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.

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.

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.

First I explored the data in section 1 by reading the filepaths of all the training data and view the counts of cars

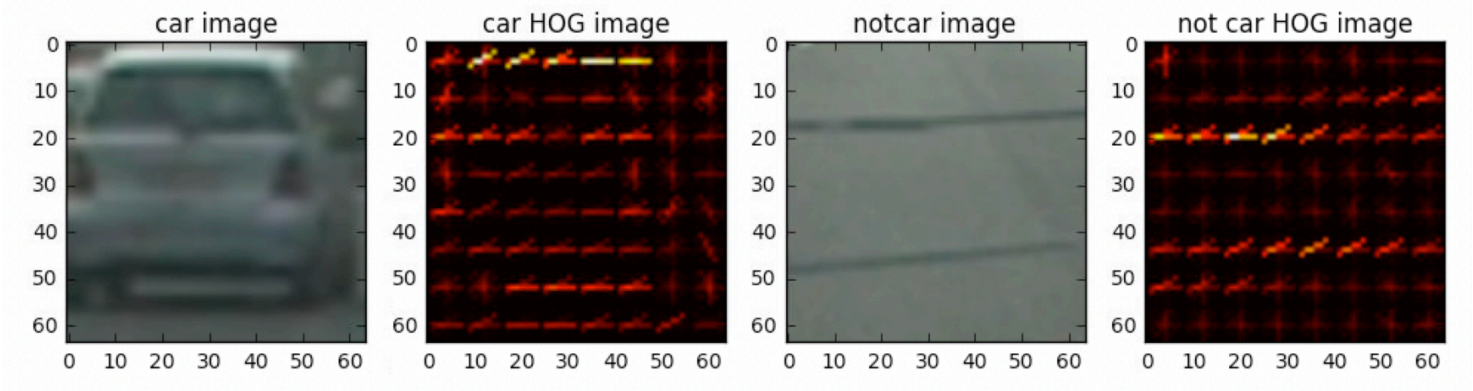
and noncars. They look to be pretty balanced in terms of quantity at a glance of file counts.

Cars: 8792 NotCars: 8968

In section 2 of the ipyhon notebook I create a function that executes the hog function from the skimage.feature library. The function takes in various parameters to try and tune such as orientation, pixels per cell and cells per block.

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:

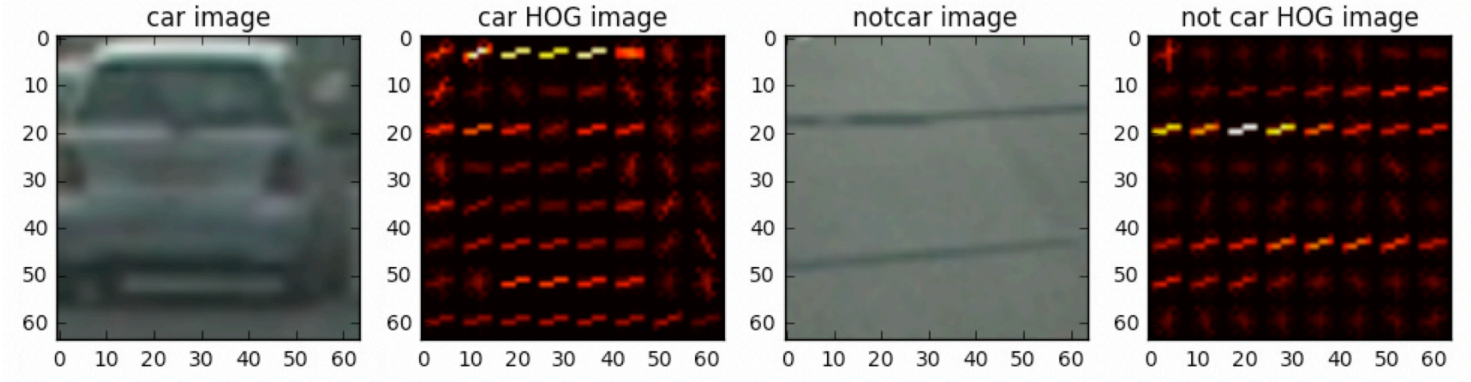
Here is a hog transform using RGB and 6 orientations, 8 px per cell and 2 cells per block



I then explored various parameters of executing the hog function with different colorspaces, orientations and pixels per cell.

