writeup.md

Writeup Template

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.

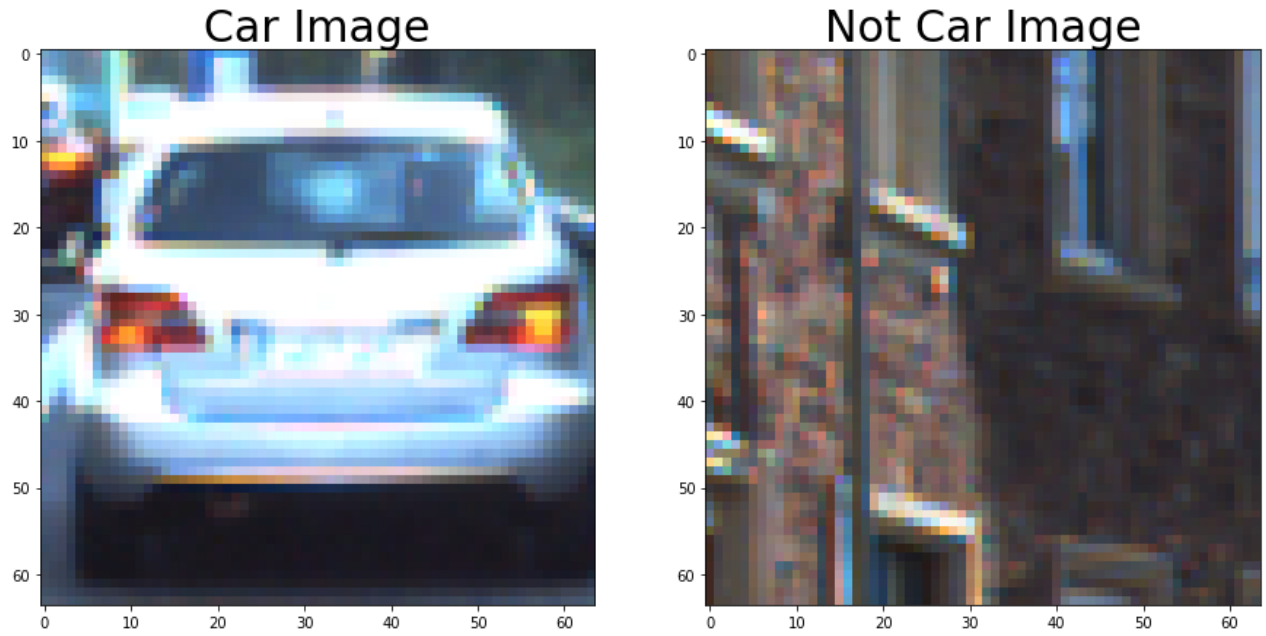
You're reading it!

Histogram of Oriented Gradients (HOG)

