
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.

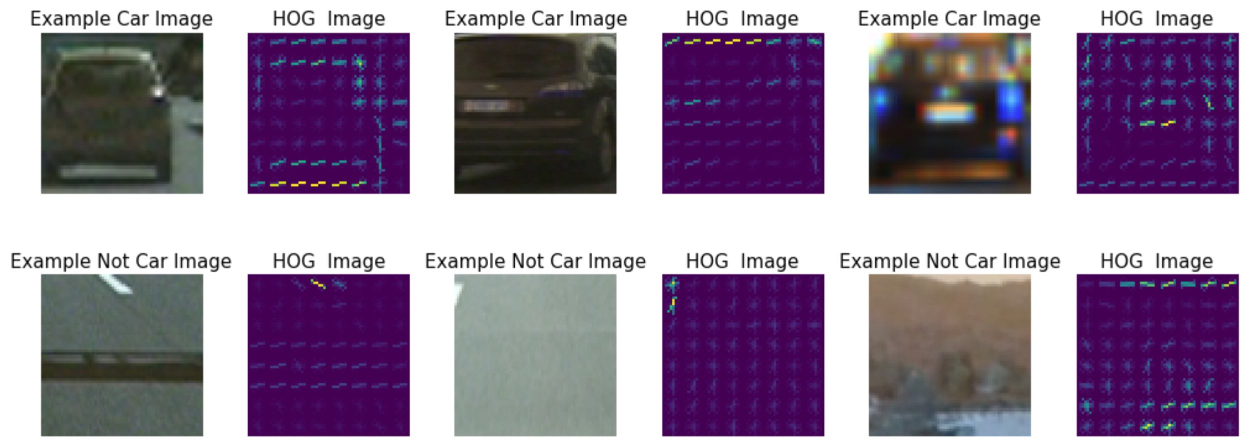
You're reading it!

###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 [373] block in jupyter notebook. The function get_hog_features ( adapted from class) extracts hog features on a single image.`

The HOG is visualised in block 376.



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

I tried various combinations of parameters. The HLS didn’t work out so well on sections having shadows or high brightness.

Finally Settled on   colorspace = 'YCrCb'

YCrCborient = 9

pix_per_cell = 8

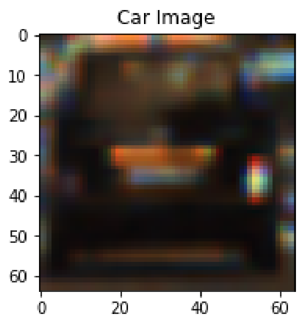
cell_per_block = 2

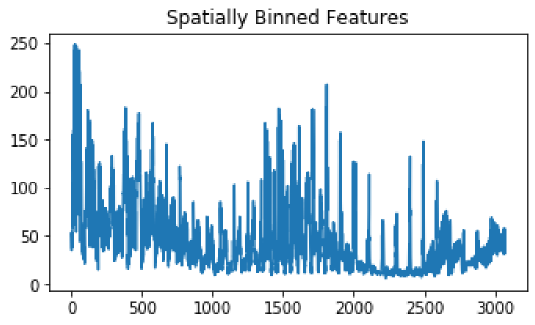
hog_channel = 'ALL' # Can be 0, 1, 2, or "ALL"

