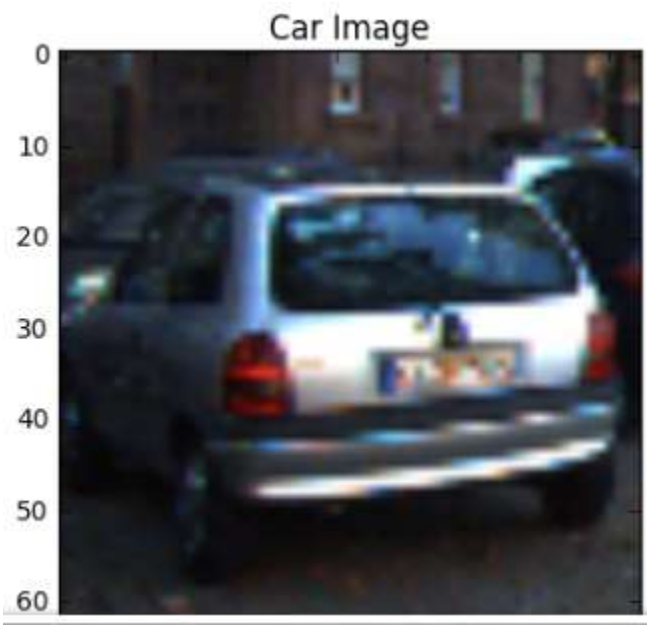
Vehicle Detection Project

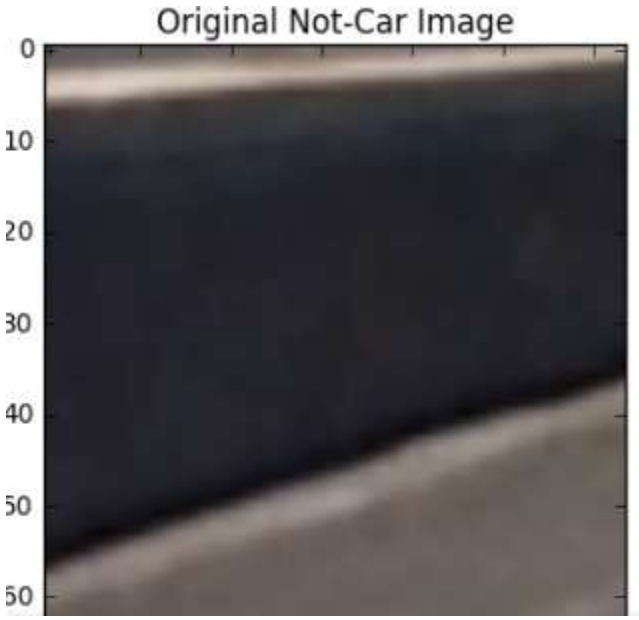
Writeup / README

Histogram of Oriented Gradients (HOG)

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

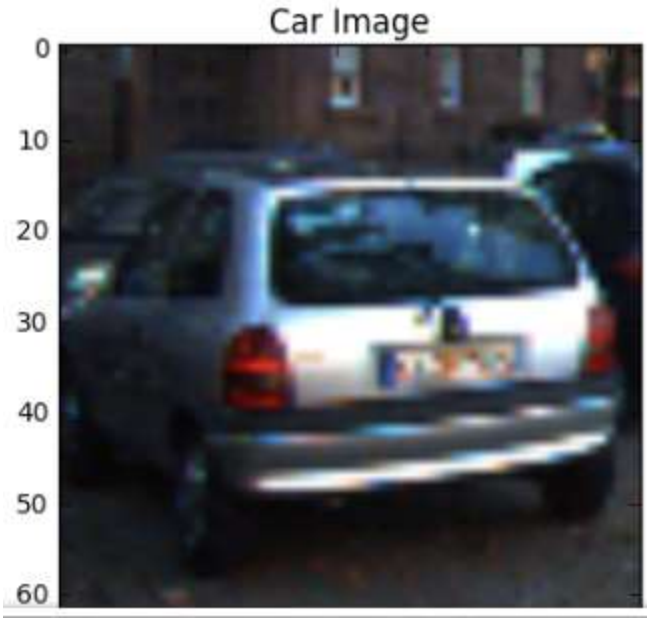
The code for this step is contained in the first code cell of the IPython notebook Train_Classifier and lines 6-94 in image_functions.py.
I started by reading in all the `vehicle` and `non-vehicle` images. Examples are shown below:

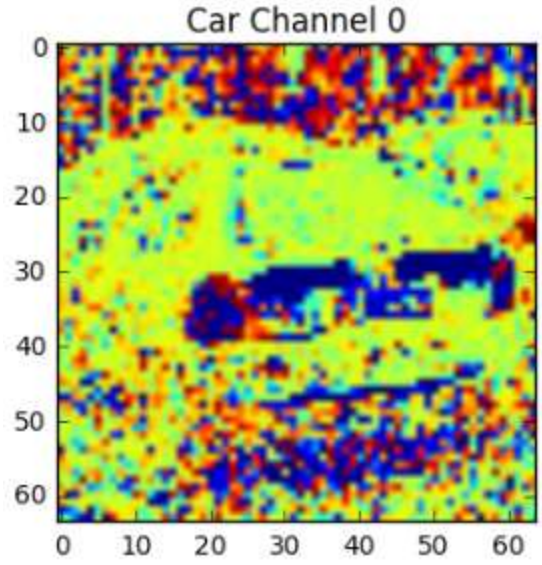


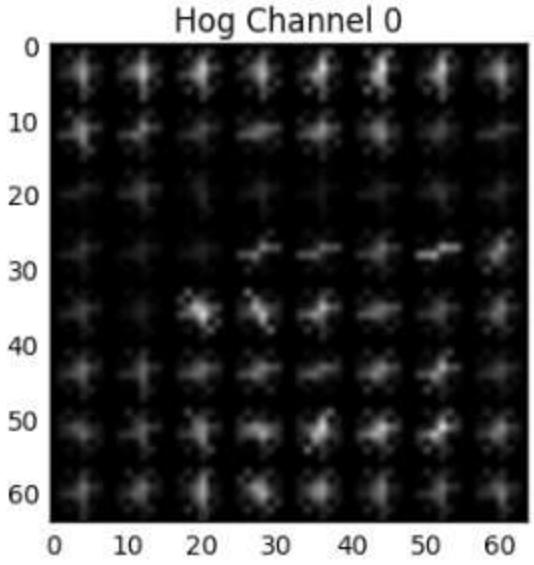


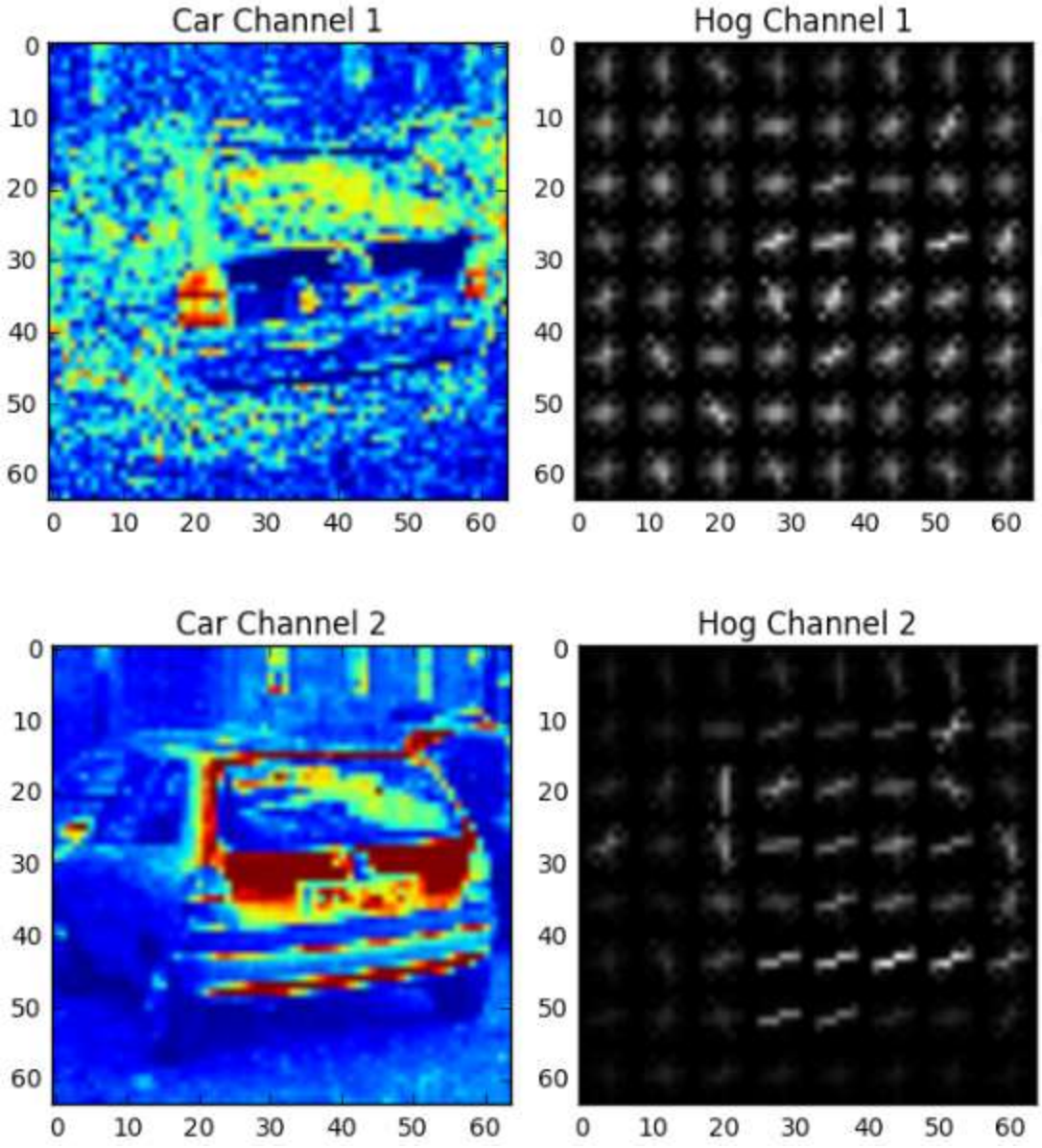
I then explored different color spaces and different `skimage.hog()` parameters (`orientations`, `pixels_per_cell`, and `cells_per_block`). From reading online, orientations cap out for accuracy at 9. From testing, 