Journal of Computing and Electronic Information Management
ISSN: 2413-1660 | Vol. 12, No. 1, 2024

Autonomous Car Behavioral Training Using Deep Neural Network

Jiayi Gao

Shanghai Polytechnic University, Shanghai 200433, China

Abstract: Autonomous driving is becoming increasingly prevalent nowadays. With the help of a number of images of car movement from the Kaggle self-driving dataset, we explore the feasibility of utilizing the images obtained to train a deep neural network to detect and predict the steering angle, which is the critical part of the car behavior. Since deep neural networks have emerged as powerful tools for training autonomous cars and learning about and improving their driving behaviors, we incorporate convolutional layers and additional layers in the deep neural network architecture so that it can capture the behaviors appropriately and provide effective results. We demonstrate that the implementation of this approach is successful and that the corresponding implementation highlights the potential of deep neural network in advancing autonomous car technology. Our comprehensive evaluation suggests that further research should concentrate on refining the network architecture and enhancing perception capabilities in order to deliver promising advances to the field.

Keywords: Deep learning, AI, Self-driving car, Model training; Neural network.

1. Introduction

With the advent of a new era focused on innovation, autonomous cars have revolutionized the automotive industry. Central to the advancement of autonomous driving is the ability to precisely perceive and interpret the surrounding environment, thus enabling a car to make informed decisions and navigate smoothly. One crucial aspect of autonomous car development is behavioral training, which involves training a car to respond appropriately under various driving circumstances to avoid jeopardizing the safety of pedestrians. Traditional approaches to behavioral training have leveraged rule-based systems, which often lack the adaptability and flexibility required by real-world real-time driving scenarios. Recent progress in deep learning technology, particularly in deep neutral networks, have shown promising results in enabling autonomous cars to learn and improve their driving behaviors. In this paper, we propose an improved approach to autonomous car behavioral training using deep neural networks. We aim at leveraging the power of deep learning algorithms to train cars to accurately perceive, interpret and react to different driving situations. By utilizing large amounts of training data, our approach enables a car to learn from a diverse range of scenarios and generalize its behavior.

This paper aims at contributing to the progress of autonomous car technology by leveraging improvements to the deep learning-based approach used for behavioral training thereby paving the way for safer and more efficient transportation systems. [1]

2. Related Work

Recently, autonomous driving has attracted the audience tremendously and the field of autonomous driving has witnessed notable progress. A pioneering piece of work is the paper titled NVIDIA End-to-End Learning for Self-Driving Cars (Bojarski et al., 2016). In this paper, the authors have trained convolutional network to learn the mapping between the input data and resulting steering angles. The network is trained on a large-scale dataset that is collected from actual driving episodes. Their results inspired many researchers in this domain.

Another prominent paper uses reinforcement learning techniques to train an agent to improve its autonomous driving abilities (Codevilla, Müller, López, Koltun, Dosovitskiy, 2017). In this paper, the authors combine a deep neural network with predictive control capabilities to enable agents (i.e., cars) to learn driving policies in a simulated environment. In this approach, the agents receive awards and penalties based on their actions.

Recent advances in multi-modal sensor fusion have also been leveraged to improve perception capabilities for cars. One of the studies (Huang, et al, 2022) uses a deep neural network architecture that integrates data from multiple sensors to improve object detection and tracking.

While a significant amount of research has already been done, there is still room for further research and development in terms of safety and real-world deployment of autonomous driving systems. [2]

3. Solution Design

In this section, we provide a quick overview of our solution and provide details about our data, neural network architecture, and training pipeline.

