
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, 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

###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 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.

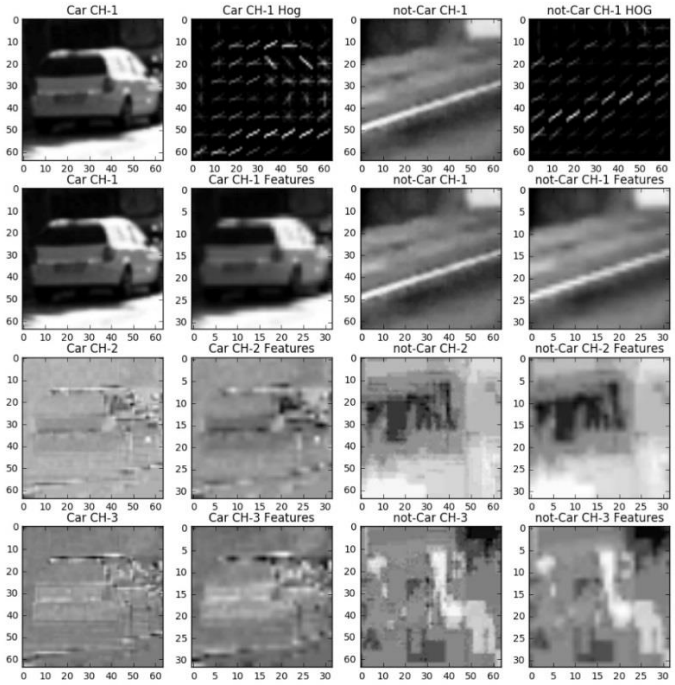
The code for this step is contained in the third code cell of the IPython notebook. I started by reading in all the `vehicle` and `non-vehicle` images. Here is an example of one of each of the `vehicle` and `non-vehicle` classes:

Non-Vehicle

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.

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 and measure the accuracy of each combination finally below combination gives me the best result with accuracy around 98.82%

color_space = 'YCrCb' # Can be RGB, HSV, LUV, HLS, YUV, YCrCb

orient = 11 # HOG orientations

pix_per_cell = 8 # HOG pixels per cell

cell_per_block = 2 # HOG cells per block

hog_channel = "ALL" # Can be 0, 1, 2, or "ALL"

spatial_size = (16, 16) # Spatial binning dimensions

hist_bins = 16   # Number of histogram bins

spatial_feat = True # Spatial features on or off

hist_feat = False # Histogram features on or off

hog_feat = True # HOG features on or off

####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).

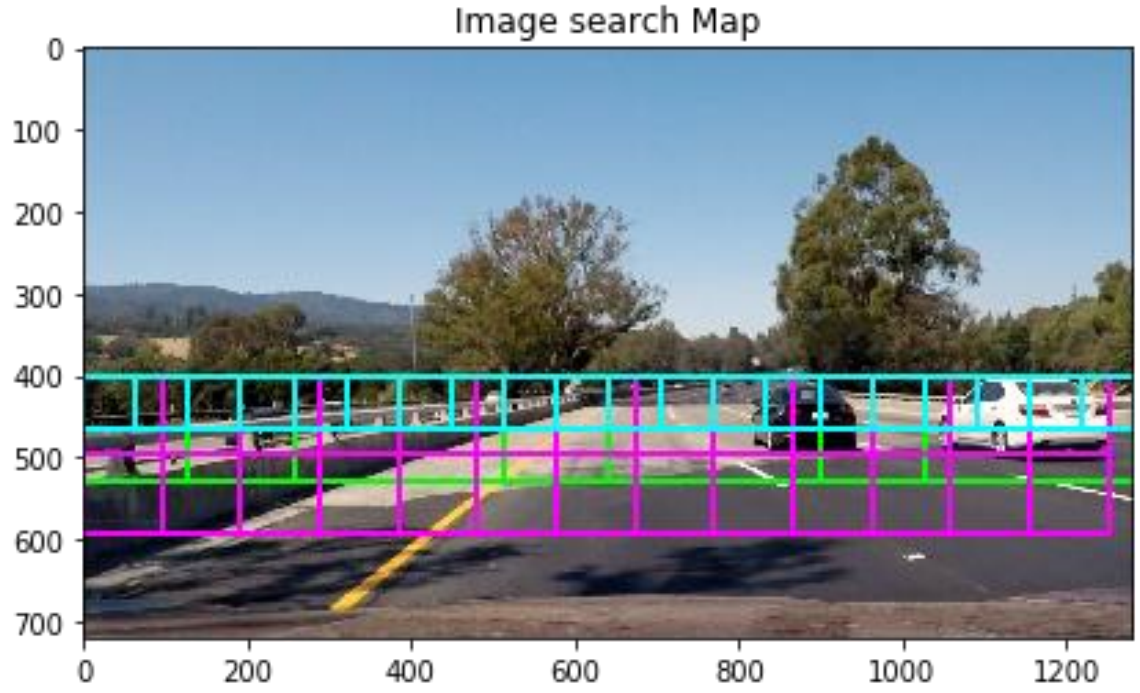
I trained a linear SVM using hog features and spatial features. I used 80% example for training and 20% for testing. Normalizing ensures that a classifier's behavior isn't dominated by just a subset of the features, and that the training process is as efficient as possible. That is why, feature list was normalized by the StandardScaler() method from sklearn.

