PROJECT 5 WRITEUP - VEHICLE DETECTION AND TRACKING

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

Writeup / README

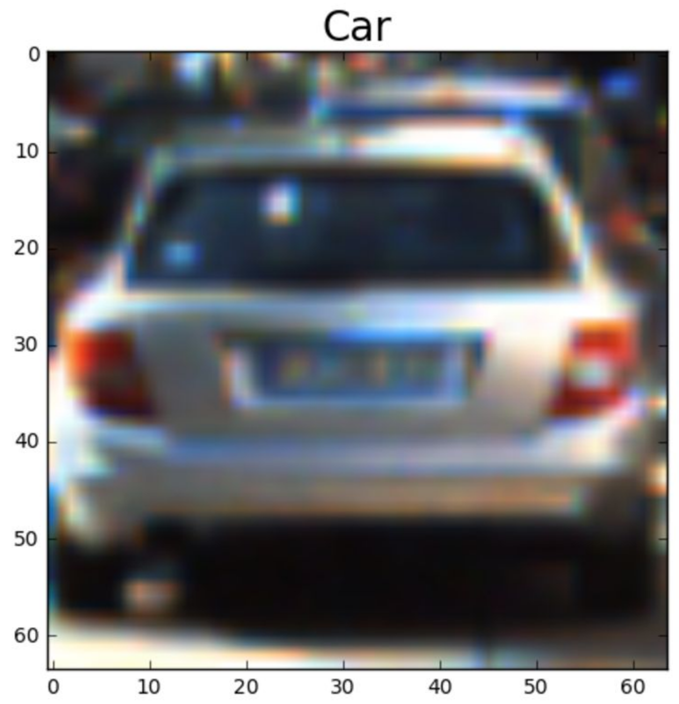
1. Provide a Writeup / README that includes all the rubric points and how you addressed each one. You can submit your writeup as markdown or pdf. Here is a template writeup for this project you can use as a guide and a starting point.

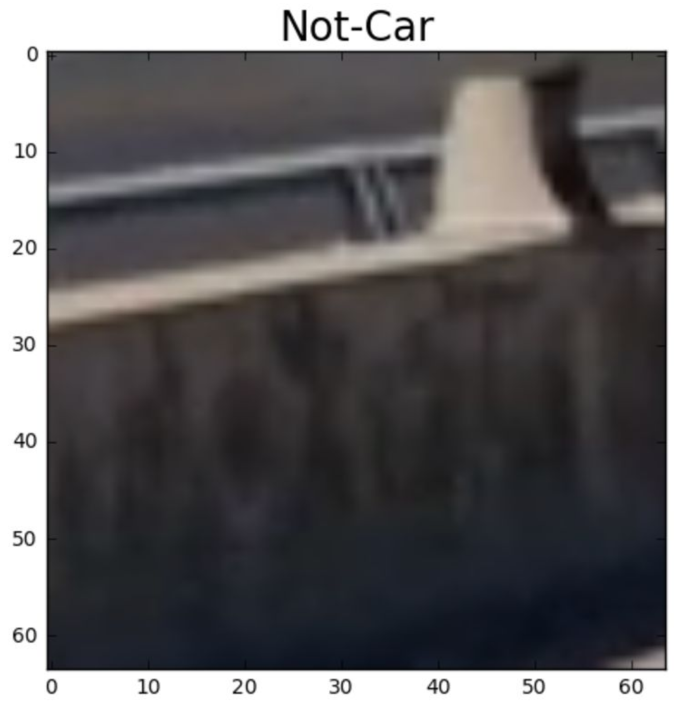
You're reading it!

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 the first, second and third code cells. 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.

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

I tried various combinations of parameters and in the end, the best results came from using:

- Color Space: YCrCb
- Orientations: 9
- Pixels_Per_Cell: 8
- Cells_Per_Block: 2
- HOG_Channel: ‘ALL’
- Spatial_Feat: True
- Hist_Feat: True
- HOG_Feat: True

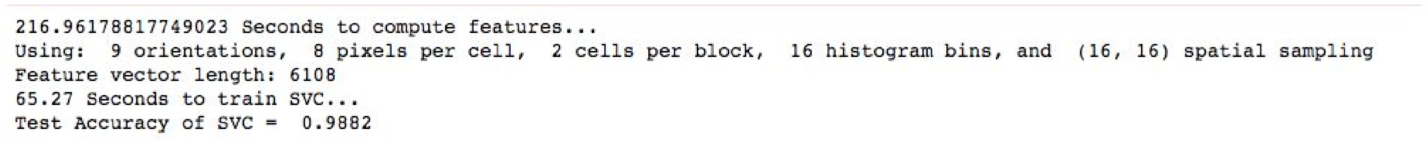
When attempting to complete this model, I spent a lot of time tweaking the parameted of the HOG function. One of the most important parameters was the color space. I first attempted RGB, but the results were sub-par. I then moved on to HLS. HLS resulted in much better detection rates. I used HLS in the last project, and I knew it worked well with gradients. When watching the video walkthrough, I saw that Ryan used YCrCb, so I decided to give that a try. I realized it was even better at picking up some of the gradients, so I stuck with that one.

