جامعة السلطان قابوس
Sultan Qaboos University

Implementing an end-to-end deep learning-based approach for self-driving

Al-Hassan. Al-Shareedah
Department of Computer Science
Sultan Qaboos University
s132186@squ.edu.om

Abstract

In this project, we propose a new implementation for autonomous safe driving and steering manoeuver by utilizing Convolutional Neural Network (CNN). This approach is based on Nvidia self-driving system[2], where they presented an end-to-end learning approach that optimizes all processing steps such as road features. Unlike the traditional approach that explicitly decomposes the autonomous driving problem into technical -human selected- components such as lane detection[1], by contrast, the end-to-end model can directly steer the vehicle from the front view camera data after training. The model architecture was modified to include normalization and dropout layers, input data was generated through Udacity self-driving simulator. Generated images and steering angle had to be pre-processed. The final implementation was successful in training the data however, the accuracy of the result was affected by the steering angle bias. A suggestion was made to include a bias parameter that discards any entry below the threshold for better data distribution.

1 Introduction

Adoption of autonomous vehicles in the near future is expected to reduce the number of road accidents and improve road safety [1]. Behavior prediction is one of the main functions of autonomous systems and it mainly revolves around understanding the past -and the future- state of the vehicle's nearby environment. This enhances their awareness of imminent hazards. Most recently, deep learning approaches have become popular due to their promising performance in more complex environments compared to conventional approaches [1]. One of the necessary tasks of autonomous systems is classifying the surrounding non-moving objects in the environment for example road lines, road signs, road layout with relation to the environment (rain, snow, sun reflection), and so on. Traditional approaches in self-driving CNN's divided the tasks into several parts such as lane detection and path planning. This approach is highly affected by feature extraction and interpretation of images and often the data is manually defined, therefore the features and rules are not optimal. Errors can also accumulate from the previous stage to the next stage and impact the overall accuracy. In [2], the performance of the end-to-end approach has demonstrated that CNN can take raw images as input and output control signals automatically. The model self-optimizes based on training data without any defined rules. In this project, a new implementation is presented to enhance the architecture proposed in [2], by modifying the model and image pre-processing.

COMP6114: Deep Learning, Spring 2020, Sultan Qaboos University. (LateX template borrowed from NIPS 2017.)

2 Related work

The project is based on Bojarski's Nvidia paper published in 2016 [2], where they developed a data collection system. The system would use images from the three mounted cameras each labeled in correspondence to their physical location (Left, Center, Right). The images are then fed to the processing module that comprises from multiple GPUs, steering, throttling, and brake paddle position are also captured through Controller Area Network (CAN). CAN-Bus is used to transfer information between Engine Control Unit (ECU) and other component of the vehicle and is standardized since 1993 and is accessed through On-Board-Diagnostic port (OBD-II) which is standard as well since 1996. Bojarski [3], further explained that the system eliminates the need for hand-coding and instead learns by observing. The steering command is captured as $1/r$ instead of r this is to prevent singularity and make the transition from left turn (negative values) to right turns (positive values) smooth. Figure 1 shows a block diagram of the training system used. The network architecture consists of 9 layers (5 convolutional, 3 fully connected layers), the convolutional layers are responsible for feature extraction while the fully connected layers produce output value which is the inverse of turning radius.

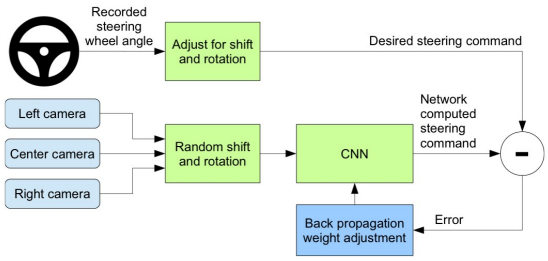

Figure 1: Block Diagram of the system

Farag, W, and Saleh, Z [4] presented "BCNet" system that is a deep learning, convolutional, seventeen layer system that expanded the architecture proposed by Bojarski's PilotNet system. Using Adam optimization and adding four dropout layers to prevent overfitting.

3 Dataset and Features

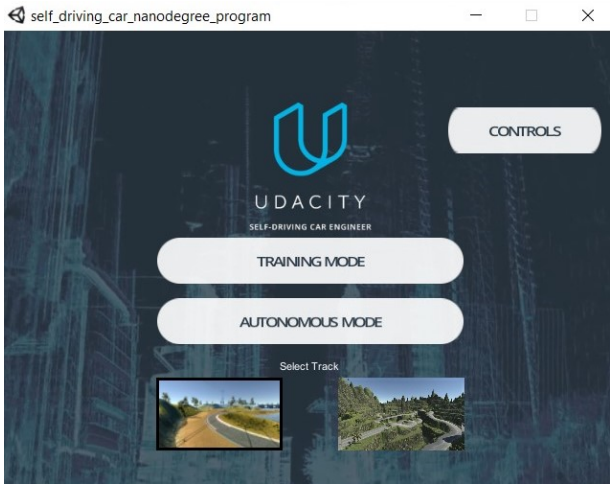

Figure 2: Main screen of Udacity simulation

The main source of input dataset was simulator generated data (Udacity driving simulator [5]) shown in figure 2. The recorded data set consists of 7,071 images that are divided equally between center, left, and right images, an example is shown in figure 3. Each image is 320x160 pixels with 3 channels for RGB colors. The data index is stored in a CSV file with 2,357 lines of record. The record is

divided between image index (center, left, anxd right) in addition to steering, throttle, brake, and speed data corresponding with each image.