Color Space	Orientations	Feature V Length	Score
YCrCb	6	4344	.997
RGB	6	4344	.994
HLS	6	4344	.935
HLS	9	6108	.997
RGB	9	6108	.994
YCrCb	9	6108	.998

He is an example of a HOG representation using parameters YCrCb and 9 orientations



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

After I tried various experiments with various parameters, I chose to use the YCrCb color space with 9 orientations, 8 pixels per cell anf 2 cells per block. This yeilded 6108 feature vector length.

The various colorspace that I tried had a very similar score result if using all channels for all of them. If I were to use a single channel on various color spaces, the score differences was much more noticeable. I decided to stick with training on all color channels. I feel this captures the most information out of the images. If I needed to really speed the performance in the future I may look into using a single channel instead.

I chose this set of parameters because this was the set that yeilded the best score while training the classifier. Also as I increased the orientations above 9, the score didnt improve, however the feature vector length increased which made our training and pipeline times increase without much gain.

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 classifier in section 5 of the code notebook. To prepare the data, I call an extract method that obtains from the image a bin spatial vector, a color histogram vector and the hog features and concatenates them together.

All these features sewed together into a single input vector will introduce variability into the value range for each part in which higher valued features can greaty dominate the training. To overcome this we normalize the values in the resulting feature vector. In this case we used StandardScaler from the sklearn.preprocessing library.

For training, we sampled 20000 random indexes from the cars and not cars training set and split them into training and test sets at 10%.

Afterwards I ran the classifier with various hog parameters and settled on the parameter set with the greatest accuracy score, which in this case was a .998 accuracy with parameters colorspace:YCrCb, 9 orientations, All

color channels, 8 pixels per cell and 2 cells per block.

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 section 8 of the notebook, I have implemented a findcars routine (similar to the lessons) that will instead of resize the search window, will rescale the image itself while reusing the a 64 by 64 static window then scaling the image back, this acheives various window sizing. The routin will also 'step' the seach window along the x and y axis of the image.

I chose to search 3 different scales. To reduce computation time and some false positives, to only search sub regions on the y-axis for each of the various scales. The smaller scales would not search as far down as the smaller scales.

Here is an examples of the windows searched with the different scales being of different color.