3.1. Overview

As depicted in Figure 1, our proposed autonomous driving simulator imports data in the form of comma delimited (.csv) files that contain images taken from different angles as determined by the location of the cameras on the vehicles. The autonomous driving simulator operates as a server that collects images as input data and processes them so as to generate a machine learning model that can be used by the clients (i.e., the cars). The models are expressed in the Python language (in the form of Jupyter notebooks). These Jupyter notebooks implement machine learning models using a deep neural network algorithm. The models are trained using TensorFlow in the backend and Keras in the frontend. Keras is able to specify model components sequentially to help structure the various network layers. In our approach, we

downloaded the Udacity simulator and used it to provide available data. Different Python libraries are required to support the deployment and training environments. [3]

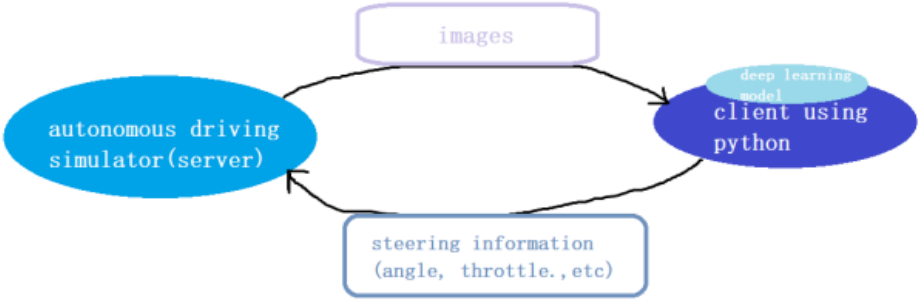

Figure 1. Solution Design Overview

Furthermore, to achieve accurate and reliable autonomous driving capabilities, the combination of a dataset, a deep neural network architecture and a training pipeline is required as documented below.

3.2. Dataset Selection and Preprocessing:

Our solution leverages the self-driving dataset provided by Udacity (https://www.kaggle.com/datasets/aslanahmedov/self-driving-carbehavioural-cloning). This dataset offers a collection of real-world driving images and functions as the foundation for training our deep neural network.

Additionally, the dataset is preprocessed to ensure that it is appropriate for training purpose. Pre-processing involves normalization and augmentation to increase the quantity of imported samples. At times, specific labels need to be added to indicate the desired driving behavior.

3.3. Deep Neural Network Architecture:

Our solution leverages a deep neural network architecture specifically tailored for automated car behavioral training. It is composed of multiple layers, including convolutional layers for extracting the features and time-distributed layers. In a nutshell, this neural network takes different factors, like the number of layers and the size of filters, into account to optimize the ability of the network to simulate sophisticated driving behaviors. [4]

3.4. Training pipeline

Our training pipeline feeds the dataset to our deep neural network for training purpose. Techniques like batch normalization, dropout and regularization are involved to avoid overfitting and improve the generalization capabilities of the network. Additionally, the training process requires a loss function such as value loss to detect and quantify errors.[5]

4. Solution Implementation

First of all, to implement our solution successfully, we import the “NumPy”, “matplotlib”, “keras”, “cv2”, “pandas” and “random” library in our Python program. We then extract the driving log (.csv) file from the data directory renaming it as “Udacity-self-driving-car-behavioral-cloning”. This file also includes many other images available to us. We define seven (7) features as columns: center, left, right, steering, throttle, reverse, speed. The “data.head()” displays the first five values of the .csv file to ensure that data is extracted appropriately. Since the path to the data is specific to a local machine, it needs to be accessible to the network. We use a “path_leaf” function to assign center, left and right accordingly. Because odd numbers tend to show center distribution and we want to visualize the quantity of steering samples, we set the number bins to 27 and keep samples per bin at 400. We then utilize the histogram function to present results intuitively. After that, we print the bin and the plot, as shown in Figure 2.

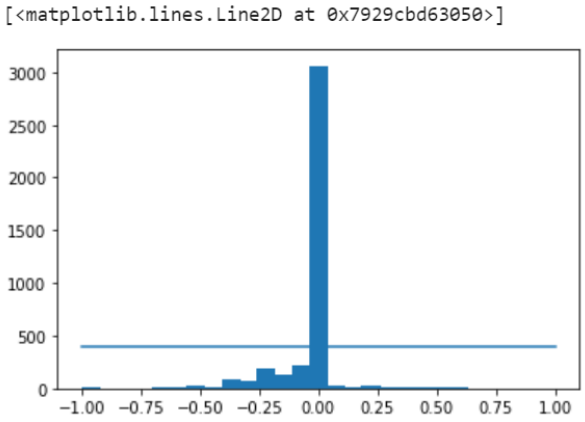

Figure 2. Print Diagram 1

Figure 2 shows that the data for steering angles is centered along “0.00” and that steering angles data in other scopes is sporadic (i.e., the line equals 400). We then output the total data that is 3930. Next, we specify a variable, which makes it possible to remove unwanted data samples. By iterating on the steering data, the output of the distribution of steering angles becomes more unified and the possibility of driving straight all the time can be ruled out. After this step, there are 1269 steering samples remaining and the plot also changes as shown in Figure 3 below.

