P5 - Vehicle Detection and Tracking

Project Write-up

In this project, the goal is to write a software pipeline to detect vehicles in a video (starting with the test_video.mp4 and later implement on full project_video.mp4).

The Project

The goals/steps of this project are as follows:

- Perform a Histogram of Oriented Gradients (HOG) feature extraction on a labeled training set of images and train a Linear SVM classifier
- Optionally, apply a color transform and append binned color features, as well as histograms of color, to the HOG feature vector
- Normalize features and randomize a selection for training and testing
- Implement a sliding-window technique and use a trained classifier to search for vehicles in images
- Run the pipeline on a video stream (starting with the test_video.mp4 and later implement on full project_video.mp4) - 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

1) Provide awrite-up that includes all the rubric points and how you addressed each one.

You're reading it!

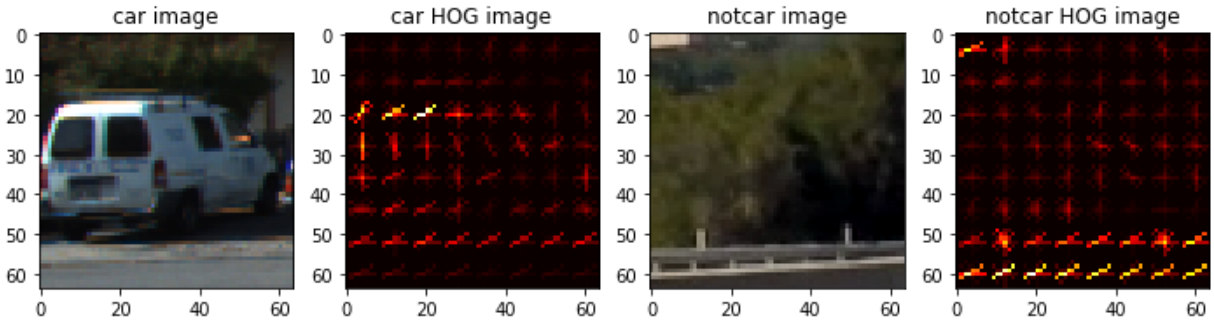
Histogram of Oriented Gradients (HOG)

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

I started by reading in all the vehicle and non-vehicle images in the second code cell of the Jupyter Notebook and the code for the get_hog_features function is contained in the third code cell.

I explored various different combinations of color space, orientation angles and pixels per cell. Slight variations from the numbers below were hard to deciper visually so I looked to the research for guidance. Here is an example of each of the vehicle and non-vehicle classes, along with their HOG images using color_space = 'YCrCb', orient = 6, pix_per_cell = 8x8, cell_per_block = 2x2 and hog_channel = 0:

In [80]:



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

As previously mentioned, I looked to the research to optimize this process. Navneet Dalal gives optimal cell size and block size recommendations (4x4 to 10x10 and 2x2 to 4x4, respectively) on page 5 of his whitepaper, based on his extensive testing using the HOG method that he helped develop. http://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf (http://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf)

He also talks about these ranges in his YouTube talk (https://youtu.be/7S5qXET179I (https://youtu.be/7S5qXET179I)) and states that 9 to 12 orientations are appropriate for our needs in this project. His paper states that beyond 9 bins "makes little difference."

These were my final choices:

- color_space = 'YCrCb'
- orient = 9
- pix_per_cell = 8
- cell_per_block = 2
- hog_channel = 'ALL'
- spatial_size = (32, 32)
- hist_bins = 32
- spatial_feat = True
- hist_feat = True
- hog_feat = True

Training a Classifier (Linear SVM)

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).

My Linear SVM classifier is trained in the eighth code cell of the Jupyter Notebook. It extracts features using extract_features() function, normalizes them, and stacks them on top of each other. I chose 10% of the data as my test set and shuffled them before training. Using the above parameters, there were a total of 8460 feature vector elements, as can be seen below, along with the results of training:

13.179030895233154 Seconds to compute features... Using: 9 orientations, 8 pixels per cell, 2 cells per block, 32 histogram bins, and (32, 32) spatial sampling Feature vector elements: 8460 1.94 Seconds to train SVC... Test Accuracy of SVC = 0.995

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?