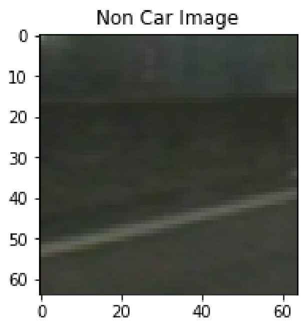
spatial = 32

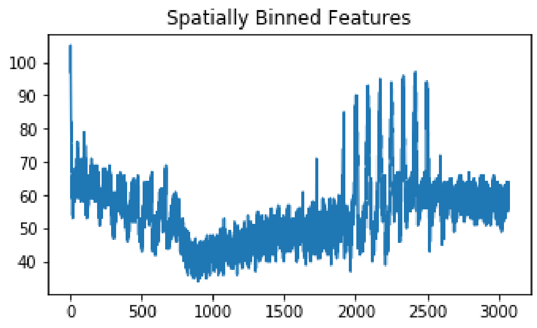
hist_bins = 32

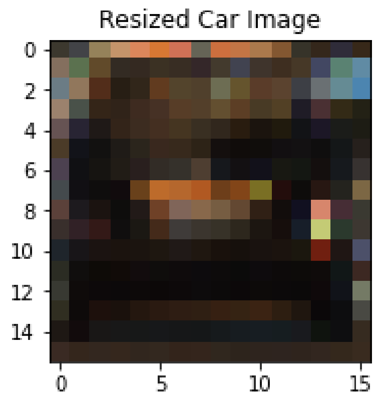
Apart from this also tried spatial bins and histogram .