8 pixels per cell worked the best. 4 pixels per cell works better, but the feature vector is too long. 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.

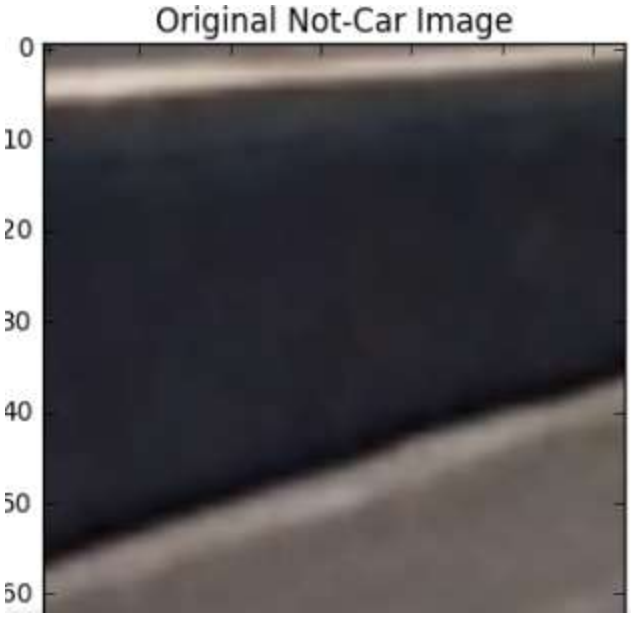
Here is an example using the `HSV` color space and HOG parameters of `orientations=9`, `pixels_per_cell=(8, 8)` and `cells_per_block=(2, 2)`:

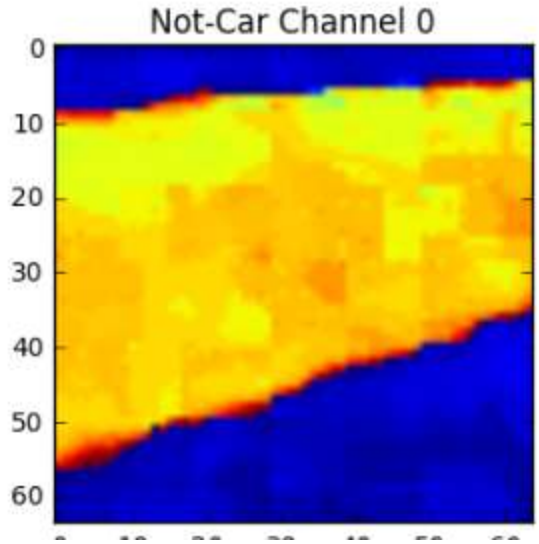

Car Image
0
10
20
30
40
50
60

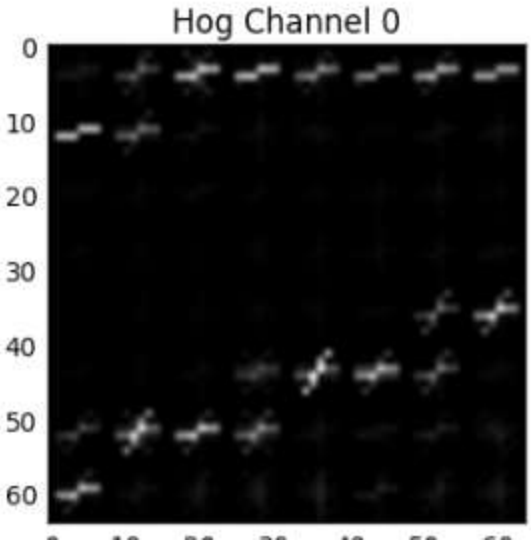

Car Channel 0
0
10
20
30
40
50
60
0
10
20
30
40
50
60

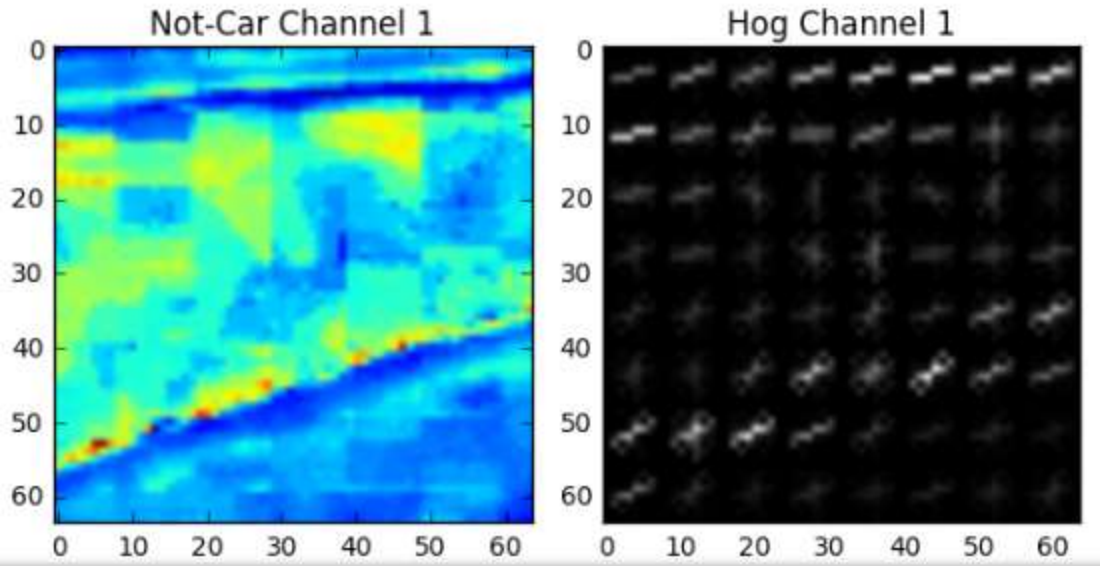

Hog Channel 0
0
10
20
30
40
50
60
0
10
20
30
40
50
60

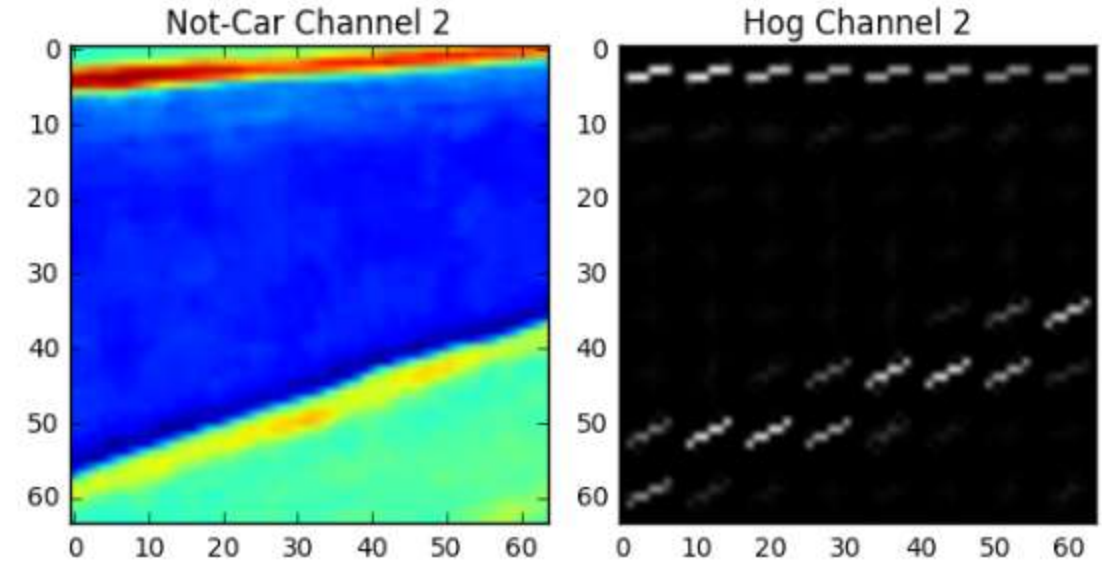

Car Channel 1
Hog Channel 1
Car Channel 2
Hog Channel 2
0
10
20
30
40
50
60


Original Not-Car Image
0
10
20
30
40
50
50


Not-Car Channel 0
0
10
20
30
40
50
60


Hog Channel 0
0
10
20
30
40
50
60





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

I tried various combinations of parameters and chose 9 HOG orientations, 8 pixels per cell and 2 cells per block. I tested various parameters