Self-Driving Car using Behavioural Cloning

Sukanya Saha

I. Introduction

Autonomous driving is a challenging yet rewarding topic to work on. In this project, I looked into different domains and technologies in the literature and finally found a suitable simulator to generate the data and a promising neural network architecture to train a self-driving car based on that simulator environment.

The motivation behind this project was to train a self-driving car using behavioral cloning in the set environments. For training and testing datasets, and Udacity's self-driving car simulator. For training the model I relied on NVIDIA's End-to-End Learning for Self-Driving Cars architecture [4] and this reference [1] for the simplified version of the network.

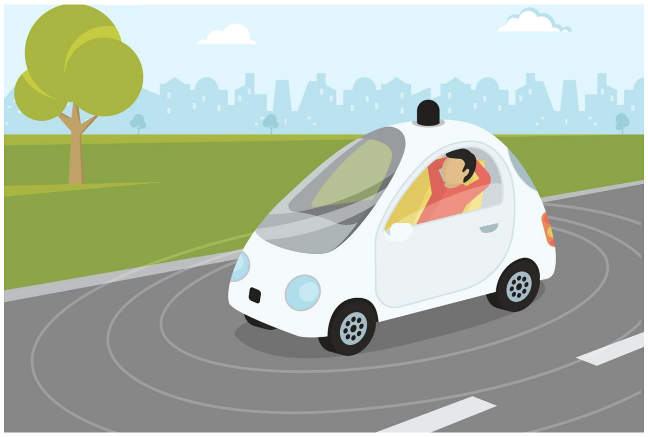


Fig. 1. Self-Driving Car [3]

II. Related Work

There has been a tremendous amount of work done in this field and it continues to expand. I focused on a few ground-breaking pieces of research in this area. One such major contribution is made by the NVIDIA paper[4]. The paper focused on training a convolutional neural network (CNN) to map raw pixels from a single front-facing camera directly to steering commands. Here the idea was to train the model to replicate human driving on local roads with or without lane markings and on highways. The paper claimed that their autonomous cars could operate in areas with unclear visual guidance such as in parking lots and on unpaved roads. They achieved this by processing all steps such as lane marking detection, path planning, and control, our end-to-end system, etc. simultaneously by training a model with maximum system performance.

The paper[4] advocated a network architecture with 9 layers, including a normalization layer, 5 convolutional layers, and 3 fully connected layers. They trained the weights of our network to minimize the mean squared error between the steering command output by the network and the command of either the human driver or the adjusted steering command for off-center and rotated images. They also added hard-coded normalization in the first layer which allowed the normalization scheme to be altered with the network architecture and to be accelerated via GPU processing. The convolutional layers were designed to perform feature extraction and were chosen empirically through a series of experiments that varied layer configurations. They used a striding window of 2x2 stride in the first three layers and a kernel of size 5x5. The last two convolution layers had 3x3 kernel size and no striding. The fully connected layers are designed to function as a controller for steering. They achieved great results through this model. However, this sophisticated architecture would be excessive for the dataset I am working with. Hence, I went ahead with a simpler network of fewer convolution layers as suggested here [1].

III. Method

A. Dataset

For the dataset, I relied on Udacity's self-driving car simulator which gave me the flexibility to generate the dataset conveniently and also to test the model in the same environment. This simulator automatically created two folders containing training data for the lake track and jungle track. I mainly focused on the Lake track. I drove with the keyboard for around 5 - 6 laps after a lot of trial and error. The simulator automatically created a folder containing the images and CSV files with the following metadata: center camera, left camera, right camera, steering angle, throttle, reverse and speed. The steering angle ranges from -1 to 1.

B. Data Augmentation

After generating the images, I added a few standard data augmentation techniques such as flipping, cropping, etc.

