Vehicle detection and Tracking 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.

Project files

1. The IPython notebook, which has all the code:
   https://github.com/khushn/carnd_p5/blob/master/vehicle_detection.ipynb

2. The output images folder:
   https://github.com/khushn/carnd_p5/tree/master/output_images

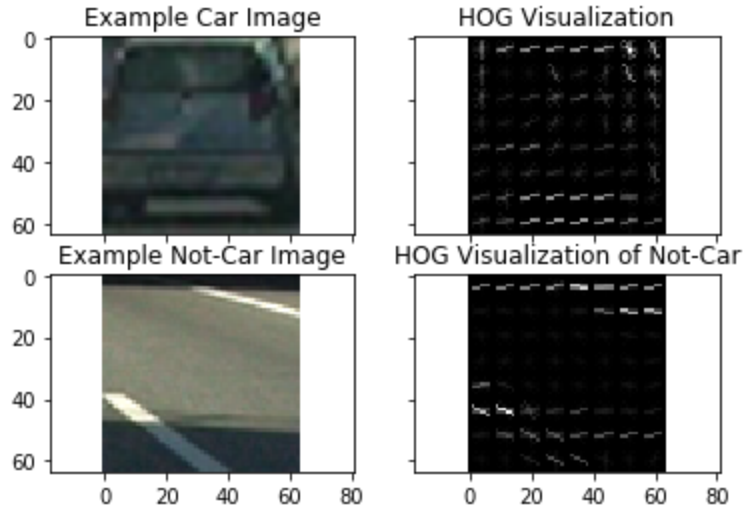
3. The vehicle detected output video:
   https://github.com/khushn/carnd_p5/blob/master/video_out.mp4

4. The lane marked first challenge video:
   https://github.com/khushn/carnd_p5/blob/master/video_vehicles_and_lanes_marked

Histogram of Oriented Gradients (HOG)

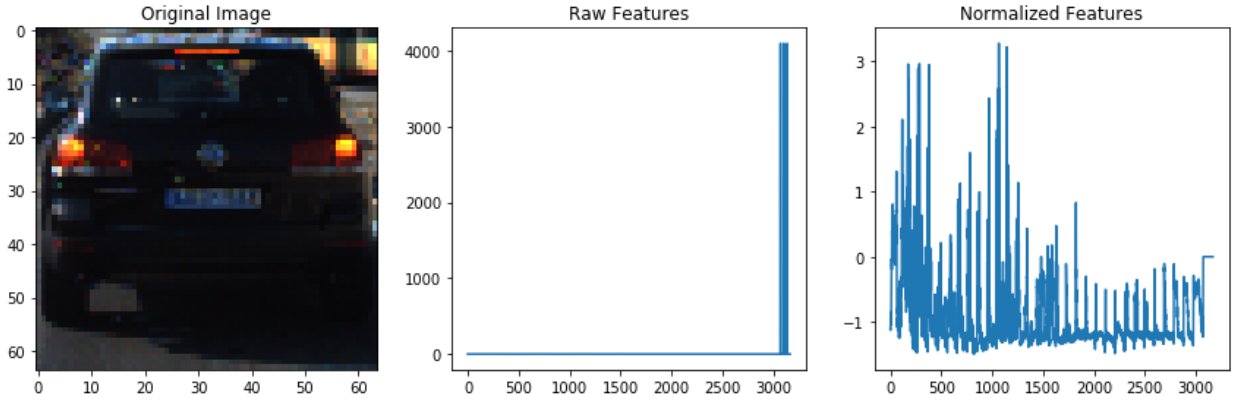
The code for extracting HOG features is in the first cell. I tried various color spaces. Example in the first cell I have used LUV. I did that for both car and not car images. The data for which is got from the training set at https://s3.amazonaws.com/udacity-sdc/Vehicle_Tracking/vehicles.zip from cars, and for not cars (https://s3.amazonaws.com/udacity-sdc/Vehicle_Tracking/non-vehicles.zip).

Sample HOG visualization below:



Combine and normalize features

Cell 2 in the notebook has the code to show, how combining and normalization of features was done. We do a bin spatial and color histogram, combine them and then normalize them. Figure shown below:



Training the SVM classifier

The cell below that has the code for training the classifier. The parameters used are:

colorspace	'YCrCb' *(Among the possibilities RGB, HSV, LUV, HLS, YUV, YCrCb)*
orient	9

pix_per_cell	8
cell_per_block	2
hog_channel	“ALL” (using all 2 were found to be better than a single 1)
Spacial features	Used
Histogram of color	Used