The data is generated by driving the car in Udacity simulator for 2 laps in track 1 with safe driving. It is also encouraged to include recovery data, meaning that when approaching a turn, initially miss the starting turning point and then recover back the car safely to the correct path. This would give the model chance to learn recovery behavior. The data set is divided 80% training and 20% validation.

Figure 3: Left, Center and Right Camera Images Respectively from the Simulator Vehicle on Track 1

Before training the model, images had to be pre-processed to be useful and meaningful throughout the learning process. The first set of pre-processing is to crop the image and remove the sky and hood of the vehicle, then resize the image to fit the input of the model, images are normalized using lambda function which divides by 127.5 and subtract by 1. To implement a min-max scaling. We also have an augmentation function that randomly generates an augmented image and applies flipping, shifting, shadow, and brightness to improve the model performance. Finally, batch generator function that generates a batch of images and associate the corresponding paths and steering angles.

4 Learning Algorithm

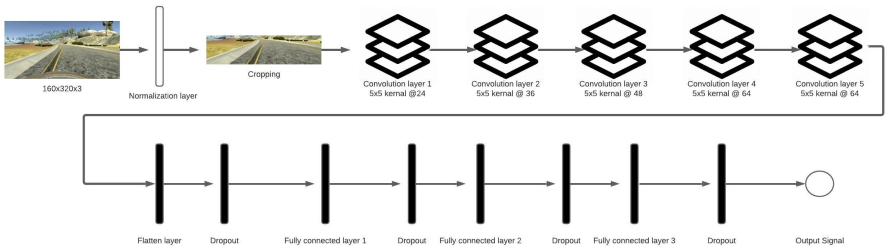

Figure 4: Model architecture

Similar to [3][4], the CNN architecture will contain a normalization layer, 5 convolution layers, and 3 fully connected layers. The model is coded using Keras on top of Tensorflow. Since this is a regression problem no pooling layer is needed. The architecture of the model is shown in figures 5. The design of the network is based on the NVIDIA model[6] and Farag BCNet [4], using both resources I was able to focus on adding modifying the original code and align it closer to BCNet architecture. Earlier we have discussed modifications in the pre-processing stage, here we will present some of the modifications in the training stage. To begin, I have added lambda normalization layer before inputting the images. Figure 5 shows the architecture of the network, followed by a dropout layer.

I have also used ELU (exponential linear unit) activation which is more accurate but slower to calculate. Mean square error was used for loss function to measure how close the model predicts to the given steering angle for each image. The learning rate was set to 1.0e-4 by trial and found to be within the sweet spot of improvement speed.

5 Evaluation

The model was compiled with the parameter shown in figure 5, training took a day and a half on a relatively old GPU (AMD Radeon R6). After training the model can successfully drive the vehicle

