Udacity Self-Driving Car Nanodegree
Term One: **Computer Vision** & **Deep Learning**
Project Five: **Vehicle Detection** & **Tracking**

Nishan Srishankar
Worcester Polytechnic Institute

2017

1 Introduction

Detecting and tracking vehicles in a video stream is the final project in the Computer-Vision and Deep Learning module. This is of primary importance for autonomous vehicles as tracking and projecting future locations of neighboring vehicles can help in creating a safe low-level path planner for the car.

2 Methodology & Implementation

This project is subdivided into five steps which are explained in detail below.

2.1 Camera Calibration and Distortion Correction

The first step of this project is the same as that in Project Four (Advanced Lane Line Detection) and deals with correcting radial and tangential distortion caused by convex lenses in cameras. The images are calibrated using standard chessboard images with tiles arrayed in a 9-column, 6-row pattern. Built-in OpenCV functions such as *findChessboardCorners* and *drawChessBoardCorners* are implemented on sample grayscale chessboard images which outputs successful returns and detected corners. It was found that corners in three chessboard images weren't detected.

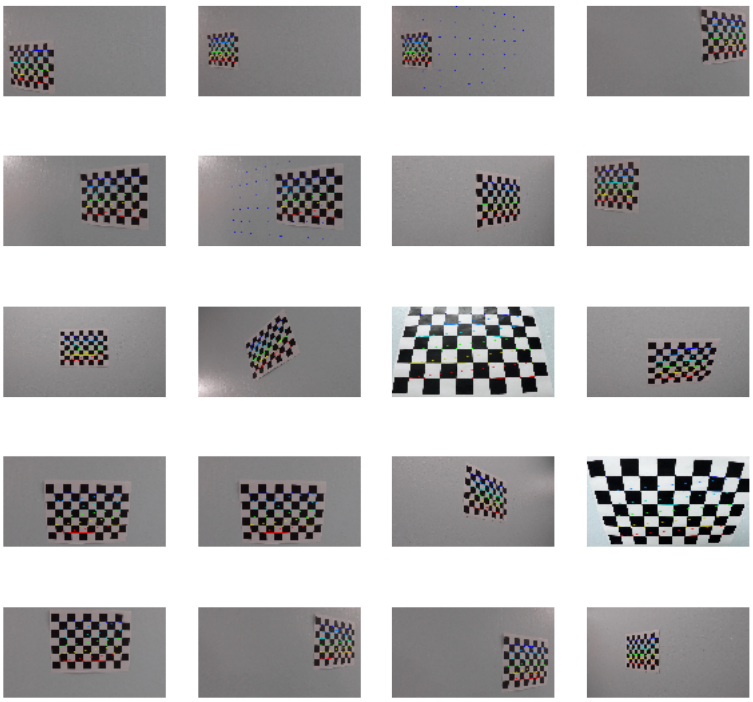

Figure 1: Chessboard Calibration by finding corners

The existing *calibratecamera* function was used to take in corners saved in an image-array "detected" using a camera, and map these to a 3D object array obtained using Numpy's *mgrid* since the pattern repeats regularly and returns an undistortion matrix and coefficients. The matrix and coefficients are saved in a pickle file to prevent the need for recalculation. This is used in *undistort* to obtain a corrected image, which can be seen in Figure 2 on chessboard images without detected corners and Figure 3 on a sample driving image. There is noticeable correction in the chessboard image, and slight correction in the driving image (recognizable by looking at the car's hood).

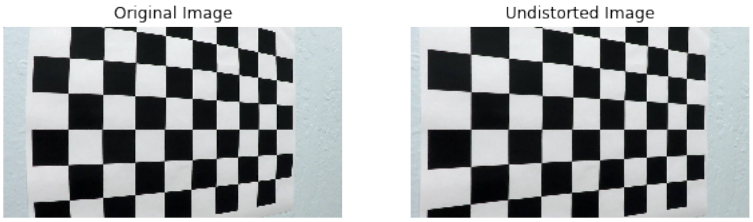

Figure 2: Undistortion on chessboard images

Figure 3: Undistortion on a real-world driving image

2.2 Dataset Analysis

The main dataset was obtained from folders of car and non-car images (only two classes need to be identified). It was found that they were of shape (64,64,3) and images were reasonable balanced across the dataset. This means that classifiers will not be biased towards the majority represented class.

