Towards Autonomous Driving System Using Behavioral Cloning Approach

Sakilam Abhiman
Department of Computer Science and Engineering (AIE)
Amrita School of Computing
Amrita Vishwa Vidyapeetham, Chennai.
ch.en.u4aie20055@ch.students.amrita.edu

B Natarajan
Department of Computer Science and Engineering(AIE)
Amrita School of Computing
Amrita Vishwa Vidyapeetham, Chennai.
rec.natarajan@gmail.com

M.Umapathy
Department of AIML
CMR Institute of Technology,
Hyderabad.
uma_bass0707@yahoo.com

Kamaljit kaur Bhagwat
Department of Mathematics
Chandigarh university
gurnoorbhagwat@gmail.com

Sudhanushu Dubey
Department of Mathematics
Chandigarh university
sudhanshusdebey@gmail.com

Saurabh Rana
Department of Mathematics(SCSET)
Bennett University
saurabhranapsm@gmail.com

***Abstract*—The automotive industry has been focused on developing autonomous driving technology, which has been a subject of research for many years. Recent advancements in Convolutional Neural Networks (CNN) have shown remarkable performance in computer vision tasks, including autonomous driving systems. Behavioral cloning is a widely adopted technique in autonomous vehicle systems, that involves training a model to replicate the driving behavior of human drivers. This paper presents research on implementing behavioral cloning using a CNN architecture for an autonomous vehicle system in the Udacity self-driving car simulator environment. Our approach employs a custom-designed CNN Model to learn driving behavior from a dataset of images captured from the simulator, and we have applied various data augmentation techniques to the dataset to enhance the model's performance. This model has a total of 27 layers, including convolutional layers, normalization layers, activation layers, dropout layers, flatten layers, and dense layers. We evaluate the model's performance on a metric, called mean square loss error and demonstrate that our approach outperforms previous works in this domain. The results of this research underscore the effectiveness of using behavioral cloning and data augmentation techniques for autonomous vehicle systems.**

***Index Terms*—Autonomous driving system, Custom designed CNN model, Convolutional Neural networks (CNN), Udacity, Behavioral Cloning**

I. Introduction

The field of autonomous learning has been gaining increasing attention in recent years, with self-driving cars at the forefront of this focus. Major automobile and tech companies have been investing significant resources in developing self-driving cars, with Google being one of the leaders in this field with a strong foundation in artificial intelligence.The closest to practical usage are Google's self-driving cars, which have been tested on the road for more than 3.2 million kilometers and had two of them as early as June 2015[14]. Another business that has made great strides in the creation of self-driving automobiles is Tesla. It was the first business to commercialize self-driving technology, and in recent years, its autopilot technology has made important strides. One of the key aspects that will determine the success of self-driving cars is their ability to learn human sub-cognitive skills. Behavioral cloning, which involves teaching the car to replicate the driving behavior of a human driver, is one of the methods being used to achieve this goal. Many deep learning-based behavioral cloning techniques have been developed recently for self-driving cars; however, gathering the necessary data to train these deep networks can be difficult. Moreover, testing these models in a real-world setting is difficult since it requires validation of the model under different weather, climate, and topographic conditions. Researchers face the challenges of collecting datasets for such scenarios and also testing them physically.

The research teams creating autonomous car prototypes encounter the same difficulties, but this might pave the way for more productive studies, such the development of command and control systems and innovative sensor combinations like LiDARs. Petabytes of data are generated daily by test fleets throughout the world, and various teams operating at once must analyze, sample, and use this data. This is another barrier to the development of autonomous cars. Every modification made throughout the development cycle could result in more

design iterations. Because it allows researchers to quickly and safely evaluate the advantages and disadvantages of different algorithms without creating a threat to human life in the event of model failure, simulation is still the best approach to overcoming these issues.

The creation of simulation testbeds for the trained models, which are quicker, less costly, and significantly more insightful than a conventional physical prototype, is required to further current research on self-driving automobiles. But creating a real-time simulation environment for autonomous vehicles presents intriguing computational difficulties that might further research into effective algorithm development. The article examines the issue that simulator datasets inevitably have and suggests a solution. Better outcomes are obtained when using end-to-end learning architectures with the processed dataset.

