Writeup Template

You can use this file as a template for your writeup if you want to submit it as a markdown file, but feel free to use some other method and submit a pdf if you prefer.

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're reading it!

Histogram of Oriented Gradients (HOG)

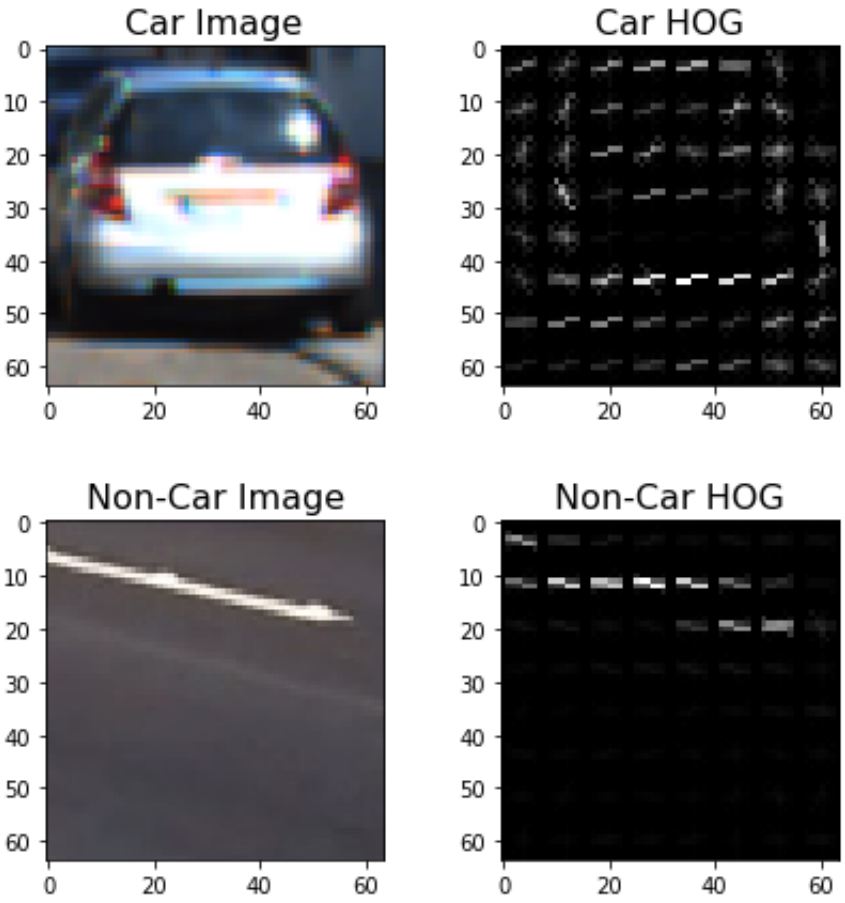
1. Explain how (and identify where in your code) you extracted HOG features from the training images.

The code for this step is contained in in lines 155 through 165 of the file called `functions.py`.

I started by reading in all the `vehicle` and `non-vehicle` images. 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`). 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 `YUV` color space and HOG parameters of `orientations=9`, `pixels_per_cell=(8, 8)` and `cells_per_block=(2, 2)`:



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

1) Color space

I discard RGB first because it varies a lot in inchanging light conditions.
I tried YUV, HLS and YCrCb which can provide ideal results and I choose YUV finally due to it have the highest accuracy (just a little higher).

2) Orientations&Pixels Per Cell&Cells Per Block

I finally settled pixels*per*cell=(8,8).
Also, larger values of than orient=9 did not have a striking effect.
Similarly, using values larger than cells*per*block=(2,2) did not improve results, which is why these values were chosen.

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 all channels of images converted to YUV space. I included spatial features color features and all three YUV channels, because using less than all three channels reduced the accuracy considerably. The final feature vector has a length of 8460 elements. For color binning patches of spatial*size=(32,32) were generated and color histograms were implemented using hist*bins=32 used. After

training on the training set this resulted in a validation and test accuracy of 98.97%.
The code for this step is contained in in lines 31 through 97 of the file called `Vehicle_Classifier.py`.

