Vehicle Detection

The goal of this project 5 of CarND course is to detect the nearby vehicles on the road. A number of test images and 1 test video are provided to test the final pipeline.

The project is done at 2 main steps with several sub-steps:

I. Detect the vehicles on the images,

- Perform a Histogram of Oriented Gradients (HOG) for feature extraction on a labeled training set of images and train a Linear SVM classifier
- Apply a color transform and append binned color features, as well as histograms of color, to HOG feature vector.
- Implement a sliding-window technique and use the trained classifier to search for vehicles in images.

II. Extend the pipeline developed for vehicle detection to videos

- Run the pipeline on a video stream
- Create a heat map of recurring detections frame by frame to reject outliers and follow detected vehicles.
- Estimate a bounding box for vehicles detected.

Related Files:

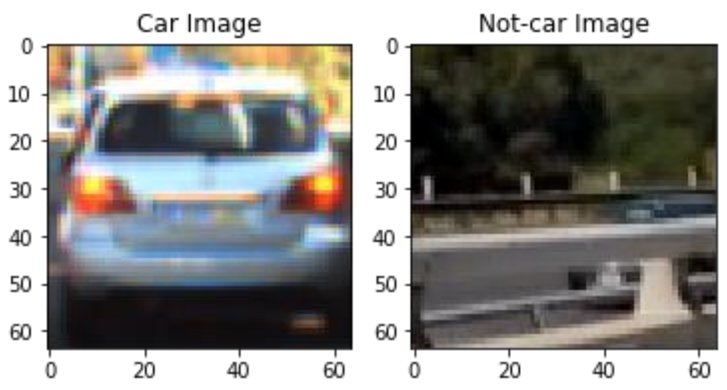
- P5_VehicleDetection.ipynb
- project_video_out.mp4
- This writeup
- output_images folder

Note: the notebook P5_VehicleDetection.ipynb includes all the codes and functions for this project; some of them are described in this report.

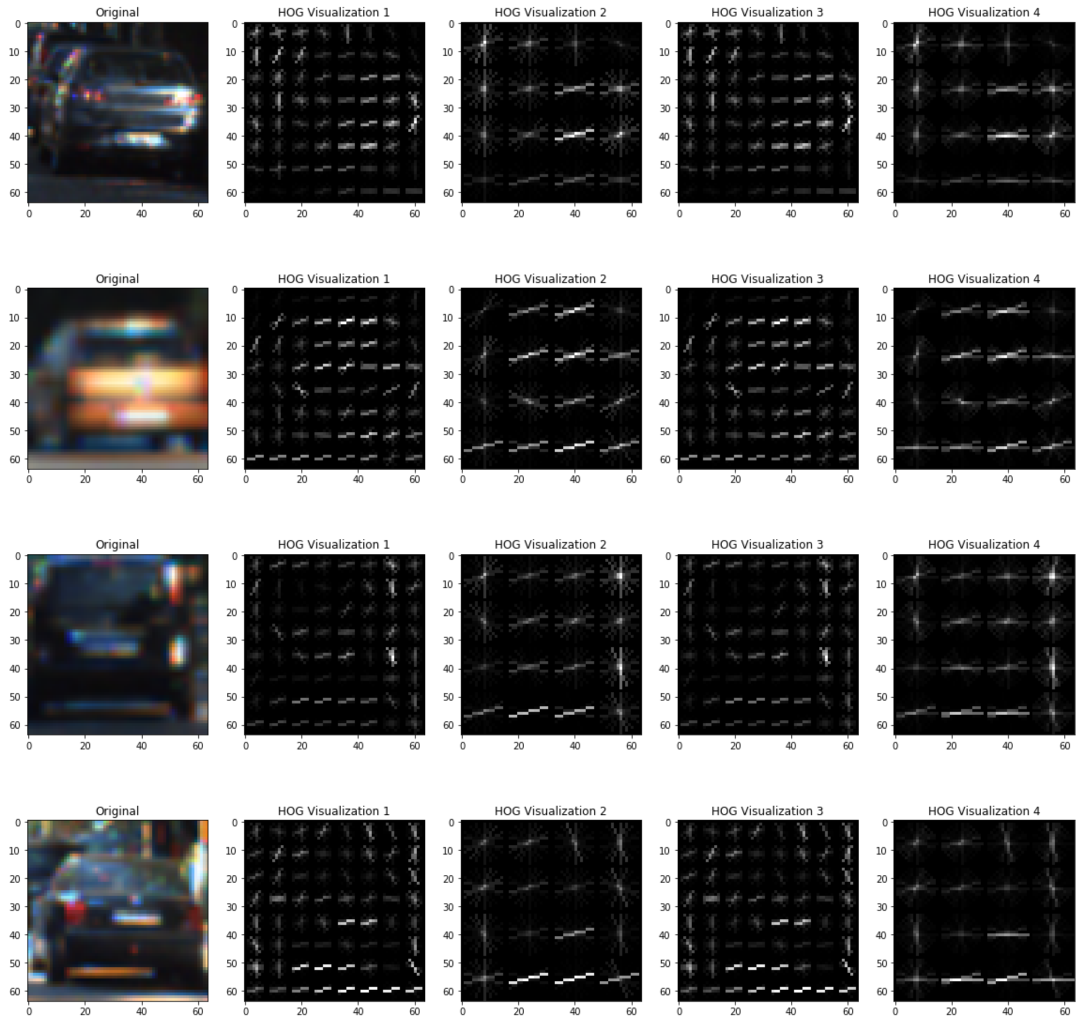
Histogram of Oriented Gradients (HOG)