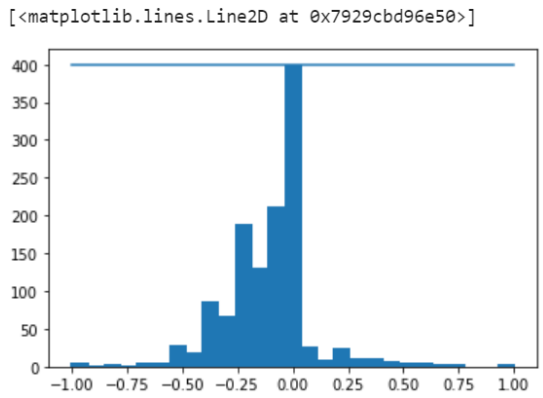

Figure 3. Print Diagram 2

We then load the images from the dataset directory into the

array to append the steering data from center, left and right images. Next, we use separate paths for images and storage arrays in order to obtain 3045 training samples and 762 valid test samples, as is demonstrated in Figure 4.

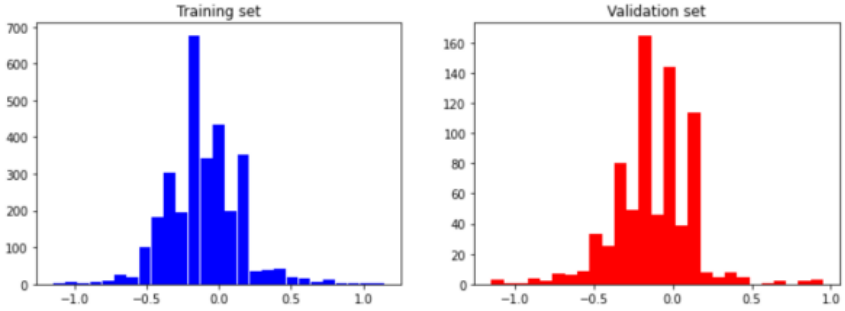

Figure 4. Comparison Of Training Set And Validation Set

The most complex problem is that of generalizing the behavior of self-driving cars on different tracks under various circumstances. To solve this problem, we use the technique of preprocessing and augmentation on images. For example, the zoomed image and shifted image. Because the external environment of a particular image cannot determine the output, the image can be tailored. And it can be zoomed at 30% as illustrated in Figure 5 below.

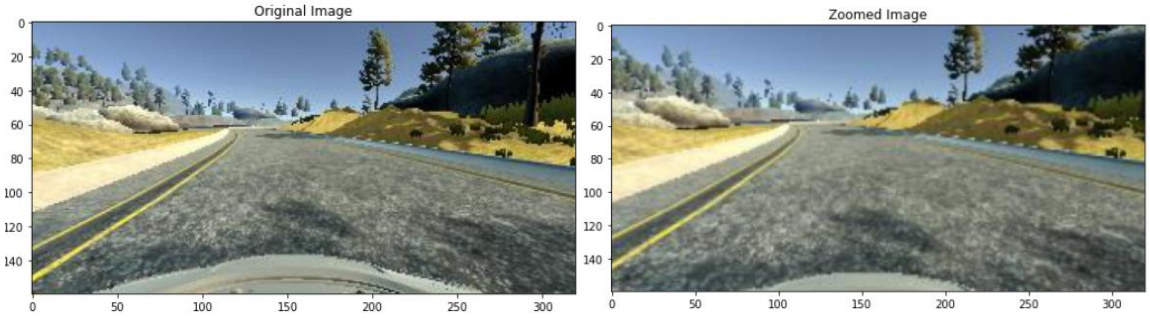

Figure 5. Original Image And Zoomed Image Of One Particular Image

The shifted image can have horizontal and vertical shift, and so it becomes a panned image, as is shown in Figure 6.