I also tried different orientations. I tried both higher and lower orientation amounts, but 9 seemed to work perfectly to get the positive detections in the final video. I chose to use 8 pixels per cell and 2 cells per block. I didnt spend much time tweaking these parameters, because the results from the above two parameters provided a sufficient classification tool. Going forward, I would spend more time analyzing the result of changing these variables to see if they would output more effective detections.

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 by using the entire dataset. I extracted the features for the car and non-car images and then stacked them together into a scaler. I then defined the label vectors. After that, I split up the data into training and testing sets and then put them into my linear SVM classifier. See cell 13 in my notebook to view the code. I've also included the output from the cell below.

Testing the SVM on the entire dataset:

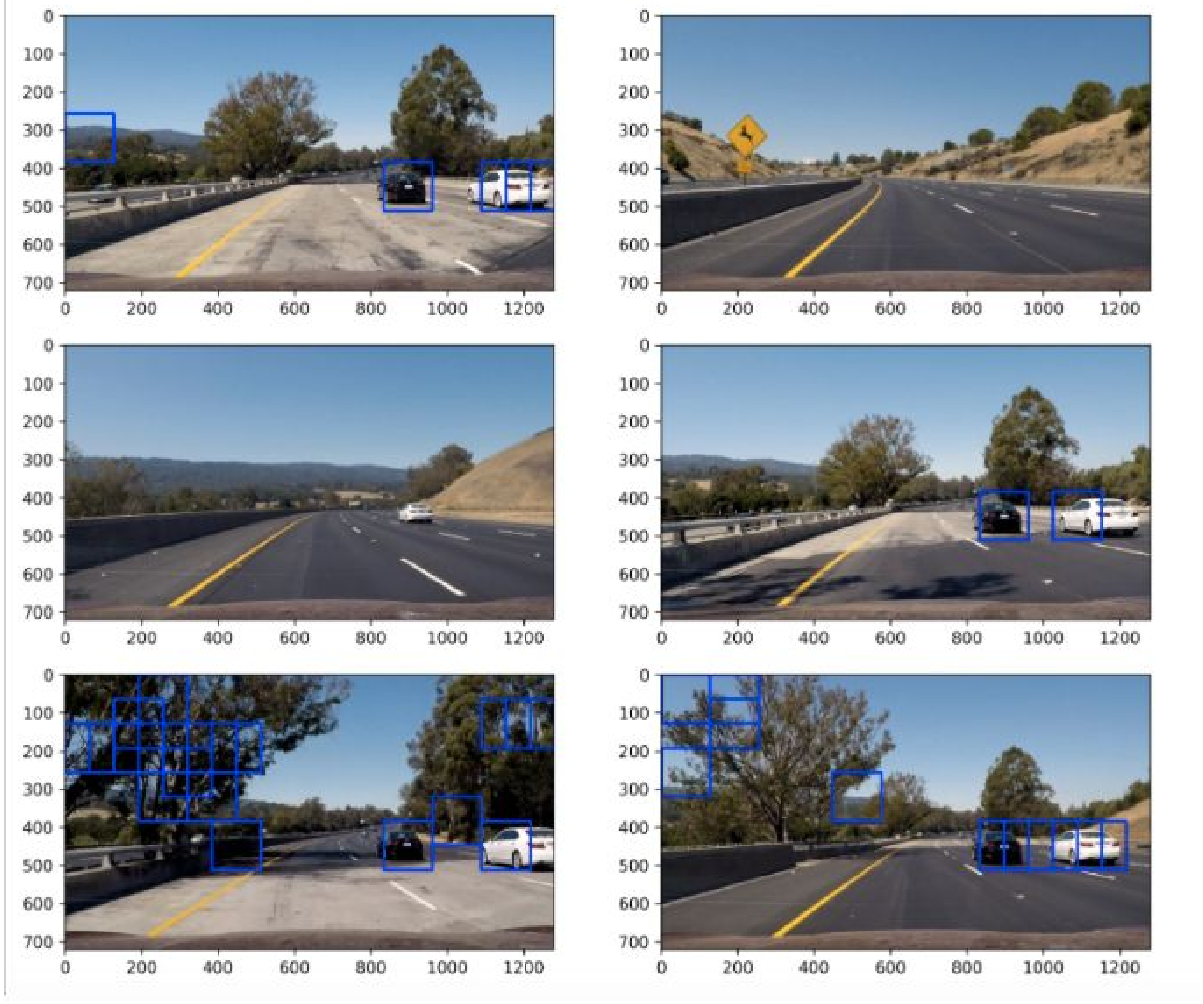


```
216.96178817749023 Seconds to compute features...
Using:  9 orientations,  8 pixels per cell,  2 cells per block,  16 histogram bins, and  (16, 16) spatial sampling
Feature vector length: 6108
65.27 Seconds to train SVC...
Test Accuracy of SVC =  0.9882
```