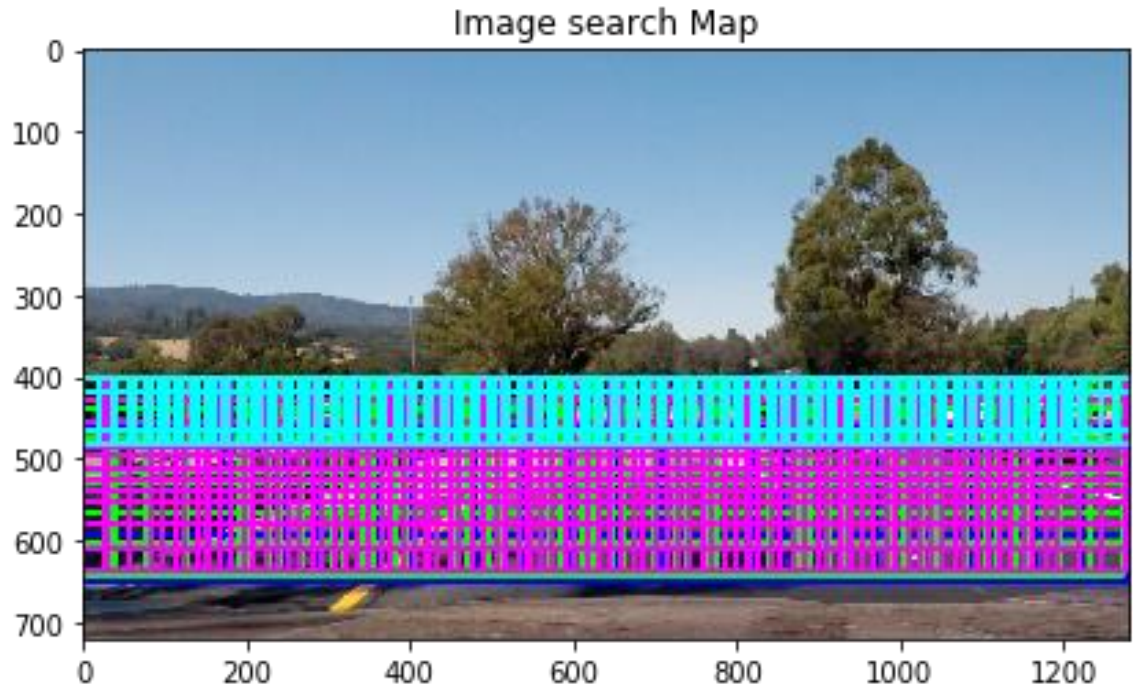
###Sliding Window Search

####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 decided to search in 4 windows size (64,64), (96,96), (128,128), (196,196) in a specific area. Below image exactly show my sliding windows search plan. Reason for this is, vehicle near to camera have bigger size and as distance increases the size of vehicle in