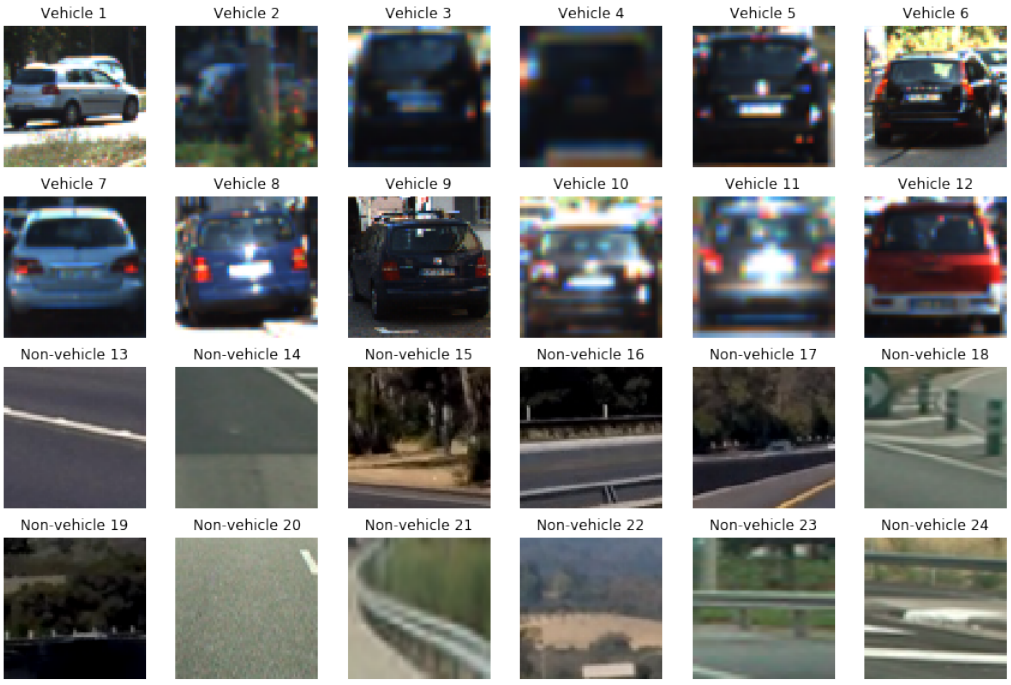

Figure 4: Initial Dataset

As can be seen in Figure 4 there are too many features/variables present which cannot be successfully or holistically represented and hence need to be worked on.

2.3 Feature Extraction

There are many different types, shapes, and colors of cars in real-life which cannot all be combined in a single dataset. Furthermore, a single car can appear differently based on viewpoint, orientation and time-of-day. For this reason, template-matching where a source

image is searched to find an exact match of the template "car" is infeasible, and will often fail. To successfully account for different possibilities of cars, an approach focusing on color-data and structural-data is implemented. A colorspace is chosen such that for different cars, there is a significant difference between cars, sections of road, and the environment. After manual iterations, *YCrCb* was selected for this variance. The image is split into its three sub-channels to obtain a histogram consisting of color data and then recombined.Spatial binning is also performed where a given image is blurred considerably, but still can be recognized as a car, and then flattened. These two features together form the color-representation of cars.

To represent structural data, *Histogram-of-Oriented-Gradient (HOG)* features are implemented. This is extremely useful in object recognition and creates a signature of occurrences of gradient and orientations in small sections of the image. The local object appearance/shape within an image can be found using distribution of intensity gradients and edge directions. For better accuracy, block-normalization (where intensity is measured across a larger area) is performed to remove effects of illumination and shadows.

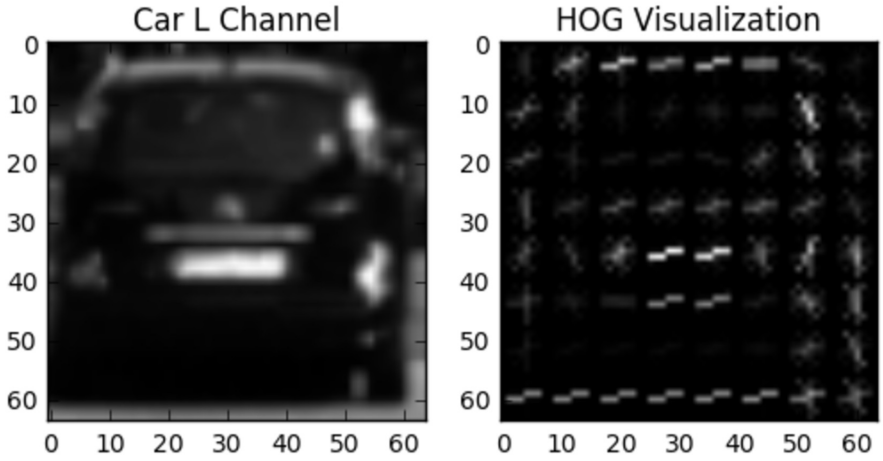

Figure 5: HOG Representations

Furthermore, all three channels of the image are used for creating the HOG structural feature, which is then concatenated with the color-information and flattened. This is the dataset that was used to train any implemented classifiers.