II. Related Works

Autonomous driving has seen significant advancements with the deep learning techniques, and behavioral cloning is one of the prominent areas of research in this field. However, researchers have attempted to improve CNN-based models for autonomous driving by exploring various approaches, such as attention models and the use of non-linear activation functions [1]. The quality of training data is crucial for CNN-based models, and researchers have proposed various methods as it suits well in many real time applications [20][21]. For instance, Khan et al. [2] proposed a novel approach to mitigate the issue of zero-biased steering angles in self-driving simulator datasets, which involves generating additional steering angles using Gaussian noise.

Researchers have investigated numerous learning strategies in the aim of fully autonomous driving. For instance, the construction of SDR, a self-driving automobile system, included reinforcement learning behavioral cloning [4][18]. Deep-learning-based networks were created to improve lane following skills [5][19].

Le Mero et al.'s evaluation of imitation learning methods for fully autonomous cars included the most recent developments in this field [6]. In addition, Ferencz and Zöldy demonstrated how deep learning techniques may be used to accomplish end-to-end lateral control [7]. The usage of CNNs for this purpose has been extensively investigated, and the Nvidia model [8] is frequently used as a baseline architecture because of its performance in the DARPA Grand Challenge. To enhance performance and interpretability, attention processes have also been integrated into behavioural cloning models [9]. While behavior cloning has shown great promise, researchers have also explored its limitations. The availability of open-source simulators such as the Udacity Simulator [10] and training datasets such as the Udacity Sample Training Data [11] have made it easier for researchers to experiment with these techniques.

III. Proposed System

Udacity has built a simulator for self-driving cars and made it open source for the enthusiasts, so they can work on something close to a real-time environment and it was developed by Udacity in collaboration with Nvidia and Unity. Udacity is based on the Unity engine and provides a basic simulation environment for testing and developing self-driving car algorithms. The Udacity features three cameras (left, center, and right) that are mounted at the front of the car and serve as the primary source of data for perception. In addition, the simulator provides information on speed, brake, throttle, and steering angle. It may be more suitable for beginners or those seeking a basic understanding of self-driving car simulation. The simulator is highly user-friendly and has a changeable resolution and control configuration. Depending on the user and computer setups, the graphics and input configurations can be altered. The Play button is pressed by the user to launch the simulator's user interface. To explore the keyboard controls, which are quite reminiscent of those in a racing game, go to the Controls tab.

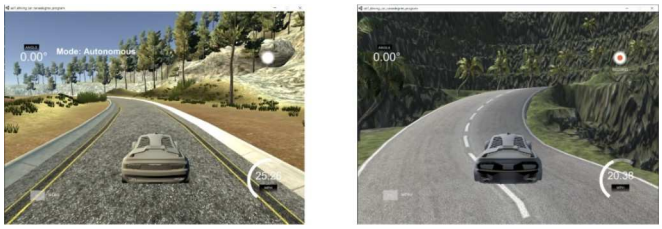

Fig. 1. Track 1 is on the right side, Track 2 is on the left side

In the driving simulator, there are two driving modes: (1) Training mode and (2) Autonomous mode. There is a choice provided to record your run while also obtaining the training dataset in training mode. The automobile is being driven in training mode, as shown by the little red symbol at the top right of the screen. The models may be tested in autonomous mode to determine if they can navigate the course without assistance from a person. Additionally, the car will quickly alert you that it has switched to manual controls if you attempt to push the buttons to put it back on the road. Once we have mastered using the keyboard to drive the automobile in the simulator, we begin by pressing the record button to begin gathering data. The gathered data will be saved in a designated folder on our system.

IV. Proposed Work

These models are configured to train the python's client and deliver the neural network outputs needed to drive the simulator car. Numerous trials and parameter changes were done in order to get the best setup. Although every model had an own set of traits and responded to every change differently. In order to do this, we built sequential models using Keras and trained the model with training data.

