##Writeup Template

###You can use this file as a template for your writeup if you want to submit it as a markdown file, but feel free to use some other method and submit a pdf if you prefer.

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.

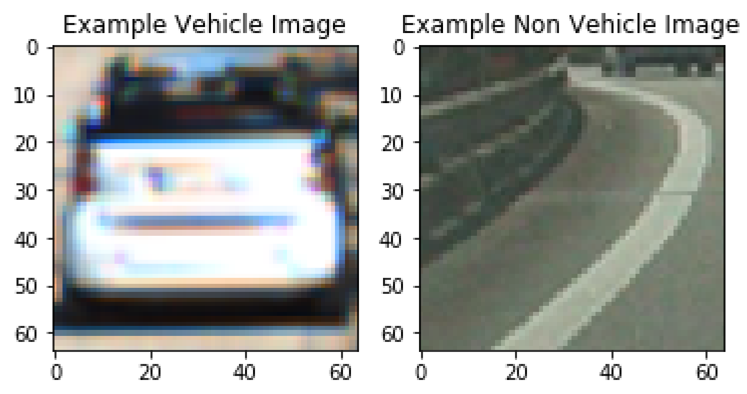
You're reading it!

###Histogram of Oriented Gradients (HOG)

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

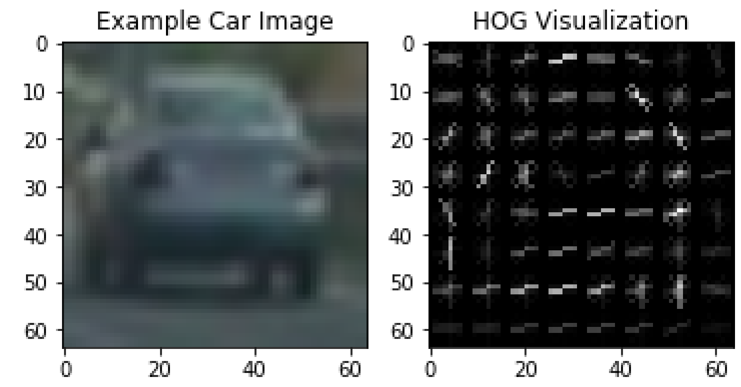
The code for this step is contained in the 1st-4th code cell of the IPython notebook.

I started by reading in all the `vehicle` and `non-vehicle` images. There are a total of `8792` vehicles examples and `8968` non-vehicles examples. Here is an example of one of each of the `vehicle` and `non-vehicle` classes:



I then generated a random index of training data and tested the image.

Here is an example using the gray image 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.

The code for this step is contained in the 6th code cell of the IPython notebook.

I tried various combinations of parameters. The parameters `orientations=32`, `pixels_per_cell=(8, 8)` and `cells_per_block=(2, 2)` worked well. I also used a linear SVC to colorspace and hog channel parameters, here are the result:

colorspace \ hog channel	0	1	2	ALL
RGB	0.9440	0.9493	0.9496	0.9685
HSV	0.9088	0.9102	0.9555	0.9828
LUV	0.9488	0.9251	0.9113	0.9876
HLS	0.9155	0.9513	0.9102	0.9887
YUV	0.9513	0.9071	0.9350	0.9870
YCrCb	0.9550	0.9172	0.9164	0.9913

So, colorspace YCrCb with all channels has best accuracy, that's my choice.

Again, I tested color classify parameters with colorspace `YCrCb` and here are the result:

spatial \ histbin	8	16	32	64
8	0.9479	0.9659	0.9733	0.9657
16	0.9389	0.9589	0.9589	0.9617
32	0.9251	0.9417	0.9507	0.9566
64	0.9189	0.9293	0.9310	0.9406

In generalï¼Œ the higher the histbin, the higher the accuracy while the lower the spatial, the higher the accuracy. Thus, I chose spatial binning of: 8 and 64 histogram bins.

####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).

The code for this step is contained in the 11th code cell of the IPython notebook.