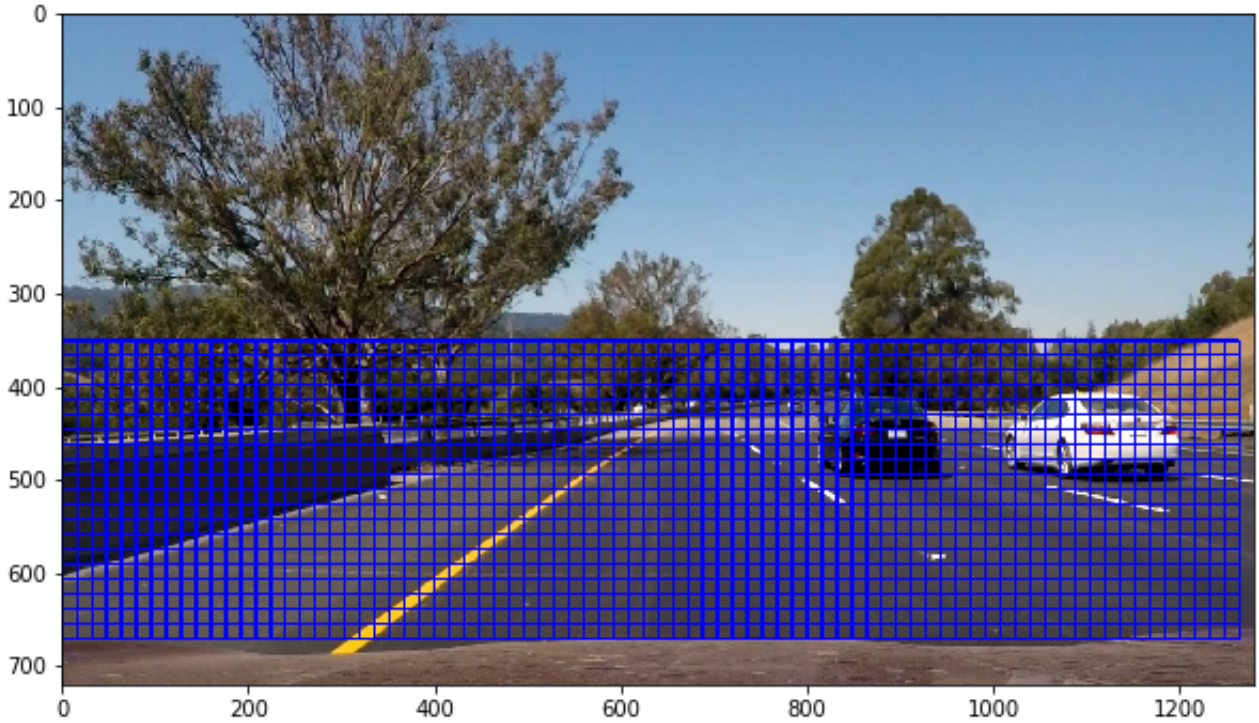
Sliding Window Search

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 adapted the method find_cars from the lesson materials which is more efficiently. The window overlap was set to 50% in both X and Y directions.
I decided to search the image by 4 parts:

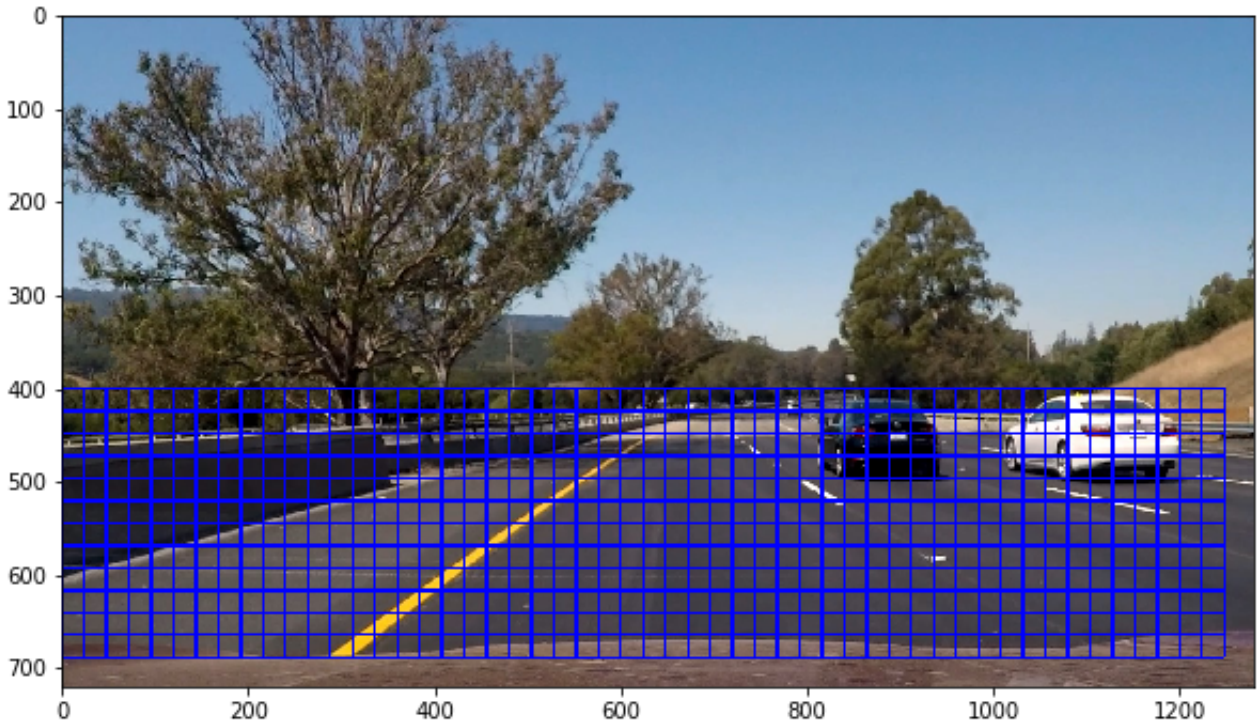
Part1:

`scale = 1.0` `y_start_stop = [350, 700]`



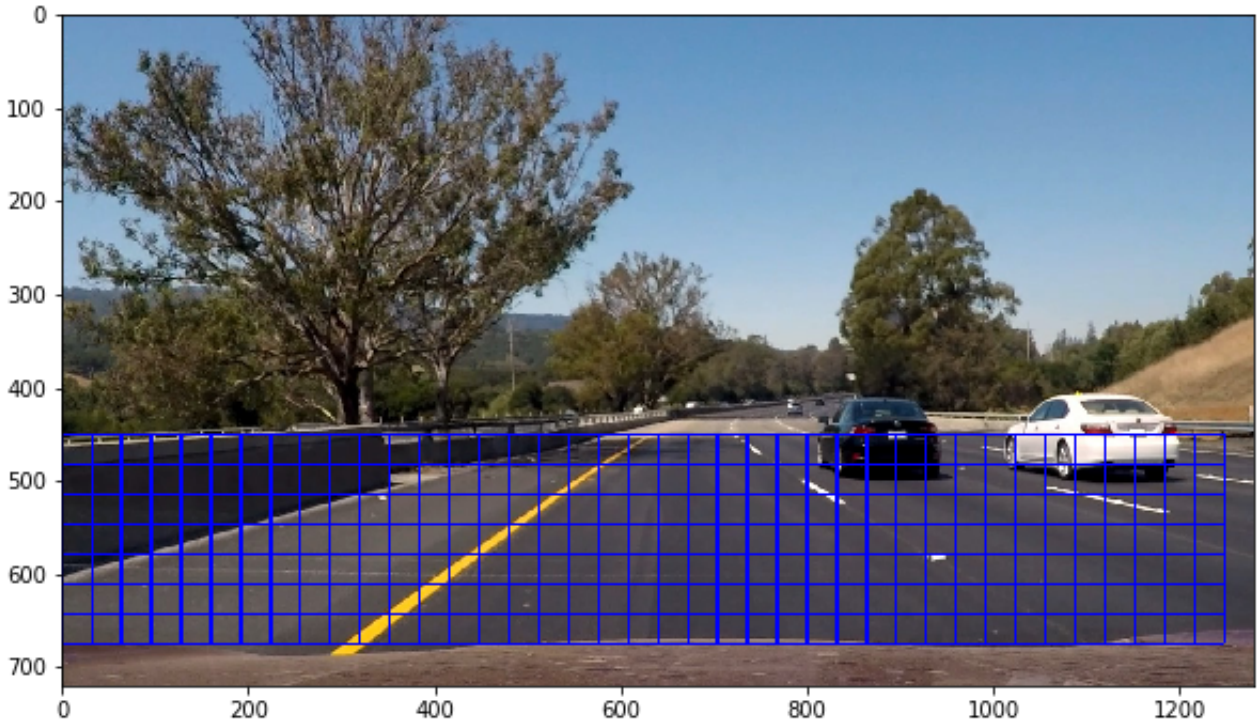
Part2:

`scale = 1.5` `y_start_stop = [400, None]`



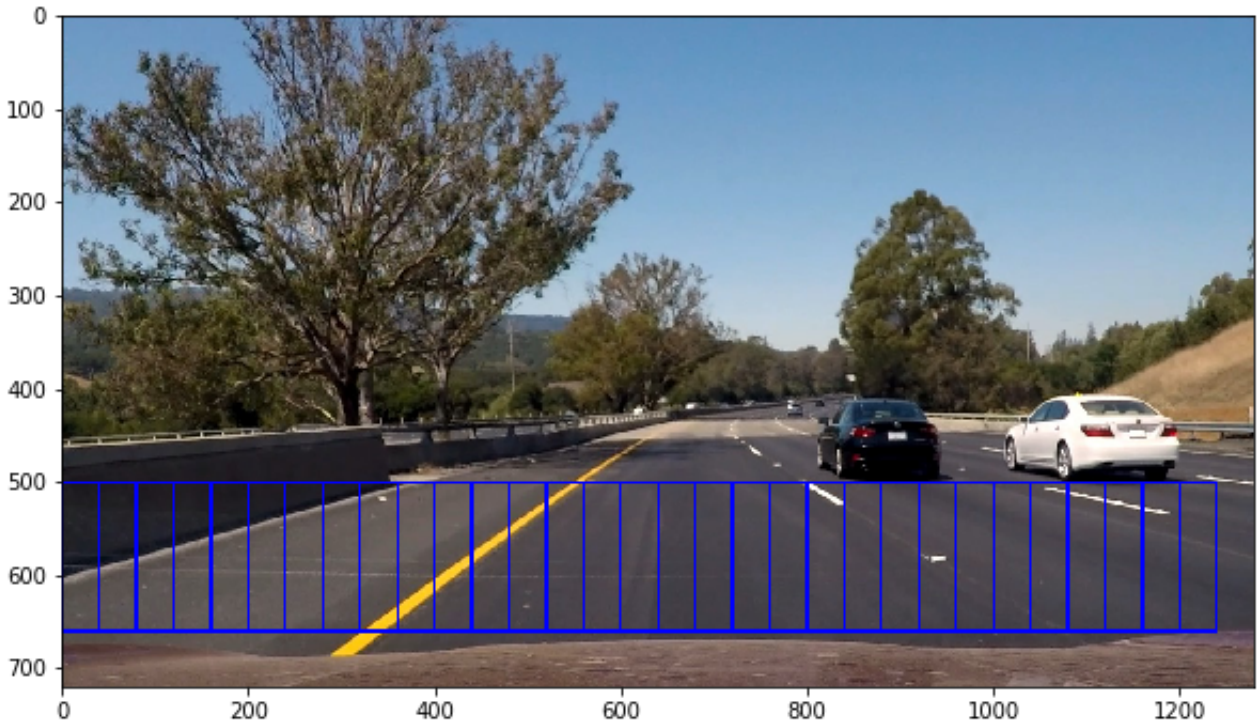
Part3:

`scale = 2.0` `y_start_stop = [450, None]`



Part4:

`scale = 3.0` `y_start_stop = [550, None]`



The code for this step is contained in in lines 205 through 227 of the file called `Vehicle_Detection.py`.

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?

