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, t 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'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.¶

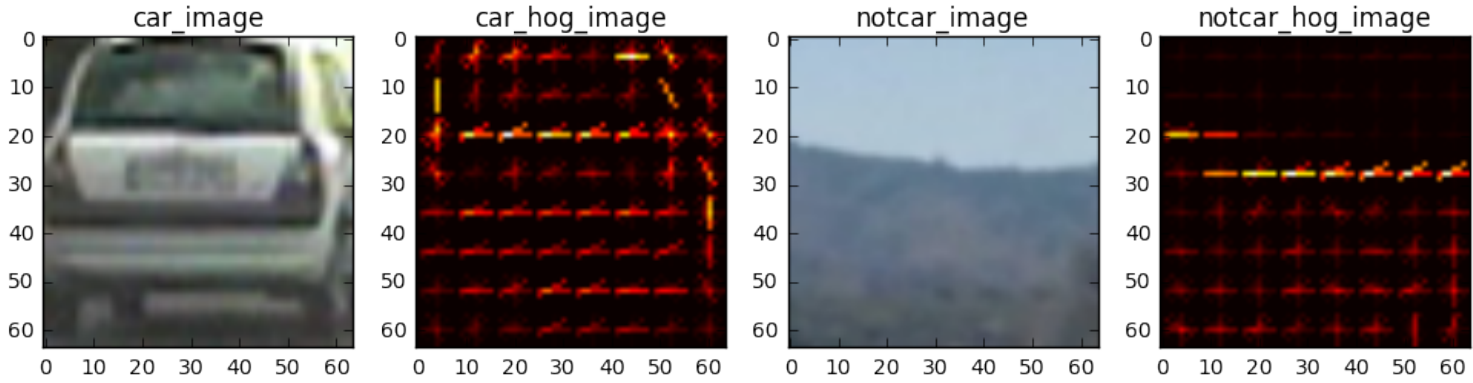
The code for this step is contained in the IPython notebook track.ipynb.

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 method `load` in the notebook):

We have 8792 vehicle samples and 8968 non vehicle samples.

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 using the `YCrCb` color space and HOG parameters of `orientations=6`, `pixels_per_cell=(8, 8)` and `cells_per_block=(2, 2)`:



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

I tried various combinations of parameters and found that following parameters work best and acrieved a 99.5% accuracy on detection in support vector classification:

```
orientation = 9
pix_per_cell = 8
cell_per_block = 2
```

OK- I actually just did few experiments with Ryan and took the best as he went through the video.

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 using test car and noncar data with feature vector size 8412 and got 99.55% accuracy.

We extract features for both car and noncar images. To get feature of a single image we go through following steps (metho name `single_img_features` in the notebook):

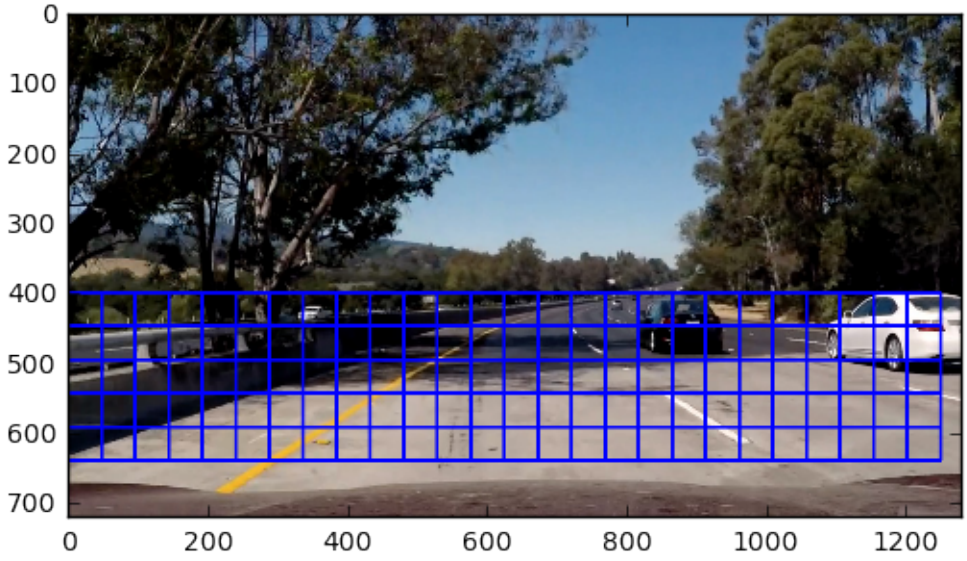
- Apply color conversion
- Compute spatial features and to the feature list
- Compute histogram features and to the feature list
- Compute HOG features and to the feature list
- Return concatenated array of features