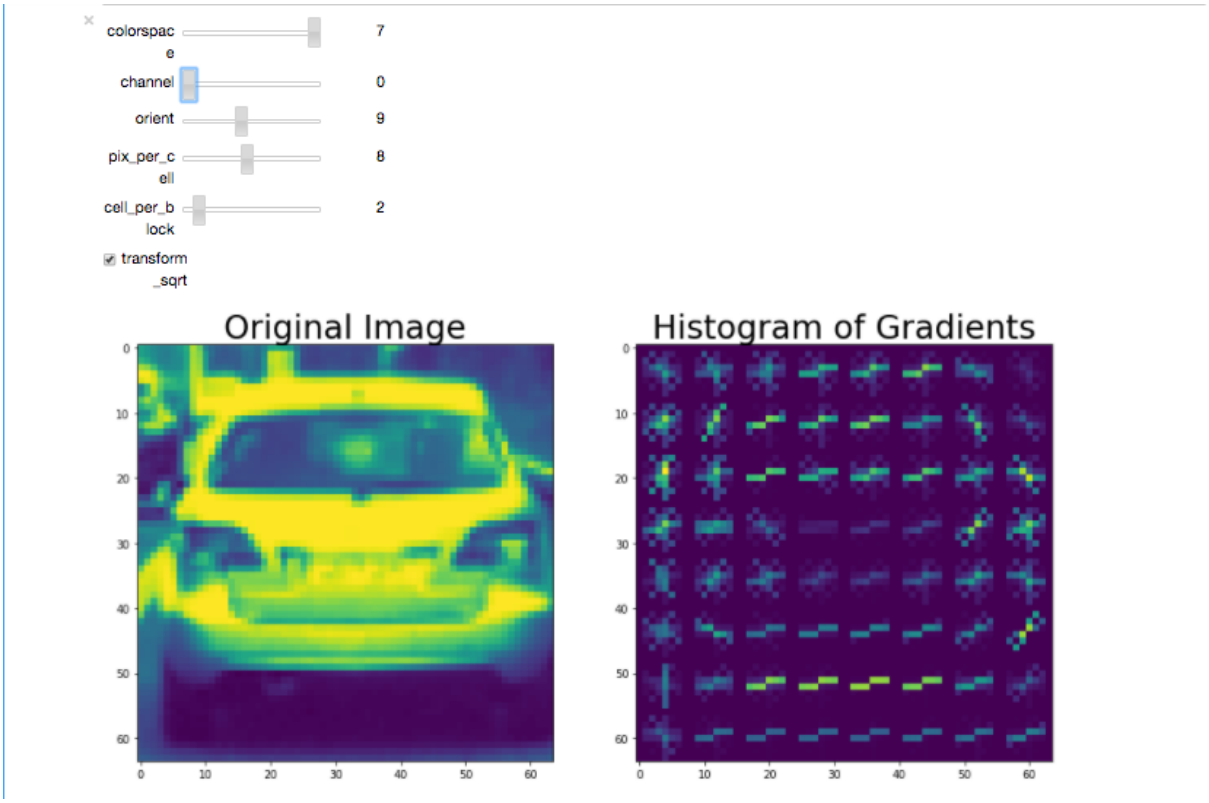
1. Explain how (and identify where in your code) you extracted HOG features from the training images.

I have done most of the work on figuring out the parameters for the Histogram of Oriented Gradients through the following notebook. It is in a different folder than my project one because I worked on it while following the lectures, as my own made-up lab project.

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:



In order to determine the right parameters that I want to use for my project, I built a small visualizing widget, following the suggestion of one of my previous code reviewers for Project 4.



This allowed me to quickly iterate through different combinations of parameters.

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

After trying several such combinations, I decided to pick the following structure for my HoG:

- color_space = 'YCrCb'
- orient = 9 # HOG orientations
- pix_per_cell = 8 # HOG pixels per cell
- cell_per_block = 2 # HOG cells per block

- hog_channel = "ALL"

Color spaces like *HSV* or *HLS* performed almost as well as *YCrCb* in terms of the model's accuracy. I decided to go with these parameters initially due to the visual hint they they were offering, in terms of properly detecting the edges of the cars, but then also because the model was offering a good enough accuracy.

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 my *P5-classifier* notebook I trained a *LinearSVC* classifier. I used the HoG, spatial and histogram features in order to train it. For data, I used all the provided cars and notcars images, split with a 20% percentage for training/testing sets. I think my approach overfitted the model, since I was having quite a hard time isolating false positives, as I will describe in the next sections. A better approach here, as hinted in the lecture notes, would have been to manually create my training and testing sets from the bigger data set I was provided with, in order to avoid overfitting.

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?

This was a very tedious process of trial and error, identifying the right scales to use for the sliding window search. I implemented this approach in the *P5-video* notebook.

The strategy I tried to use when selecting the scales was simple: use a larger scale (i.e. smaller window sizes) closer to the center of the image, where farther cars will be, and a smaller scale (i.e. larger windows) closer to the bottom of the image, where closer cars will be. Therefore, I eventually settled on the following values: 2 and 1.5, respectively.