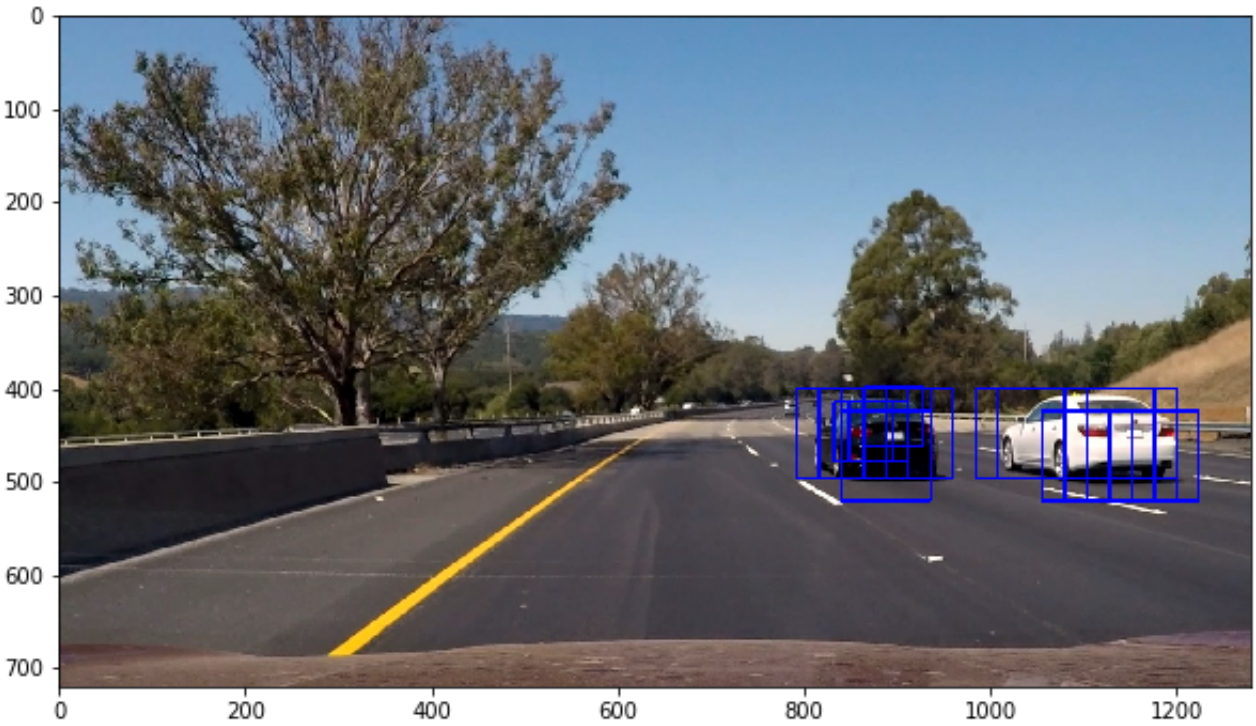
The results of passing all of the project test images through the above pipeline are displayed in the images below:


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200



Video Implementation

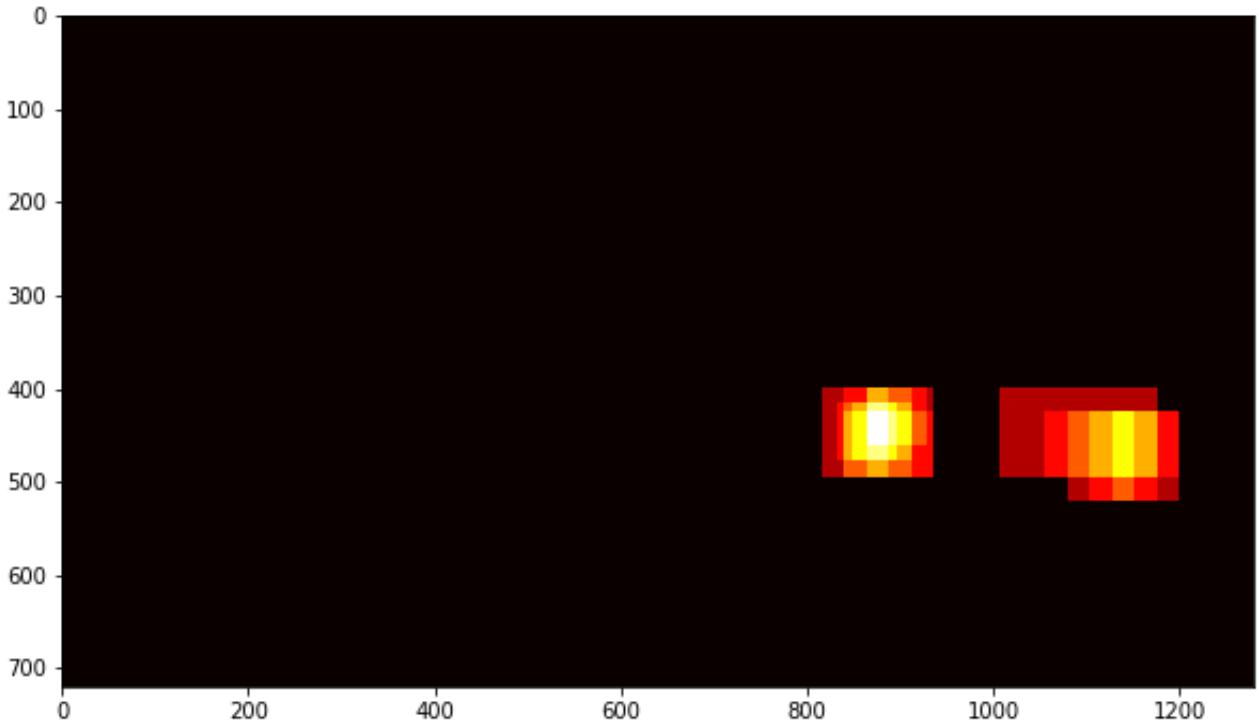
1. Provide a link to your final video output. Your pipeline should perform reasonably well on the entire project video (somewhat wobbly or unstable bounding boxes are ok as long as you are identifying the vehicles most of the time with minimal false positives.)