on both tracks. The initial training with 7,071 images was found to be small and was replaced with another data set of size 33,066 images. drive.py which is the script to control the vehicle had to be updated to run. Images had to be transformed in an array with float32 data type. For the test run, the reader can refer to the video uploaded alongside this report for visual evaluation. After evaluating the model, the steering angle distribution in the pre-processing stage appears to be uneven since the vehicle skew in the opposite direction of the road turn direction. As shown in figure 6, we can see how the steering angle is biased towards 0 (straight) and negative values (left), this -along with the augmentation process- helped to escalate the problem.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lambda (Lambda)              (None, 66, 200, 3)        0
_________________________________________________________________
conv2d (Conv2D)              (None, 31, 98, 24)        1824
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 14, 47, 36)        21636
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 5, 22, 48)         43248
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 3, 20, 64)         27712
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 1, 18, 64)         36928
_________________________________________________________________
dropout (Dropout)            (None, 1, 18, 64)         0
_________________________________________________________________
flatten (Flatten)            (None, 1152)              0
_________________________________________________________________
dense (Dense)                (None, 100)               115300
_________________________________________________________________
dense_1 (Dense)              (None, 50)                5050
_________________________________________________________________
dense_2 (Dense)              (None, 10)                510
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 11
=================================================================
Total params: 252,219
Trainable params: 252,219
Non-trainable params: 0
```

Figure 5: Model structure

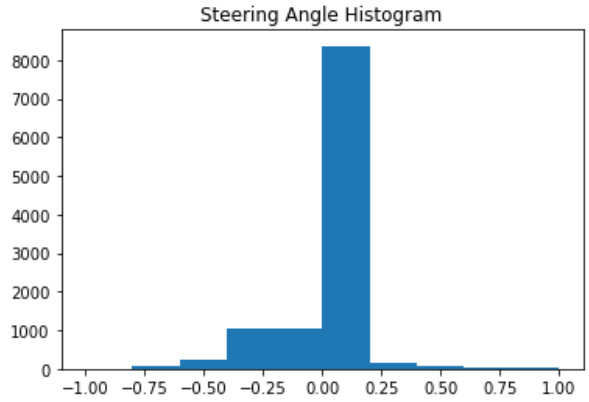

Figure 6: Model structure

This issue was addressed in [7], by adding a parameter “bias” that is between the range [0, 1] in the batch generator that mitigates the bias towards zero. For every example loaded from the training set, a random threshold r=np.random.rand() is computed. If steering(i) + bias < r then the example “ i “ is discarded from the batch. This way we can ground the batch data more evenly as shown in figure 7. This solution would be tested if not for the time constrain of this project since model run time takes up to a day and a half.

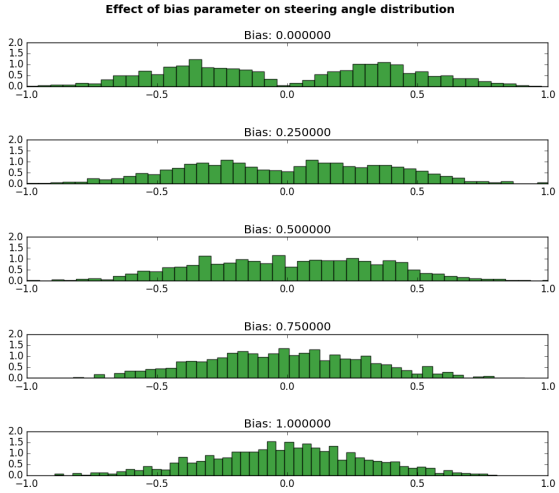

Figure 7: Model structure

6 Conclusion/Future Work

In this project, I have successfully implemented an end-to-end self-driving convolutional model that was introduced by the Nvidia team. The data was generated through Udacity self-driving simulator by recording a 15-minute drive and generate 3 images (left, center, and right) with the corresponding throttle and steering positions. The data was preprocessed through various methods such as cropping, flipping, shifting, and augmentation. The model consisted of a normalization, five convolutional, and three fully connected layers. The model was trained twice, the first run had few images, and the second run revealed a data distribution issue where data was biased towards the straight and left turn. through search, the problem can be addressed by different methods one of which was proposed by adding a bias parameter in the batch generator in the pre-processing stage. In the future, I would like to expand my pre-processing technique and test the solution in [7], there's also the risk of overfitting since the Udacity simulator provides only one track which makes it difficult for the model to generalize. The model can also incorporate a more aggressive L1 regularization to address this issue.

Acknowledgement

Special thanks to user hdmetor[6] who provided the base code for my project and to Dr. Wael Farag, Zakaria Saleh their paper[4] was a fundamental resource for completing my project.

References

[1] Mozaffari, S. & O., Dianati & P., Mouzakitis (2020) Deep Learning-Based Vehicle Behavior Prediction for Autonomous Driving Applications: A Review. *IEEE Transactions On Intelligent Transportation Systems*, pp. 1-15. https://doi.org/10.1109/tits.2020.3012034.

[2] M. Bojarski & D. D. Testa & D. Dworakowski, & B. Firner, & B. Flepp, & P. Goyal, & L. D. Jackel, & M. Monfort, & U. Muller, & J. Zhang, & X. Zhang, & J. Zhao, & and K. Zieba. (2016) End to end learning for self-driving cars.

[3] M. Bojarski, & P. Yeres, & A. Choromanska, & K. Choromanski, & B. Firner, &L. D. Jackel, & and U. Muller.Explaining (2017) how a deep neural network trained with end-to-end learning steers a car. CoRR,abs/1704.07911,. URL: https://arxiv.org/abs/1704.07911

[4] W. Farag and Z. Saleh, "Behavior Cloning for Autonomous Driving using Convolutional Neural Networks", 2018 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT), 2018. Available: 10.1109/3ict.2018.8855753 [Accessed 14 December 2020].

[5] Udacity Simulator (2017) [online] https://github.com/udacity/self-driving-car-sim (accessed 26th November 2020).

[6] hdmetor/Nvidia-SelfDriving [online] https://github.com/hdmetor/Nvidia-SelfDriving (accessed 29th November 2020).

[7] ndrplz/ self-driving-car [online] https://github.com/ndrplz/self-driving-car/tree/master/project_3_behavioral_cloning (accessed 14th of December 2020).