I have split the dataset for training and testing. I used 90% for training and 10% for testing. Splitted using scipy `train_test_split` method and trained the classifier.

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?¶

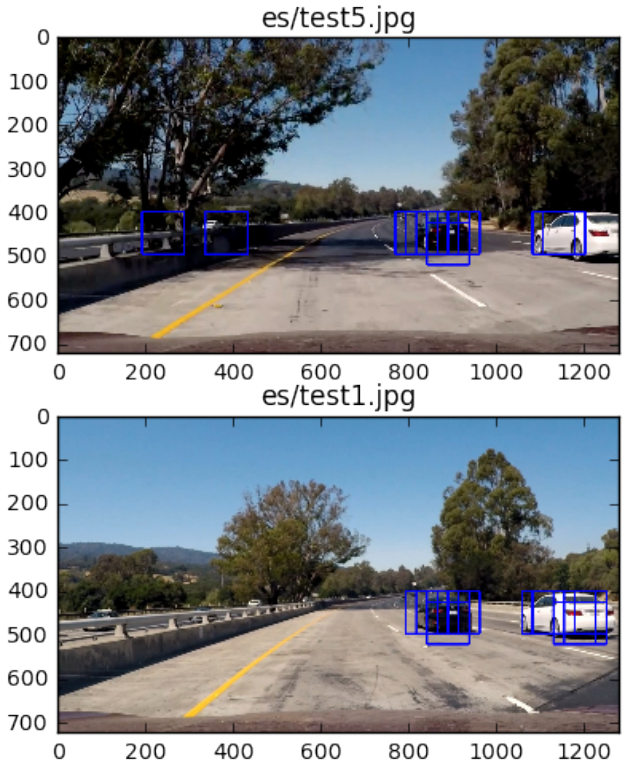
I have taken bottom 256 pixel high area starting at row 400 and ending at row 656 and taking all columns from the search image. Then I have used 96x96 window to slide through the entire search image with 50% overlap for each box to the next box. The windows are crteated using method name `slide_window` in the notebook.



I have tried a few combination like 64x64, 128x128 and 96x96 window sizes and found that 96x96 works best for test images and then 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?¶

Ultimately I searched on two scales using YCrCb 3-channel HOG features plus spatially binned color and histograms of color in the feature vector, which provided a nice result. Here are some example images:



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 are links to my video results:

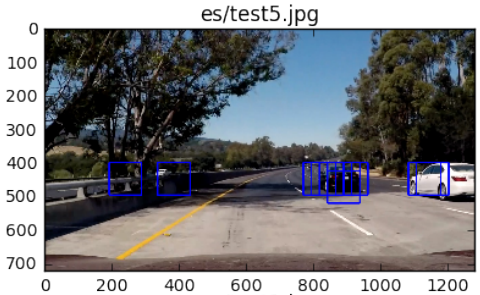
- Without threshold (some false positive): https://youtu.be/C4GEiQzyWlc.
- With threshold (some false negative) https://youtu.be/9d_ImqfHGEs.