and found that these settings worked the best. 4 or 2 pixels per cell works the best, but results in a longer training time. I chose HSV color space because it performed sufficiently and is used in my previous project. This will reduce execution time.

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).

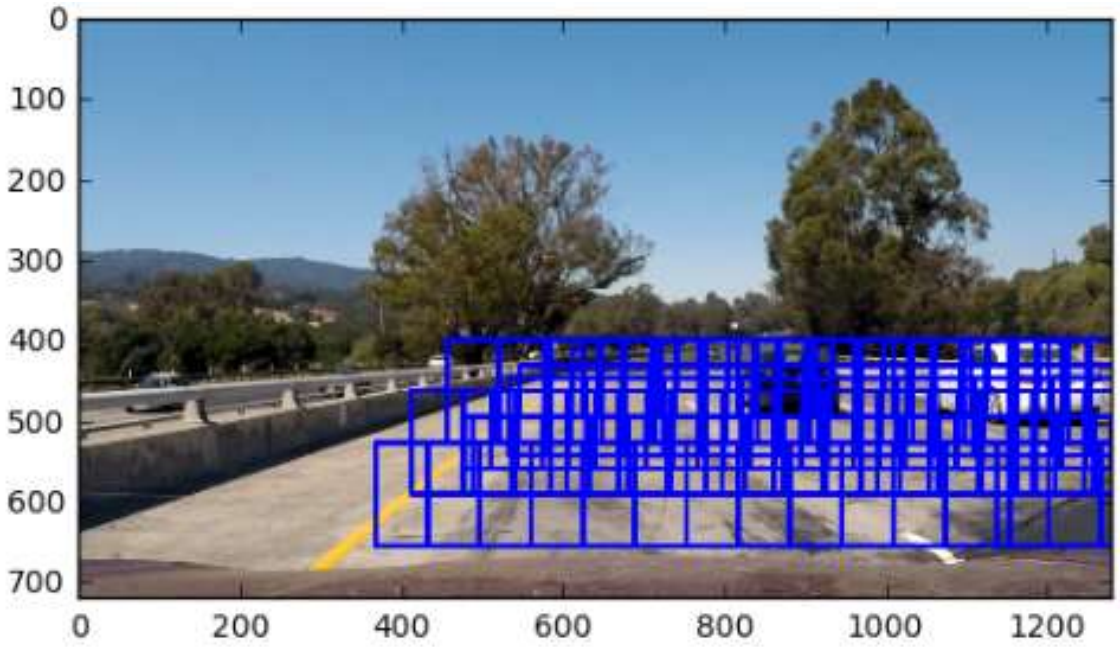
I trained a linear SVM using sklearn SVC. This can be found in the 3rd cell of the Train_Classifier.ipnb notebook. The features are extracted using functions found in the initial part of Image_functions.py. Each images is loaded, then the color histogram, spatial features and HOG features are loaded into a feature vector array and concatenated together.