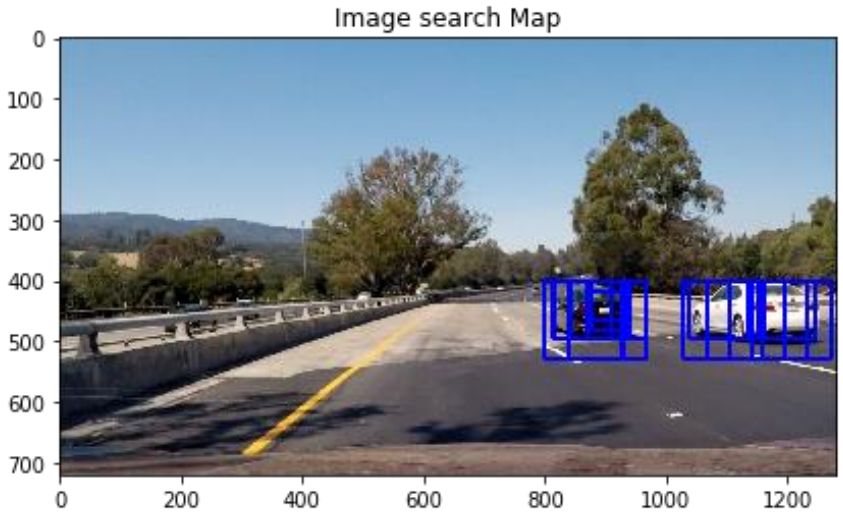
decrease. So, I take smaller windows for far distance vehicle and bigger windows for near vehicles.

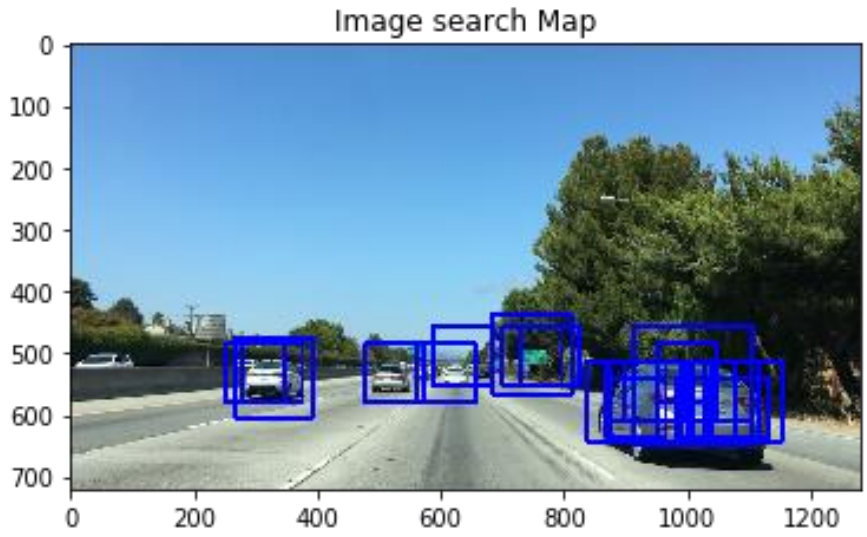


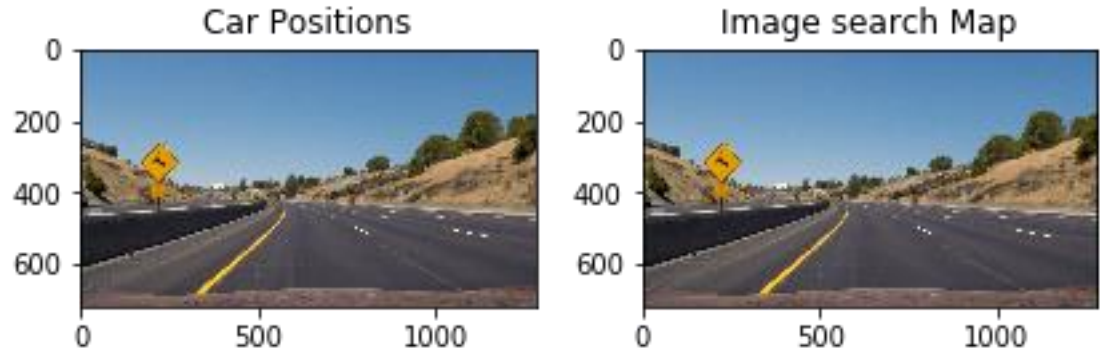


####2. Show some examples of test images to demonstrate how your pipeline is working. What did you do to optimize the performance of your classifier?

Ultimately, I searched on using YCrCb ALL-channel HOG features plus spatially binned color in the feature vector, which provided a nice result. Here are some example images.







I reduce the search area so that classifier don't do unnecessary search. I also choose small windows size such that vehicle far from camera gives better result. It also provide some false positive, we can use threshold and heap map to reduce the 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.)

