Project 5: Vehicle Detection Project Report

Objective:

In this project, our goal was to write a software pipeline to detect vehicles in a video. Since the video is nothing but the sequence of images at some specified frame rate, so, adopting the logical approach, I first developed pipeline and tested it on the images first. The images and videos are the same as I used in the Project 4 (Advanced Lane Finding 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.
- Estimate a bounding box for vehicles detected.
- Run your pipeline on a video stream and create a heat map of recurring detections frame by frame to reject outliers and follow detected vehicles.

The inputs provided:

As stated above, the moderately large set of images (that contained both car and non-car images) were provided for training our classifier. There are 8792 cars (that are distributed in different shapes, colors, distances and orientations) and also around 8968 non-car images. The 80/20 rule was used to split the dataset into training and validation set (i.e.20% of the combines ~17000 images, 20% belong to the validation set and remain are used for training).

The output images from various steps are present in the “output images” folder with descriptive names.

Description of the pipeline

In this section, a brief account of the steps required to implement a pipeline is provided. Most of the ideas, inspiration, suggestions and the code snippets were taken from the lectures. The provision of support material helped a lot to understand the concepts and reduced the development time.

1. ***Visualization*** : Even this is an optional task and has least to do with the overall processing algorithm, but the it gives clear picture how the dataset looks like in actual.
2. ***Extracting HOG Feature***: The Histogram of Oriented Gradient (HOG) feature descriptor is widely used for object detection because it provides a unique signature to various objects. The scikit-image package has a built in function to extract Histogram of Oriented Gradient features. The description of the algorithm being implemented under the hood is beyond the scope of this report. The scikit-image ***hog()*** function takes in a single color channel or grayscaled image as input, as well as various parameters. These parameters include orientations, pixels_per_cell and cells_per_block.
3. ***Applying the HOG Feature:*** The HOG feature is applied to nearly 18000 images (both cars and otherwise). The dataset is also split into the training and validation set using 80/20 rule. The following shows some of the simulation results.

   ```
   It takes 35.41 seconds to extract HOG features.
   Size of Training Data 14208
   Size of Validtion/Test Data 3552
   Using: 11 orientations 16 pixels per cell and 2 cells per block
   Feature vector length: 1188
   ```

4. ***Train the classifier:*** In this code section of the python notebook, I train the classifier u sing the Support Vector Machines (Linear SVC is used that is another implementation o f Support Vector Classification for the case of a linear kernel). Since it uses a linear kern el to categorize the dataset, there is not field of kernel in the SVC command. It takes ar ound 1.3 seconds to train the classifier whereas its accuracy is 98~99%.
5. ***Using the above Linear Support Vector Classifier (LinearSVC) to Detect Cars in an I mage:*** in this section, a method is defined that extracts the features using hog-subsam pling and make predict as its result. To avoid spending time in uselessly searching for c ars in sky and trees, a window is defined to search in the lower half of the image, this is done by introducing the ystart and ystop parameters.
6. ***Draw Rectangles on the image***: Rectangles are drawn on the detected regions of the i mage. As suggested in the lecture we need to swipe through the whole horizontal plan e under the sky and treeline. Because we expect our cars to be located anywhere in this region. So, we need to scan the whole region. And since the size and shape of the car will be also not remain throught, we need to keep changing the window size to accurat ely track the car. Therefore, at this stage, there are chances of false predictions as well t hat will be discussed in the later sections.
7. ***Apply the Heatmap based on the rectangle locations***: The purpose of this is merely t o identify the “hot-pixels” in the image. If your classifier is working well, then the "hot"

parts of the map are where the cars are, and by imposing a threshold, you can reject ar eas affected by false positives.

8. ***Draw the rectangle Again:*** Finally Draw the rectangle on the detected hot pixels and o verlay it on the actual image to finally detect the cars in the image.

To avoid duplicity, the images as a result of various stages in the pipeline are not described here:

Possible cases of failures and remedies:

I think the pipeline is not expected to work in the urban and congested environment, where there are not only cars, but four-wheelers of other categories as well. Also if the module scans the whole images, it may wrongly detect the high rising but in-view bill-boards that contain modern vehicles advertisements. In such cases, the number of false detections may increase even more. What I can think, as a solution, is that we should need train our classifier of more rigorously using more and variety of data. And we may need to experiment with different SVM if there is a case to classify four-wheeled objects into more categories.