The scales that I decided are

scale 1.00 from y-400px to y-550px

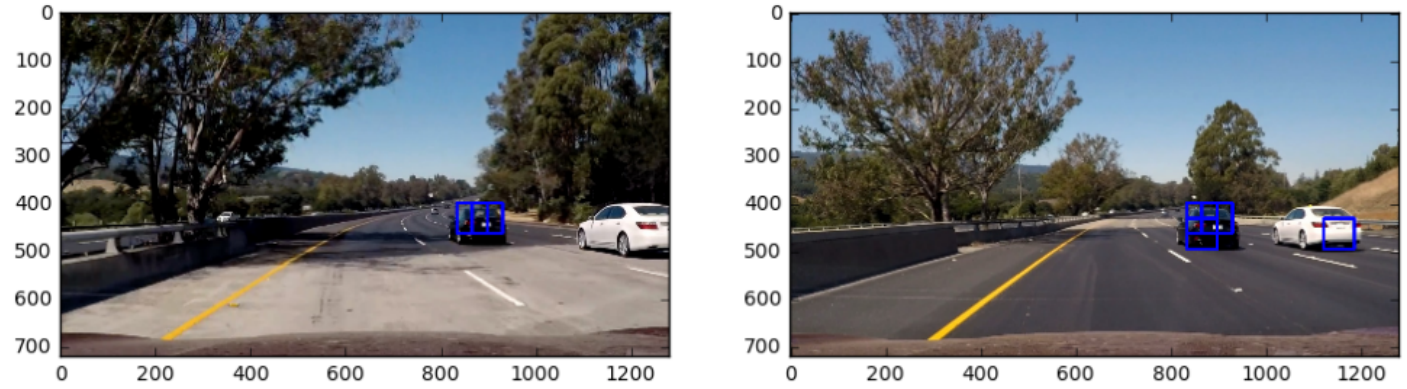
scale 1.33 from y-400px to y-620px

scale 1.75 from y-400px to y-720px

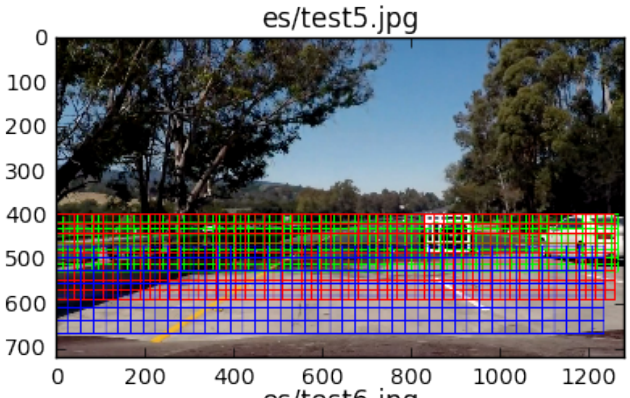
Also included are some heatmaps that have been output. The heat maps are computed by adding a +1 to the cell that have been predicted as being a car. The more predictions in that space, the brighter the heatmap and more prominent the confidence the algorithm as of a car being present in that space.

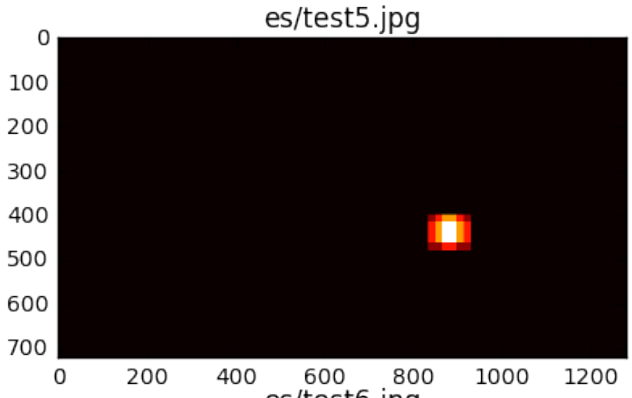
Here is an image with a car detected using the sliding window search (before settling on scales and ranges)

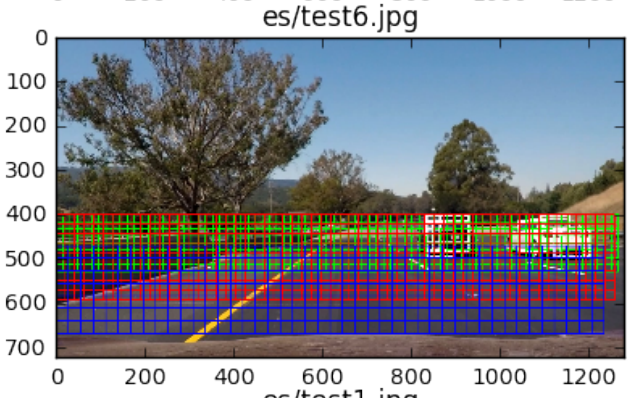
```
1487962154.3937387 seconds to process one image searching 273 windows
```