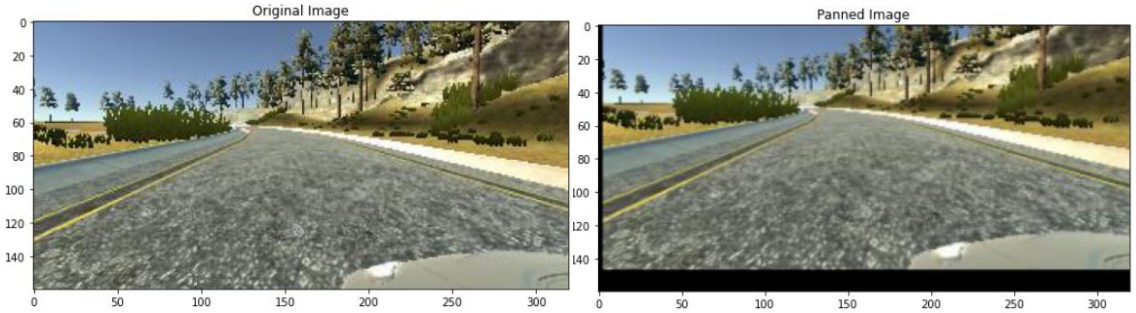

Figure 6. Original Image And Panned Image Of One Particular Image

Then, brightness-adapted augmentation proves to be suitable as it comes to generalizing to various weather conditions, such as cloudy or rainy weather as illustrated in Figure 7 below.

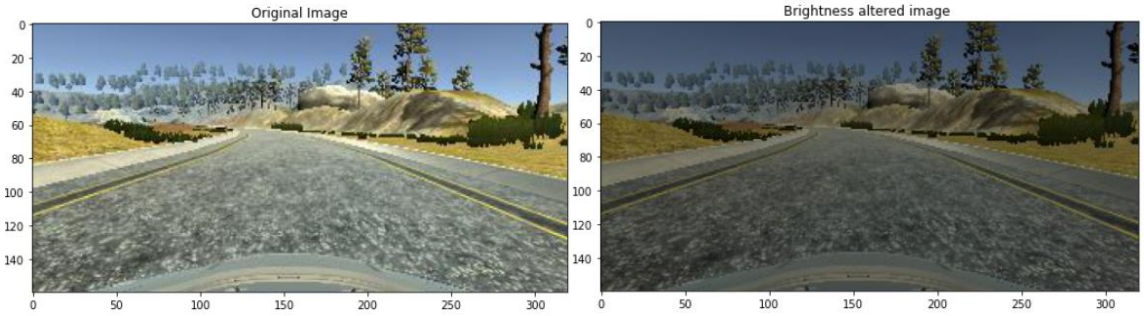

Figure 7. Original Image And Brightness Altered Image Of One Particular Image

Next, we consider the flipped image because the model can be trained on both left turns and right turns and we set the steering angle to 0.15 as illustrated in Figure 8 below.

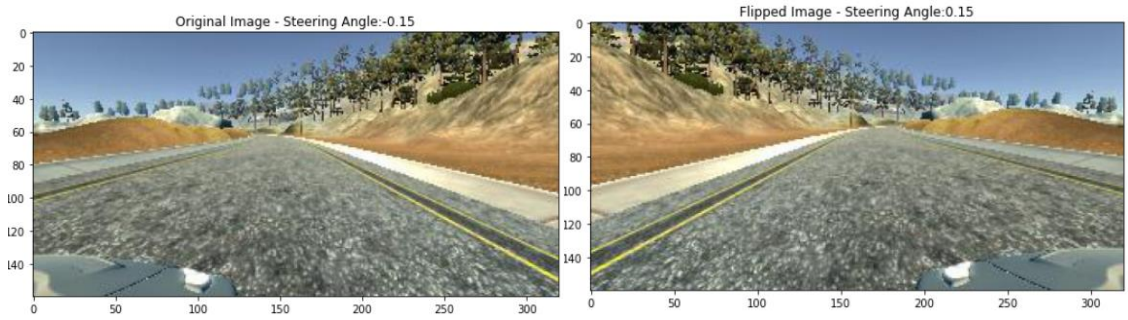

Figure 8. Original Image And Flipped Image Of 0.15 Steering Angle

After that, we process some images by removing irrelevant features. Figure 9 illustrates the effect.