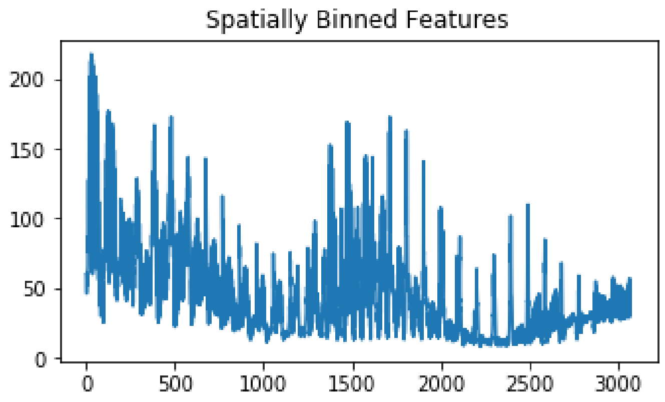


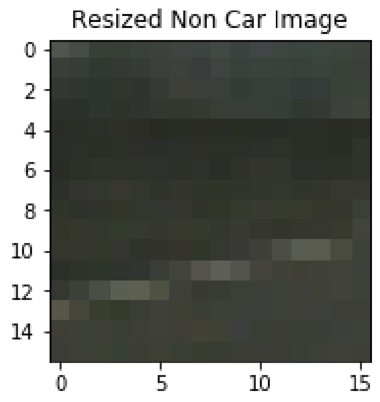


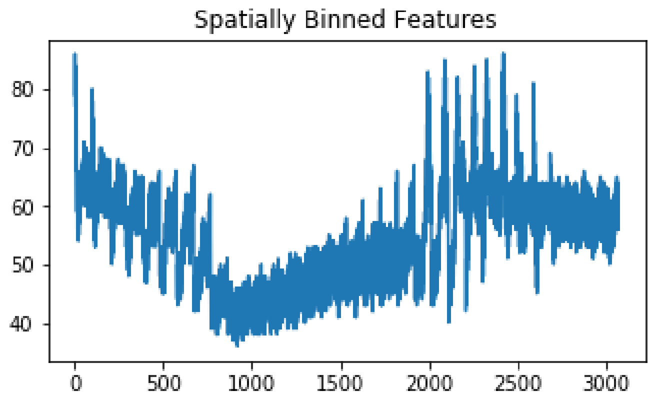


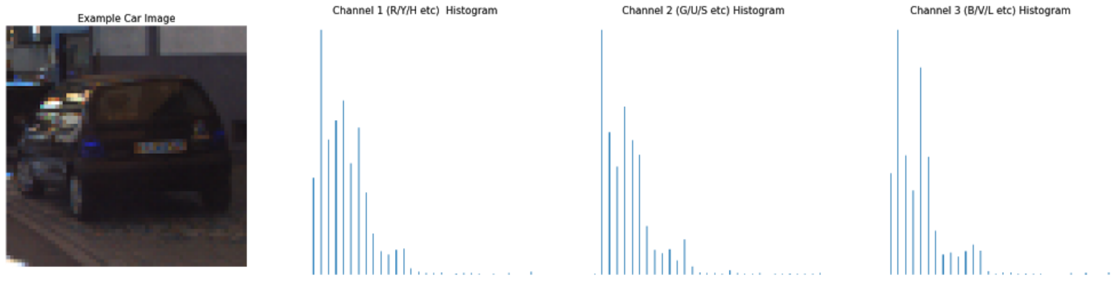


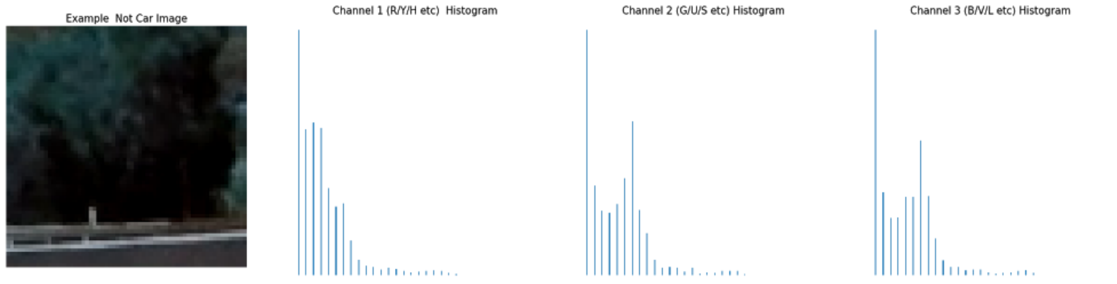


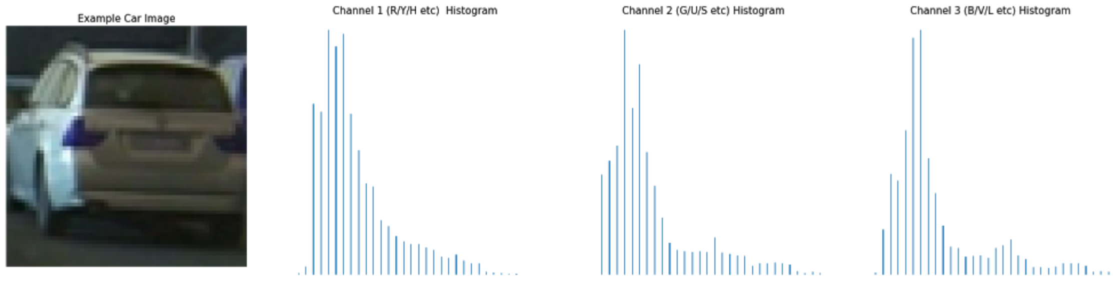


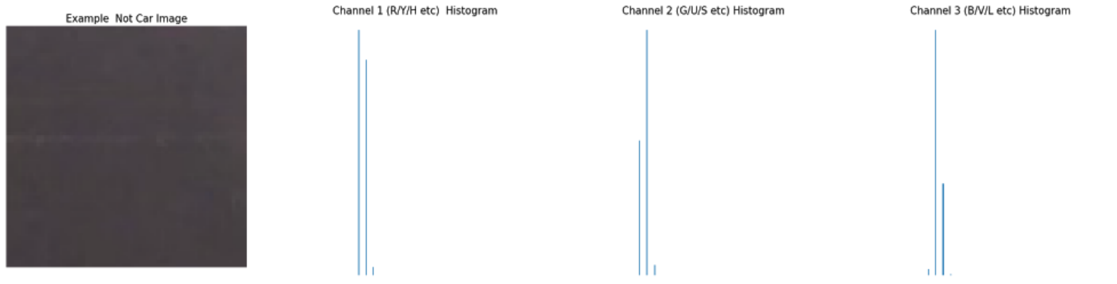


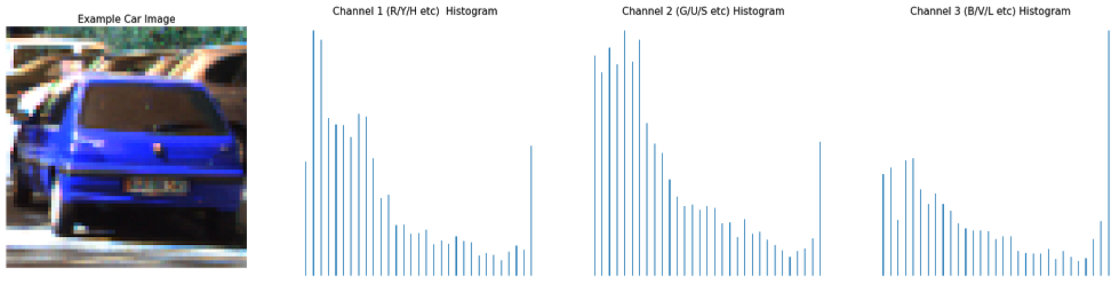


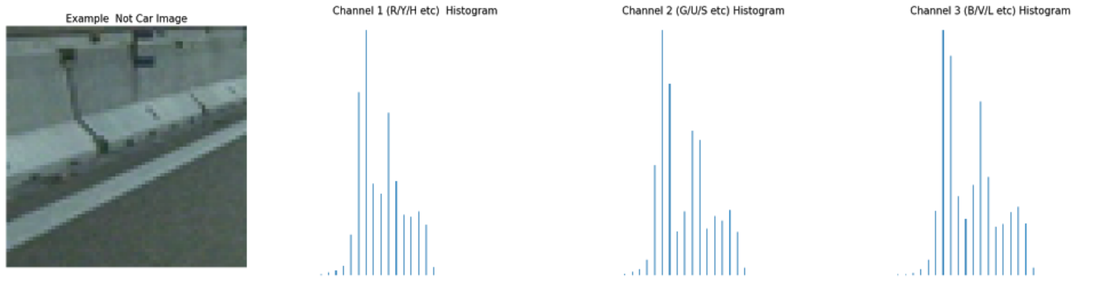

Example Car Image
Channel 1 (R/Y/H etc) Histogram
Channel 2 (G/U/S etc) Histogram