Furthermore, many epochs were tested, however these just helped to increase the model's overfitness. In other words, while the model does a great job of identifying details in the training data, its inability to adapt to the new dataset eventually stands in the way of its success. The most important

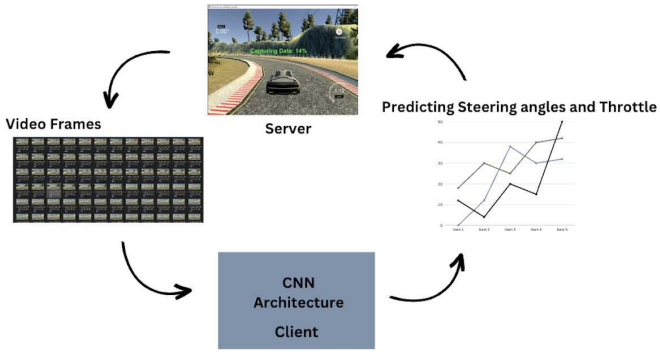

Fig. 2. Architecture for Implementation

prerequisite for every project is a dataset. This research does not use any open-source datasets; instead, a original dataset is made utilising the simulator.

A. Collecting Data

In the Udacity self-driving car project, there are actually three camera sensors mounted on the front of the car, which are used to capture images from different perspectives:

- Center camera: This camera is pointed straight ahead and captures the center of the road. It is the primary source of information for lane detection and tracking.
- Left camera: This camera is pointed slightly to the left of center and captures the left side of the road. It is used to detect vehicles and obstacles in the left lane.
- Right camera: This camera is pointed slightly to the right of center and captures the right side of the road. It is used to detect vehicles and obstacles in the right lane.

The recording of the driving in simulator is converted into image frames. The data is collected using these 3 cameras mentioned above. Using these images te steering angle, throttle and speed is determined. All the information about the images and parameters which are obtained are stored in a csv file, which we will be getting after stoping the recording of the simulation.

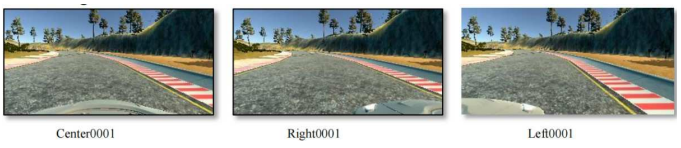

Fig. 3. Images captured by left, center and right sensor

B. Dataset

The dataset used for training the proposed Nvidia model was collected using an open-source driving simulator called the Udacity driving simulator.The data includes recovery data, which records the process of steering the car back to the centre

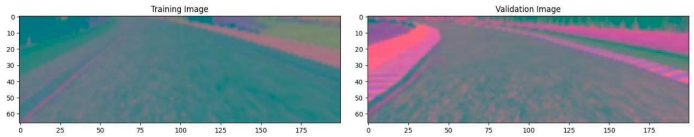

Fig. 4. Images from Training and Validation dataset

of the track after approaching the edge, allowing the model to learn recovery behaviour. The data was generated by manually driving the car around Track-1 in the simulator around four times with safe driving behaviour. A 3553-record csv file with the dataset index is used. The dataset size, after unzipping, is 140MB. The dataset consists of 3553 images captured from the center, left, and right cameras of the car, each having a resolution of 160x320 pixels with RGB color channels. Further the duplicates have been removed from the dataset.

center	left	right	steering	throttle	brake	speed
IMG/center_2016_12_01_13_	IMG/left_2016_12_01_	IMG/right_2016_12_0	0	0	0	22.14829
IMG/center_2016_12_01_13_	IMG/left_2016_12_01_	IMG/right_2016_12_0	0	0	0	21.87963

Fig. 5. Driving log generated after capturing data

The researchers have developed various subroutines to visualize and analyze the data, which help in identifying any potential flaws in the preprocessing stage. Any flaws in the data can lead to confusion in the model, which could result in poor performance. The dataset is split into two distinct portions: the training data, which makes up 80% of the total data, and the validation data, which comprises the remaining 20% which is into set of 2,820 and 705 images respectively