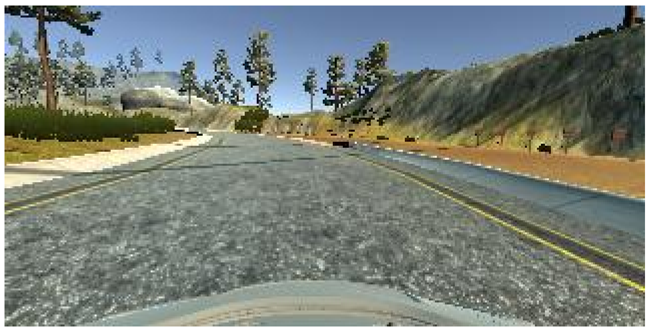


Fig. 2. Data Augmentation - Original Image

1) Flipping: For this step, I used the generated images, specifically the center camera images, and flipped them. Then I used the flipped counterpart with a negative steering angle of that of the original steering angle of the center image. This flipping of the images technique helped to balance the dataset so that model will not be biased towards any specific condition. This idea came from the reference medium blog-post [1].

2) Data Offset: Another technique that worked on this dataset is offsetting the steering angle to make the dataset more normally distributed. To do that I used an offset of 0.4 to the steering angle of the left camera images and -0.3 to the steering angle of the right camera images.

3) Cropping: The third data augmentation technique was to crop all the images and keep the original steering angel values. Since model training does not require surrounding details such as natural scenarios, sky, etc. This step helped to reduce the image ratio which helped with model training performance. To remove redundant details I cropped the image's 60 pixels from the top and 20 pixels from the bottom. This has been used in the reference medium blog-post too [1].

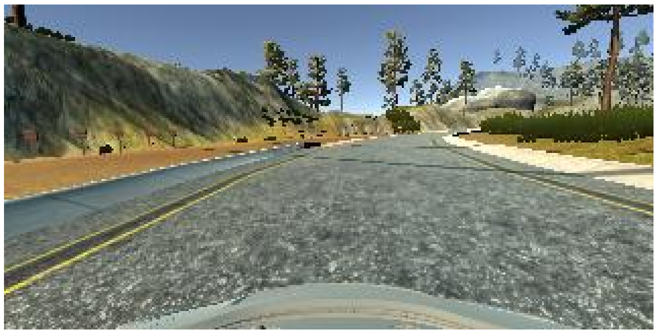


Fig. 3. Data Augmentation - Flipping

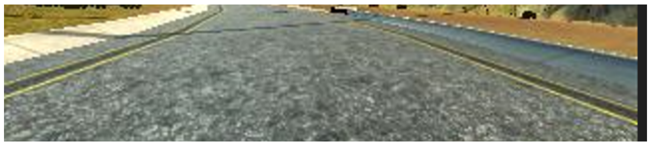


Fig. 4. Data Augmentation - Cropping

C. Architecture

This project is based on the idea that the predictor is a specific area cropped from the images and the label is the steering angle. Here the goal is to run the model on each simulator track till the finish line. The underlying architecture used here is based on the NVIDIA paper [4]

But I used the modified architecture mentioned in the reference [1]. For a car to identify steering angles, it is important to identify features like road edges. In CNN such high-level features are extracted at initial layers.

The modified architecture is based on this idea. Here, the layers have been reduced from the NVIDIA CNN model. As suggested in the reference, first I tried with three convolution layers and three dense layers. Then I tried with two convolution layers, and two dense layers and my final try was one convolution layer and one dense layer. With the last configuration, I was able to achieve a similar performance as my previous trials with a shorter training time of around 10-15 minutes.

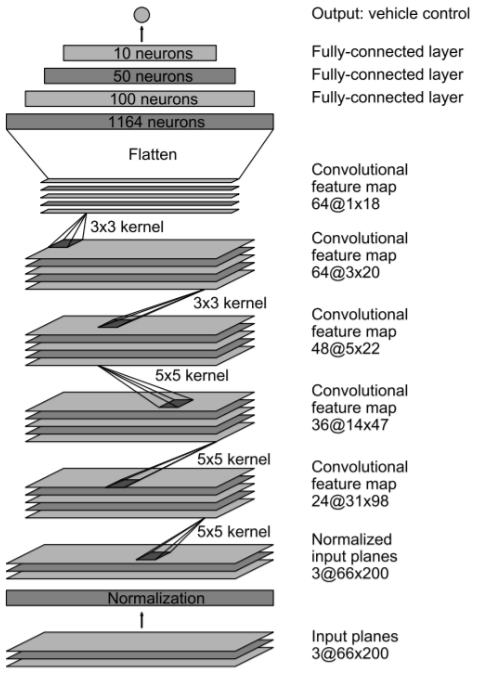


Fig. 5. NVIDIA architecture [4]