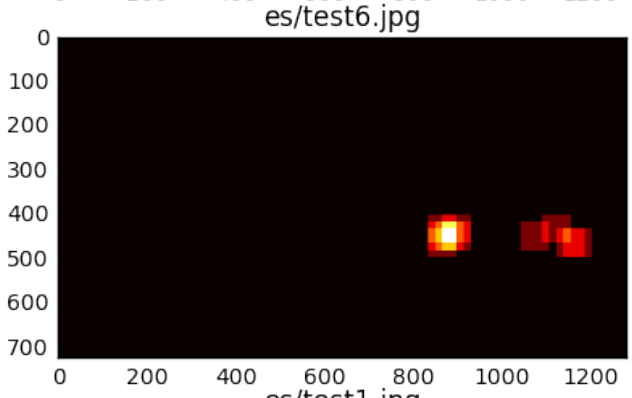


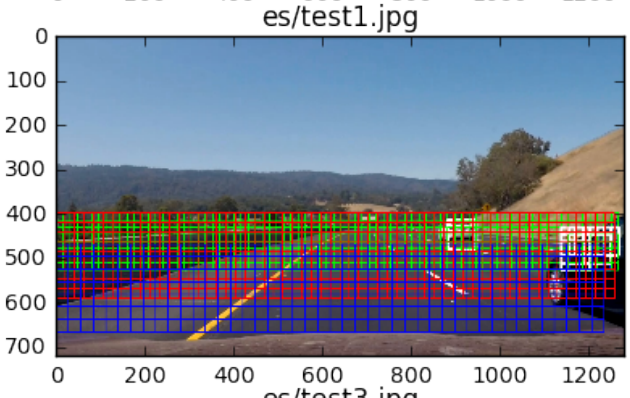
Here are some images of the windows printed in various colors and settled scales and ranges.

