
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.

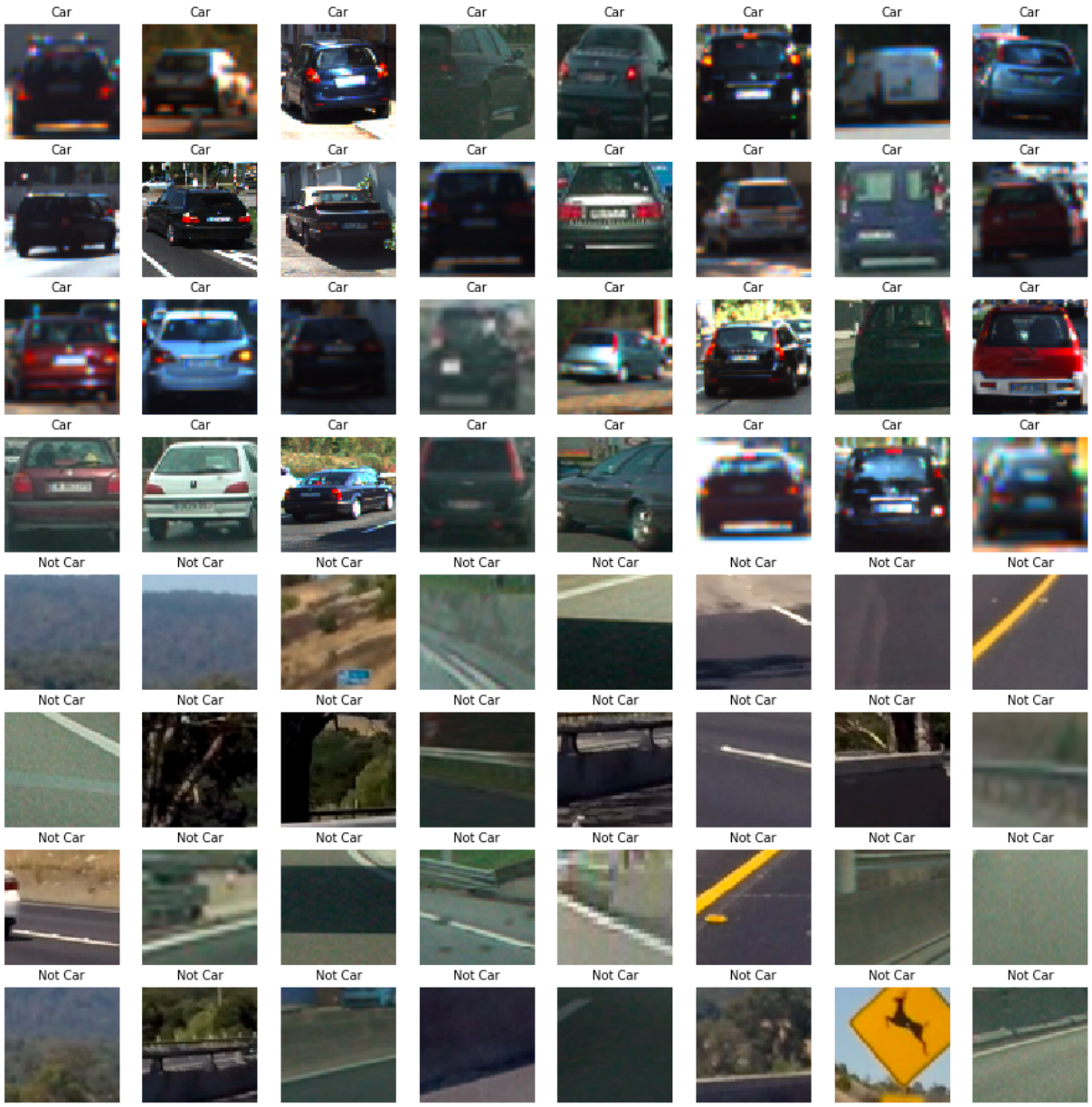
You're reading it!

