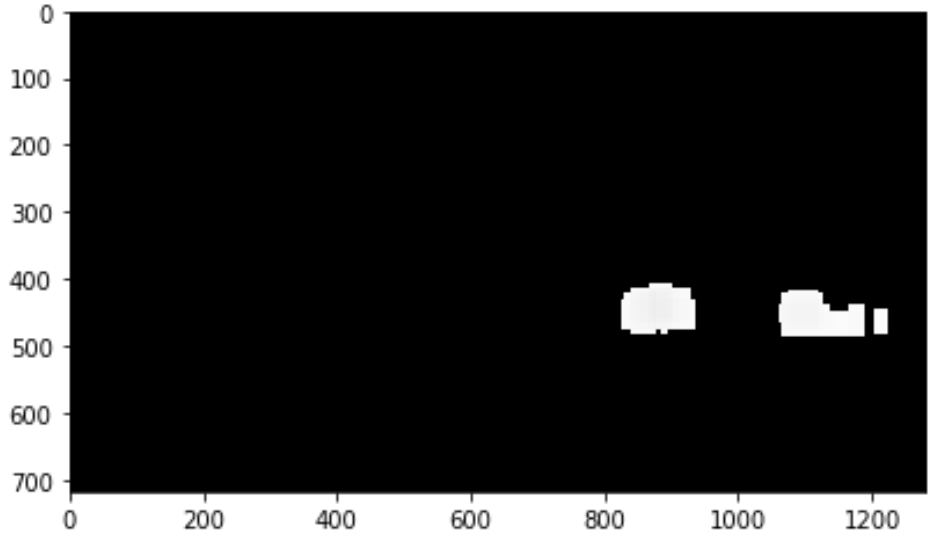
2. Describe how (and identify where in your code) you implemented some filter for false positives and some method for combining overlapping bounding boxes.

Some effort was done already to minimize false positives using a heatmap and threshold in the pipeline, but it was not enough. The overlapping bounding boxes were resolved by using the function label() from scipy.ndimage.measurements to find the cars. To filter false positives, the image heatmap was averaged over three consecutive frames. The implementation could be found on In [11]

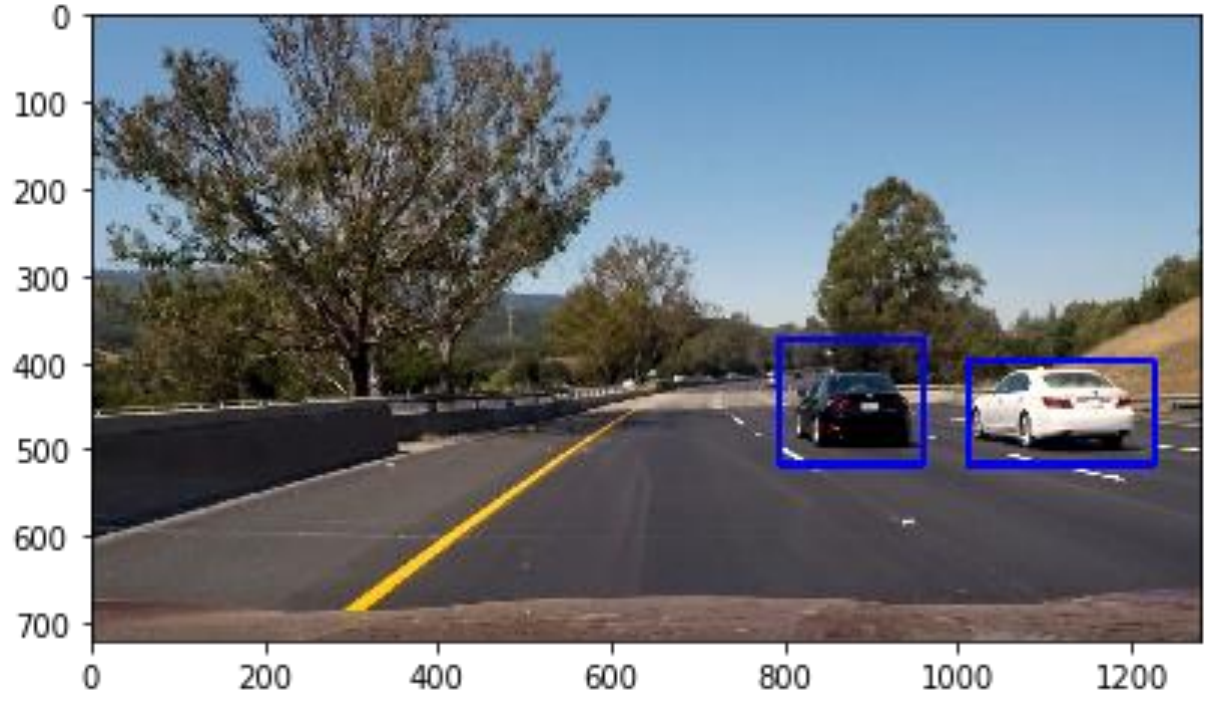
Here are six frames and their corresponding heatmaps:



Here is the output of scipy.ndimage.measurements.label() on the integrated heatmap from all six frames:



Here the resulting bounding boxes are drawn onto the last frame in the series:



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?

Here I'll talk about the approach I took, what techniques I used, what worked and why, where the pipeline might fail and how I might improve it if I were going to pursue this project further.

(a) Use a decision tree to analyze the redundancy on the feature vector and try to decrease its length eliminating redundancy.
(b) The performance of the pipeline could be improved by trying to decrease the amount of space to search for windows.

(c) More than one scale could be used to find the windows and apply them on the heatmap.

(d) The windows size could change for different X and Y values to minimize the number of windows to process.