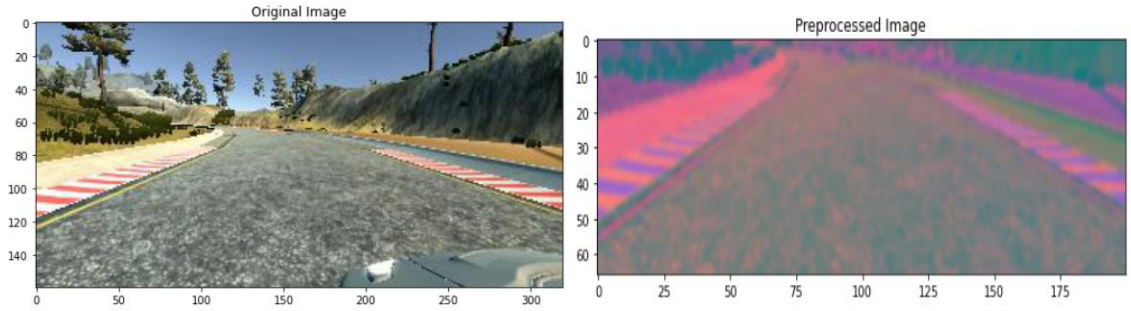

Figure 9. Original Image And Preprocessed Image Of One Particular Image

To set up the models to train the client using Python and generate the neural network, we need to follow various configuration steps. We use sequential models on Keras with DNN layers to train the data. We then set the epochs (i.e., number of iterations) to ten, set batch size (i.e. number of image samples that pass through the network) to 100, and set learning rate to 0.0001 because it reflects the coefficient change in the network. [6]

Besides all of this, it is clear that combinations of layers can be utilized to better implement the neural network. To that effect, we introduce a more advanced model provided by NVIDIA and make adjustments to the model to prevent overfitting. We then define the model structure by defining the model objects. In particular, we add the convolutional layer and we apply sub-sampling. Because it is the first layer and the images are large, we set horizontal movement and vertical movement to 2 pixels and stride length accordingly. Also, the activation function “elu” is needed and the inside shape is defined as (66,200,3). The second layer is quite similar, with 36 filters and kernel size (5,5). Because there are three more layers in the CNN, we remove the sub sampling from the 4th and 5th layer. We then add a flatten layer to convert the output layer from the convolutional layer obtained before to a one-dimensional array. Next, we compile the model and use low learning rate to increase accuracy. The dropout layer can set the input of random value to zero for every update. So we can generate the training data because the model can learn from the same data. We also have to separate the convolutional layer from the fully connected layer. Now, we can refine the training process by importing training data x_train and y_train, which requires more epochs to be effective. Figure 10 demonstrates the detailed information of training parameters. [7]

Layer (type)	Output Shape	Param #
conv2d_5 (Conv2D)	(None, 31, 98, 24)	1824
conv2d_6 (Conv2D)	(None, 14, 47, 36)	21636
conv2d_7 (Conv2D)	(None, 5, 22, 48)	43248
conv2d_8 (Conv2D)	(None, 3, 20, 64)	27712
conv2d_9 (Conv2D)	(None, 1, 18, 64)	36928
dropout_4 (Dropout)	(None, 1, 18, 64)	0
flatten_1 (Flatten)	(None, 1152)	0
dense_4 (Dense)	(None, 100)	115300
dropout_5 (Dropout)	(None, 100)	0
dense_5 (Dense)	(None, 50)	5050
dropout_6 (Dropout)	(None, 50)	0
dense_6 (Dense)	(None, 10)	510
dropout_7 (Dropout)	(None, 10)	0
dense_7 (Dense)	(None, 1)	11