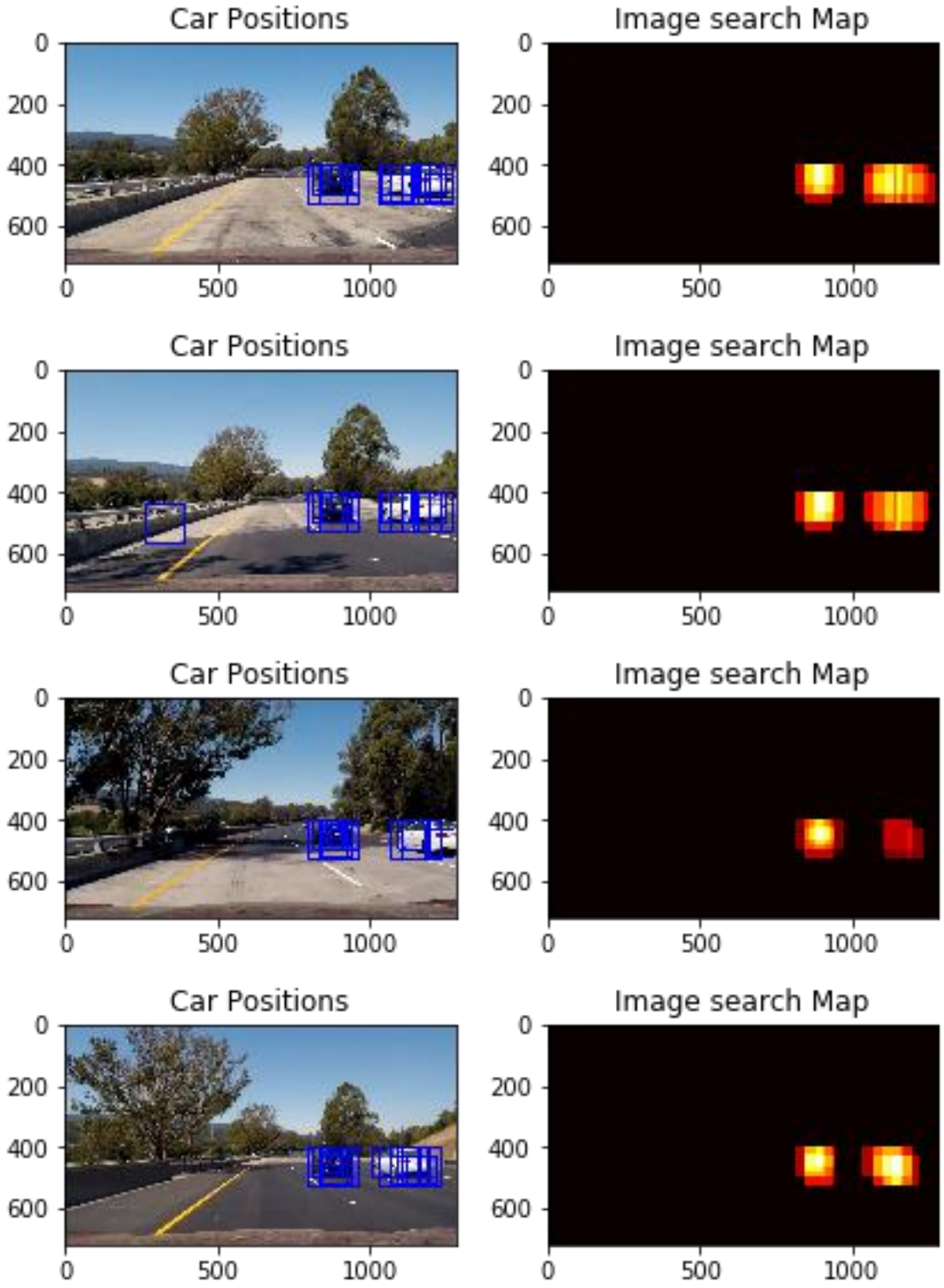
- Submission files have videos you can see it.

####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.