###Histogram of Oriented Gradients (HOG)

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

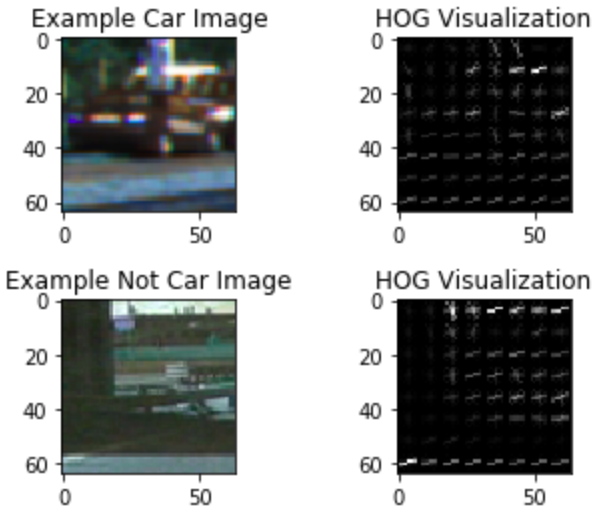
The code for this step is contained in the first code cell of the IPython notebook (Block 222).

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:



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=9`, `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 . Here is a list of few of them:

HOG Extraction							SVC Training	
Configuration Label	**Colorspace**	**Orientations**	**Pixels Per Cell**	**Cells Per Block**	**HOG Channel**	**Extract Time**	**SVC Training**	**Test Accuracy of SVC**
1	YCrCb	9	8	2	ALL	11.46	1.49	0.945
2	RGB	9	8	2	ALL	8.18	4.53	0.92
3	YCrCb	9	16	2	ALL	3.89	0.5	0.94
4	HSV	9	8	2	ALL	9.03	0.4	0.955
5	HLS	9	8	2	ALL	9.56	1.1	0.955
6	HLS	11	8	2	ALL	9.59	1.29	0.96
7	YUV	11	8	2	ALL	9.12	1.44	0.95
8	HLS	11	16	2	ALL	4.14	0.59	0.97

Finally Settled on YUV, Orientations = 11, Pixels Per Cell = 8, Cells Per Block = 2

####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 326. I have used Linear SVM

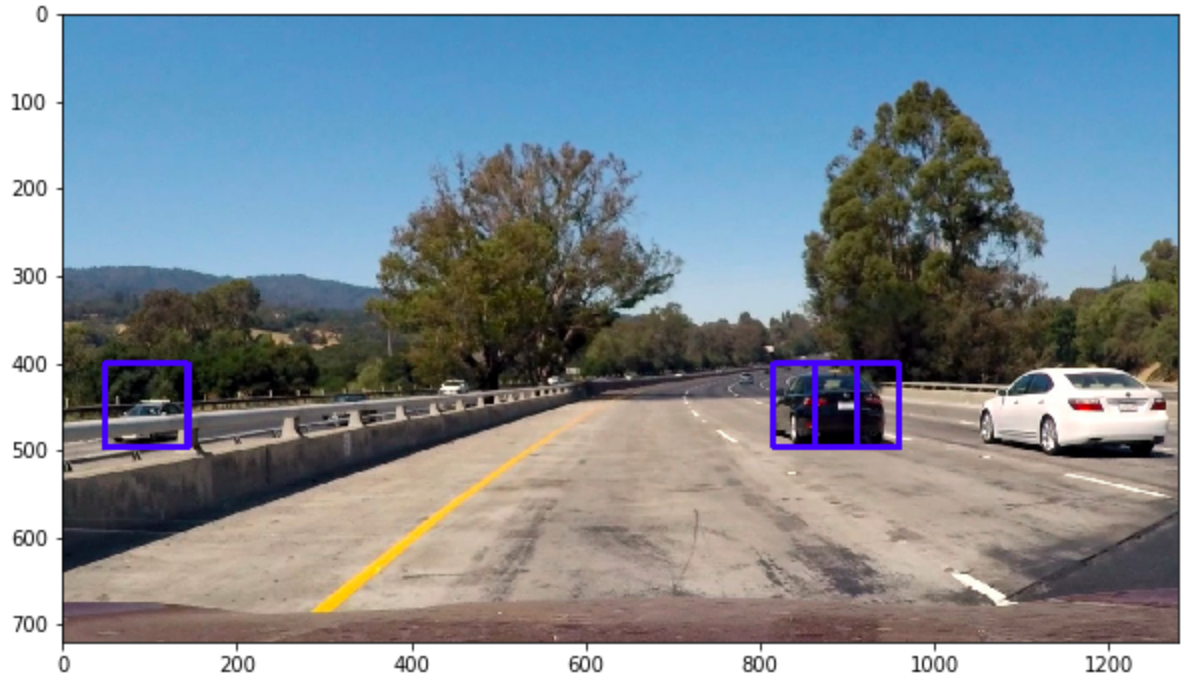
svc = LinearSVC()