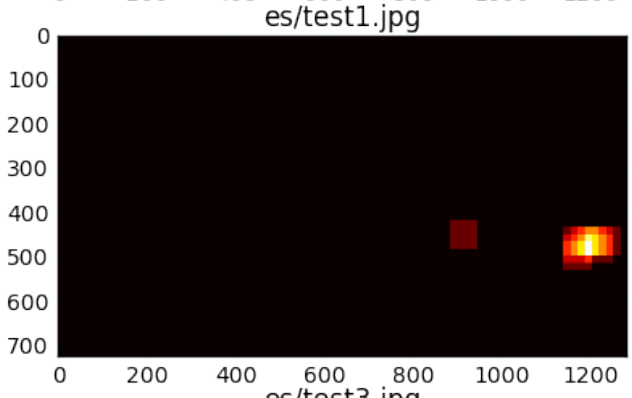


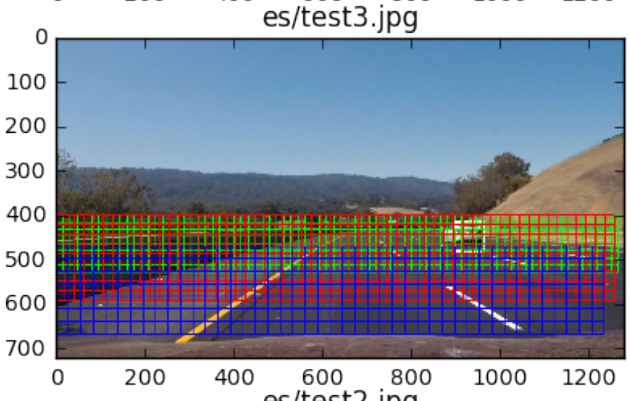


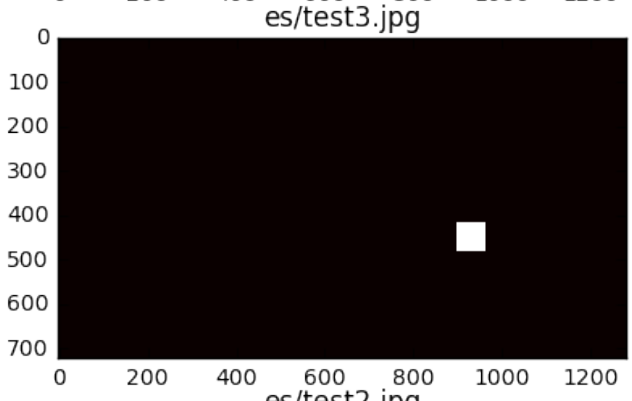


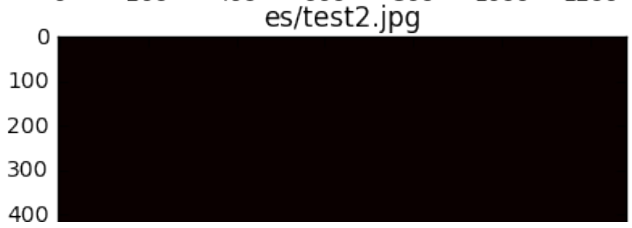


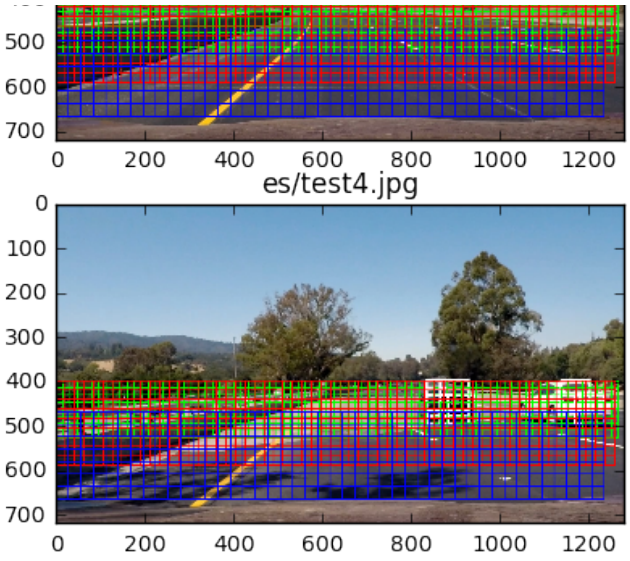


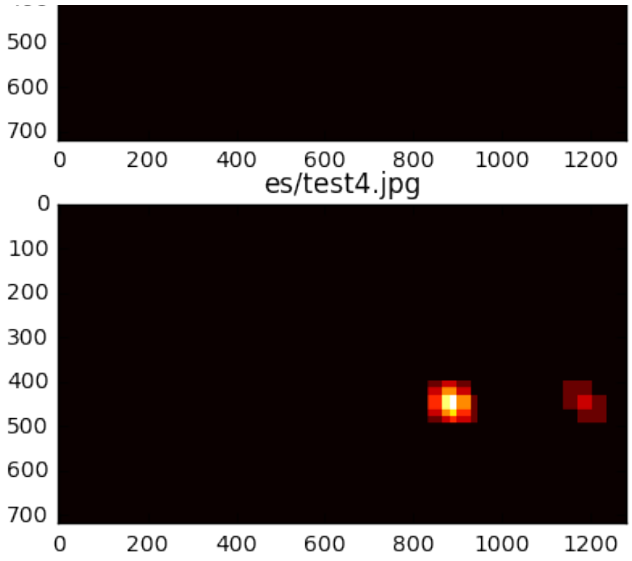








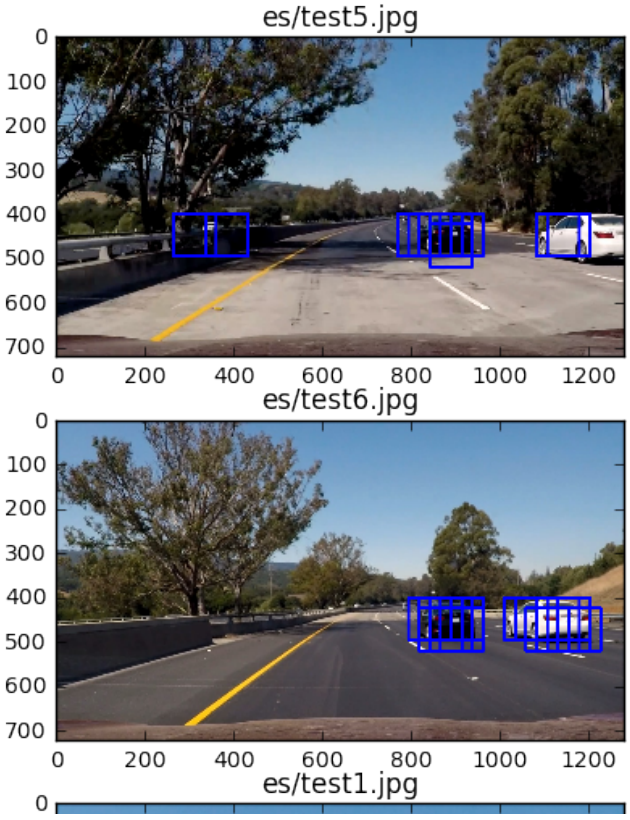