The data is then normalized using sklearn standard scaler and split into training and testing data to avoid overfitting. The data is randomly shuffled and trained using sklearn linear svc. Lastly, the svc and normalization data is saved using joblib.dump so that the classifier only needs to be trained once.

Sliding Window Search

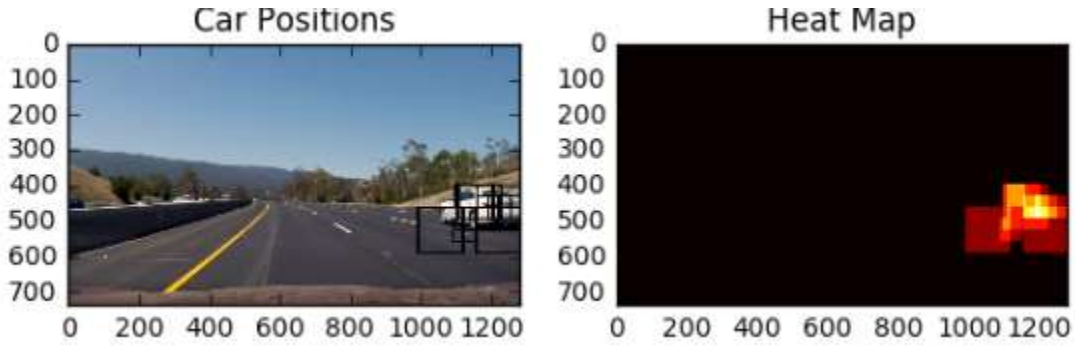
1. Describe how (and identify where in your code) you implemented a sliding window search. How did you decide what scales to search and how much to overlap windows?

The slide_window function in lines 100-159 of image_functions.py is used to extract window data. This function is called in cell 6 of the Master_Notebook. This function took the most experimentation. I ended up choosing 3 size windows: 128x128, 96x96 and 64x64 pixels. The smaller windows only search towards the top of the image. The larger windows search the whole area. After completion, 176 windows are used as shown below.



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 HSV 3-channel HOG features plus spatially binned color and histograms of color in the feature vector, which provided a nice result. I used 16 spatial features and 32 x 32 histogram features. Here are some example images and the corresponding heatmaps.



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.)

