Udacity Self-Driving Car Nanodegree
Term One: **Computer Vision & Deep Learning**
Project Three: **Behavioral Cloning**

Nishan Srishankar
Worcester Polytechnic Institute

2017

1 Introduction

This project deals with the creation of an end-to-end convolution neural network to allow a car to drive in a track using the driving simulator after seeing a few training laps. This is a prime example of behavioral cloning as the network learns to mimic and generalize the driving behavior of an user.

2 Methodology & Implementation

The methodology to bring about behavioral cloning is to capture a dataset of the user driving/manually controlling the car, processing and augmenting this image, and passing the image dataset as features to a CNN which tries to learn the respective steering angle labels. This is described in detail in the following subsections.

2.1 Data Capture

Data was obtained with the intent of using it in two different methods: in the first, both Track One (beach) and Track Two (jungle) are used to collect data whereas in the second method, only Track One data is captured and then tested on an "unseen" Track Two (to mimic a test set). The provided Udacity driving log was used as a starting point, which was then augmented using laps of forward-lap driving, reverse-lap driving, and recovery driving. Track One has a bias of left-turns hence using a reverse lap dataset helps the neural network to generalize better. Furthermore, recovery driving is useful for the neural network to learn how to successfully recover from perilous positions from the corner/edges of the road back to the center of the road.

The number of images used for training is increased by using images from the left- and right- cameras (in addition to the central cameras), albeit with a small bias added to the steering angle label. Furthermore, features that correspond to steering angles of 0 rad (straight driving), unacceptably high steering angles, extremely slow driving (speed < 5 mph) are discarded to prevent the Neural network from learning possibly dangerous driving. A further aspect which could be removed are images that have large corresponding braking values which points to reckless driving.

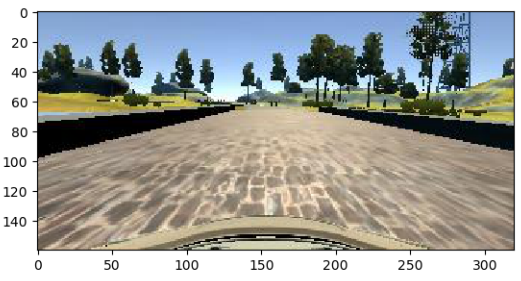

Figure 1: Initial Image obtained from the center camera

2.2 Data Augmentation

To generate more images, the initial dataset is randomly augmented using a combination of translations, vertical flips, brightness/contrast modifications, and corruption with Gaussian noise. Augmentation is helpful due to differences in Track One vs. Track Two; Track Two, for instance, consists of shadowy regions as well as regions where the road is in an incline or decline which is not seen in Track One. This is mitigated by translations and contrast modification. Vertical flips of images ensure that left, and right turns are equally represented in the dataset. The addition of noise helps generalize the data a bit better.

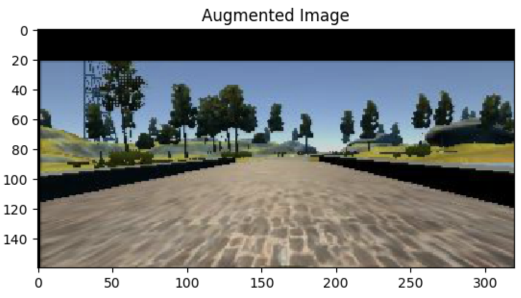

Figure 2: Slightly augmented image

2.3 Balancing Classes

As can be seen in Figure 3, some steering angle classes are extremely under-represented resulting in an unbalanced dataset. Two methods are used to balance the dataset. First, all images with a significant steering angle are vertically flipped (to mimic turning in an opposite direction). Secondly, classes are pruned such that to obtain a slightly more balanced dataset as can be seen in Figure 4.