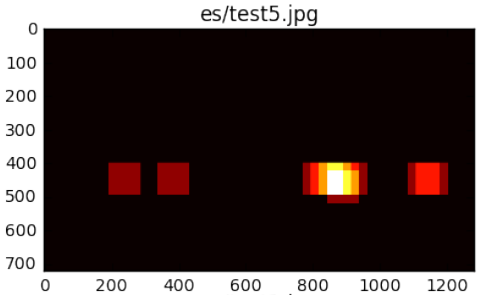
2. Describe how (and identify where in your code) you implemented some kind of filter for false positives and some method for combining overlapping bounding boxes.¶

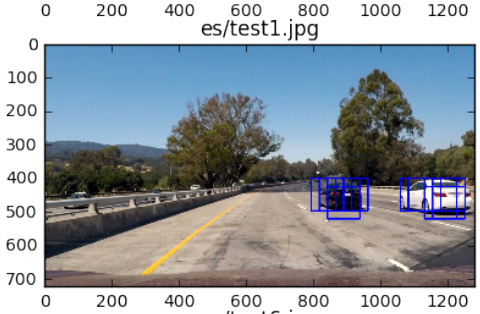
I recorded the positions of positive detections in each frame of the video. From the positive detections I created a heatmap and then thresholded that map to identify vehicle positions. I then used `scipy.ndimage.measurements.label()` to identify individual blobs in the heatmap. I then assumed each blob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

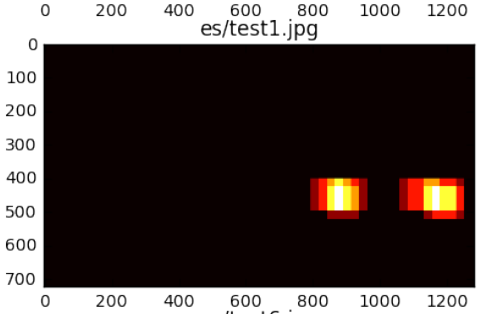
Here's an example result showing the heatmap from a series of frames of video (extracted as image stored in test_images folder), the result of `scipy.ndimage.measurements.label()` and the bounding boxes then overlaid on the frame of video:

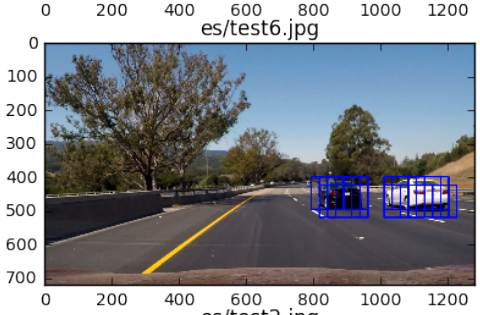
Here are few frames and their corresponding heatmaps:¶

