Vehicle Detection
[![Udacity - Self-Driving Car NanoDegree](https://s3.amazonaws.com/udacity-sdc/github/shield-carnd.svg)](http://www.udacity.com/drive)

In this project, your goal is to write a software pipeline to detect vehicles in a video (start with the test_video.mp4 and later implement on full project_video.mp4)

The 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.

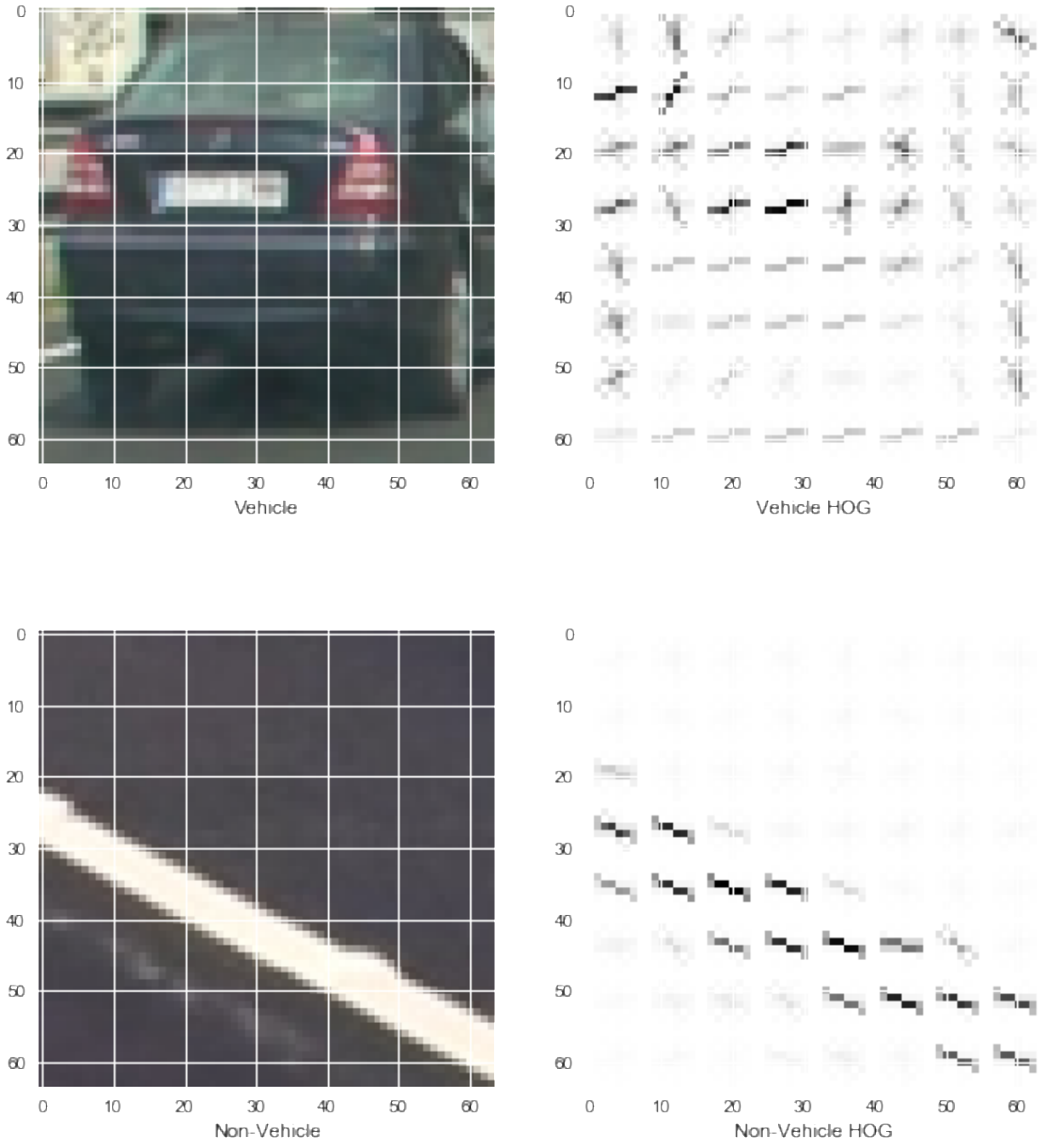
Here are links to the labeled data for [vehicle](https://s3.amazonaws.com/udacity-sdc/Vehicle_Tracking/vehicles.zip) and [non-vehicle](https://s3.amazonaws.com/udacity-sdc/Vehicle_Tracking/non-vehicles.zip) examples to train your classifier. These example images come from a combination of the [GTI vehicle image database](http://www.gti.ssr.upm.es/data/Vehicle_database.html), the [KITTI vision benchmark suite](http://www.cvlibs.net/datasets/kitti/), and examples extracted from the project video itself. You are welcome and encouraged to take advantage of the recently released [Udacity labeled dataset](https://github.com/udacity/self-driving-car/tree/master/annotations) to augment your training data.