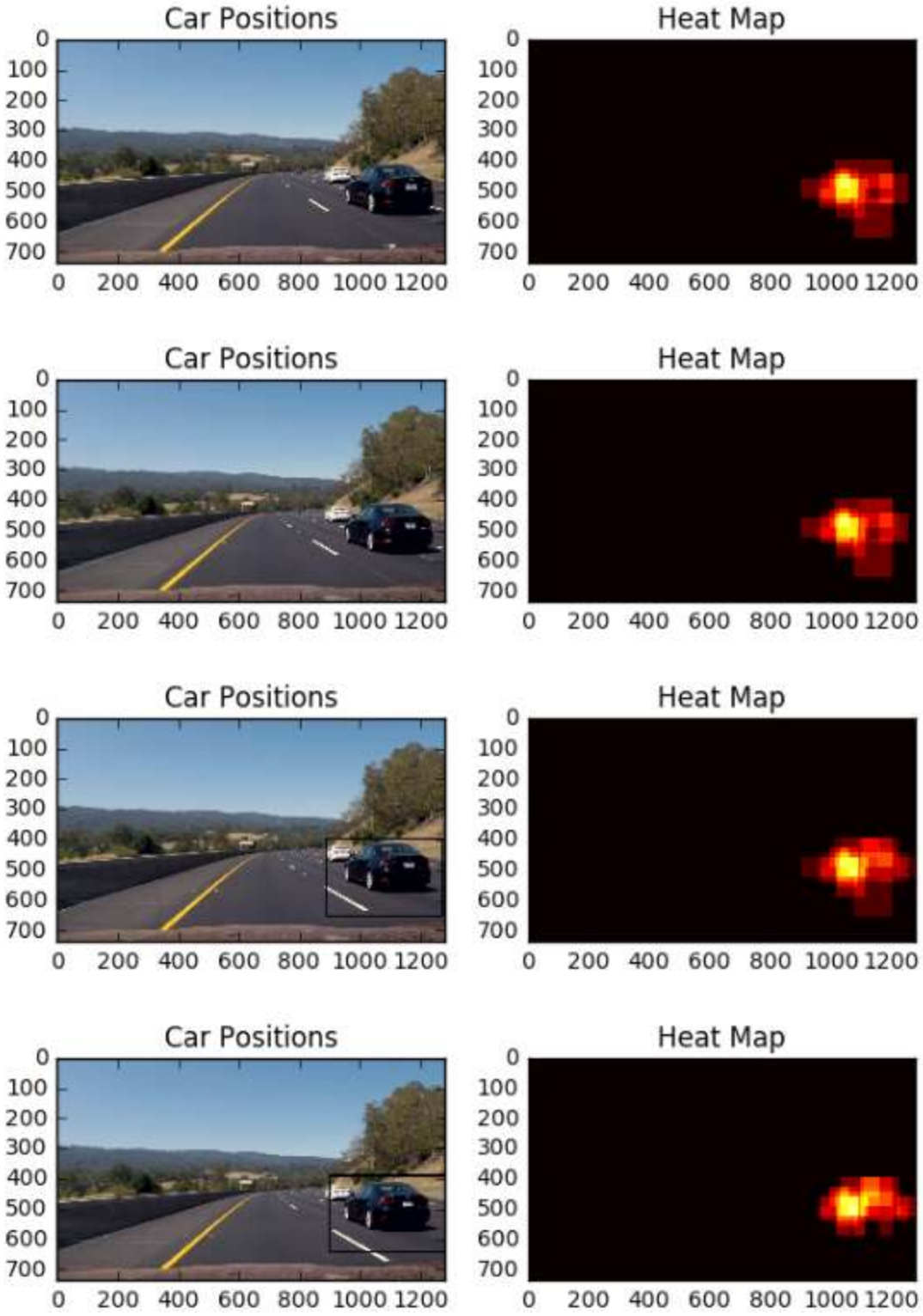
Here's a link to my video result

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.

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 using the function add_heat in cell 5 of the Master_notebook. The heatmaps are averaged over 5 video frames.
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. This code can be found in cell 3 of the Master_notebook. I also thresholded this function. A vehicle must be detected for at least 2 out of 3 frames. If not, the vehicle is labeled but not shown.

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

Here are four frames and their corresponding heatmaps:



The frames on the left represent the output of the video and the corresponding averaged heatmaps.

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 faced a large problem in choosing window sizes and placement. The code produced substantially different output when I adjusted the windows slightly. I also faced a thresholding problem. When the threshold for the heatmap was set too high, vehicles were not correctly identified. When it was set too low, I got too many false positives.

The project does my best to balance these issues. As you can see, it still has room to improve. My goal for the future will be to correctly identify the cars 100% of the time with no false positives.