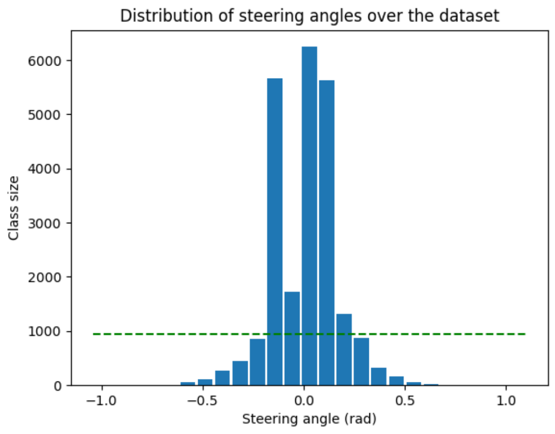

Figure 3: Example histogram of a sample dataset

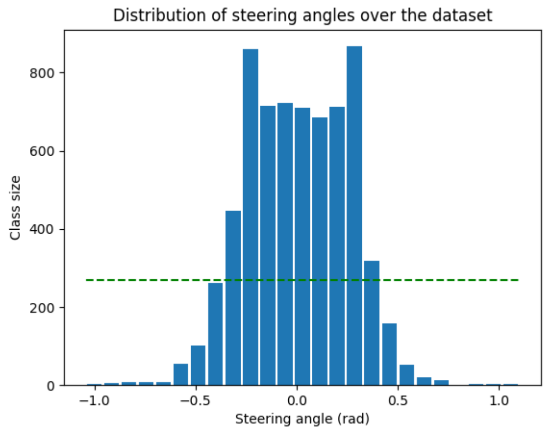

Figure 4: Example histogram of a balanced dataset

2.4 Preprocessing

Since Keras is utilized as a neural network library to run on top of Tensorflow, pre-processing is done in the first layer of the Sequential model. Preprocessing involves cropping out regions of the raw image as seen in Figure 5 that correspond to the sky or the hood of the car which are not useful to the architecture, and normalizing the image to have zero mean and a small standard deviation.

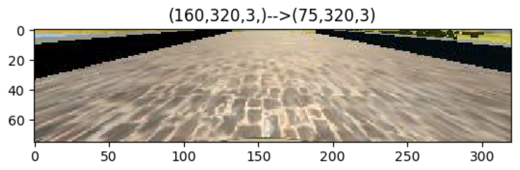

Figure 5: Cropped Image

2.5 Convolution Neural Network

The Convolution Neural Network that was used as a reference was NVIDIA's end-to-end convolution neural network (Figure 6) specifically designed for self-driving simulations. The CNN takes in input images in YUV space to map pixels from a front-facing camera to steering commands, and requires minimal training data from humans. This network learns to drive in traffic, on local roads with or without lane markings, and on highways. Furthermore, the network has around 27 million connections, and 250 thousand parameters though takes very little time (approximately 50 seconds) to complete an epoch.

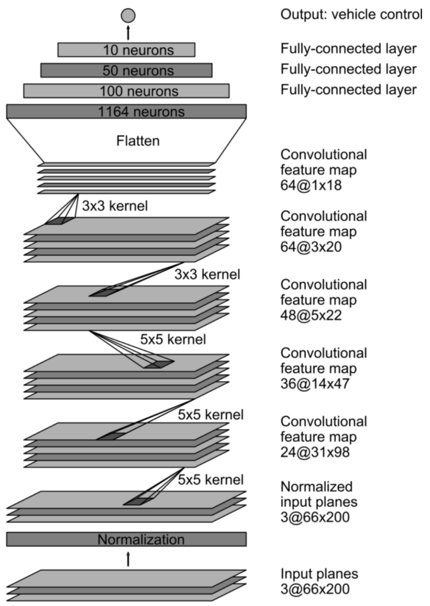

Figure 6: NVIDIA End-to-End CNN Architecture

The final CNN architecture used was streamlined slightly to reduce the number of parameters by sub-sampling/using strides greater than one, modifying/reducing output feature maps, and implementing dropout and Early-Stopping to generalize and prevent over-fitting.

ELU was utilized as a nonlinear activation instead of RELU as this helps in speeding up learning. An Adams optimizer was used instead of vanilla Gradient-Descent since this intrinsically implements learning-rate decay as well as momentum and is very useful for better convergence and to prevent overshooting. Furthermore, Mean-Square-Error (mse) was utilized as a loss metric instead of Softmax cross-entropy as this is a regression problem compared to the previous classification problems with Traffic Signs.

Table 1: Tuned Hyper-parameters

Hyper-parameter	**Detail**
Epochs	Varied depending on dataset used (approximately 3-10)
Batch Size	64
Learning Rate	1e-4
Early Stopping	validation loss
Early Stopping min threshold	1e-4
Early Stopping patience	3

2.6 Controller

A Proportional-Integral (PI) controller is used to set and control a desired speed (that is tinkered with). Additionally, to smoothen trajectories, a very basic implementation of momentum is used where the current steering angle is weighted with steering angle previously obtained. Throttle control is also achieved by modifying a multiplier to the throttle obtained from the controller when the steering angle or speed goes above a specific threshold. Images fed into *drive.py* are converted from RGB to YUV space whereas in OpenCV, images are converted from BGR to YUV space. Modified controllers that run the .h5 file are either $mellow - drive.py$ or $modified - drive.py$.

3 Results & Improvements

The obtained video for driving on Track One can be seen on Youtube. The car remains on the center of the road on Track One for most of the path, but waits a little too long to turn near the dirt track (but still manages to nonetheless). The issue that was seen with Track Two (jungle track) was that there were sections of the track where the car was effectively 'blind' and couldn't see the next section of the road- this was the case during either sharp curves or steep inclines followed by sharp curves. In this case, it stands to reason that cropping of the raw image (before input into the Neural Network) needs to be a bit more conservative or non-existent.

The performance of this pipeline could be better improved by using pre-trained weights from a different architecture (transfer learning) which is then fit to our application or by using either a driving wheel or a joystick to better manipulate the car during manual mode and generate smoother steering and hence better steering angles.