Total params: 252,219
Trainable params: 252,219
Non-trainable params: 0

Figure 10. Layer Type, Output Shape And Parameters Of Training

5. Results

Our architecture can be evaluated by value loss because there is loss in each epoch of the training. Furthermore, Keras can provide val_loss that means average loss after one epoch. Because we set the epoch to ten, Figure 11 below shows the detailed loss and val_loss during every epoch.

```
Epoch 1/10
300/300 [==============================] - 301s 1s/step - loss: 0.1281 - val_loss: 0.0511
Epoch 2/10
300/300 [==============================] - 295s 986ms/step - loss: 0.0666 - val_loss: 0.04
66
Epoch 3/10
300/300 [==============================] - 292s 975ms/step - loss: 0.0647 - val_loss: 0.04
50
Epoch 4/10
300/300 [==============================] - 305s 1s/step - loss: 0.0602 - val_loss: 0.0399
Epoch 5/10
300/300 [==============================] - 303s 1s/step - loss: 0.0597 - val_loss: 0.0404
Epoch 6/10
300/300 [==============================] - 289s 964ms/step - loss: 0.0580 - val_loss: 0.04
08
Epoch 7/10
300/300 [==============================] - 281s 940ms/step - loss: 0.0564 - val_loss: 0.03
97
Epoch 8/10
300/300 [==============================] - 283s 947ms/step - loss: 0.0549 - val_loss: 0.03
90
Epoch 9/10
300/300 [==============================] - 288s 961ms/step - loss: 0.0551 - val_loss: 0.03
96
Epoch 10/10
300/300 [==============================] - 298s 997ms/step - loss: 0.0548 - val_loss: 0.03
96
```

Figure 11. The First Round Of Loss And Val_loss During Every Epoch

To better understand the loss, we compare the training and validation by presenting both loss in a line chart as illustrated in Figure 12 below.

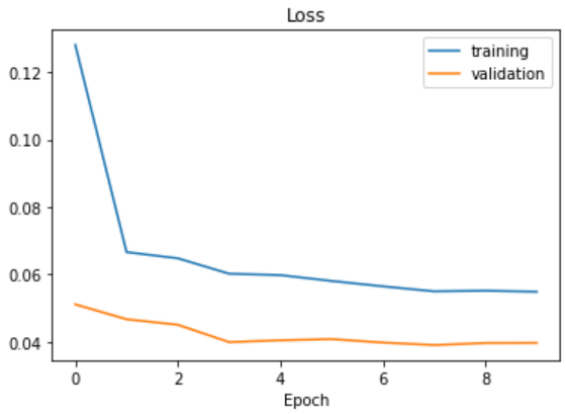

Figure 12. The Loss Comparison Of Training And Validation During Every Epoch In Round One

From Figure 12 above, the value loss in training plummets from epoch zero to epoch one but subsequently shows a gradual decline from epoch two to epoch ten. However, the loss in validation remains relatively stable during every epoch. To make this result more persuasive, we ran the model an extra three rounds as shown in Figure 13, 15 and 17 below:

```
Epoch 1/10
300/300 [==============================] - 311s 1s/step - loss: 0.0924 - val_loss: 0.0474
Epoch 2/10
300/300 [==============================] - 314s 1s/step - loss: 0.0628 - val_loss: 0.0427
Epoch 3/10
300/300 [==============================] - 325s 1s/step - loss: 0.0599 - val_loss: 0.0401
Epoch 4/10
300/300 [==============================] - 322s 1s/step - loss: 0.0574 - val_loss: 0.0436
Epoch 5/10
300/300 [==============================] - 299s 999ms/step - loss: 0.0560 - val_loss: 0.03
97
Epoch 6/10
300/300 [==============================] - 297s 992ms/step - loss: 0.0561 - val_loss: 0.04
00
Epoch 7/10
300/300 [==============================] - 295s 985ms/step - loss: 0.0554 - val_loss: 0.03
98
Epoch 8/10
300/300 [==============================] - 305s 1s/step - loss: 0.0533 - val_loss: 0.0439
Epoch 9/10
300/300 [==============================] - 306s 1s/step - loss: 0.0539 - val_loss: 0.0382
Epoch 10/10
300/300 [==============================] - 318s 1s/step - loss: 0.0527 - val_loss: 0.0352
```