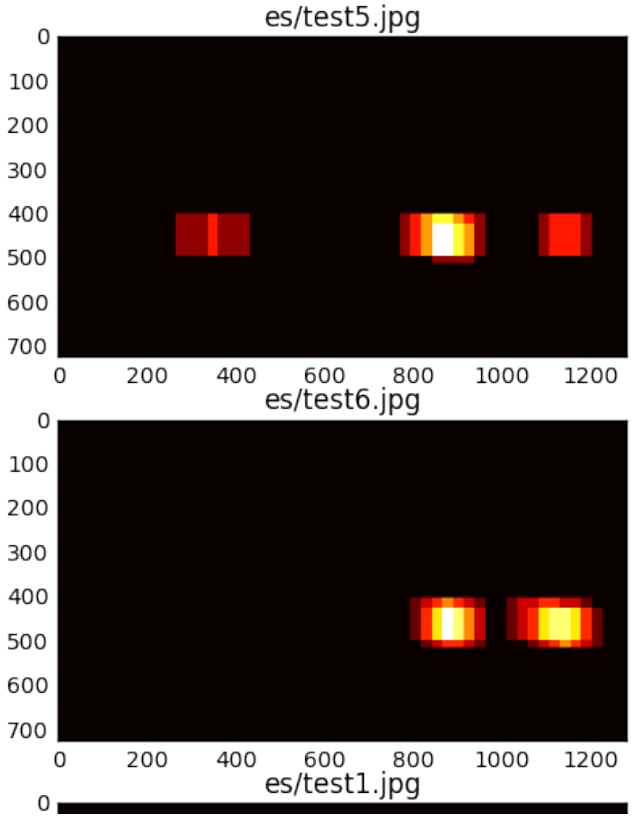


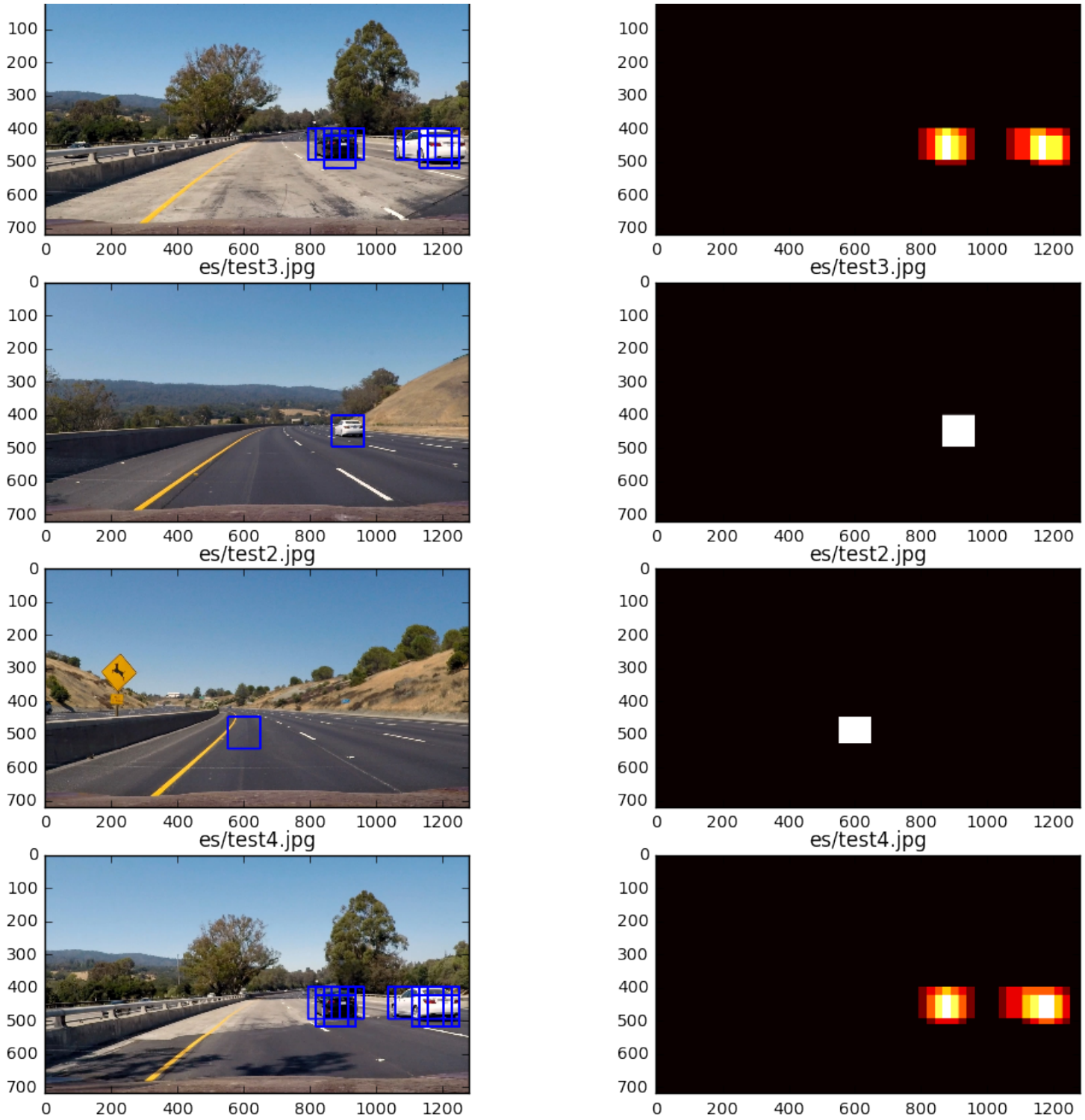
2. Show some examples of test images to demonstrate how your pipeline is working. What did you do to optimize the performance of your classifier?

I chose to leave my classifier with YCrCb colorspace with 9 orientations and searching the 3 scales that are described above with a step of 2 cells. To optimize the classifier I reduced the search space for each sliding window scale and decided to not go higher than roughly 6k feature vector space.

Here an output of a processed 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.