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 second code cell of the IPython notebook. 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 on_test_images using the `YCrCb` color space and different HOG parameters as described in cell 3 of the project jupyter notebook:



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

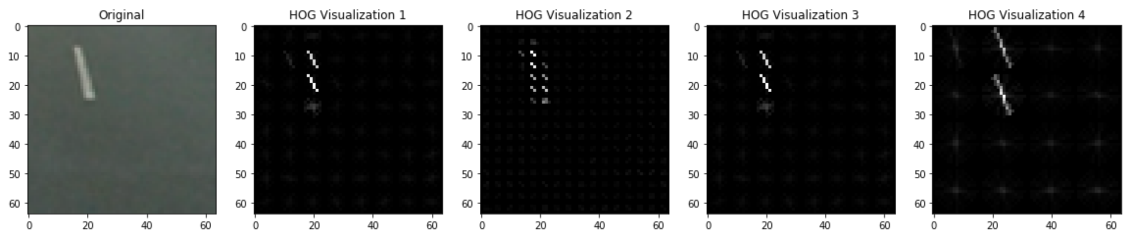
For HOG feature extraction I started with the following parameters as baseline:

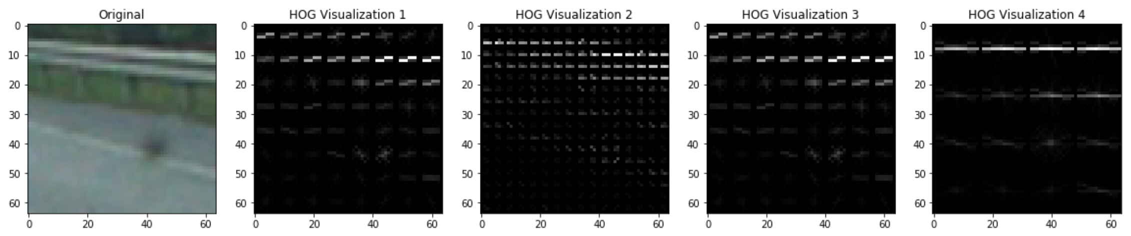
orient= 9, pix_per_cell= 8, cell_per_block= 2