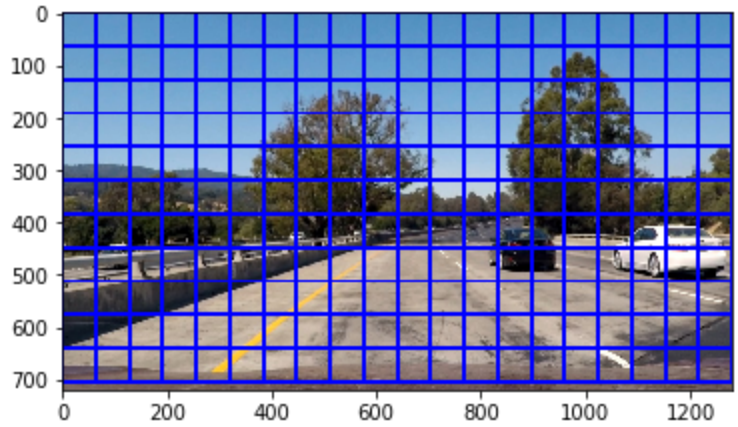
The results of training:

```
training data profile:
no. of cars:  8792
no. of not-cars:  8968
9.34 Seconds to extract HOG features...
Using: 9 orientations 8 pixels per cell and 2 cells per block
Feature vector length: 5292
0.25 Seconds to train SVC...
Test Accuracy of SVC =  0.995
My SVC predicts:  [ 1.  0.  1.  1.  1.  1.  0.  0.  0.  1.]
For these 10 labels:  [ 1.  0.  1.  1.  1.  1.  0.  0.  0.  1.]
0.00528 Seconds to predict 10 labels with SVC
```

So it was able to get a good accuracy of .995 on the test set. I used 20% from the training data for computing the test accuracy.

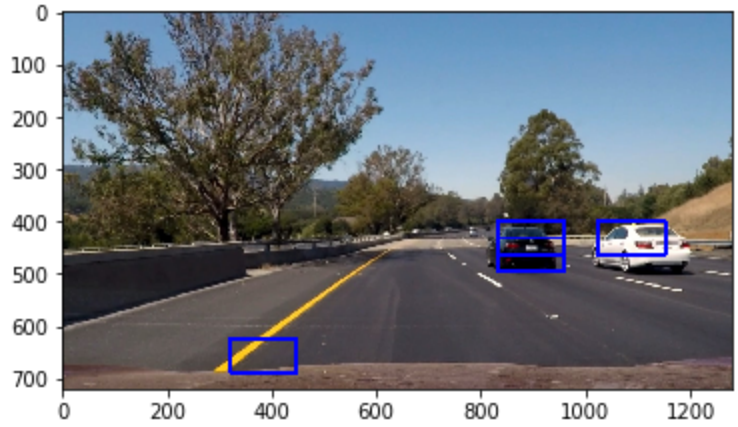
Sliding window search

The 4th cell in the notebook has the sliding window implementation.



Trying the classifier on an image

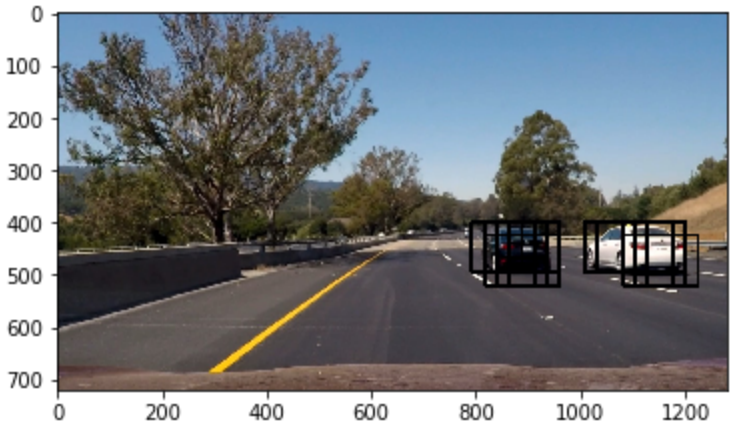
Below is a sample image of just trying the trained classifier. For that, we need to extract features in the same way, as we do for training. Then normalize them using the same scalar object reference. And then predict the vehicles using a sliding window approach mentioned above.



The above image was processed using the approach of computing HOG params as needed. We optimize the approach to be more efficient, as advised in the lesson on sub-sampling.

HOG sub sampling

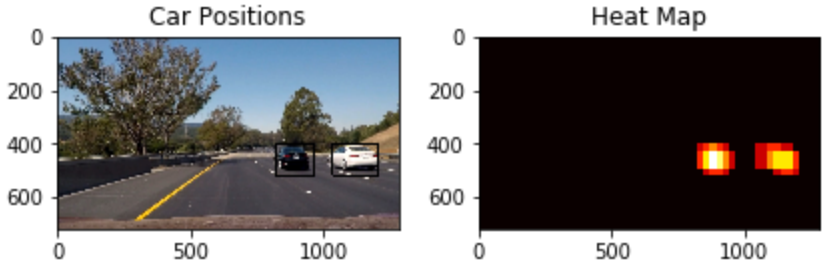
This approach is more time/cpu efficient as we compute all the HOG features once, and then use it by combining them as necessary.