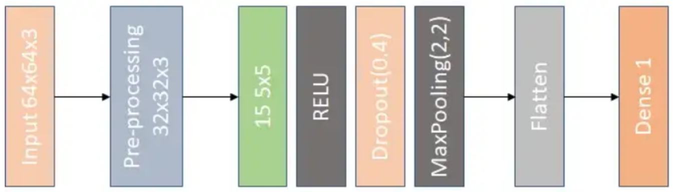


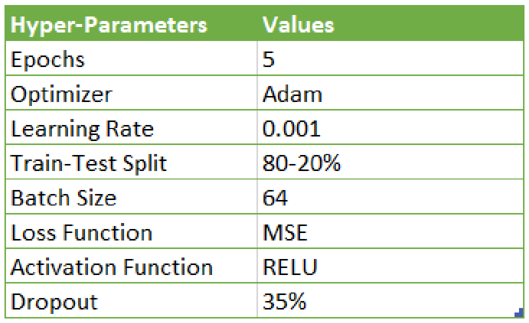
Fig. 6. Modified NVIDIA architecture [1]

The final model consists of a single convolution layer followed by a dropout layer, a max pooling layer, and a single dense layer as shown below.

1) Outcome: With the simplified version of the architecture, the accuracy remained similar to that of the NVIDIA model. Here are some of the hyper-parameter values that worked with the simplified version of the model-

For evaluation metrics, the NVIDIA paper mentions autonomy metrics based on how many times human intervention was required in addition to MSE (Mean Squared Error). However, for simplification purposes, I have used just the MSE of the predicted steering angle. The MSE for the simplified model reached 98% for train data and approximately 97% for the test data.

The most challenging part of this project was



Hyper-Parameters	Values
Epochs	5
Optimizer	Adam
Learning Rate	0.001
Train-Test Split	80-20%
Batch Size	64
Loss Function	MSE
Activation Function	RELU
Dropout	35%

Fig. 7. Final Hyper-Parameters [4]

to generate the data especially because I am not a gamer. I struggled a lot to complete a full lap without falling into the lake. Then, even though I was able to complete the lap it was wiggly and the model training on this dataset did not perform well. Finally, after perfecting the laps I generated the dataset which enhanced the model accuracy from 80% to more than 90%.

Finally, the model accomplished finishing the lap successfully without falling or needing any human intervention. The output video is available in the GitHub repo[2].

D. Conclusion

Self-driving is an ever-evolving field and I tried to get some hands-on in this area for which this project was appropriate. Even though there were initial challenges but finally the results were impressive. Future improvement in this project would be to enhance the model accuracy and model training efficiency. Another step would be to look into more recent papers and advanced simulators. A major drawback of the behavioral cloning approach in real-world applications is reliability and dependence on human driving skills. Since humans drive differently in various internal and external situations the dataset could be biased. Also, it lacks generalization based on dynamic objects or extreme weather conditions. On top of that, people around the world have varying driving standards based on crowds, local regulations, and traffic which demands a larger dataset with a good representation of all driving standards. In spite of the drawbacks and initial setbacks, overall, this project was a success for me to get started in the field with great potential for future enhancements.

REFERENCES

[1] Behavioral cloning: Make a car drive itself. https://towardsdatascience.com/my-learning-from-udacity-sdc-nanodegree-do-we-really-need-a-complex-cnn-and-hours-of-training-4f80e28af90b. Accessed: 2022-11-30.

[2] Github repo. https://github.com/sukanyasaha007/MS-Independent-Study-Fall-2022. Accessed: 2022-11-30.

[3] Self-driving car poster. https://cdn.vox-cdn.com. Accessed: 2022-11-25.

[4] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars, 2016.