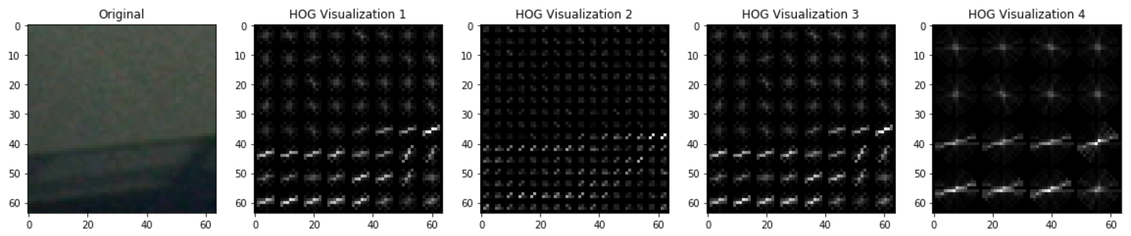
The result of the above parameter combination is on the 2nd column of the above figure. Then, I changed the above parameters (result on 3rd to 5th column in above figure) to search for the better combination of them that provides better shape detection for car, finally I landed on the below parameters after several try on car and notcar images that showed a better robustness in distinguishing between the shape of cars and notcars:

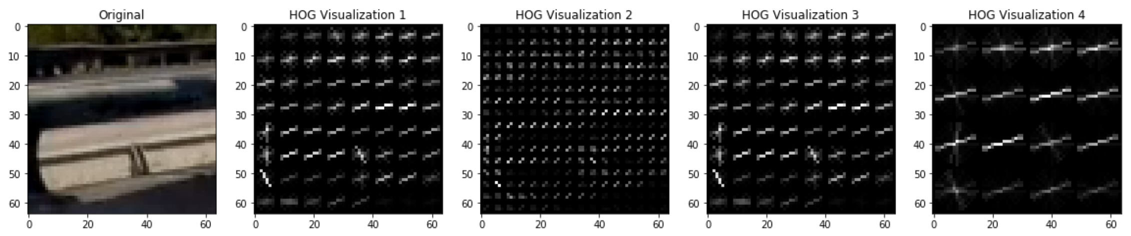
orient= 15, pix_per_cell= 8, cell_per_block= 4

The output result on notcar images are shown below:

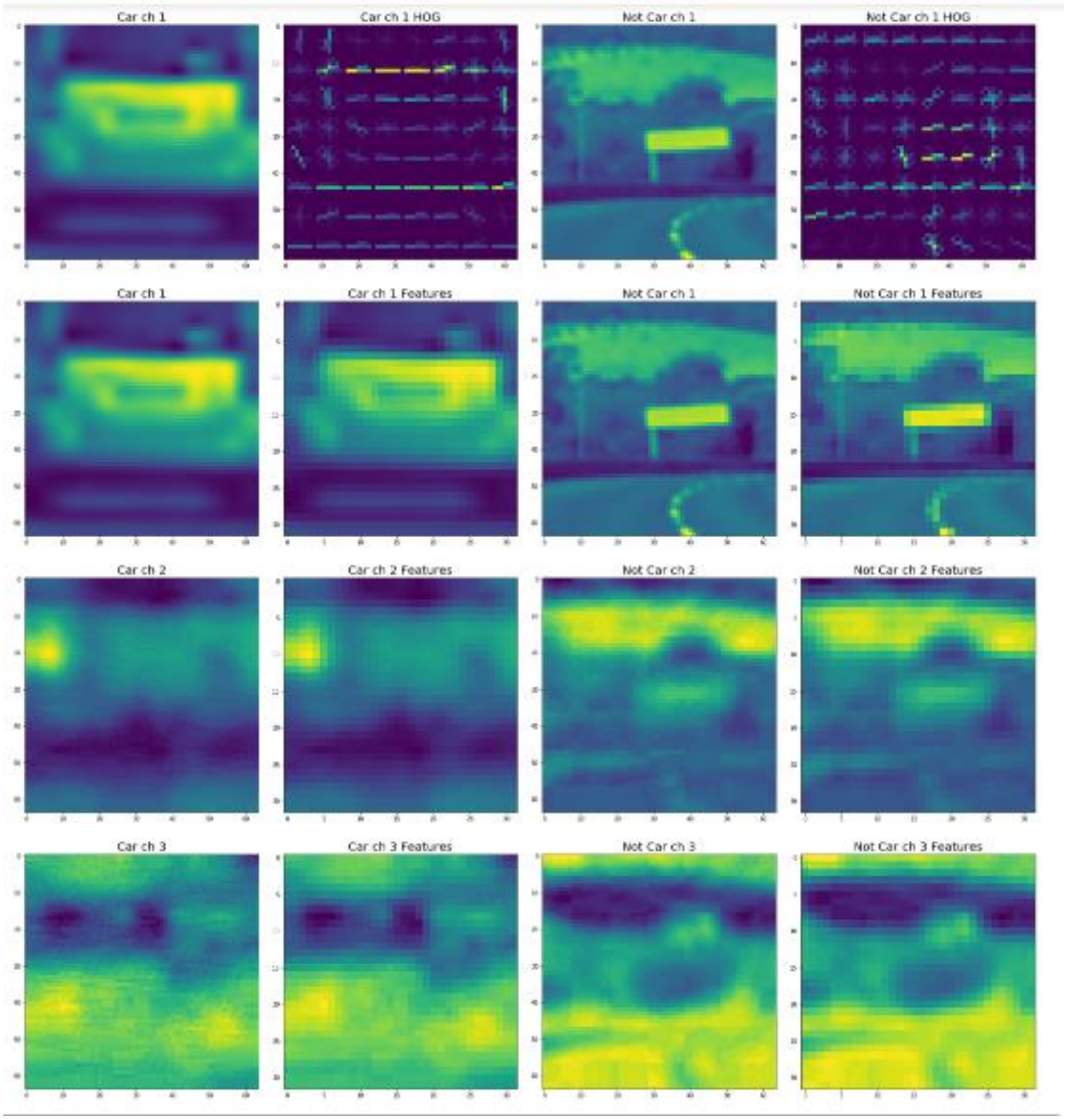








I also looked at the HOG in YcrCb color channel and other color channels and their features to find out the best combination of HOG feature and color channels. Some example of the result is below. Looking at the results I found out HOG and channel 1 of YcrCb can be better features for vehicle detection:



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 (SVC) using the following code:

```
# Use a linear SVC
svc = LinearSVC()
# Check the training time for the SVC
t=time.time()
svc.fit(X_train, y_train)
t2 = time.time()
```

The extracted features for this classifier are HOG, bin spatial and histogram of colors. The accuracy after training on only 1000 training data reached to %96 after 9 second of training:

```
9.0 Seconds to extract HOG features...
Using: 9 orientations 8 pixels per cell and 2 cells per block
Feature vector length: 1764
1.94 Seconds to train SVC...
Test Accuracy of SVC =  0.96
My SVC predicts:  [ 0.  1.  1.  0.  1.  1.  1.  0.  0.  1.]
For these 10 labels:  [ 0.  1.  1.  0.  1.  1.  1.  0.  1.  1.]
0.0 Seconds to predict 10 labels with SVC
```

Different numbers of training images are used and yield different results:

```
Using: 15 orientations 8 pixels per cell and 4 cells per block
Feature vector length: 21168
49.15 Seconds to train SVC...
Test Accuracy of SVC =  0.9975
My SVC predicts:  [ 1.  1.  0.  0.  1.  0.  1.  0.  1.  1.]
For these 10 labels:  [ 1.  1.  0.  0.  1.  0.  1.  0.  1.  1.]
0.67189 Seconds to predict 10 labels with SVC
```

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?

First for efficiency not all part of the image is searched for vehicles; given that the camera position is fixed then the search area is limited to only the lower part of the images where the vehicles can actually exist; after trying different positions for ystart I decided to search part of image at 380 pixels or higher to detect far and small enough vehicles on the image.