Following feature extraction, *Scikit-Learn* was used to split the dataset into train- and test-sets to be implemented on the classifier in a 0.8-0.2 ratio. The dataset was analyzed again and it was seen that there were 6156 features, 14,208 images in the train set and 3552 images in the test set. The classes were also nearly equally balanced across both train (49.61%-50.39%) and test (49.1%-50.9%), which is extremely satisfactory.

To prevent any one feature from dominating the rest of the features in the dataset, Scikit-Learn's *StandardScaler* was used to normalize the data by fitting and transforming it.

To reduce the number of features, a Scikit-learn pipeline was used to pass *GridSearchCV* into a *Principal-Components Analysis* algorithm and then into an initial random classifier. PCA (used in Eigenfaces for instance), uses an orthogonal transformation to create a set of

linearly uncorrelated variables. It also helps in removing unnecessary features and brings out latent features (combinations of initial features) and helps to summarize the dataset. There is a slight loss of information with this dimension reduction step, but it is acceptable. GridSearchCV was used to loop through an array of possible variations of number of features of the dataset that would provide optimum results (high accuracy) while minimizing data-loss.

2.4 Classifier

The choice of classifier was of vital importance to this pipeline. Linear Support-Vector Machines, Kernel Support Vector-Machines, AdaBoost, and an Ensemble were looked into. The Linear SVM (without a kernel change) works very well with HOG features, and provides 99.32% accuracy in 18.4 s, which is highly suited to our use. The Ensemble-algorithm consisted of two LeNet-5 architectures created using Keras, which was then wrapped using Scikit-learn's *KerasClassifier* to obtain a *VotingClassifier* of two neural networks that were run over 25 epochs and had a batch size of 100. This was definitely overkill, but was done for its own sake. Finally, a trained Single-Shot Multibox Detector Neural Network was implemented as well. This had the additional benefit of creating tighter bounding-polytopes and performing real-time detection compared to offline detection that the machine-learning classifiers implemented.

2.5 Tracking Functionality

Given a augmented dataset and a trained classifier, the next step of the pipeline is to identify and track vehicles that pass into the field of vision of the camera. The first step was to create a Region of Interest of an image frame. This ROI has the most likelihood of being occupied with cars and is basically the lower-half of the image.

A sliding-window algorithm was then implemented on this narrowed region to generate the list of all possible valid windows which can be occupied by cars. This is then fed into the $detect - cars$ function that performs the same feature manipulations as those done for the dataset, and is fed into the classifier. Since the classifier outputs binary values (two classes, with "1" label for car detected), windows that result in a label of "1" are stored as valid bounding boxes. This list is even more constrained by obtaining the probability the classifier associates with a particular image-label pair, and setting a lower threshold to those that can be accepted.

Since the classifier isn't perfect, there are instances where false positives or multiple detections are obtained. False positives are areas which are flagged as having a car, despite being a patch of road. Multiple detections are instances when the same car object has multiple, overlapping detections. To combine multiple detections and remove false positives, a heatmap is constructed to depict "hot" windows which are very likely sections of the image that contains a car. Furthermore, thresholding is done to the heatmap to remove false positives even more. The threshold value is different for the image and the video pipeline. The final step of the pipeline is to draw a cohesive bounding box that tracks the car as it moves across the frame.

To make this more robust on video frames, heatmaps are smoothed across 5 frames to get an average heatmap and remove outliers. The images that result from this pipeline can be seen in Figures 6, 7, and 8.

Figure 6: Bounding Boxes

Figure 7: Heatmap

Figure 8: Thresholded Heatmap

A further feature that was implemented (inspired from the calculations in Project Four) was an estimate of the distance of the detected vehicle from the car camera. This was done by knowing that the car camera was at the center of the image, bounding-box coordinates of the detected vehicle, and an estimate of pixel-to-meter ratio based on US Highway information.

3 Results & Improvements

This project implements the classical approach of computer vision and machine learning for object detection.As can be seen, it successfully detect vehicles in an image and tracks this location over frames. It was also seen (especially in cars in the opposing lane of the highway) that partially observed vehicles were classified as non-cars. Finding a way to classify such objects, together with smoothing tracking, and implementing a real-time vehicle detection and tracking scheme would be possible improvements to the pipeline. The pipeline is being tested with a deep neural network such as YOLO and SSD (not within the scope of this project) to determine real-time performance. The video obtained from this project can be seen on Youtube.