I trained a linear SVM using `GridSearchCV`. I combined and normalized both color features and hog features with `StandardScaler()`.

The best parameters are: {'C': 0.1, 'kernel': 'linear'}.

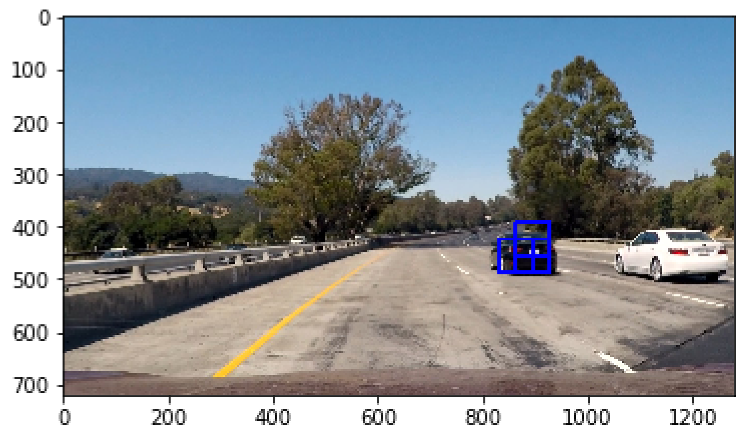
The test accuracy of SVC is 0.9966.

###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?

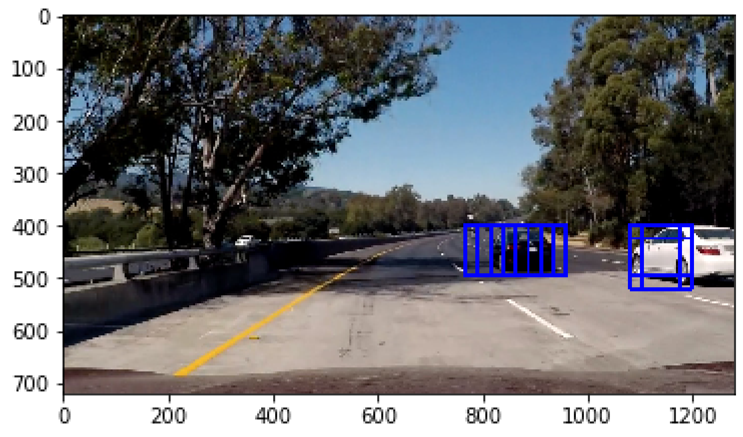
The code for this step is contained in the 14th code cell of the IPython notebook.

I decided to search lower half of the window with 96*96px box, and here is the result:



####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 used combined and normalized features from both color features and hog features with parameters above, 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.)

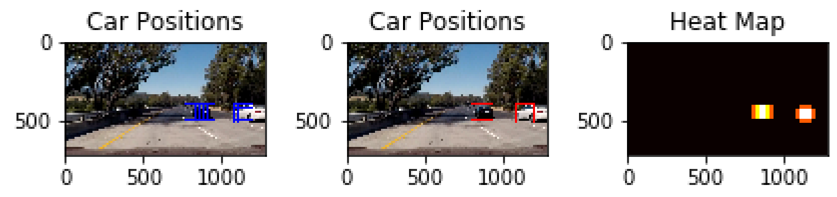
Here's a [link to my video result]()

####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 append the positions of positive detections in each image to a list. Using the list of positive detections I created a heatmap and then thresholded that map to reject areas affected by false positives. I also used a class to fliter false positive detection that flashed by. 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.
At first test, the output was no good enough, so I went back and used cv2.flip to augment the data, and changed spatial binning from 8 to 16 to save memory. I also used different scales to get better accuracy.

Here's an example result showing the heatmap from a test image.

The first image with blue box is original result, the second image with red box is the result of `scipy.ndimage.measurements.label()` and the bounding boxes, the third image is the heatmap result.



###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?

I think the pipeline is too slow, I think using CNN with GPU will faster.