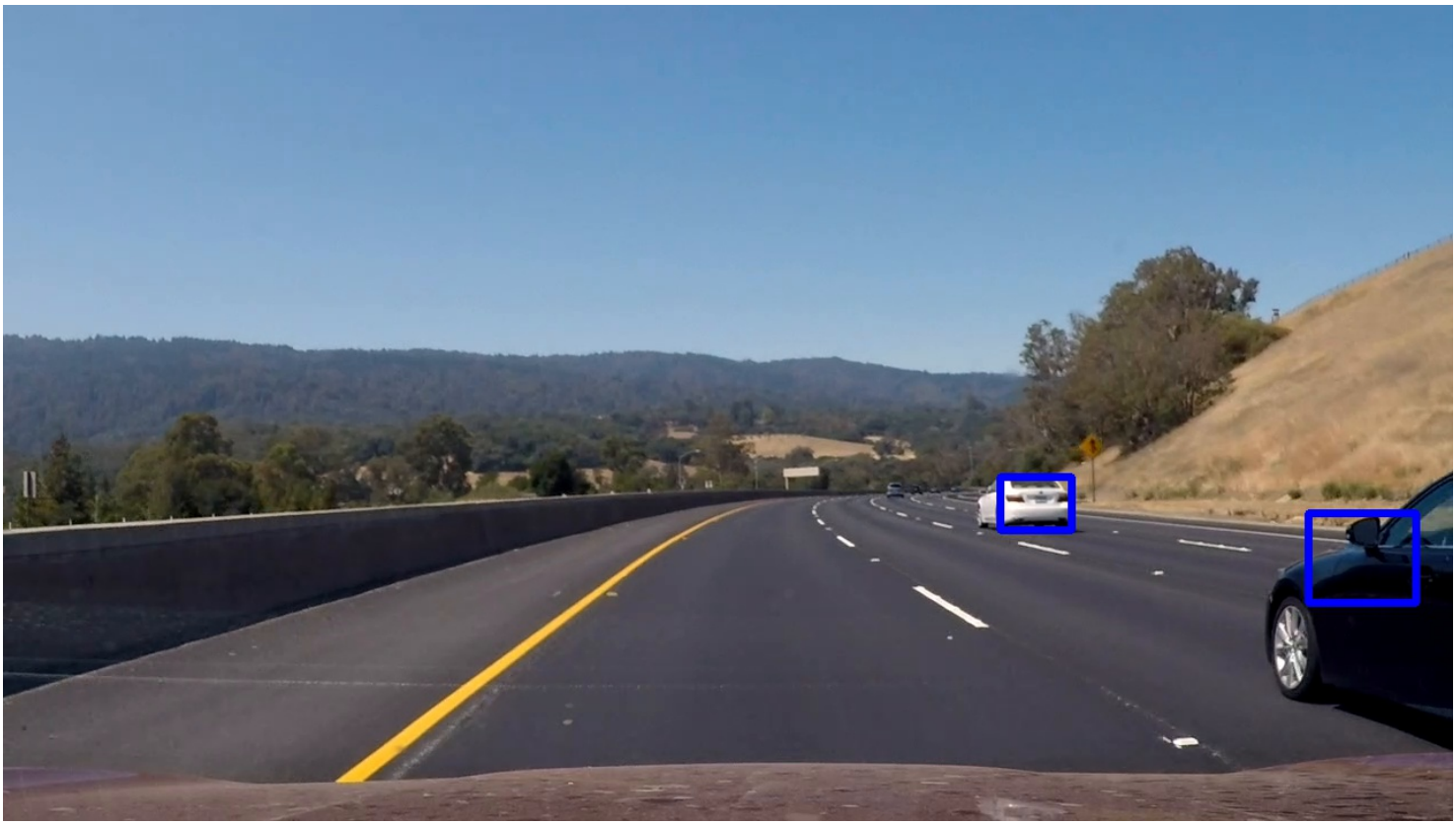
In sections 9 and 11 in my notebook I record the last 10 heat maps generated from my search windows and extract heatmaps routing from find cars from frame to frame in a FIFO queue. I then take the sum of this stacked heatmap and apply a threshold. Ultimately what this is doing is only rendering the bounding boxes for detected heat that had been consistently detected over a period of several frames. This effectively eliminated false positives from the output images of the video.

Here is a before and after image of my classifer of choosing hog params and window search parameters:

After tuning the HOG training classifer and window searching parameters, my pipeline success rate increased

Before:

After:



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.
Some problems I faced is false positives on the oppsite driving cars (on the other side of the road), I wasnt sure if this would actually be a false positive or a positive positive because it actually is a car, however we would probably want to treat these ones differently.

I would also try to do more scans of windows and scales of the image but im afraid that this will make the process way too slow.

For the window scales, I tried to make a smaller scale window search sizes, however this would not only end up not detecting the cars accuratley, it would cause an extreme amount of false positives. By analyzing how big the cars "should" be at any particular distance from the camera, I was able to gauge a better box size at different y-axis ranges.

This pipeline will probably fail if there was many different objects in the camera space that may look like cars to

the classifier, samples being billboards, the walls/windows of city buildings, or even just driving down a "main" street of a busy town or city. There there would be much more false positives taht would confuse the classifier.

If I were to continue this project I would probably feed the classifier images of stated above including buildings and billboards. I would also attempt to use a deep learning network to greatly increase the power of the pipeline.