Figure 13. The Second Round Of Loss And Val_loss During Every Epoch

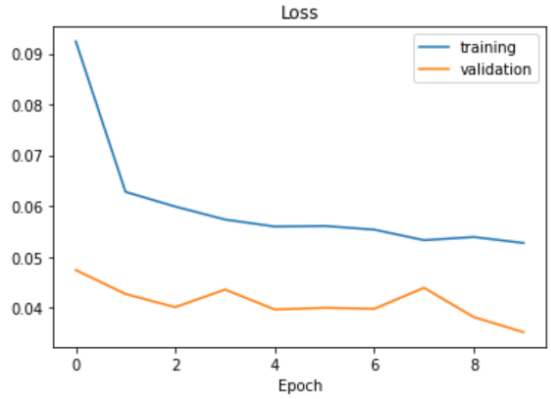

Figure 14. The Loss Comparison Of Training And Validation During Every Epoch In Round Two

```
Epoch 1/10
300/300 [==============================] - 311s 1s/step - loss: 0.1144 - val_loss: 0.0458
Epoch 2/10
300/300 [==============================] - 277s 927ms/step - loss: 0.0677 - val_loss: 0.03
86
Epoch 3/10
300/300 [==============================] - 273s 910ms/step - loss: 0.0619 - val_loss: 0.03
76
Epoch 4/10
300/300 [==============================] - 266s 887ms/step - loss: 0.0608 - val_loss: 0.03
62
Epoch 5/10
300/300 [==============================] - 262s 875ms/step - loss: 0.0610 - val_loss: 0.03
84
Epoch 6/10
300/300 [==============================] - 265s 885ms/step - loss: 0.0564 - val_loss: 0.03
48
Epoch 7/10
300/300 [==============================] - 259s 867ms/step - loss: 0.0577 - val_loss: 0.04
03
Epoch 8/10
300/300 [==============================] - 262s 876ms/step - loss: 0.0573 - val_loss: 0.03
29
Epoch 9/10
300/300 [==============================] - 263s 879ms/step - loss: 0.0554 - val_loss: 0.03
55
Epoch 10/10
300/300 [==============================] - 265s 884ms/step - loss: 0.0561 - val_loss: 0.03
52
```

Figure 15. The Third Round Of Loss And Val_loss During Every Epoch

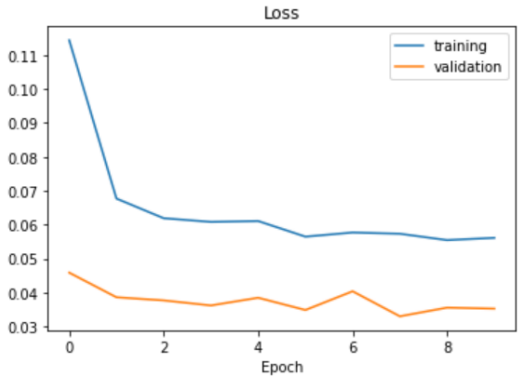

Figure 16. The Loss Comparison Of Training And Validation During Every Epoch In Round Three

```
Epoch 1/10
300/300 [==============================] - 274s 909ms/step - loss: 0.1553 - val_loss: 0.05
11
Epoch 2/10
300/300 [==============================] - 259s 866ms/step - loss: 0.0705 - val_loss: 0.04
26
Epoch 3/10
300/300 [==============================] - 260s 867ms/step - loss: 0.0649 - val_loss: 0.04
40
Epoch 4/10
300/300 [==============================] - 262s 876ms/step - loss: 0.0643 - val_loss: 0.03
95
Epoch 5/10
300/300 [==============================] - 257s 860ms/step - loss: 0.0615 - val_loss: 0.03
78
Epoch 6/10
300/300 [==============================] - 259s 864ms/step - loss: 0.0605 - val_loss: 0.03
64
Epoch 7/10
300/300 [==============================] - 271s 905ms/step - loss: 0.0591 - val_loss: 0.03
51
Epoch 8/10
300/300 [==============================] - 262s 876ms/step - loss: 0.0586 - val_loss: 0.03
57
Epoch 9/10
300/300 [==============================] - 268s 896ms/step - loss: 0.0578 - val_loss: 0.03
47
Epoch 10/10
300/300 [==============================] - 266s 890ms/step - loss: 0.0584 - val_loss: 0.03
49
```