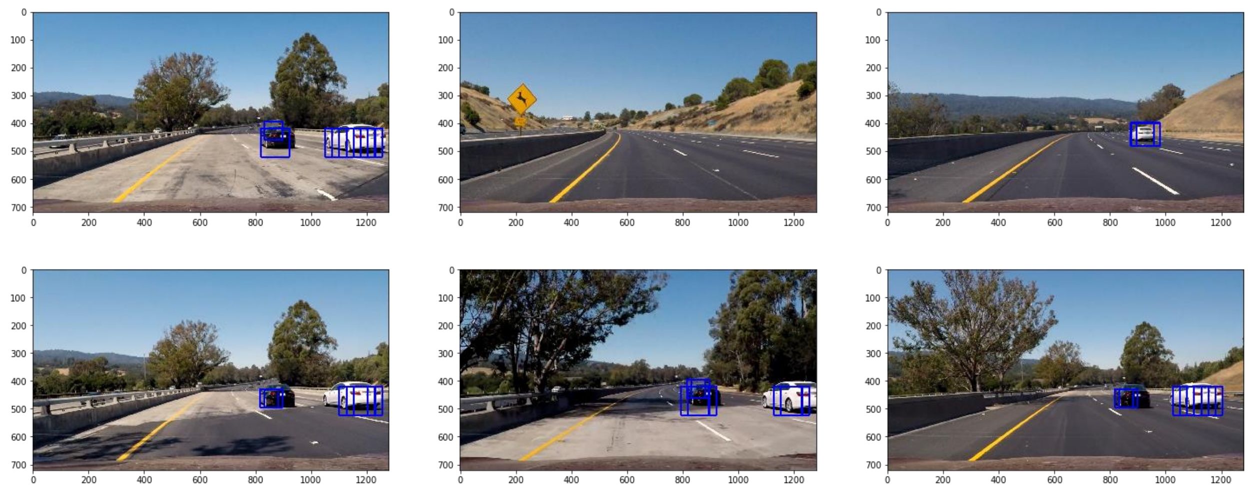
I decided to search random window positions at random scales on the search area of images and came up with the below combinations for ystart, ystop and scales:

```
ystart = [380, 400, 410, 420, 430, 430, 440, 400, 400, 500]
ystop = [480, 500, 500, 556, 556, 560, 560, 560, 560, 650]
scales = [1, 1.3, 1.4, 1.6, 1., 2.0, 1.9, 1.3, 2.2, 3.0]
```

This combination is selected considering the fact that the farther vehicles will appear smaller and approximately in the middle of the image (380 to 430). Also, nearby vehicles will appear bigger and need a larger scale.

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?

Ultimately I searched on all the above mentioned scales using YCrCb 3-channel HOG features plus spatially binned color and histograms of color in the feature vector, which provided a good result. The results of window search with these parameter combinations on the test images are as below, which detects enough windows for every vehicle (and interestingly no false positive):



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.)

The function video_processor() is created to process each frame of the video, it's input is the video frame images and it outputs an image with detected vehicles boxed in it. The video is stored in the repository with the name project_video_out.mp4.

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 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; a threshold of 1 is used meaning at least 2 detections is required for a vehicle; the result on test images is shown below:

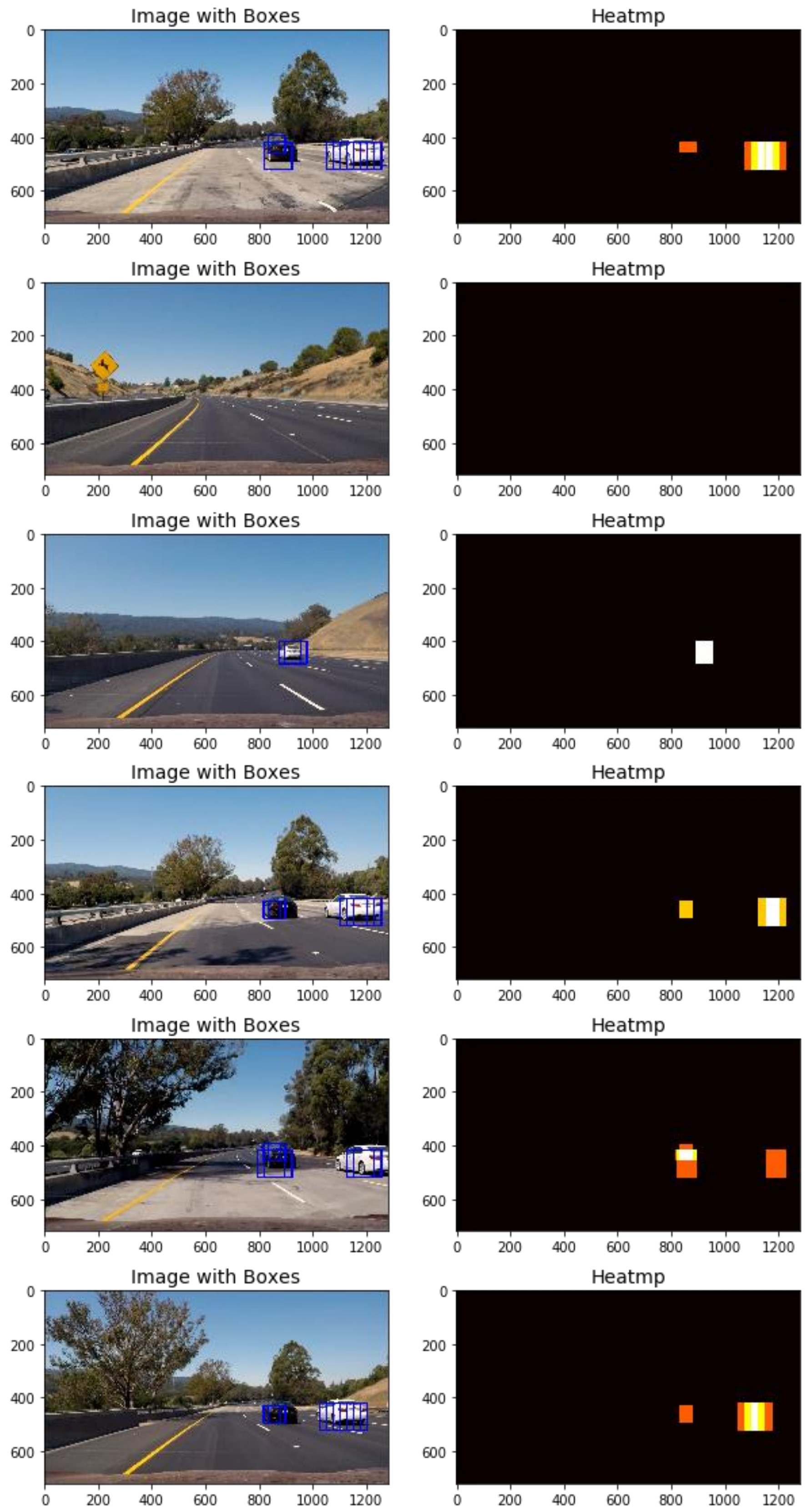
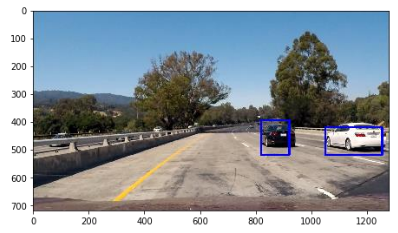
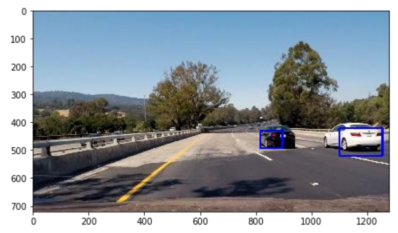
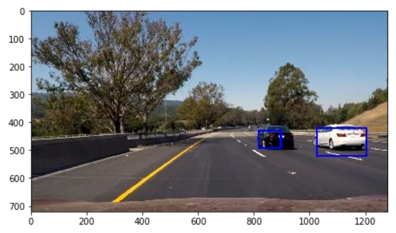


Image with Boxes
Heatmp
Image with Boxes
Heatmp
Image with Boxes
Heatmp
Image with Boxes
Heatmp
Image with Boxes
Heatmp
Image with Boxes
Heatmp
0
200
400
600
0
200
400
600
800
1000
1200

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. Here's the result on the test_images (so far no false positive and no false negative):







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?

While using more images for training SVM classifier will increase the reliability of the classifier, I noticed the time to train will increase with size of training set exponentially which needed higher computation power.

Another observation was that the classifier performance is very sensitive to choice of parameters.

The performance of this pipeline can be improved by making the bounding boxes less unstable. This can be achieved by looking at more number of subsequent frames in the video at the same time, but this might increase the processing time.