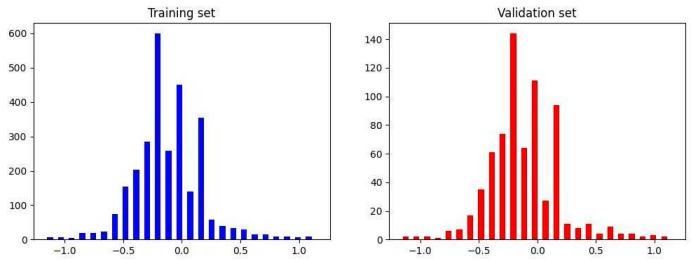

Fig. 6. Training set and Validation set

C. Data Augmentation

The data augmentation techniques are used to enhance the diversity of a dataset without collecting new data. Data augmentation is a widely used technique in machine learning to increase the size of a dataset by applying various transformations to the existing images. We have augmented the data using seven functions that apply different image transformations to the dataset - Zoom, Pan, Random Brightness, Shadow Effects, Shear Transformation, Random Transformation and Random Flip. These functions utilize the Affine transformation to apply random scaling, translation, brightness, and flip to the input images. The purpose of these transformations is to generate new images that are similar to the original ones but have some

variations that can help improve the model's performance. Used functions are stated below:

- Zooming in during data augmentation allows the model to focus on smaller details within the image, enhancing its ability to recognize objects of varying sizes, where we can observe in the Fig. 7.
- When horizontal flip augmentation is used, the picture is mirrored along the vertical axis, which is very useful when object orientation is not critical, as seen in Fig. 8.
- Panning, or image translation, involves shifting the image horizontally and/or vertically, creating new samples that expose the model to objects appearing at different positions within the image, where we can observe in the Fig. 9.
- Brightness adjustment augmentation modifies the pixel values to increase or decrease the overall brightness of the image, improving the model's robustness to variations in lighting conditions, where we can observe in the Fig. 10.
- Shadows image augmentation is an approach employed to introduce diversity within an image by incorporating simulated shadow effects, where we can observe in the Fig. 11.
- Jitter image augmentation is a method for introducing unpredictability and improving a picture's resilience. It entails subjecting the pictures to arbitrary transformations, as seen in Fig. 12. These changes include translation, rotation, scaling, and flipping.
- Shear image augmentation is used to create variations of an image by applying shear transformations. Shear transformation is a type of affine transformation that slants or tilts an image along a specified axis, where we can observe in the Fig. 13.

The random augment uses a combination of these transformations randomly with a 50% probability for each transformation. This generates a set of augmented images for a randomly selected image from the data set.