Here's a link to my video result

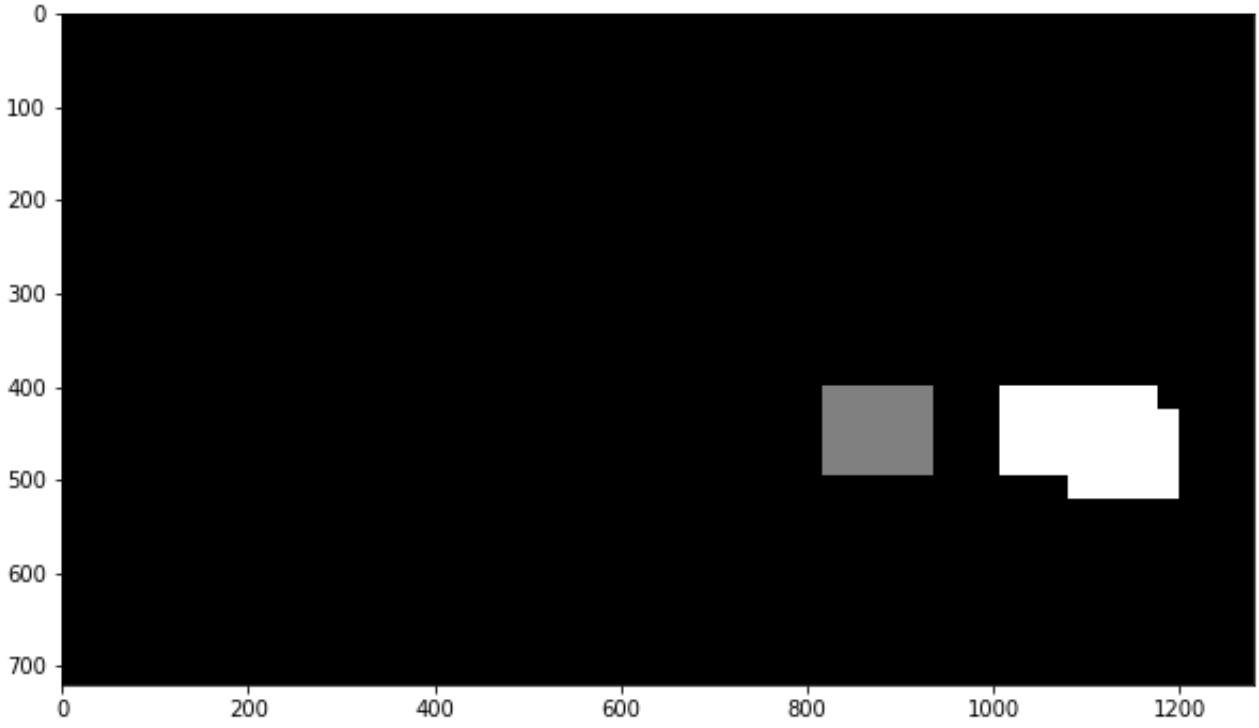
2. Describe how (and identify where in your code) you implemented some kind of filter for false positives and some method for combining overlapping bounding boxes.

I recorded the positions of detections in each frame of the video by a class called `Vehicle_Detect`. It recorded the last 15 frames detections positions and add all the positions to heatmap. The threshold for the heatmap is set to 1 + len(det.prev_rects)//2. This idea comes from `https://github.com/jeremy-shannon/CarND-Vehicle-Detection` which have excellent performance.

Here is one frame and corresponding heatmap:



Here is the output of `scipy.ndimage.measurements.label()` on the integrated heatmap:



Here the resulting bounding boxes are drawn onto the frame:

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?

The first problem is I trained the classifer with accuracy like 98% but when it comes to real image it have so much false positive detections. I spend too much time on this issue. I tried different color space and different parameters, check my code again and again, finally I found the problem comes from data preprocessing:
1. I didn't shuffle the data correctly.
2. To saving time, I didn't use the whole dataset which also cause so much false positive.
After I solved it, others is just small tips: such as to deal with Nan features in color space other than RGB, tuning the search area of different scales.
Still, in the test video, my pipeline is not robust enouth which have two issues to be solved:
1. Minor false positives still exist.
2. The detection cubes didn't follow the cars rapidly when it move its positions .
These problem can solved after tuning the filter for false positives.