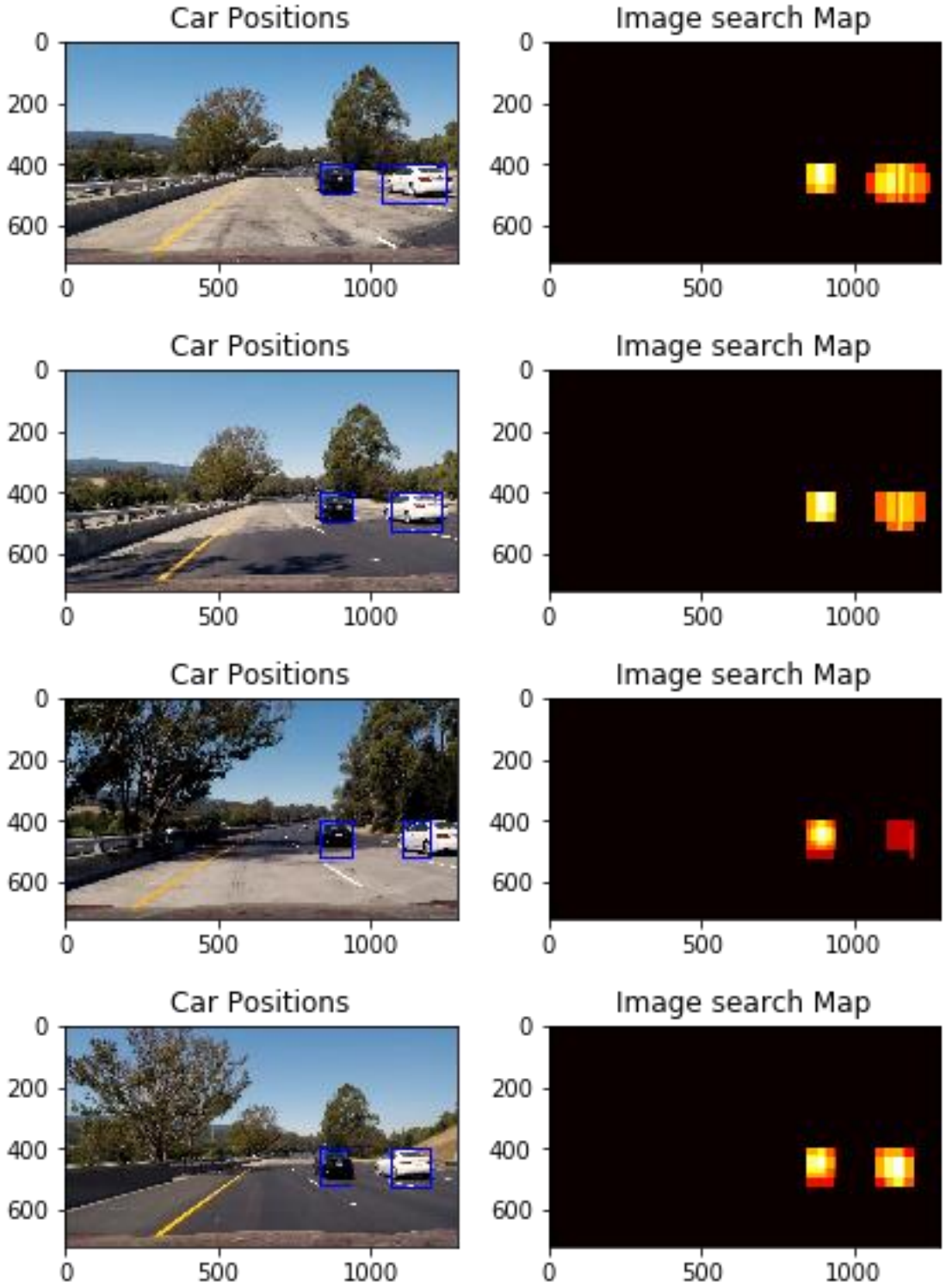
I recorded the positions of positive detections in each frame of the video. From the positive detections I created a heatmap and then thresholded that map to identify vehicle positions. I then used `scipy.ndimage.measurements.label()` to identify individual blobs in the heatmap. I then assumed each blob 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, the result of `scipy.ndimage.measurements.label()` and the bounding boxes then overlaid on the last frame of video:

One way I could do this is to store the heat maps of the N most recent frames in a list, take the sum of those heat maps, and set a threshold on the combined heat map to get the bounding boxes for each frame. This helps not only to filter out false positives, but it also makes the boxes to appear much more stable across frames.

Here are frames and their corresponding heatmap:



Here the resulting bounding boxes are drawn onto the last frame in the series:



###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?

- Classification algorithm may fail in different light conditions, we need to improve the classification algorithms.
- We are supposed to use this method in real time, so we need to improve the algorithms because it takes more time.
- It is possible to improve the classifier by additional data augmentation, hard negative mining, classifier parameters tuning etc.
- My classification algorithm takes more time, (I am running it on my pc with Intel i7 and 8gb ram.) I think to reduce the time factor in next time in this project. It takes around 1 hour to process all the videos. I think my feature vector is more feature, next time I try tune the parameter to reduce the size of feature vector then its takes less time.