The result is slight better, because another change in this piece of code (provided by Udacity, actually) is that the scaling of sliding windows is provided. And we make use of it by setting the value to 1.5, which seems to be optimal.

False positives

We eliminate the false positives by generating the heat map. Basically drawing the rectangle, in the overlapping area of multiple rectangles. We set the threshold=2.



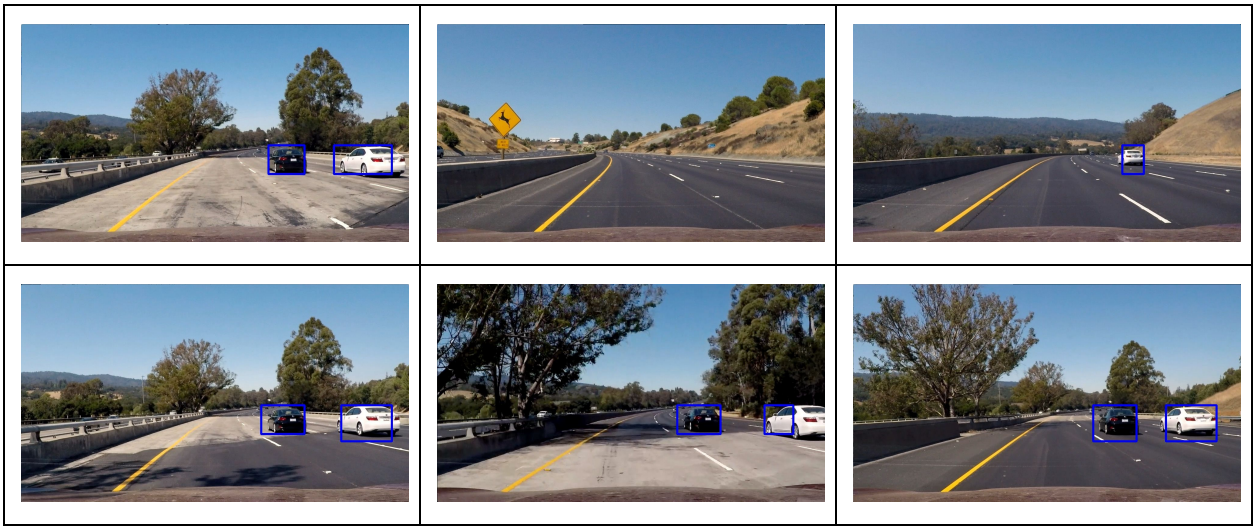
If you see the left image above the car positions are now identified properly.

process_vehicles() Function

We combine the pipeline into process_vehicles() function. Which takes in an image and returns the cars detected image.
For advanced part, I modified the function to invoke a `process_lanes()` function, which takes care of lane marking (code reused from project 4, and present in a separate python file process_lanes4.py

Processing the test images

We call process_vehicles() on all the test images and generate the output in https://github.com/khushn/carnd_p5/tree/master/output_images folder. Samples shown below:



Processing the video

Confident that all the test images worked fine, I applied the same thing on the video. First on the test video.
I noticed that processing was quite slow, almost 1.2 iterations/second. And it took almost 25 minutes to process the video. Since I know, that the video has some 40 frames per second, I decided to skip 10 frames, for processing. Then the processing time became manageable.
The output video is present here:
Video_out.mp4 (github link:
https://github.com/khushn/carnd_p5/blob/master/video_out.mp4)

Advanced part

I combined the vehicle detection and lane finding (using project 4th code) to be able to generate a single video. Which is present in file:
Video_vehicles_and_lanes_marked.mp4
(github:
https://github.com/khushn/carnd_p5/blob/master/video_vehicles_and_lanes_marked.mp4)

Discussion

If you notice in the video as the white car goes further up, it is not identified. Which can be done by scaling down the size of the sliding window. Which is noted as further work for this project.

Also some frames of the video has some wrong identifications. They are (thankfully) only a few, but still are there. To fix, that we can filter out vehicle detections outside of lanes. Since we managed to integrate the two. Once the lanes are detected. We can add some more thresholding in the form of a triangle (corresponding to the lanes area) apart from the vertical one, which we use in the project.

Also some other approaches like changing the threshold of the heatmap could be done. Yet another possible approach is that showing the boxes, only if detected over successive frames.