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 4th code cell of the detect_vehicle.ipynb notebook.

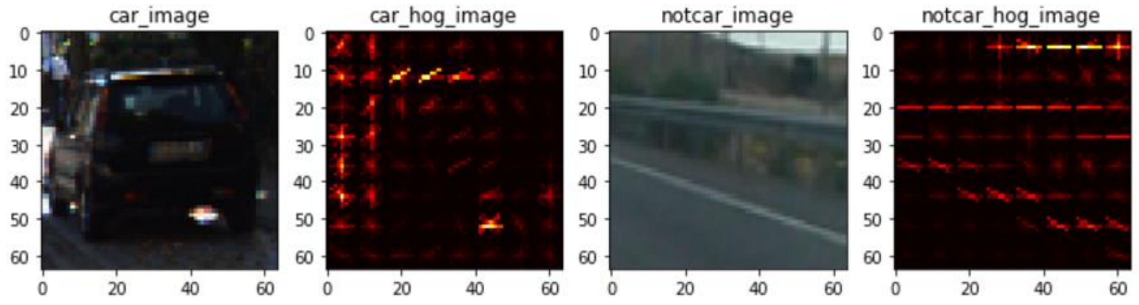
I started by reading in all the vehicle and non-vehicle images. Here is an example of one of each of the vehicle and non-vehicle classes:

Vehicle

Non-Vehicle

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. The hog features are extracted individually from each of the color channels.

After testing on several colorspace, I finished with YCrCb as it tend to give more distinct features than the other colorspace. Here is an example using the `YCrCb` color space and HOG parameters of `orientations=8`, `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 an after many rounds of using different color channel and as well the orientation bins, below are the final parameters for the spatial bins, HOG & Color histograms (also shown in the cell code 5 of the detect_vehicle.ipynb notebook).

i) color_space = 'YCrCb'
ii) orient = 8
iii) pix_per_cell = 8
iv) cell_per_block = 2
v) hog_channel = ALL
vi) spatial_size = (16,16)
vii) hist_bins = 48

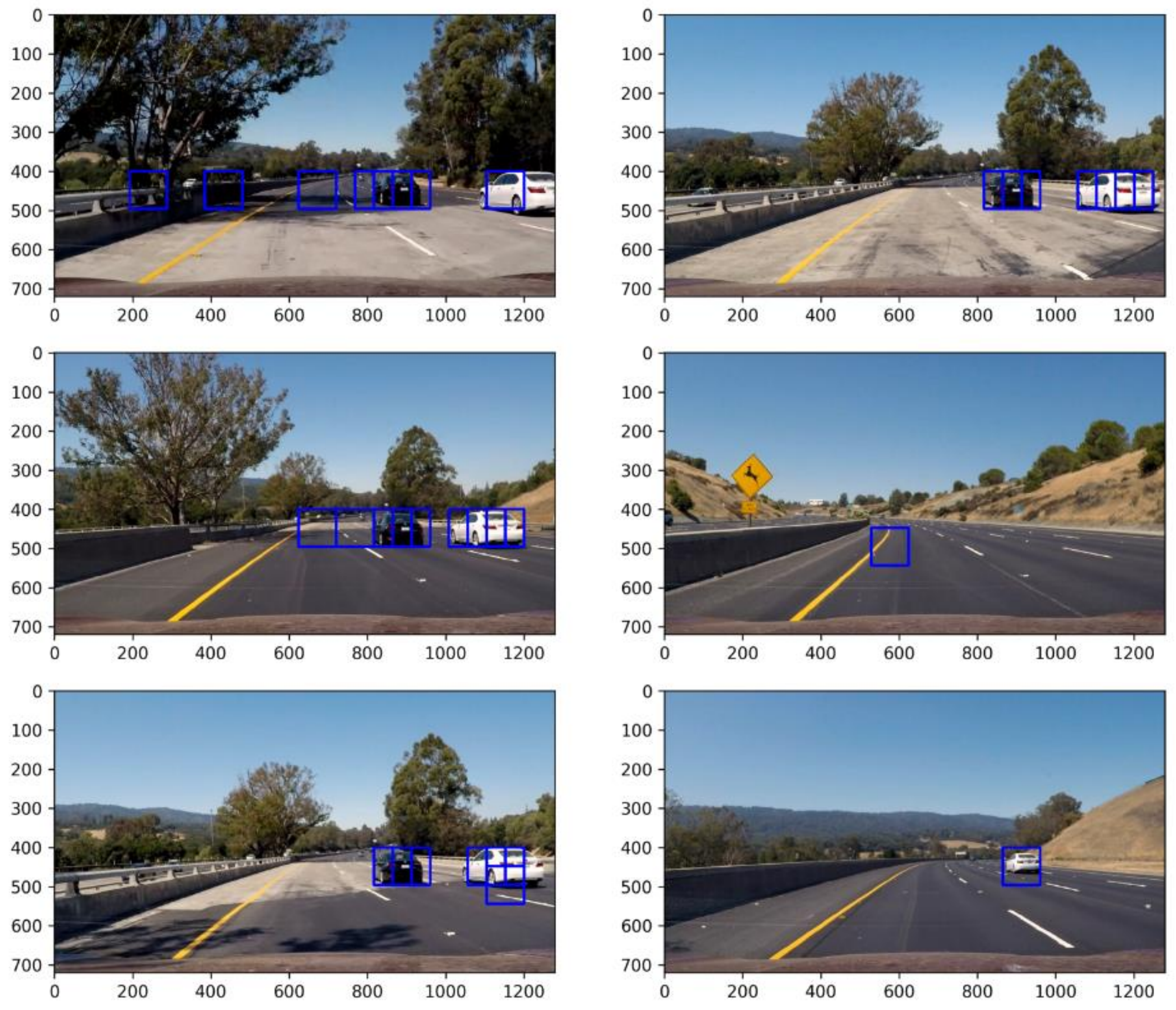
3. Describe how (and identify where in your code) you trained a classifier using your selected HOG features (and color features if you used them).

I trained a linear SVM using the LinearSVC function with the final accuracy percentage of 99.16%. StandardScaler was used once all the features were extracted to scale the features down evenly with a zero mean. The implementation of the classifier training is at the cell code 5 of the detect_vehicle.ipynb notebook.

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 approach is very similar to what was covered in the course. I tested a simple implementation of the sliding window in cell code 6 of the detect_vehicle.ipynb notebook. Initially, I tried different overlap values until I come up with the final value of 0.5. As well, the cars were mostly on the bottom half of the image I set the sliding window to search for cars within the image shape which is based on the y_start_stop coordinate. As a result of this simple implementation, below are the result of testing on the test images set:



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?

Based on previous question images, there are several false positives and overlapping boxes.

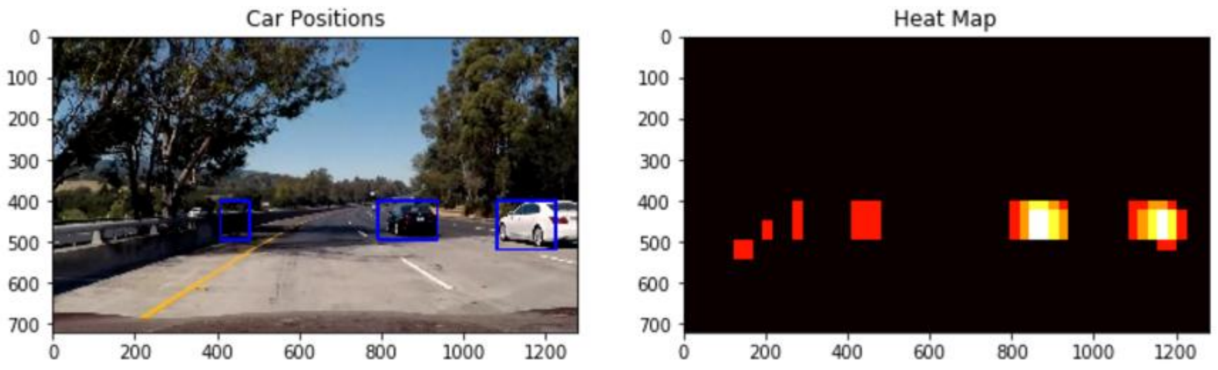
I used a different sliding window technique which is now based on the HOG sub-sampling window search. This time, I tried different scales of sliding window and eventually a combination of 64 x 64, 80 x 80, 96 x 96, and 112 x 112 pixels sliding windows are implemented with `scales = [1., 1.25, 1.5, 1.75]`. The combination of various sized windows makes sure generating enough number of bounding boxes for each detected object, therefore is beneficial for the next step of ruling out false positives.

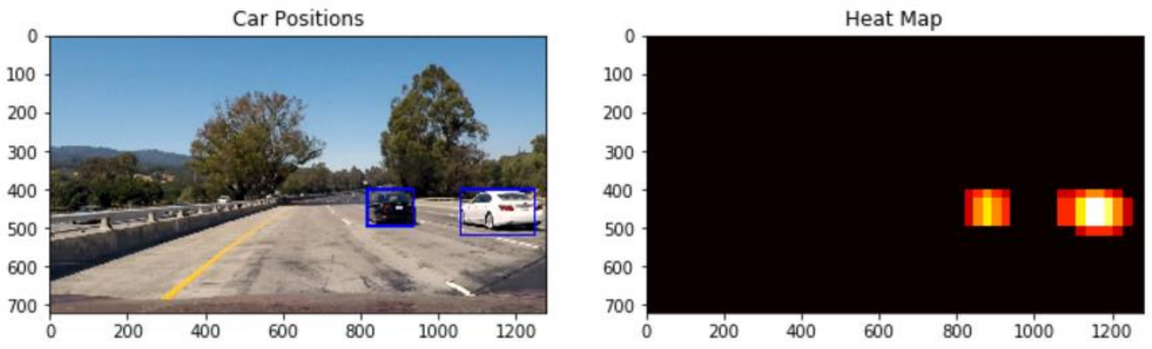
Next step, a heatmap are created by combining overlapped boxes, and thresholded with a criterion to rule out false positives. Then a label function is applied to identify each detected object from the thresholded heatmap

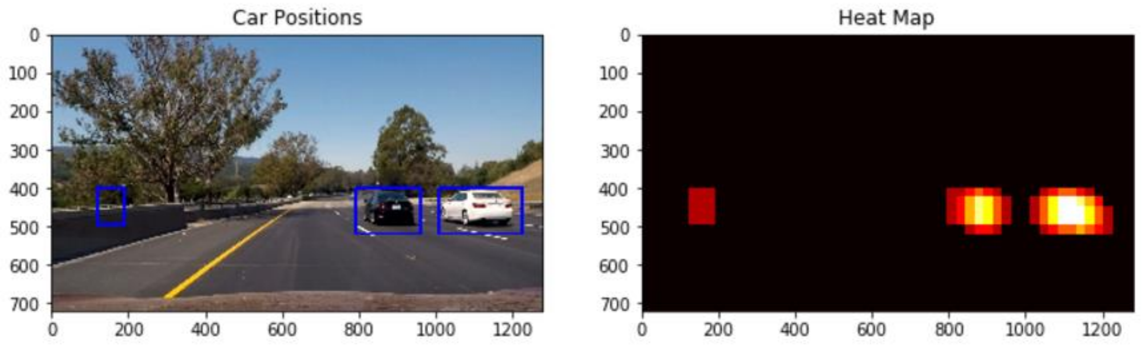
As well, in draw_labeled_bboxes() function in cell no. - 9 of detect_vehicle.ipynb, I check whether the minimum y-coordinate values is in between 300 and 600. If not, then don't draw the box. Also, minimum x-coordinate value should be less than 1220. One more check of the area of the box has been used to remove small false positive boxes. If the area of the box is less than 2500, then don't draw it.

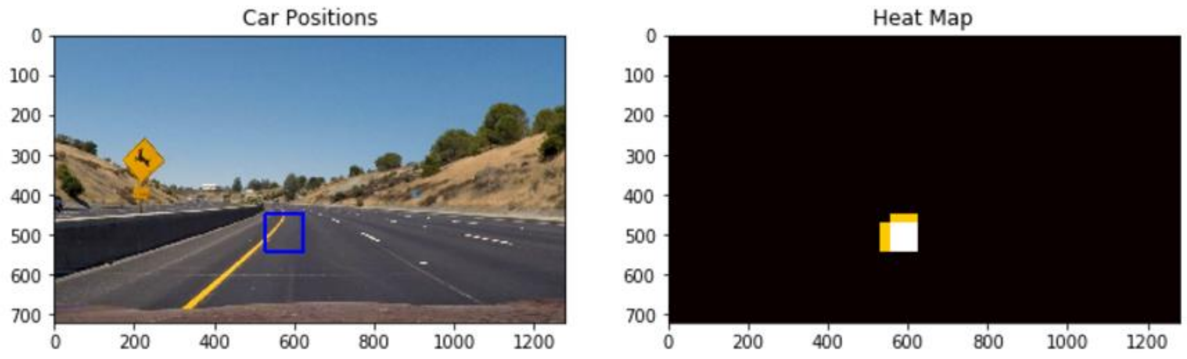
The entire implementation of this codes are in cell code 10 and uses the find_cars function in cell code 8 of the detect_vehicle.ipynb notebook

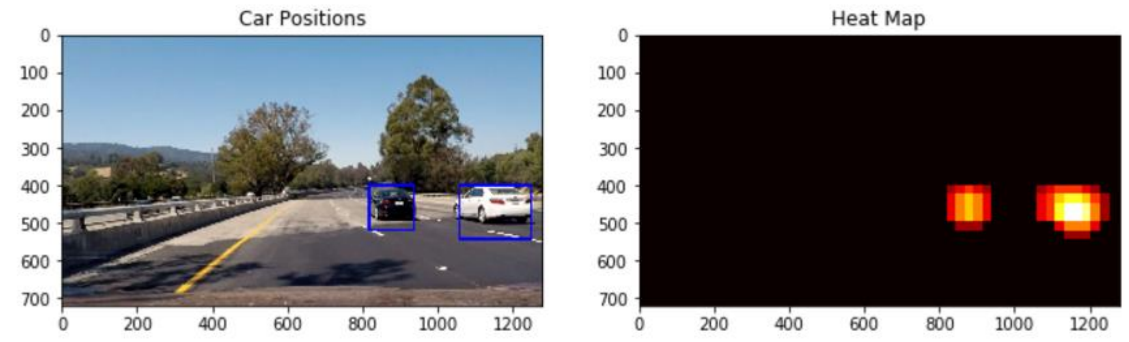
Here are some the output of the images:

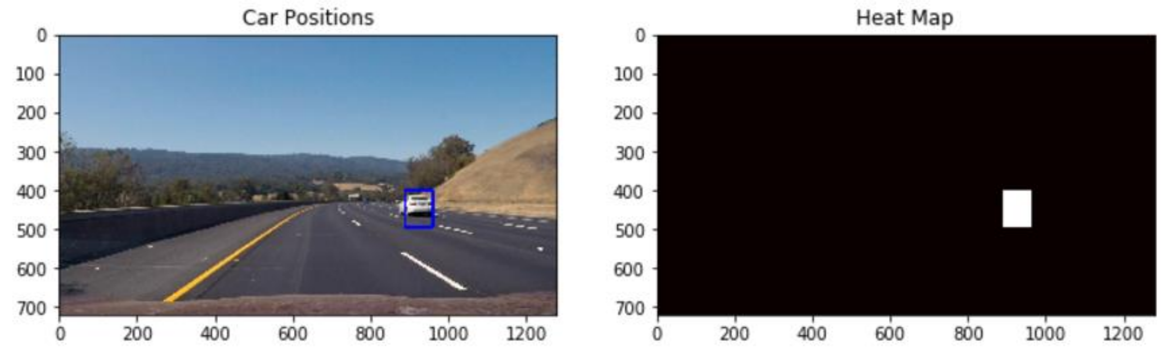

Car Positions
Heat Map


Car Positions
Heat Map


Car Positions
Heat Map


Car Positions
Heat Map





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 video result is in the file project_video_result.mp4 file

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.

To tackle the problem of requiring that a detection be found at or near the same position in several subsequent frames, the pipeline does a whole image HOG sub-sampling window search for every 8 frames, and a reduced window search for every 2 frames which only scans the region of interest at where vehicle objects previously detected.

The vehicle boundary boxes of recent 8 frames are stored in a Python `deque` object with a length of 8. The current boundary boxes are calculated from a thresholded heatmap of accumulated boundary boxes over the recent 8 frames. These steps managed to smooth the drawing of vehicle boundary boxes and are shown in cell code 12 of the detect_vehicle.ipynb notebook

---

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 pipeline used in the current project is very specific to the project video. All the different parameters have been tuned keeping that fact in mind and thus might not work on other recorded video frames.

More work can be done on smoothing out the detection boxes. The is still some visible jittering between frames. I am still seeing some false detections on the left side of the screen, this is something that could potentially fixed by optimizing the multi-window search function for better speed and accuracy.