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 create a sliding window function and a window search function, and applied both to test images to see the results. One of the first things I did was restrict the y axis of the image to solely focus on the road. As mentioned in the video lectures, there is no need to search the sky for a car. I also applied an HOG subsampling method for each of the windows in the search. Here is the first batch of results I received:

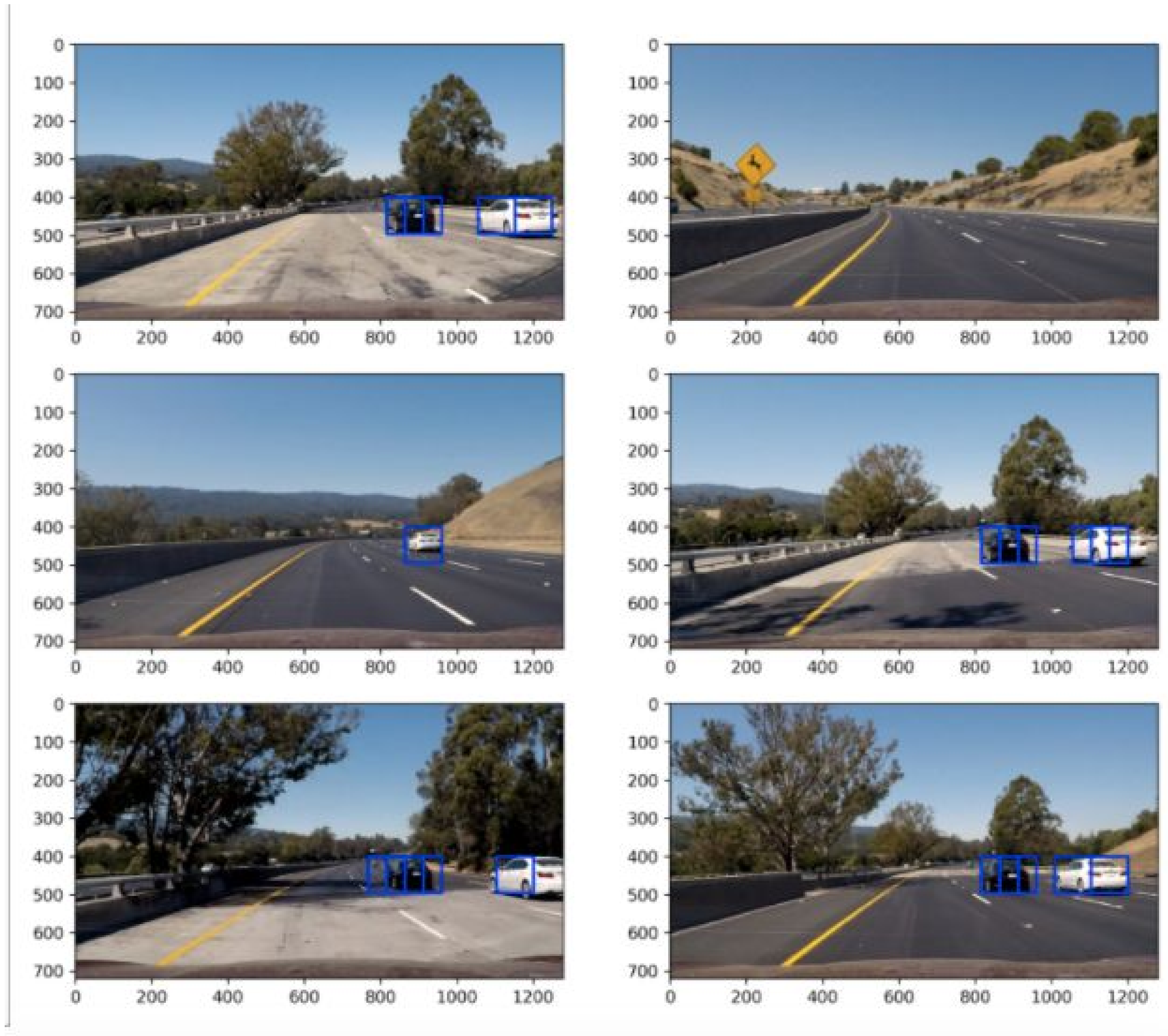

:

I chose my windows size by testing multiple different sizes. I finally chose (96,96) because it seemed to output the least amount of false positives. I chose my overlap size based on what we were given back in the lecture videos, because it seemed to work fine in my tests. I used 4 scales in the search to located the vehicles, and I appended them all to a list called “box_list” as shown in the following code:

```
box_list = []
box_list.append(find_cars(img, 380, 500, 1.0, svc, X_scaler, orient, pix_per_cell, cell_per_block,
spatial_size, hist_bins))
box_list.append(find_cars(img, 380, 570, 1.5, svc, X_scaler, orient, pix_per_cell, cell_per_block,
spatial_size, hist_bins))
box_list.append(find_cars(img, 380, 630, 2.0, svc, X_scaler, orient, pix_per_cell, cell_per_block,
spatial_size, hist_bins))
box_list.append(find_cars(img, 380, 700, 2.5, svc, X_scaler, orient, pix_per_cell, cell_per_block,
spatial_size, hist_bins))
```

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?**

As you can see, the classifier kept mistaking the trees for vehicles. To fix this, I decided to restrict the y axis to only focus on the road and not the sky and trees. I changed the window size to (96,96) and restricted the y-axis to (400 and 656). I searched on four separate scales using YCrCb, and ‘ALL’ HOG features. These parameters were decided upon after many tests, and these variables resulted in the fewest number of false positives. I also included spatially binned color and histograms of color in the feature vector. Here’s the result:



Video Implementation

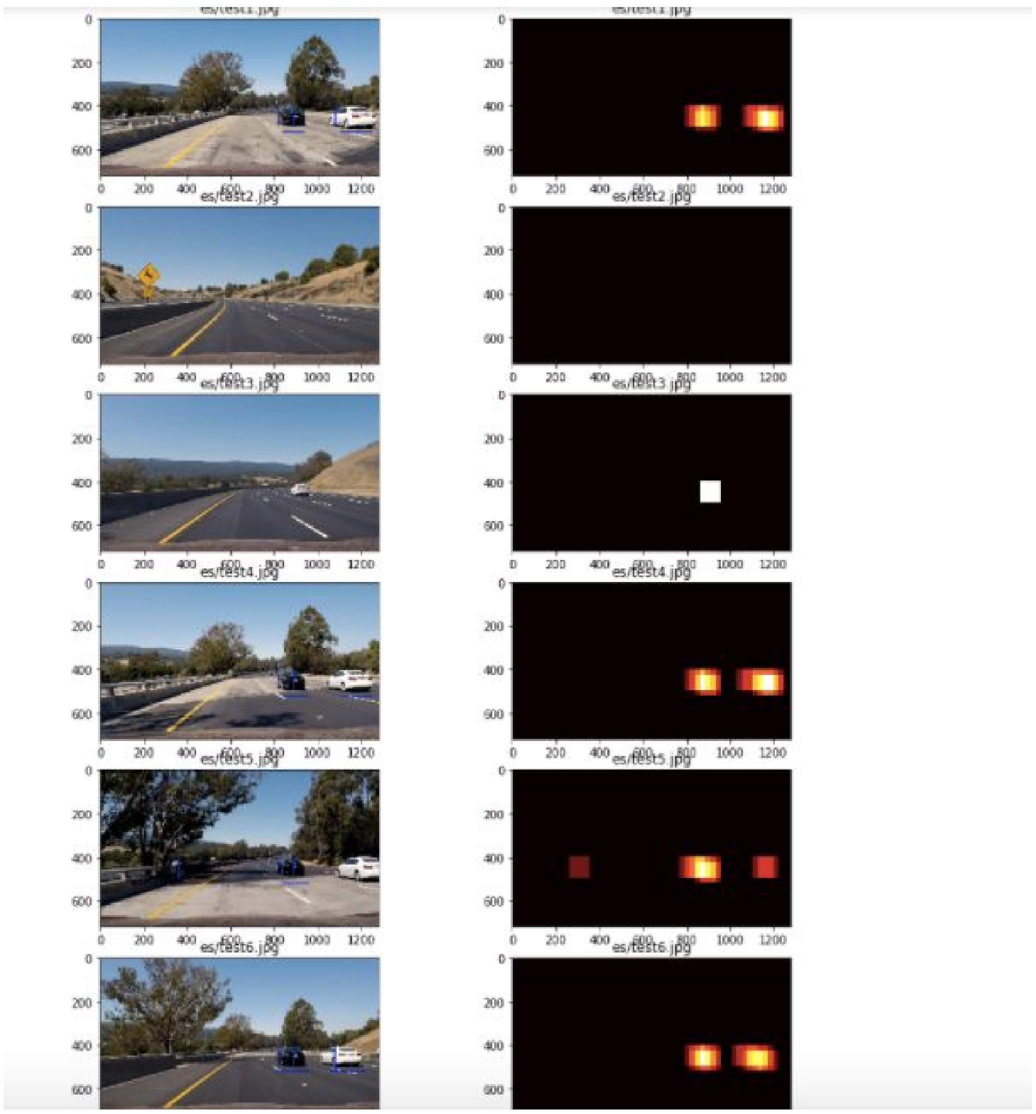
1. Provide a link to your final video output. Your pipeline should perform reasonably well on the entire project video (somewhat wobbly or unstable bounding boxes are ok as long as you are identifying the vehicles most of the time with minimal false positives.)