The sliding window technique is used in this project to check for the existence of a vehicle based on the required features of a vehicle through multi-scale window sizes. In order to make the implementation faster, I have chosen to only slide through an area where cars would actually be located, cutting out everything above the horizon and anything near where the front hood of the car ends. Overlap was set at 50% and here again I will defer to the expertise of Navneet Dalal, who showed that 50% - 75% overlap provides material improvement in what's referred to as the "miss rate" in his whitepaper on HOG for human detection.

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?

In order to remove false positives, I was fortunate to have marked improvement simply by combining two slide windows, one at 64x64 and the other at a size of 128x128. Again, the overlap rate was set at 50% and my y range for searches was set at 400 to 656 to minimize unnecessary sampling.

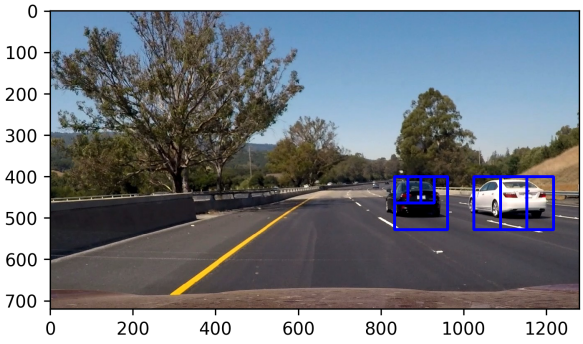
I saved the final set of images from each step in my pipeline. Those can be found in the output_images file included with my submission.

In [82]:

```
0.0 1.0
1.6351149082183838 Seconds to process one image searching 330 windows
0.0 1.0
1.2225840091705322 Seconds to process one image searching 330 windows
0.0 1.0
1.2430789470672607 Seconds to process one image searching 330 windows
0.0 1.0
1.3025097846984863 Seconds to process one image searching 330 windows
0.0 1.0
1.2889559268951416 Seconds to process one image searching 330 windows
0.0 1.0
1.2126598358154297 Seconds to process one image searching 330 windows
```





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:
https://www.dropbox.com/s/4sph758vwulgch6/project_video_output.mp4?dl=0
(https://www.dropbox.com/s/4sph758vwulgch6/project_video_output.mp4?dl=0)

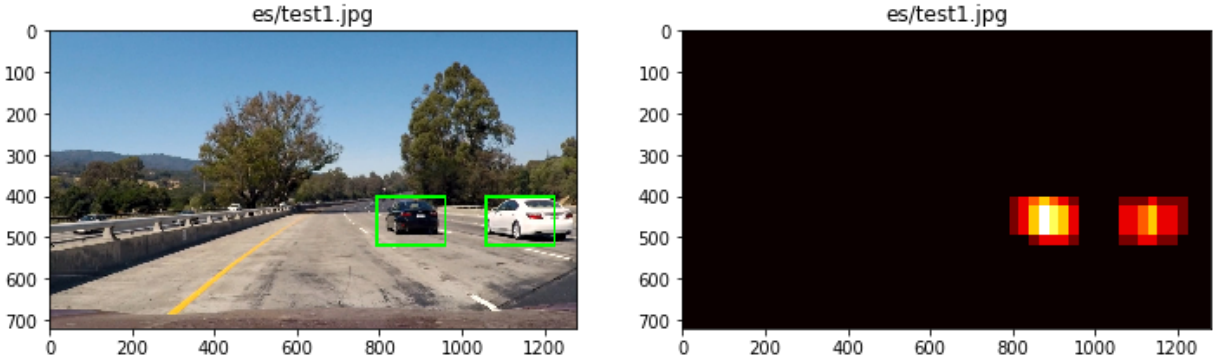
I'm kind of impressed that it is picking up vehicles on the other side of the barrier even though they are only in full view for a split second!