Channel 3 (B/V/L etc) Histogram


Example Not Car Image
Channel 1 (R/Y/H etc) Histogram
Channel 2 (G/U/S etc) Histogram
Channel 3 (B/V/L etc) Histogram


Example Car Image
Channel 1 (R/Y/H etc) Histogram
Channel 2 (G/U/S etc) Histogram
Channel 3 (B/V/L etc) Histogram


Example Not Car Image
Channel 1 (R/Y/H etc) Histogram
Channel 2 (G/U/S etc) Histogram
Channel 3 (B/V/L etc) Histogram


Example Car Image
Channel 1 (R/Y/H etc) Histogram
Channel 2 (G/U/S etc) Histogram
Channel 3 (B/V/L etc) Histogram


Example Not Car Image
Channel 1 (R/Y/H etc) Histogram
Channel 2 (G/U/S etc) Histogram
Channel 3 (B/V/L etc) Histogram

####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).

Please see codeblock 381 where classifier training is defined. Block 380 combines all feature extractions - HOG, Histogram and Spatial Binning to create concatenated feature vector.

I have used Linear SVM

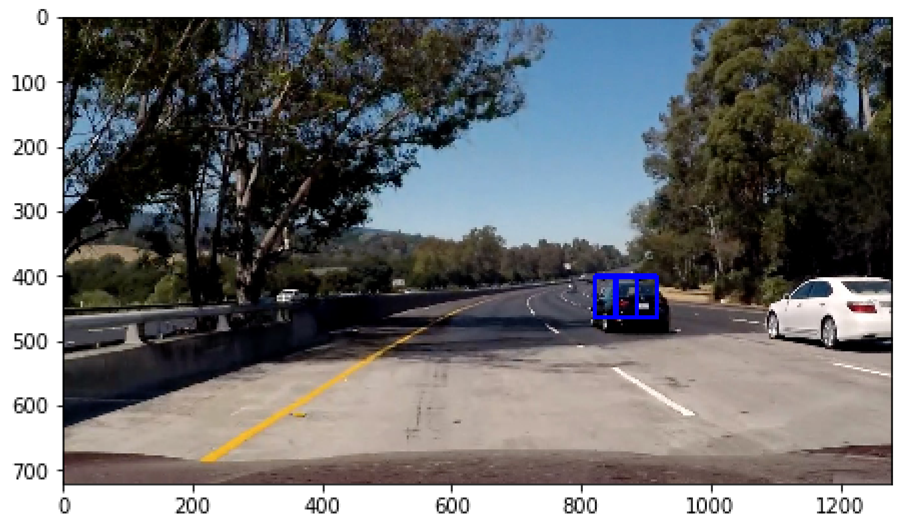
svc = LinearSVC()