Here's a link to my video result

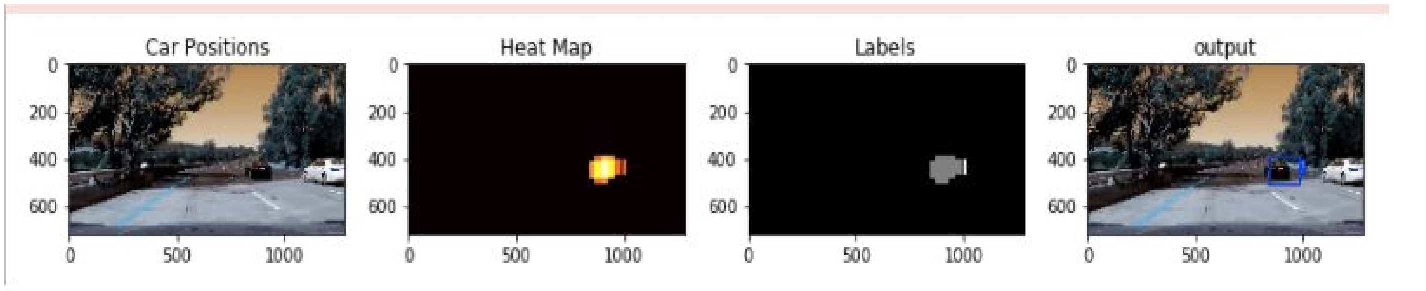
2. Describe how (and identify where in your code) you implemented some kind of filter for false positives and some method for combining overlapping bounding boxes.

First I generated heatmaps for each of the positive detections. I then thresholded the map to identify the vehicle positions. I then used `scipy.ndimage.measurements.label()`to identify individual blobs in the heatmap. I knew that each blob corresponded to a vehicle in the image. I constructed bounding boxes to cover the area of each blob detected.

Here are the six test images and their corresponding heatmaps:



And here is the heatmap thresholding and labeling applied to test image 5:



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?

I spent a tremendous amount of time working on this project and testing out different parameters to see which ones resulted in the best detections. One really important thing that I noticed was that the classifier wasn't able to detect vehicles directly parallel to the car. How does a car know where vehicles are if they aren’t in front of it. I feel like this is something really important for a car to take into consideration for instances such as switching lanes or making turns.

Also, I am curious to know how the classifier does in low light conditions such as night time. The only illumination would be the headlights and possibly the streetlights, which could make it difficult to extract the HOG features and classify the cars.

One last implementation I could make to this project is to apply a function to smooth the boxes being drawn. The video stream shows the boxes being drawn too often, and it looks jumpy. By averaging over several video frames, I could make the bounding boxes appear more smooth and consistent.

To complete this project I followed along with the walkthrough video and coded all of my own cells in the notebook. I gained a solid foundation of understanding from watching the video and following along. I understand what each line of code does within the notebook, and I am happy with the result of this project. I plan to return to the lectures to review the different classifiers and get a deeper understanding of HOG as well. I hope you enjoyed checking out my project and learned something new about using all these different tools to detect vehicles in camera images!