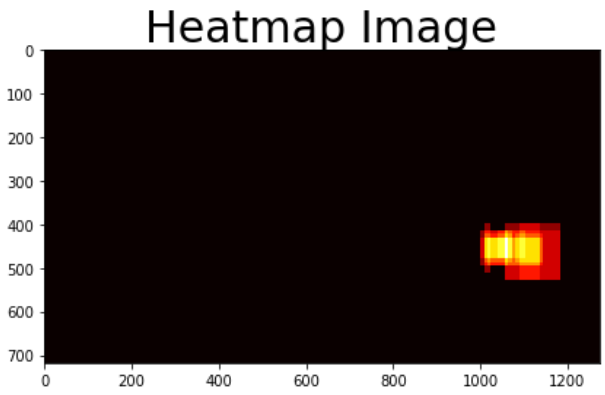
In order to get to these, I used *a ton* of test images and sample videos (shorter cuts from the project video) due to the very large turnaround time in between iterations on the main project video.

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?

Here are some examples of windows using multiple scales, along with their respective heatmaps and bounding boxes:


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


Detection Windows Image
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

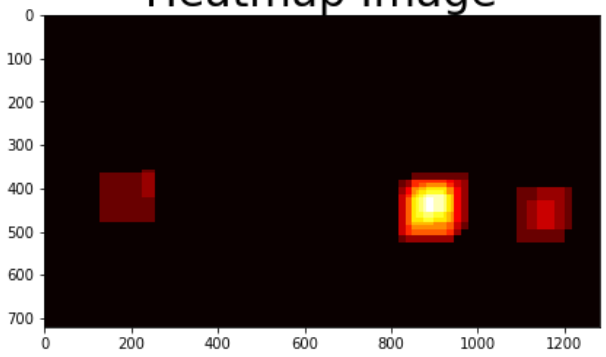

Heatmap Image
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


Detection Boxes Image
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

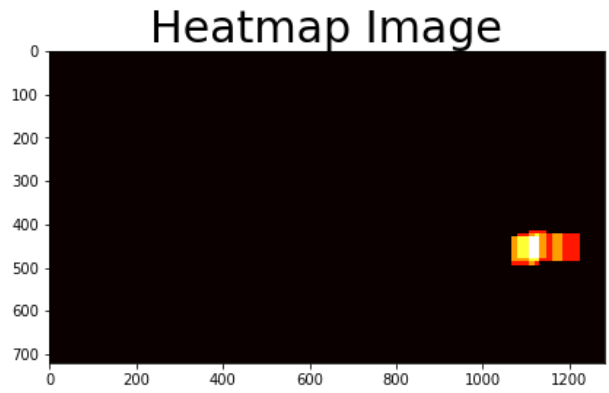
Original Image

Detection Windows Image

Heatmap Image



Detection Boxes 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.)

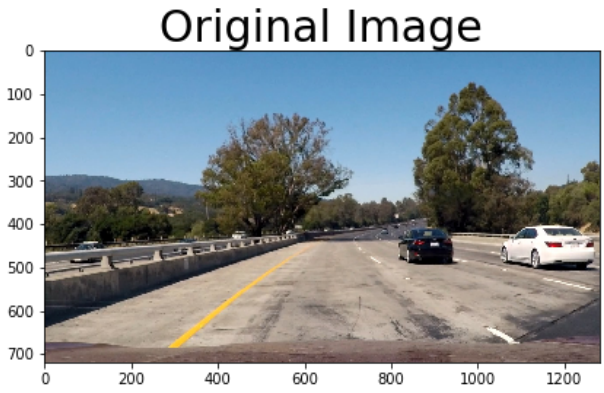
Here's a link to my video result

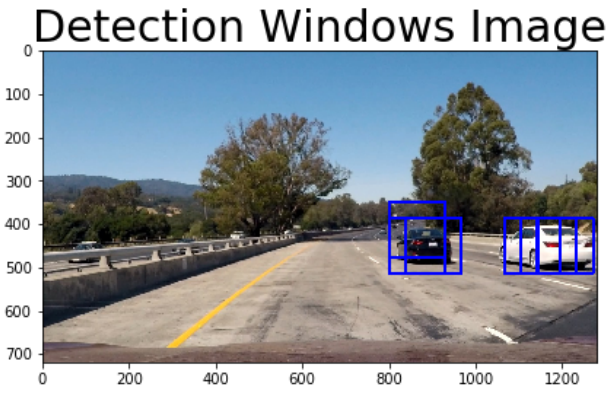
####2. Describe how (and identify where in your code) you implemented some kind of filter for false positives and some method for combining overlapping bounding boxes.