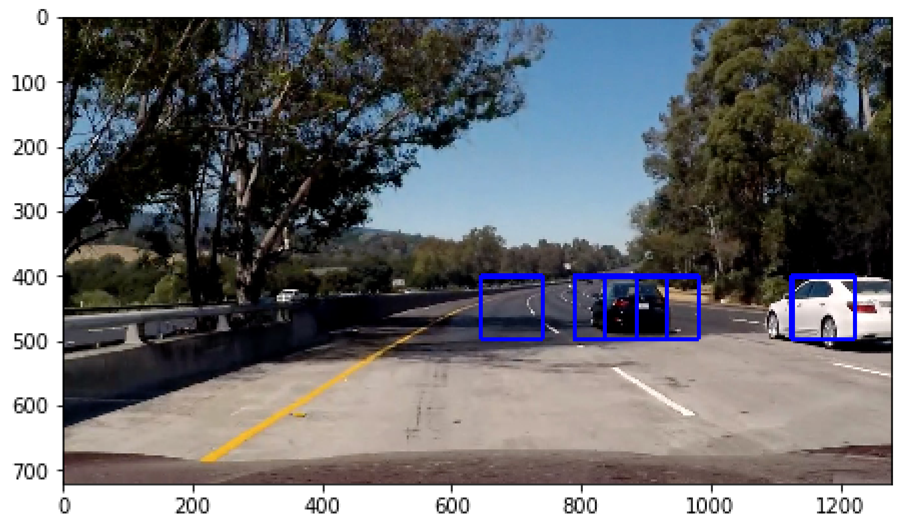
###Sliding Window Search

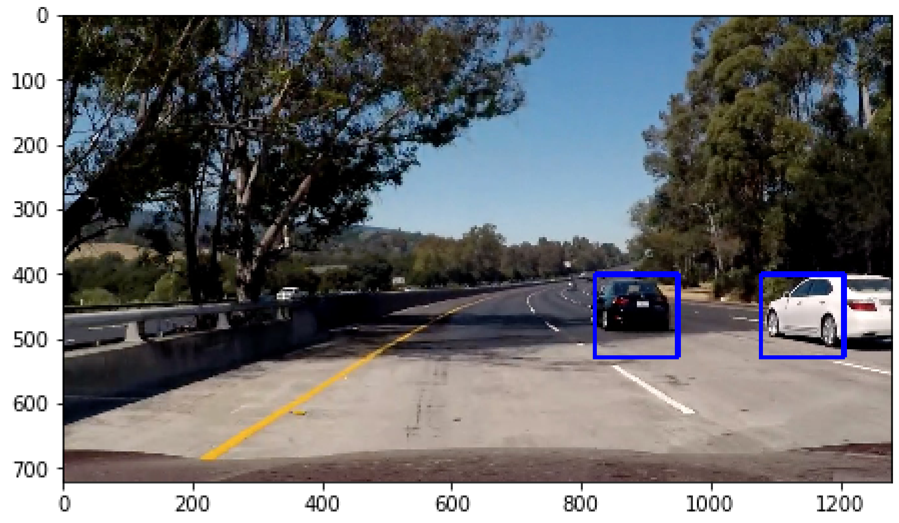
####1. Describe how (and identify where in your code) you implemented a sliding window search. How did you decide what scales to search and how much to overlap windows?

Please see Code Block 394.