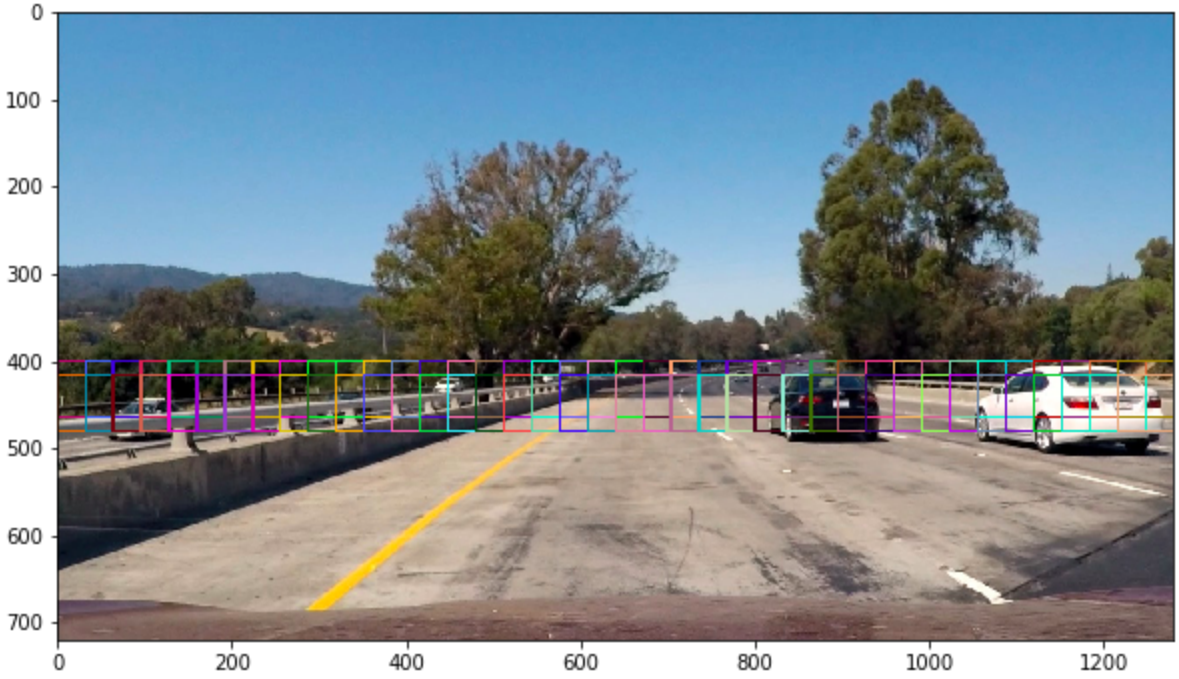
###Sliding Window Search

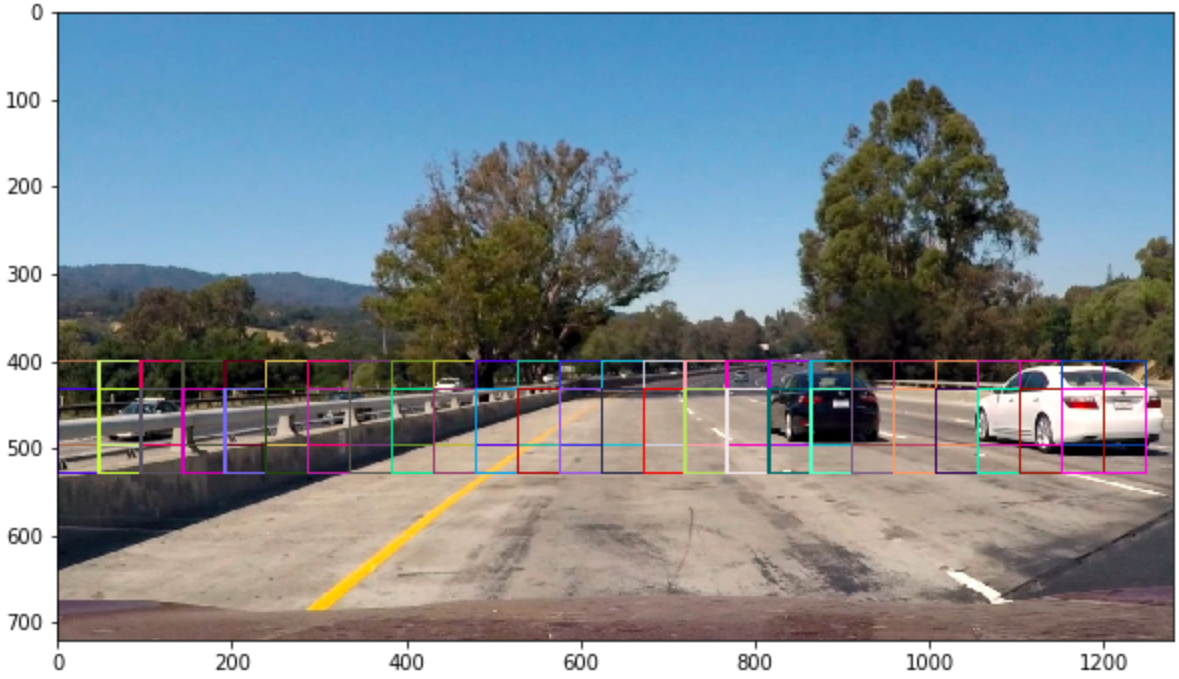
####1. Describe how (and identify where in your code) you implemented a sliding window search. How did you decide what scales to search and how much to overlap windows?

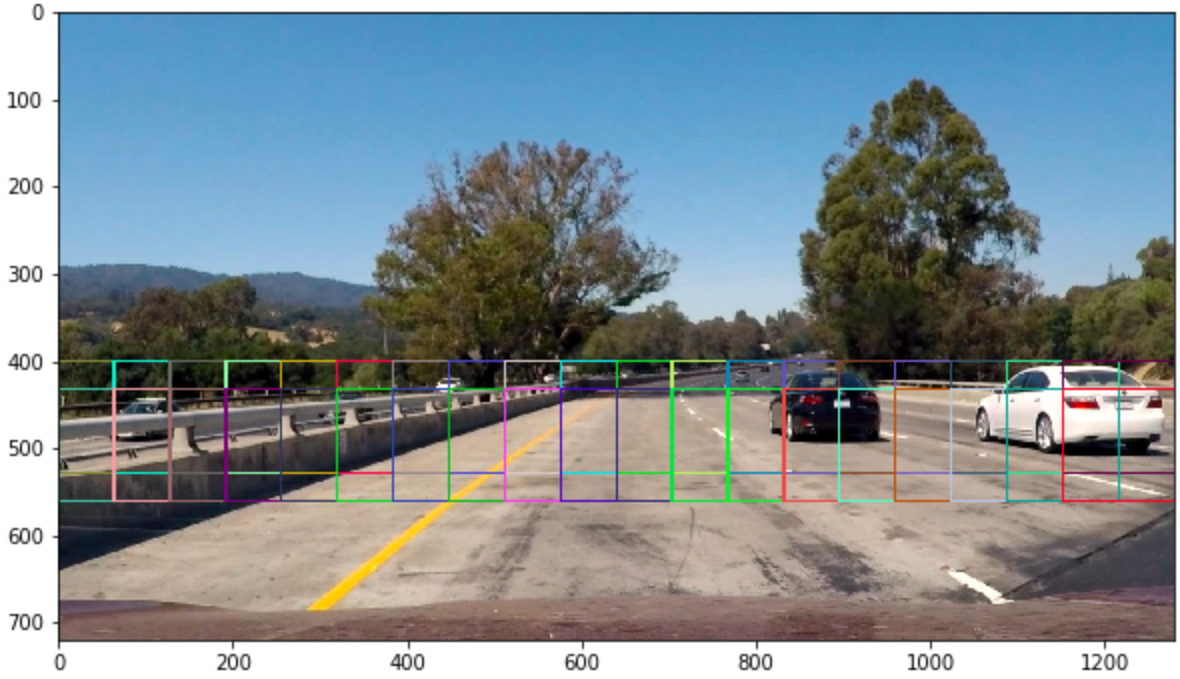
Please see Code Block 327. First Implemented the standard window search over the desired image portion as discussed in class.

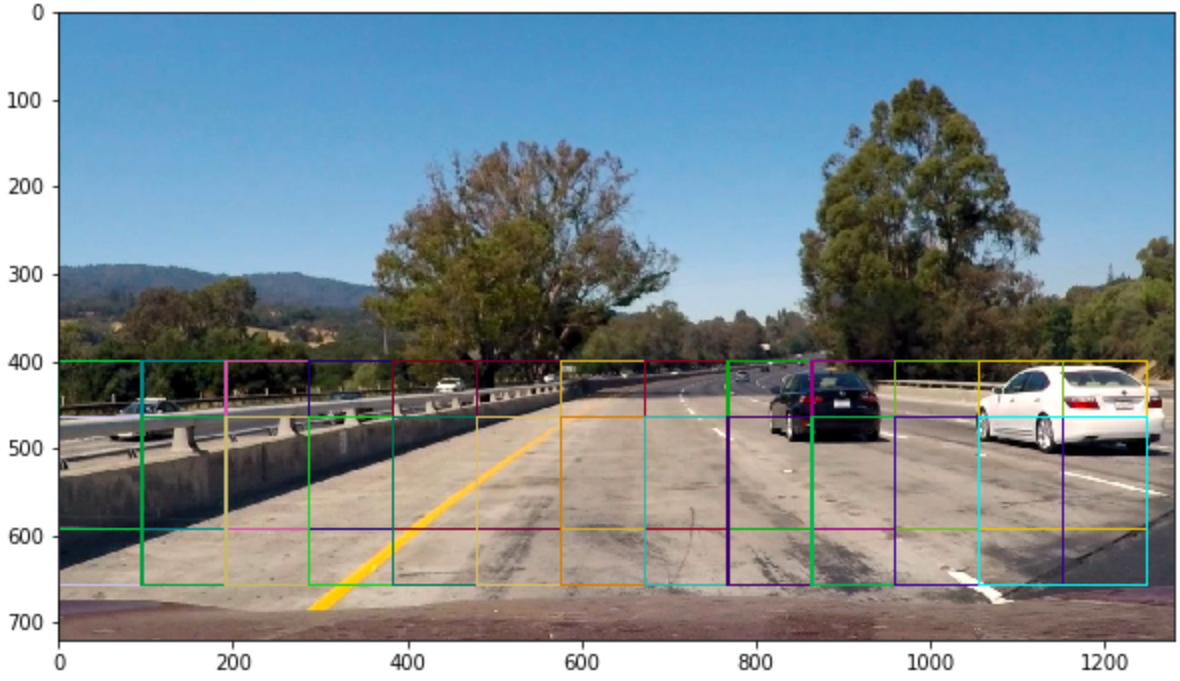
Later on implemented multiple size window scaling at different heights in image to detect cars easily.



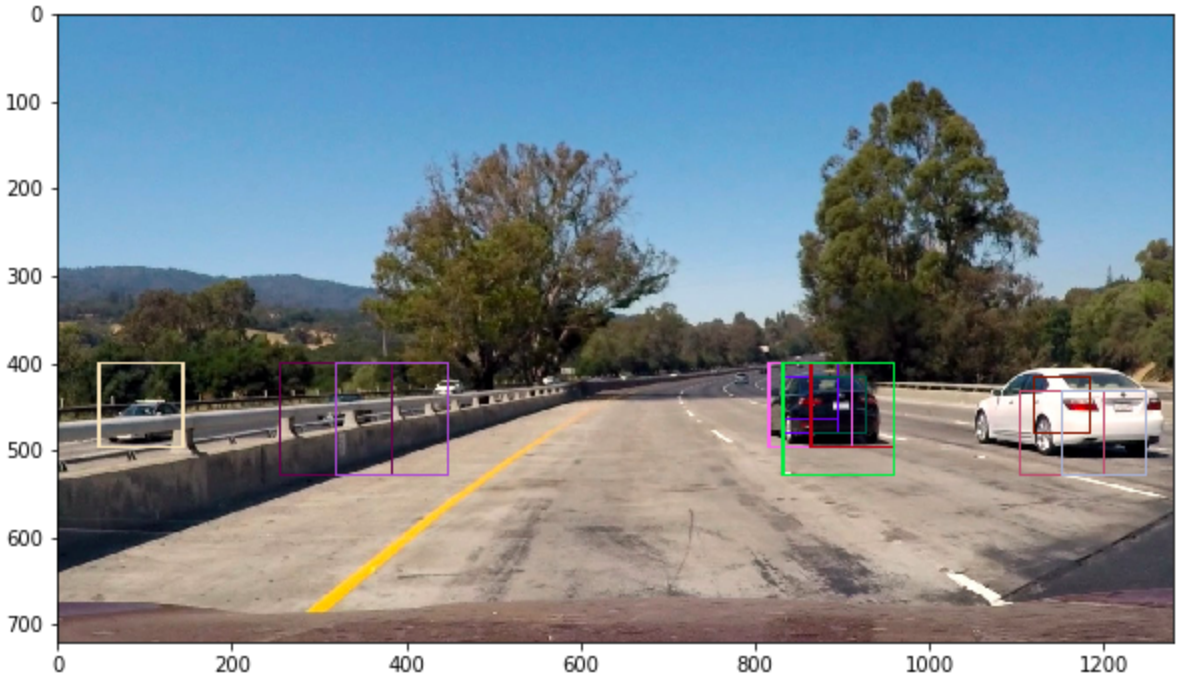

0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

The merged detections of all scaled windows is as here:



####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 YUV 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:

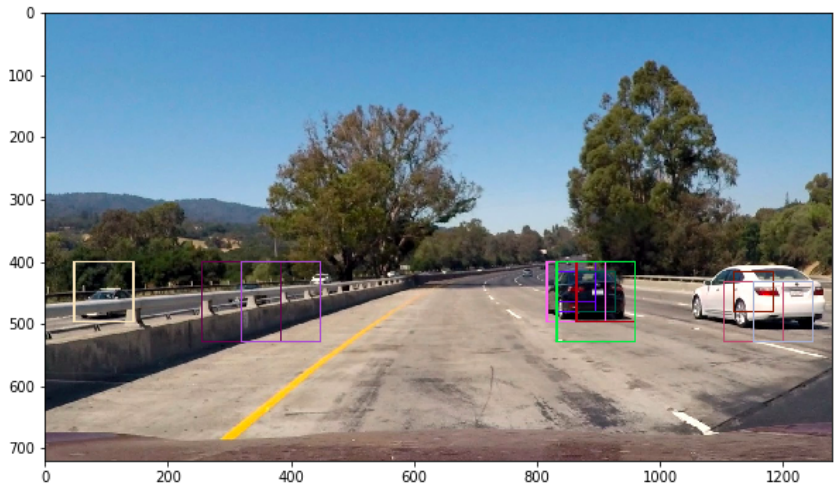
Video Implementation

####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's a [link to my video result]

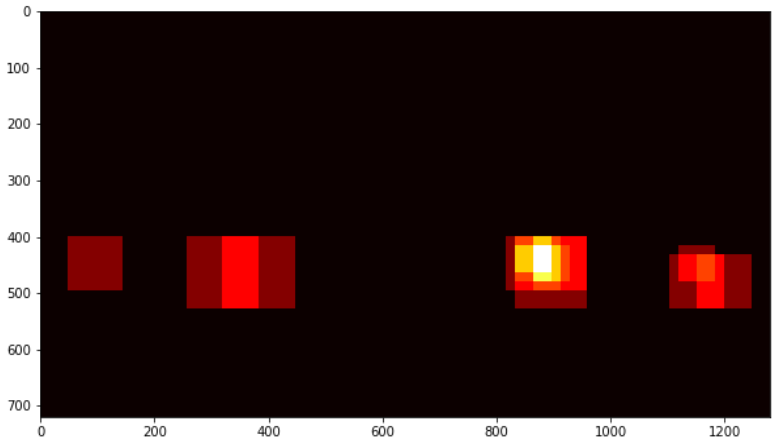
####2. Describe how (and identify where in your code) you implemented some kind of filter for false positives and some method for combining overlapping bounding boxes.

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. 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.

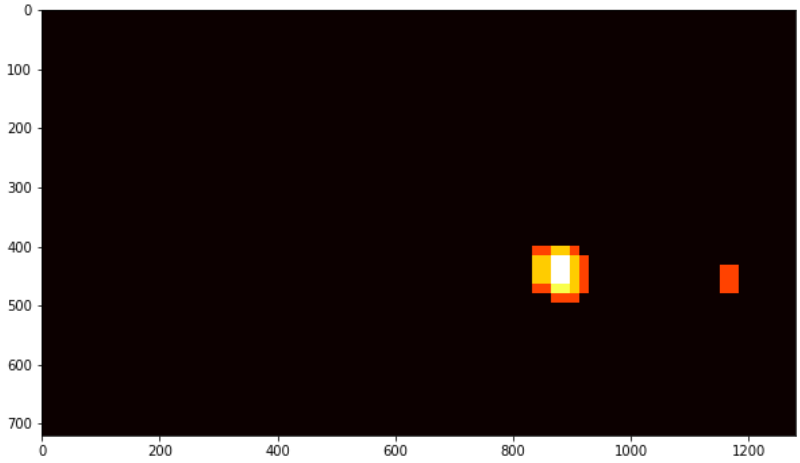
Here's an example result showing the heatmap from a series of frames of video, the result of `scipy.ndimage.measurements.label()` and the bounding boxes then overlaid on the last frame of video



Original Image with Window Detections

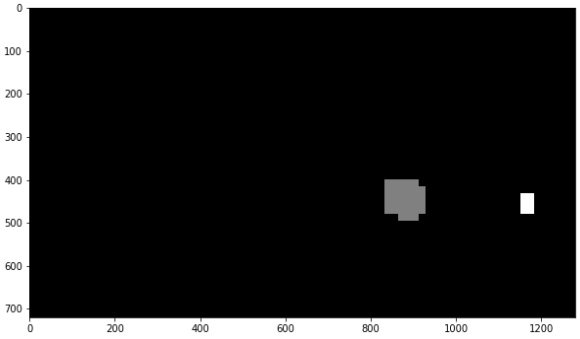


Heatmap

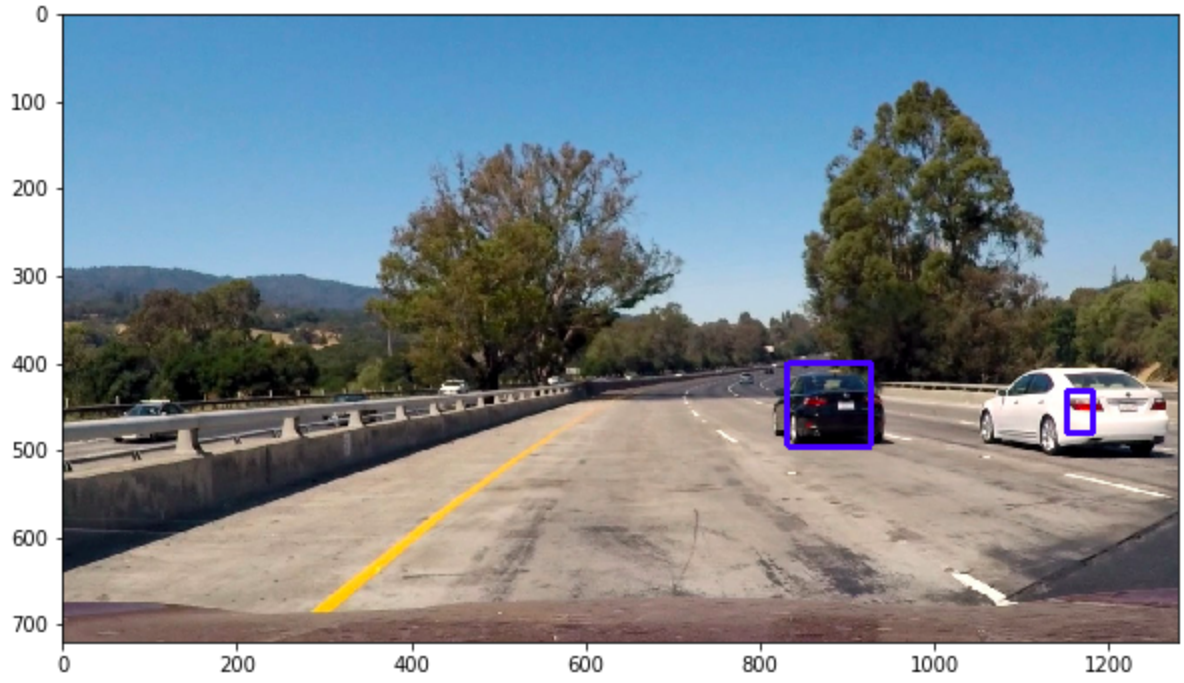


Thresholded Heatmap

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 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?

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 affects.
To make it more robust, as discussed in class, a video pipeline dependent on the previous video frames can be implemented to avoid any false positives. 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.