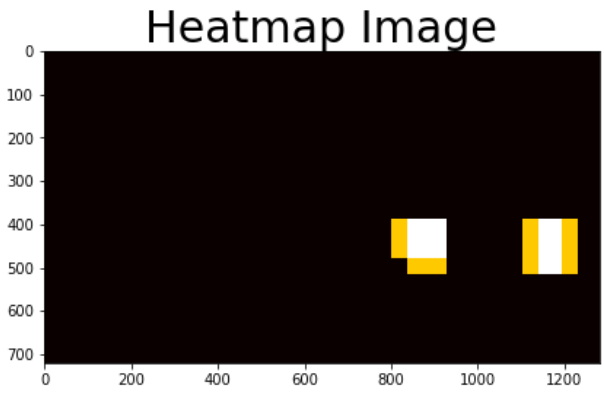
This was probably the challenging part of my implementation. I think the biggest issue for me was that my model slightly overfitted on the training data, and was detecting false positives with *a lot* of windows (not just 1, so that I can easily threshold it through heatmaps).

Initially, my approach was straightforward. I recorded the positions of positive detections in each frame of the video. From the positive detections I created a heatmap and then thresholded that map to identify vehicle positions. I then used `scipy.ndimage.measurements.label()` to identify individual blobs in the heatmap. I then assumed each blob corresponded to a vehicle. I constructed bounding boxes to cover the area of each blob detected.

These were some results of this approach:


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

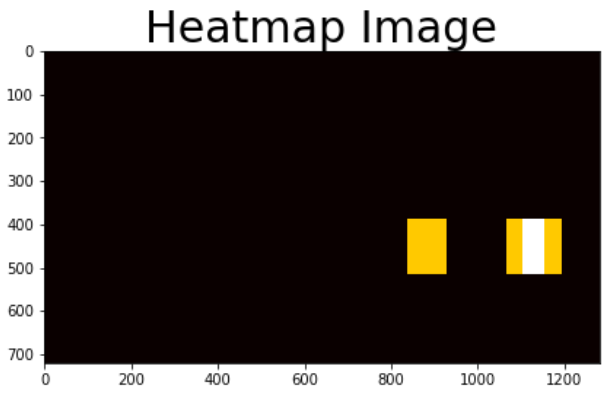

Detection Windows Image
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


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

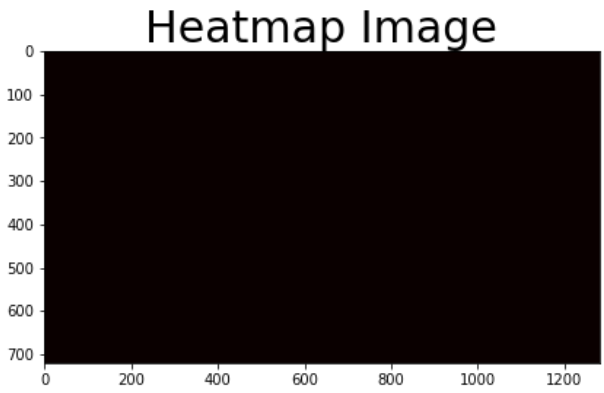

Detection Boxes Image
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


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


Detection Windows Image
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

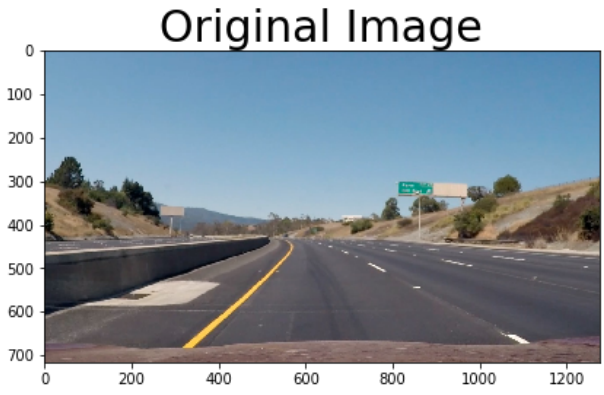

Heatmap Image
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


Detection Boxes Image
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

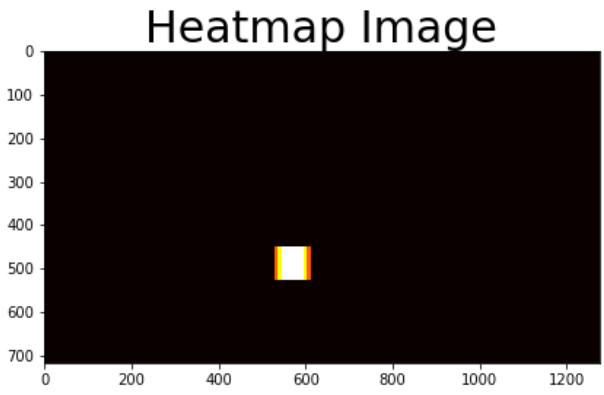


But this did not seem enough, given how my model was detecting multiple windows per false positive, over a few consecutive frames.

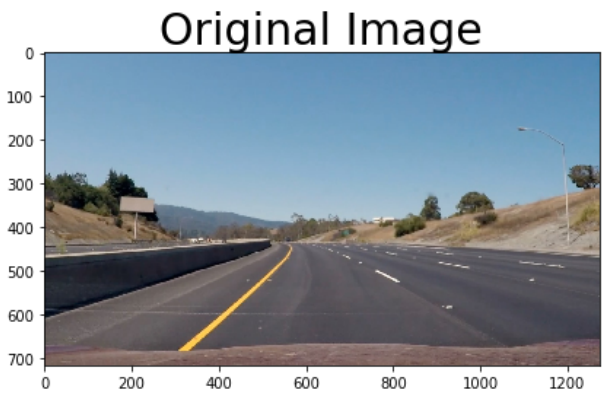
Here are some examples of the issues:

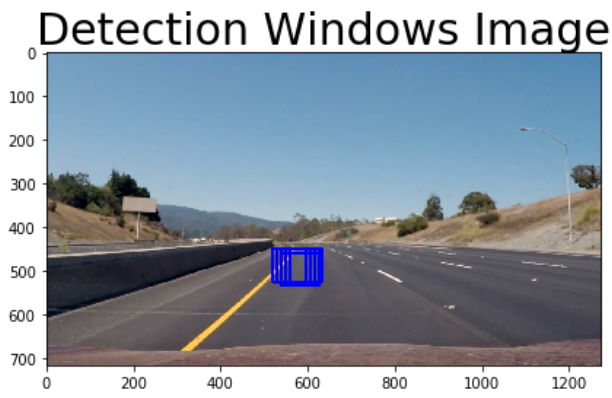

Original Image

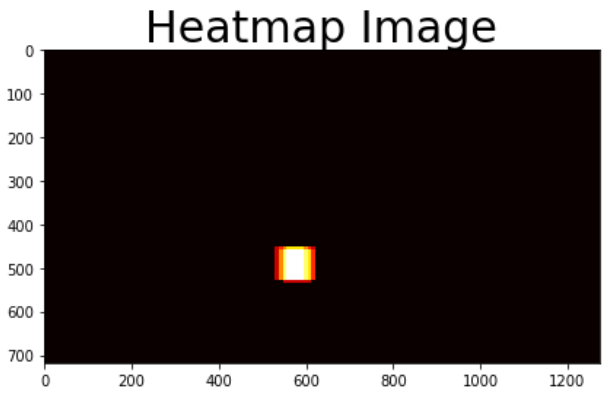

Detection Windows Image


Heatmap Image


Detection Boxes Image







Therefore, I collected a lot of test images and short cuts of the project video where I noticed bigger issues with false positives.