Figure 17. The Fourth Round Of Loss And Val_loss During Every Epoch

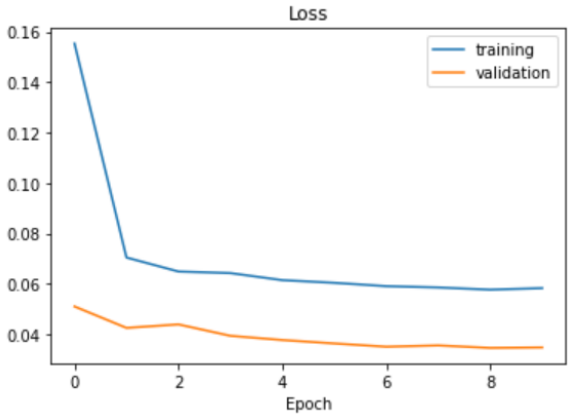

Figure 18. The Loss Comparison Of Training And Validation During Every Epoch In Round Four

Although there may be little discrepancy during every round, the loss of every epoch is holistically parallel, as is presented in Figure 14, 16 and 18. Thus, this model that uses deep neural network is appropriate to use as part of an autonomous behavioral training process.

6. Conclusion

We have presented a comprehensive approach for autonomous behavioral training using a deep neural network while leveraging a self-driving dataset. We also implemented and tested our deep neural network to enable a vehicle to learn and improve its behavior.

This paper is based on the dataset obtained on Kaggle under the field of autonomous vehicles (namely self-driving car on Kaggle). By using the Kaggle self-driving car dataset, we are able to train the deep neural network on a wide range of real-world driving scenes, which can be conducive to behavioral prediction in future. The deep neural network architecture really plays a vital part in analyzing complex driving behaviors. The training pipeline that incorporates optimization algorithms and techniques like batch normalization and dropout ensures efficient machine learning. Additionally, through a comprehensive evaluation and testing results, we have shown the effectiveness and reliability of our trained deep neural network model. Our model shows high accuracy, responsiveness and adaptability.

One of the shortcomings in our approach is that we have only focused on and implemented the “steering” behavior of the car, which is not the only factor to consider for behavioral training. Future research direction should include refining the deep neural network architecture, exploring advanced optimization algorithms and incorporating extra sensor data from many parts of the car (such as input from a radar), so as to enhance the car’s perception abilities, thus creating a better environment for self-driving deployment. Finally, investigating techniques for continuous learning and adjustment to dynamic driving conditions is an important next step to ensure that cars can periodically improve their driving behavior.

References

[1] Bojarski M ,Testa D D ,Dworakowski D , et al.End to End Learning for Self-Driving Cars. [J]. CoRR, 2016, abs / 1604. 07316.

[2] Codevilla F ,Matthias Müller, Dosovitskiy A ,et al.End-to-end Driving via Conditional Imitation Learning[J]. 2017.

DOI:10.1109/icra.2018.8460487.

[3] Huang K , Shi B , Li X ,et al.Multi-modal Sensor Fusion for Auto Driving Perception: A Survey[J]. 2022.

DOI:10.48550/arXiv.2202.02703.

[4] Song X , Wang P , Zhou D ,et al.ApolloCar3D: A Large 3D Car Instance Understanding Benchmark for Autonomous Driving [J].IEEE, 2018.

DOI:10.1109/CVPR.2019.00560.

[5] Upamanyu K , Ramaswamy I P .The Effects of Image Augmentation on Efficiency of a Convolutional Neural Network of a Self-Driving Car[J].rScroll, 2021(2).

DOI:10.47611/JSRHS.V10I2.1486.

[6] Smolyakov M V , Frolov A I , Volkov V N ,et al.Self-Driving Car Steering Angle Prediction Based On Deep Neural Network An Example Of CarND Udacity Simulator[C]//Advanced Industrial Conference on Telecommunications. IEEE, 2018.

DOI:10.1109/icaict.2018.8747006.

[7] Wang D ,Wen J ,Wang Y , et al.End-to-End Self-Driving Using Deep Neural Networks with Multi-auxiliary Tasks [J]. Automotive Innovation,2019,2(2):127-136.