Some example images for testing your pipeline on single frames are located in the `test_images` folder. To help the reviewer examine your work, please save examples of the output from each stage of your pipeline in the folder called `ouput_images`, and include them in your writeup for the project by describing what each image shows. The video called `project_video.mp4` is the video your pipeline should work well on.

My first step was to choose and load training my data. Ultimately, I chose to only use the KITTI vision benchmark suite for vehicle data in addition to the the entire non-vehicle dataset. More discussion of my data choice can be found in the following sections, and the code to load my training data can be found in 15-17 cells of the ipython notebook.

Secondly, I extract the training data from the training images in 19-20 cells of the ipython notebook. This calls the function 'extract_features' which can be found no.9 cell of the ipython notebook.

Thirdly, for training image HOGs, the 'get_hog_features' function no.4 cell of the ipython notebook. My main focus at this point was, however, to get the feature extraction working and I changed my parameters several times with different combination. The image below shows a training image and its HOG, with my final parameters.



Then, I chose a linear SVM, and used HOG, spatial and histogram features. Then, I splitted the whole dataset into training and test data which constitutes 70 % and 30 % in no.27 cells of the ipython notebook.
I got around 98% accuracy. Below is the image for the proof.

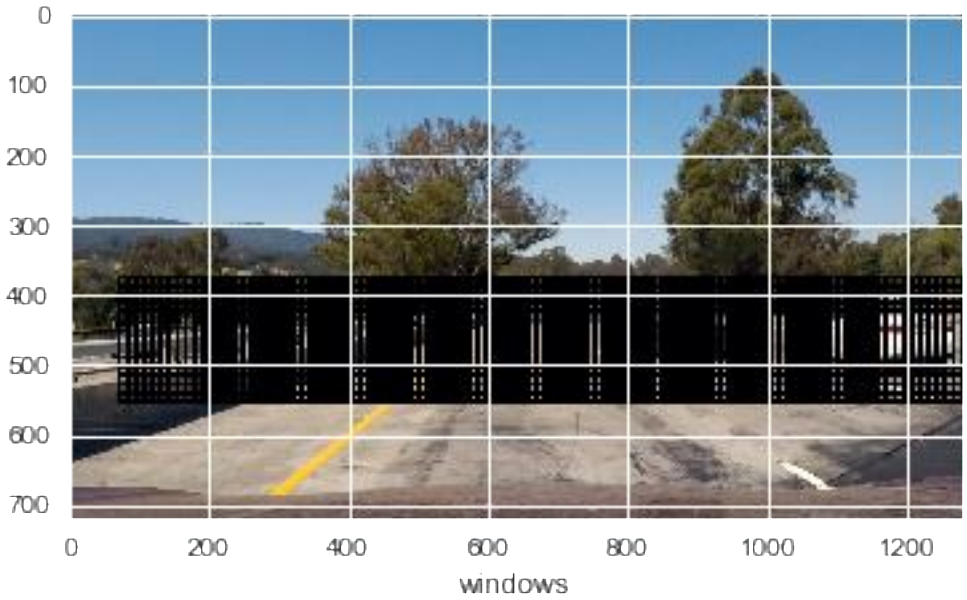
```
print(confusion_matrix(y_test,predictions))
```

```
[[2609   57]
 [  70 2592]]
```

```
print(classification_report(y_test,predictions))
```

	precision	recall	f1-score	support
0.0	0.97	0.98	0.98	2666
1.0	0.98	0.97	0.98	2662
avg / total	0.98	0.98	0.98	5328

Then, I worked with sliding window technique. My solution consists of two layers of windows, one 64x64 px and one 128x128px. These two grids were slightly staggered on the x axis and both overlapped 90%. sliding windows implementation is in no.10 and 37 cells of the ipython notebook.



Then I did the video implementation. with collections.deque, I kept track of the heatmaps of 10 adjacent frames and then averaged them. I subsequently performed my thresholding and labeling

on these heatmaps. This can be seen above and inside my pipeline function in no.37 cell of the ipython notebook.
My output video is in project_result.mp4.

Discussion:

Firstly, with respect to extracting the features, I played with all the parameters to find out how each combination is working. Then, in training, I tried with SVM, Linear SVM, decision trees. But I stick with the linear svm. And Finally, with respect to sliding windows. I tried to minimize the portion (not to scan the sky's and trees).