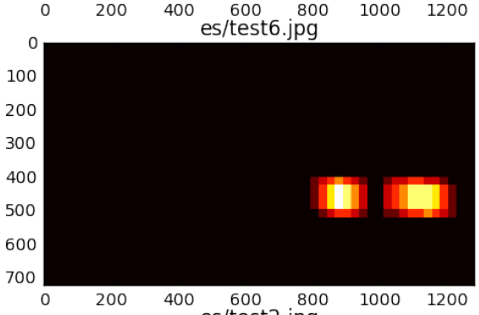


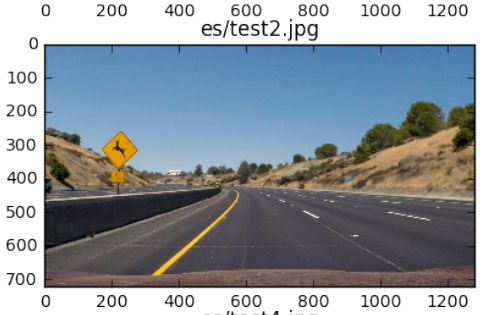


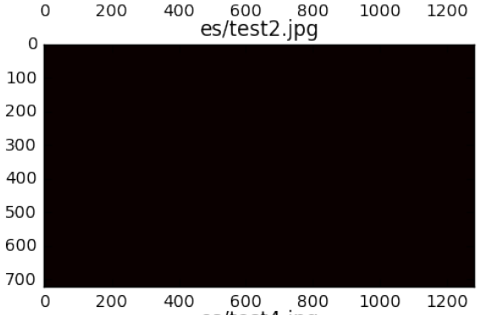


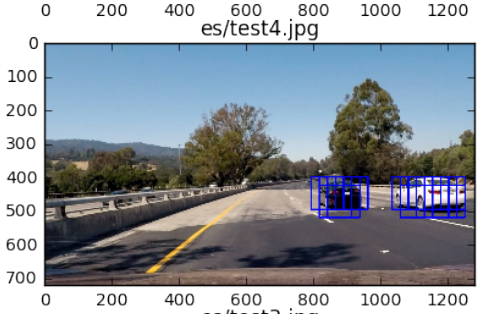


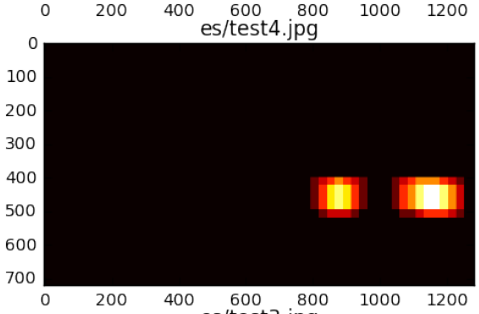


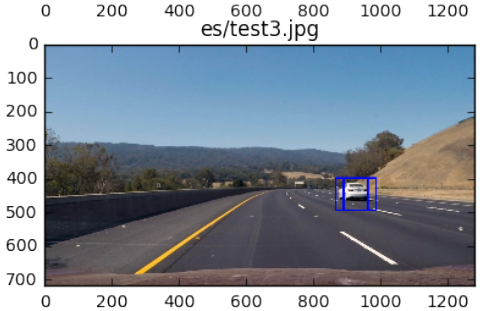


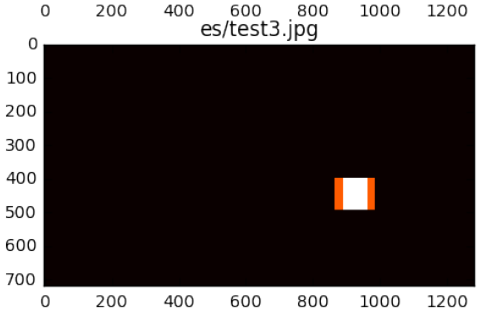




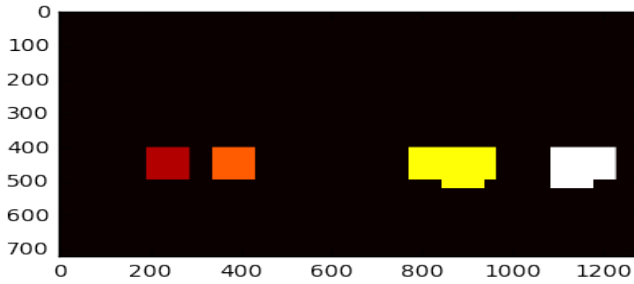