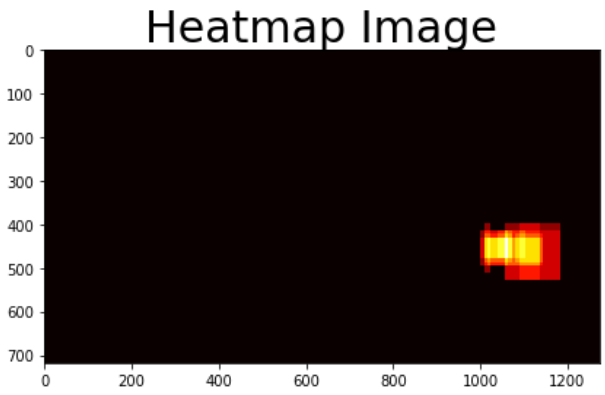
I realized that probably my best shot at dealing with these false positives would be to look over multiple consecutive frames and average the heatmap values. I have done that using a bounded deque, with a maximum length of 20. I do not think this approach is generally realiable, as getting to this number involved another trial and error process, and I do not think it will generalize well. But it worked for the majority of my video...

Having 20 frames and doing an average on them helped me filter out the false positives and also smooth the bounding boxes on the cars.

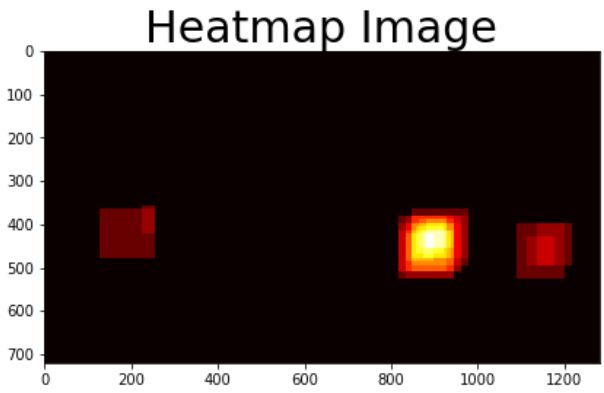
Here are some examples of the heat map, as also presented earlier:


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

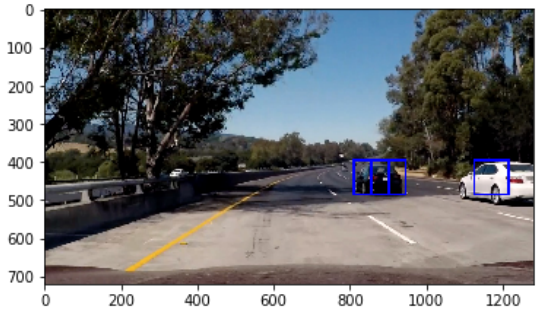

Detection Windows Image
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

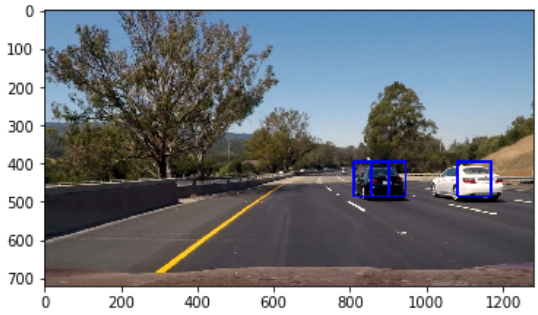

Heatmap Image
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


Detection Boxes Image
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200



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?

As stated before, I think my biggest issue was overfitting the model. I did not have enough time to manually construct my training set from the initial data set, and I realized that this could be a problem from what I've read in the lecture notes. This led to a very big number of false positives in the project video.

I also tried training my model with the smaller set used throughout the lectures, but it did not produce good results overall.

In order to deal with these false positives, given their "very strong" predictions, with multiple windows across consecutive frames, I decided to look over a history of frames, hoping that over 20 frames the false positives will be sparse. This strategy was successful, after a bunch of trial and error attempts through a lot of testing images and shorter videos, of course.

I would really like to improve this pipeline somehow, as to not rely on the number of historical frames in order to filter out false positives though. Just the simple approach of filtering them out on each frame, using a heatmap threshold, did not work well for me for the stated reason of having way too many windows detecting a false positive.

Another issue was the very slow iteration process in between my code changes. This was due to me using a laptop to process the videos and, probably, also due to some overhead added by the python notebook. A better approach here would have been to use an AWS instance to process the videos. I overcame this (somewhat) by looking at tricky parts of the video, having the project video trimmed.