Used find_cars function as defined in class. The functions moves over a defined region and slides the window across the region to match the feature vector of this sample with our required feature vectors to classify it as a car or non car. The region of interest where the sliding windows method is reduced to the lower half of image. Further this scanning is reduced to right half only as the left half detection don't matter as they are on the opposite road across the divider. Below are the detections on same images when using different scaled windows at different heights in image.



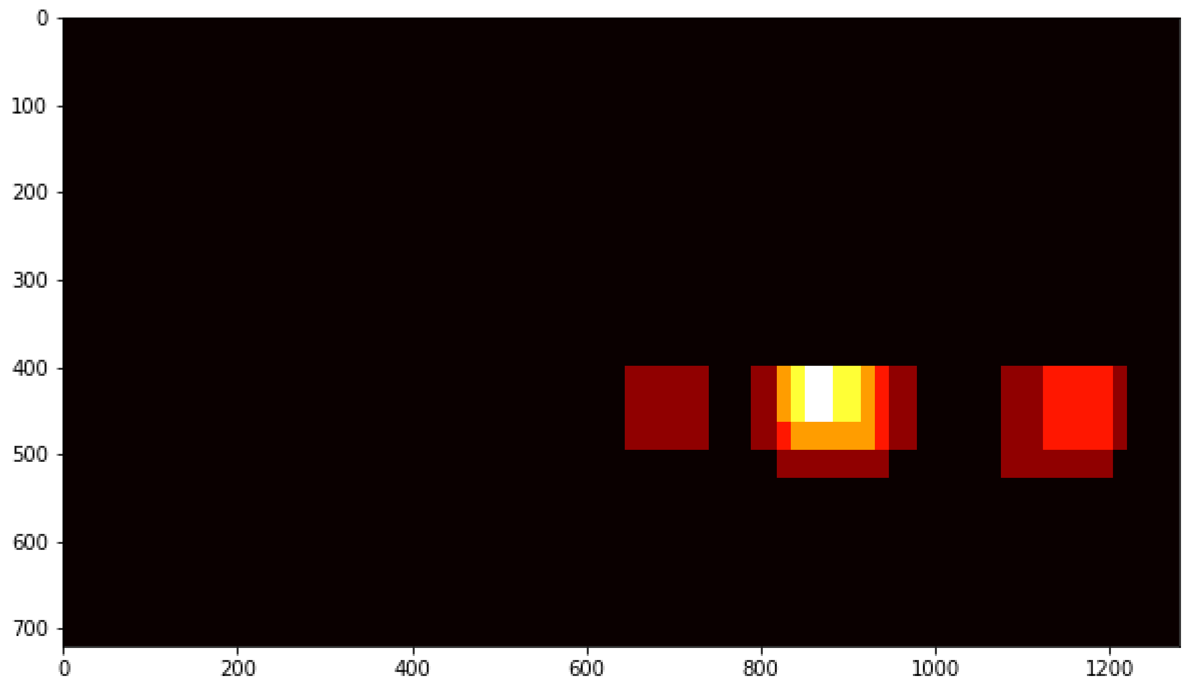




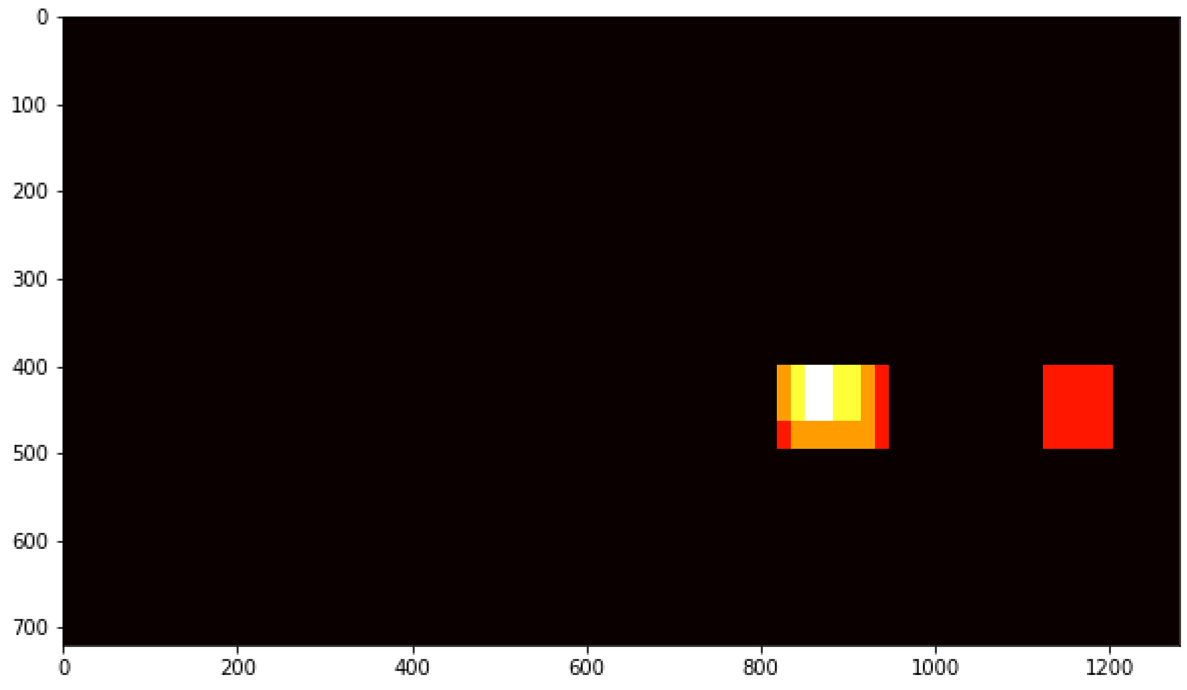
####2. Show some examples of test images to demonstrate how your pipeline is working. What did you do to optimize the performance of your classifier?

Ultimately I searched on several scales (1.0,1.5,2.0,2.5 etc)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:

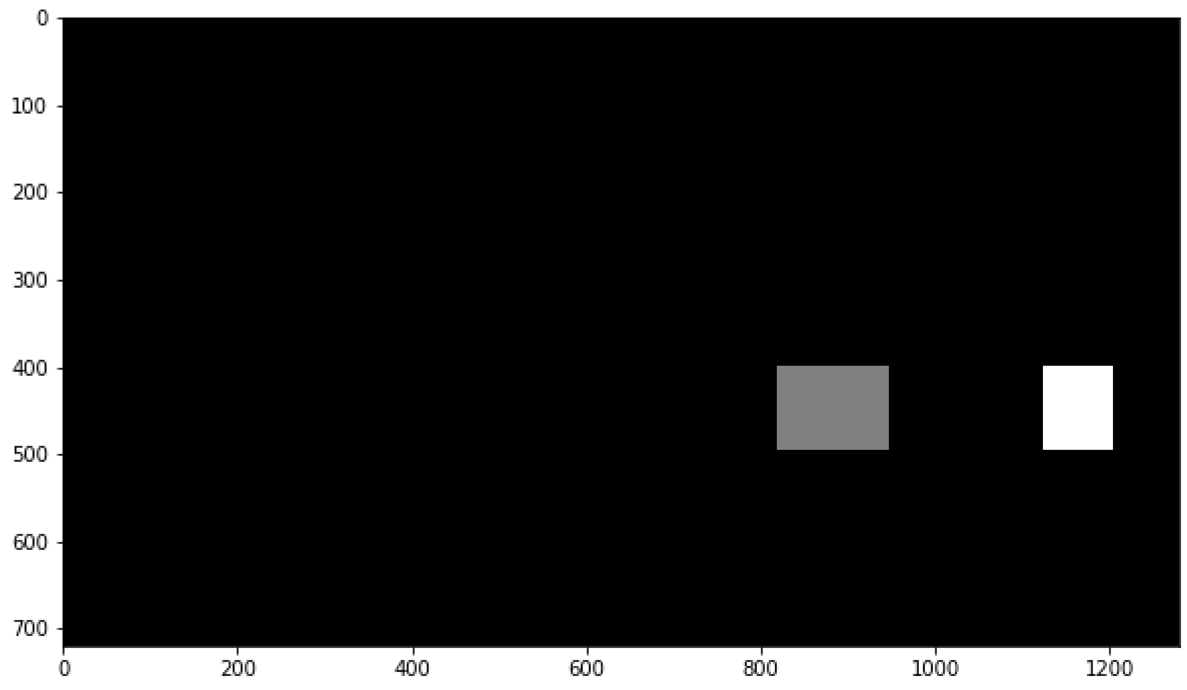
Merged Heatmap of all windows detection



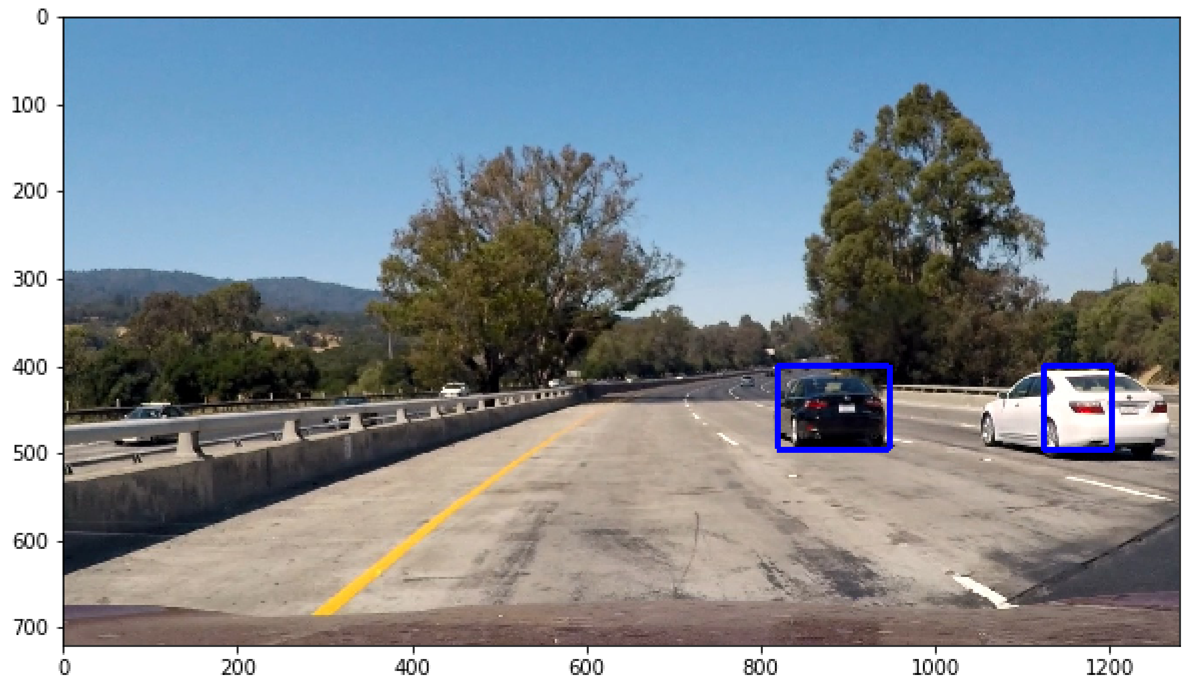
Applied Heatmap Threshold



Applied Scipy Labels



Generate Bounding Boxes on Labels



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

I have also added a video pipeline dependent on the previous video frames. The heatmaps are added over a period of 25 frames and then thresholded. This is to ensure only persistent objects (cars) show up in the labels. Random false detections are avoided using this method. This method also smoothens the detection over frames.

####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

---

###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 usually has chances of failing in areas that have high shadows or lighting. Moreover the parameters used to tune the classifier play a very significant role. After many attempts I was able to get this output. Minor change in a parameter such as cells per block or no of pixels per cell result in drastic effects.
I think testing with other color channels might also help in some cases. The current implementation is quite prone to false positives. Using a larger dataset may also help in avoiding false positives.