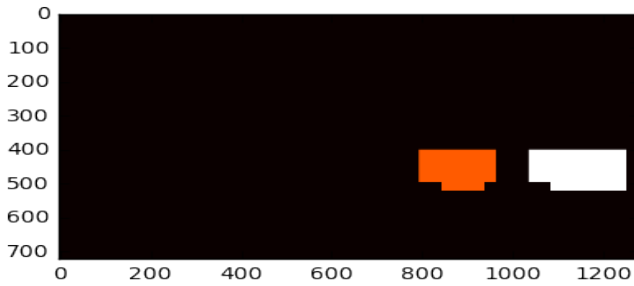
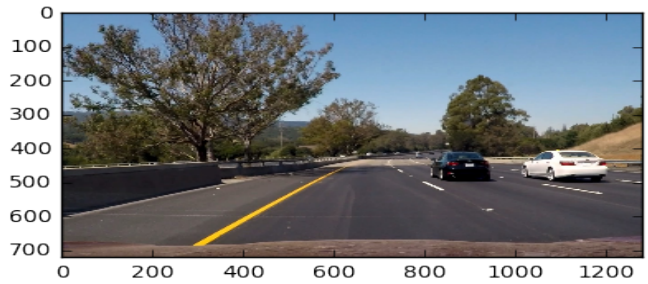
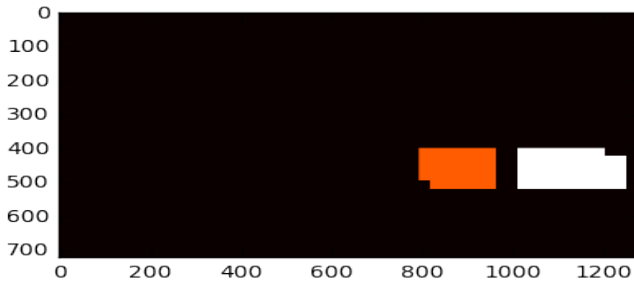




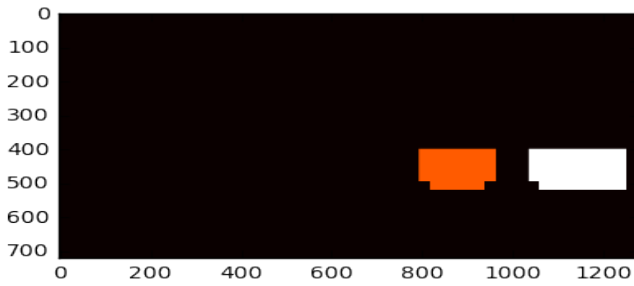
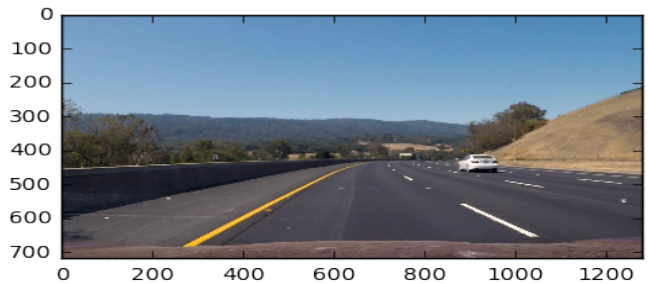
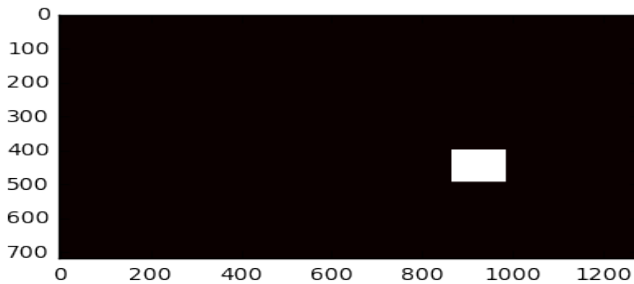


Here is the output of `scipy.ndimage.measurements.label()` on the integrated heatmap from all six frames:¶