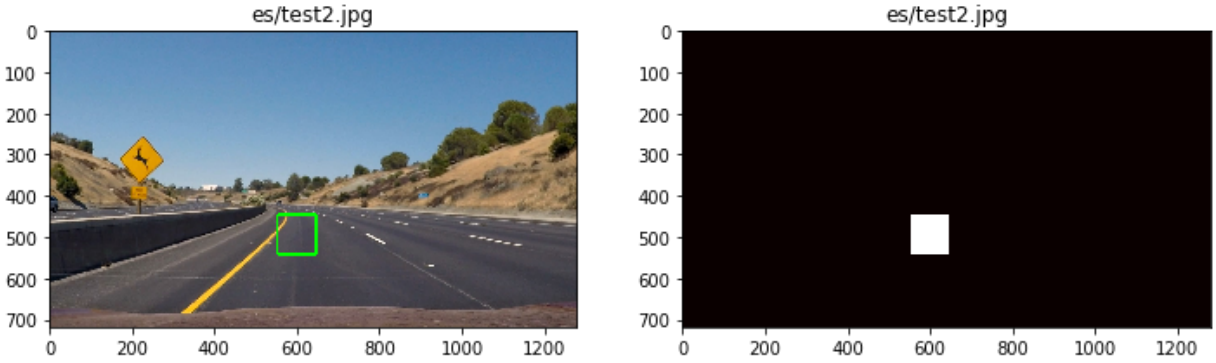
2) Describe how (and identify where in your code) you implemented some kind of filter for false positives and some method for combining overlapping bounding boxes.

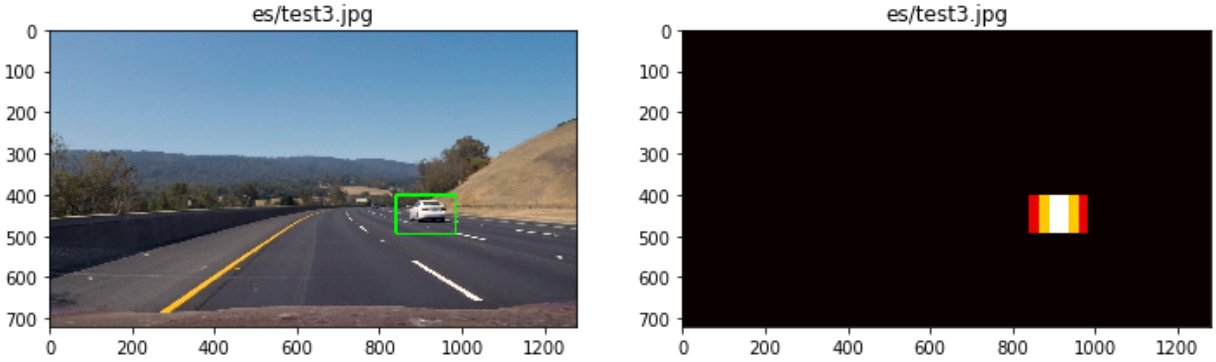
In the eleventh code cell of the Jupyter Notebook I recorded the positions of positive detections in each frame of the video. From the positive detections I created a heatmap and assumed each blob corresponded to a vehicle. I constructed bounding boxes to cover the area of each blob detected.

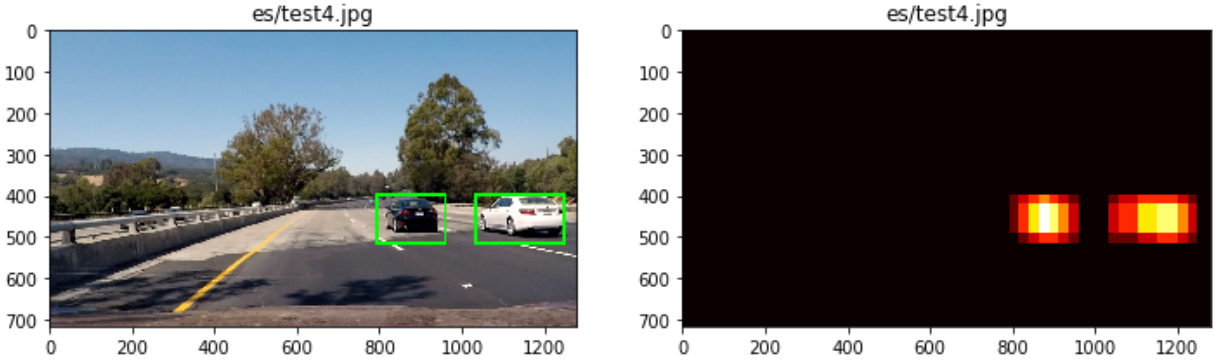
Here's an example result showing the heatmap from a series of frames of video and the bounding boxes overlaid on a frame from the video I created:

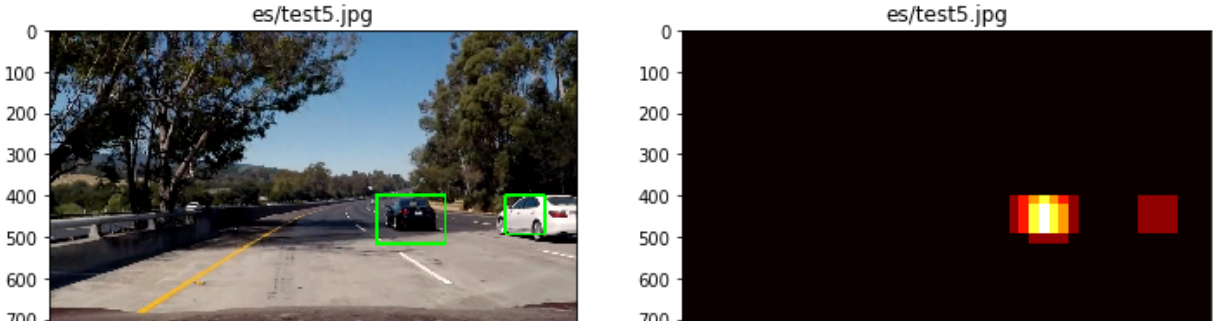
In [87]:

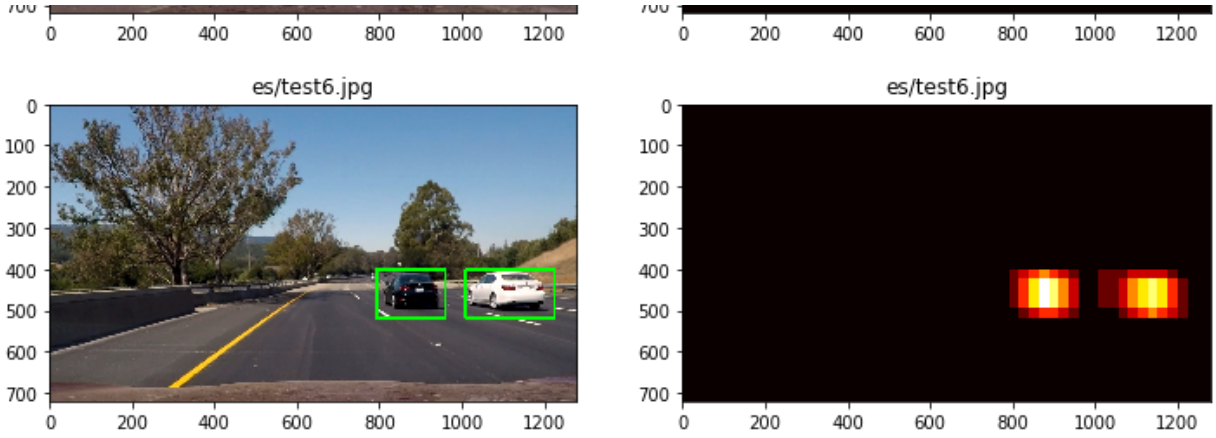

es/test1.jpg
es/test1.jpg


es/test2.jpg
es/test2.jpg


es/test3.jpg
es/test3.jpg


es/test4.jpg
es/test4.jpg


es/test5.jpg
es/test5.jpg



In [90]:

Out[90]:

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 video shows that object detection (vehicles, in this case) is being carried out reasonably well. Given the deadline for this project, I didn't get to tinker as much as I would have liked. I will revisit this model to attempt to eliminate the jitteriness of my bounding boxes. I'm happy that I was able to eliminate the pretty regular false positives that I was getting in the beginning, though I still count three false positives in my video.

This was a great first experience with image detection using a classic computer vision methods and a simple SVM classifier.but I'm really hoping we get to see how deep learning could provide more flexibility and much more space to tune parameters. Of course we lose our full intuition on intermediate steps due to the "black-box" behavior of neural nets. I think a combination of these two methods is what makes the task both more feasible and intuitive.

I would definitely like to try using a neural net for the classification problem instead of a linear SVM.

In [ ]: