**\*\*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.

###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](https://github.com/udacity/CarND-Vehicle-Detection/blob/master/writeup\_template.md) 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 HOG function is defined in the section-3 of the code. It is basically combined with the rest of the classification data channels including color and the base pixel data as well. I have defined the parameters used for the HOG and other filters in the Section-5 and finally the combined pipeline has been defined in section-7. The following parameters were used for the HOG class after much hit and trial.

cspace = 'YUV'

orient = 11

pix\_per\_cell = 16 # HOG pixels per cell

cell\_per\_block = 2 # HOG cells per block

hog\_channel = 'ALL' # Can be 0, 1, 2, or "ALL"

spatial\_size = (4, 4) # Spatial binning dimensions

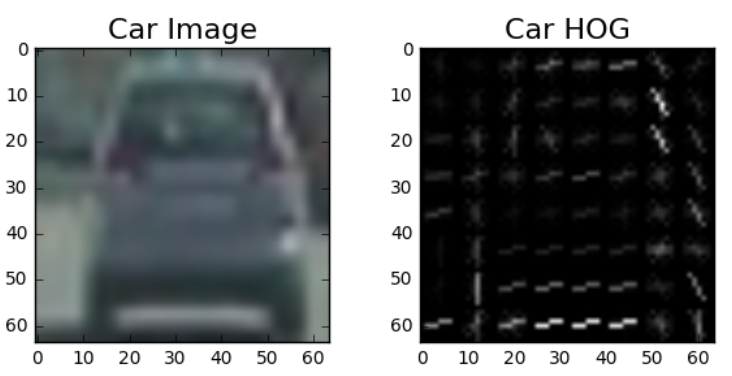
hist\_bins = 16 # Number of histogram bins

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 above given parameters



![alt text][image2]

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

I tried various combinations of parameters and...

####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 mentioned functions and the above given parameters. Basically using all three of the YUV channels for HOG and also 16 bins for the color histograms and finally a 4,4 spatial bin as well. This helped the trainer give the required importance to all of the features since the autotune feature was used. Here is the snippet of the result of the SVM training

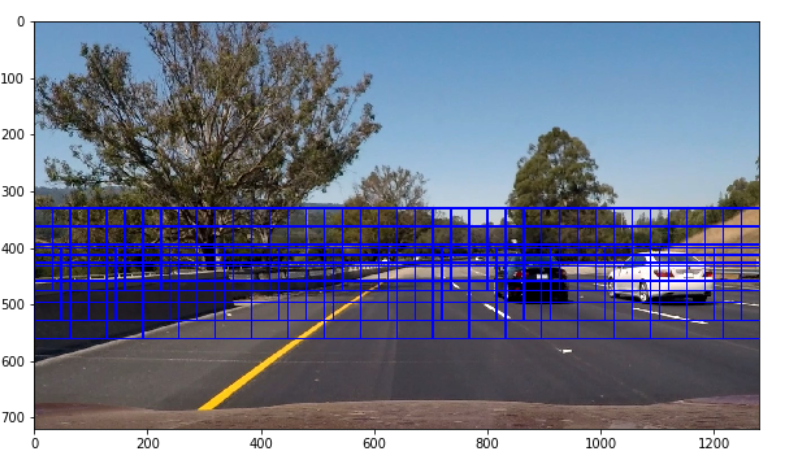
Using: 11 orientations 16 pixels per cell and 2 cells per block Feature vector length: 1284 0.03 Seconds to train SVC... Test Accuracy of SVC = 1.0

###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 implemented the sliding window inside the sliding window function in section-7 find cars function and then in the process\_image function. I basically decided to cover all the scales that can accommodate the cars in the test images and the project\_video. The overall was decided by the step size which was chosen to be 2.Aslo I only did the sliding window approach on a part of the image where the cars are visible. As you can see in the image the window sizes were varied as we moved down towards the bottom of the image to include the perspective of distance and size.

This was gathered using the given test images.



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?



The above are my final results with the multiple bounding boxes combined and the heatmap threshold applied. To get to the above result, I tried the image through multiple color spaces and then finally settled on YUV. I also reduced to the spatial bin size as that seemed to improve the results. I varied the scale to fit the perspective and add the distance based sensitivity required for the algorithm. This actually also reduced the number of total windows and made the algorithm faster.

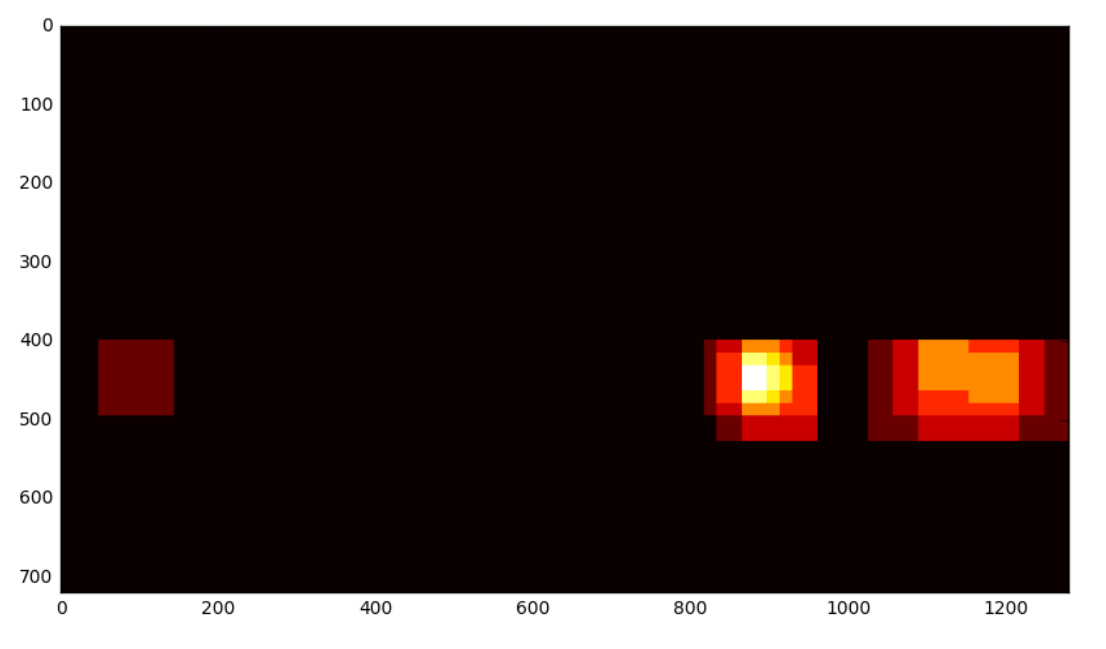
I also used the draw bounding box function to combine multiple boxes together after applying heat threshold and labelling them. I also changed the number of HOG bins to 11 to improve the gradient direction orientation

Video Implementation

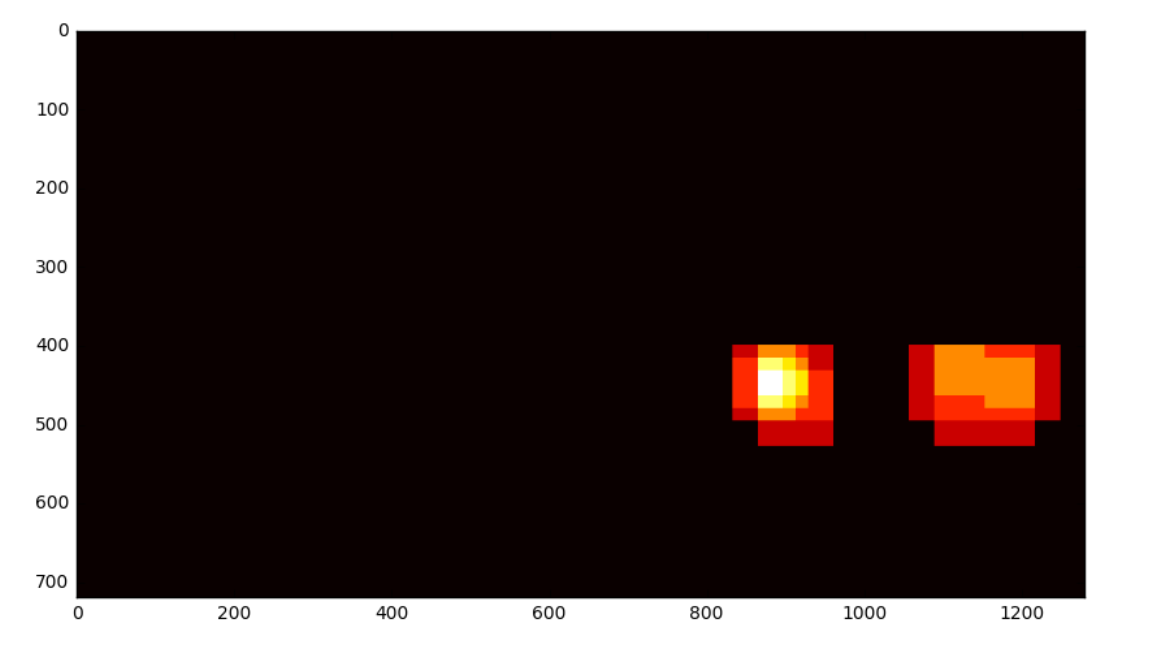
#1. Provide a link to your final video output. Your pipeline should perform reasonably well on the entire project video (somewhat wobbly or unstable bounding boxes are ok as long as you are identifying the vehicles most of the time with minimal false positives.)

The video is attached with the folder.

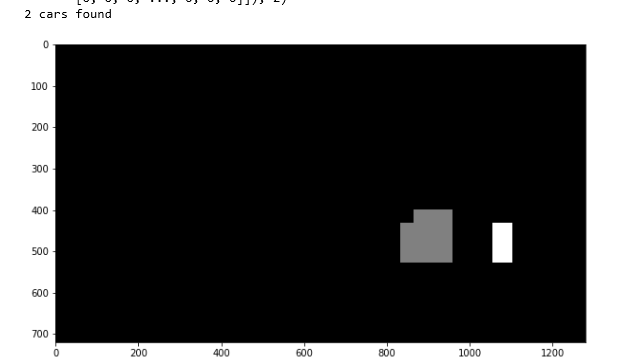
####2. Describe how (and identify where in your code) you implemented some kind of filter for false positives and some method for combining overlapping bounding boxes.

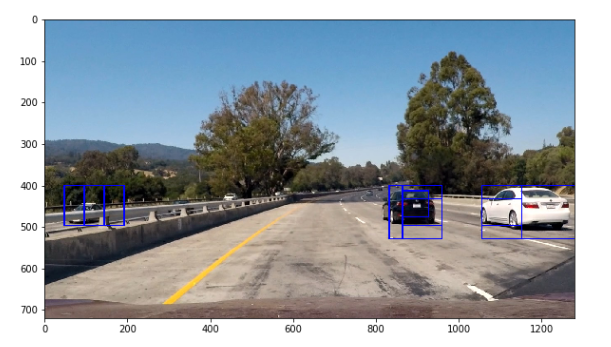
I recorded the positions of positive detections in each frame of the video. From the positive detections I created a heatmap and then thresholded that map to identify vehicle positions. I then used `scipy.ndimage.measurements.label()` to identify individual blobs in the heatmap. I then assumed each blob corresponded to a vehicle. I constructed bounding boxes to cover the area of each blob detected.

Detections after the threshold applied.



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 is how the bounding boxes were combined



The bounding boxes were combined using the minimum and th emaxumum numbered pixel covered by all the bounding boxes on that car label. This allowed us to effectively combine all the detections together.

###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?

Here I'll talk about the approach I took, what techniques I used, what worked and why, where the pipeline might fail and how I might improve it if I were going to pursue this project further. I believe that finding the right parameters is quite a big problem and the pipeline might fail to be robust enough. Moreover resolution becomes an issue when the cars are too far. Also since I have not included any filtering features the pipeline is not able to remember the position of the vehicles around the car and is basically starting afresh in each of the frame. Also I noticed that the cars in the dataset were short on certain colors and hence recognizing those colors would be difficult.