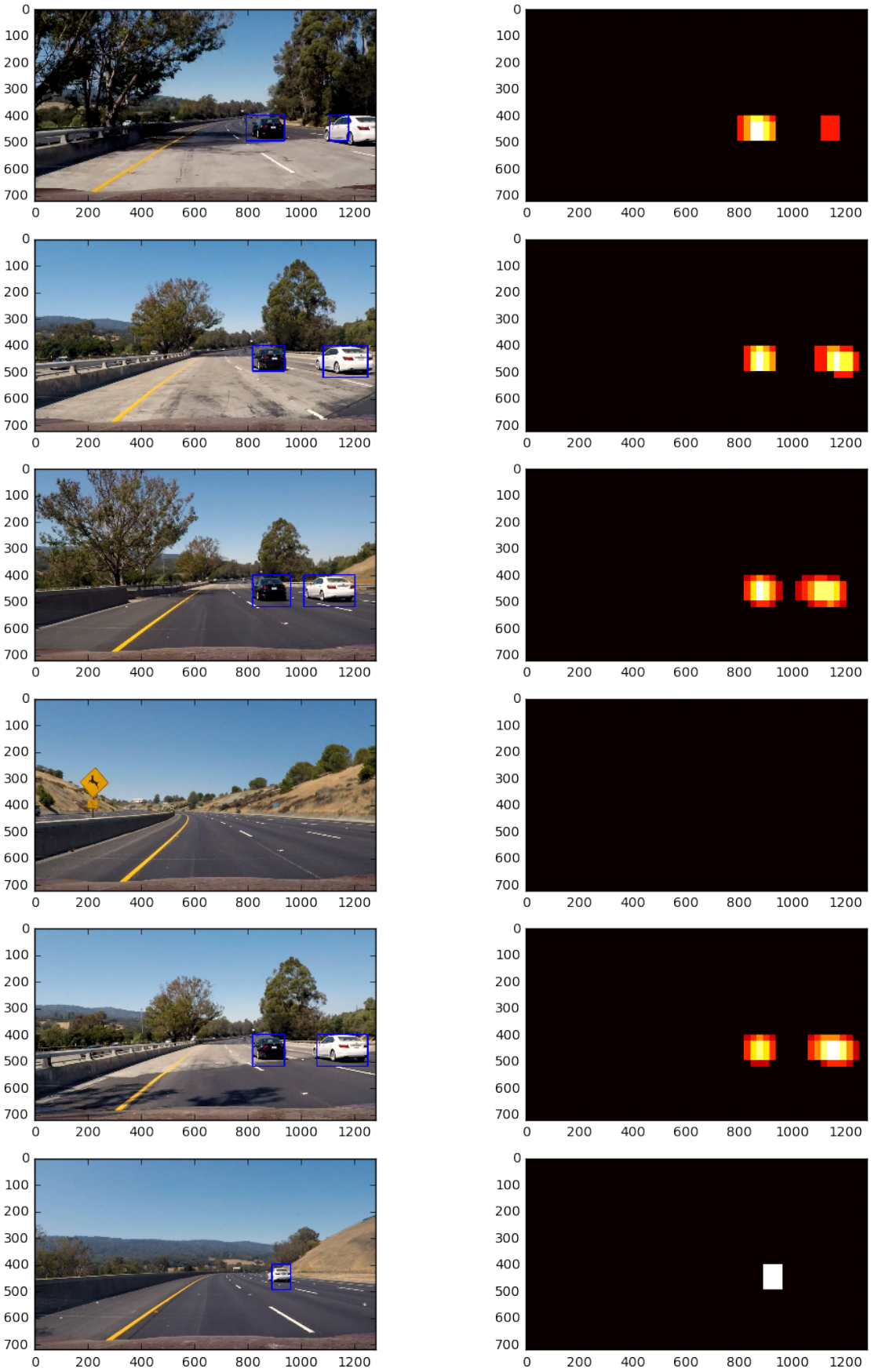





Here the resulting bounding boxes are drawn onto the frame:¶



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?¶

There were be false positive and false negative results. If the classifier returns false negatives/positives a lot the pipeline wont be able to track a car since we do not consider previous frame results to asses the probability of the result. Also a fixed window size wont result in good result if the input provided is scaled to different sizes.

Tracking a car from one frame to other and averaging over few frames will result in better detection and eliminate the false negatives. Also the threshold image may use a higher threshold for locations without car from previous frame and lower threshold near location where car was found. This would reduce false negatives as well as false positives. Tracking vehicles from frame to frame will make the detection robust against occassional false negative/ false positive results.

A combination of different sized windows would make the pipeline work with different input scale.