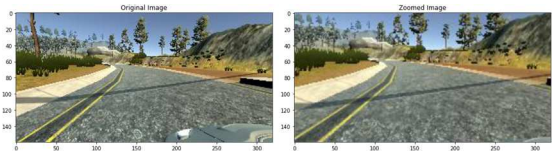

Fig. 7. Augmenting the original image by zooming

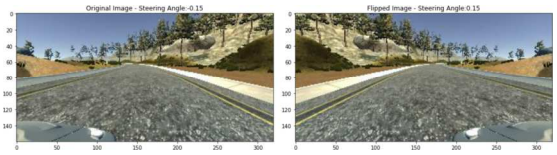

Fig. 8. Augmenting the original image by flipping

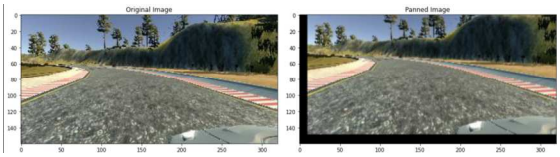

Fig. 9. Augmenting the original image by Panning

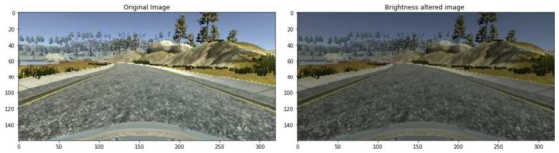

Fig. 10. Augmenting the original image by altering Brightness

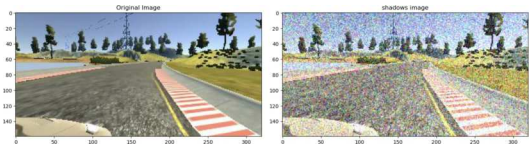

Fig. 11. Augmenting the original image by adding simulated shadow effects

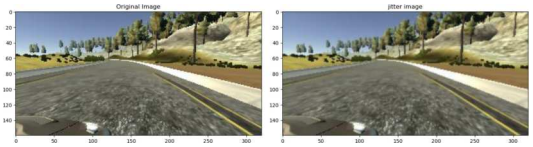

Fig. 12. Augmenting the original image by adding random transformations

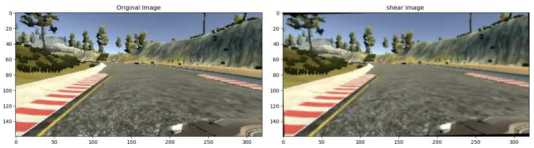

Fig. 13. Augmenting the original image by adding shear transformation

D. Data Pre-processing

In order to utilize the front camera's images in the training and validation data sets, it is necessary to perform pre-processing steps that can enhance the images and make them more suitable for the learning process. These steps aim to improve the training results and minimize the computation required. The pre-processing steps that were implemented are listed below in the order in which they were executed. Image pre-processing is an essential step in developing a self-driving car model as it allows the model to focus on the relevant information in the image while reducing noise and unnecessary data. In this particular implementation, the pre-processing function performs the following steps on an input image:

1) Crop the top 60 pixels and the bottom 25 pixels to select the portion of the image that contains the road.
2) Convert the RGB image to YUV color space, which

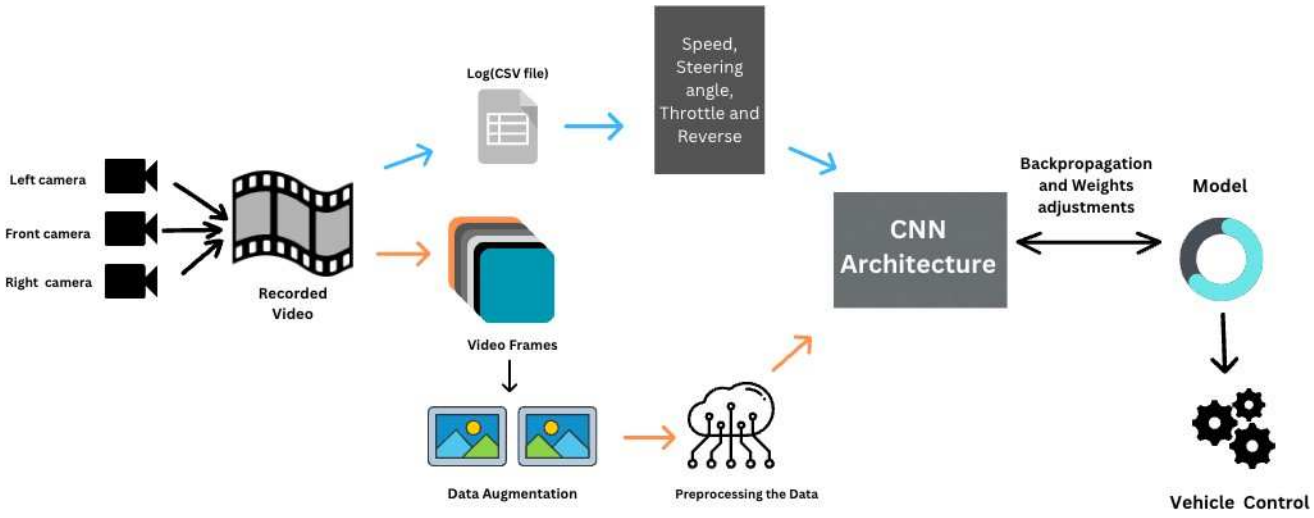

Fig. 14. Training the Neural Network

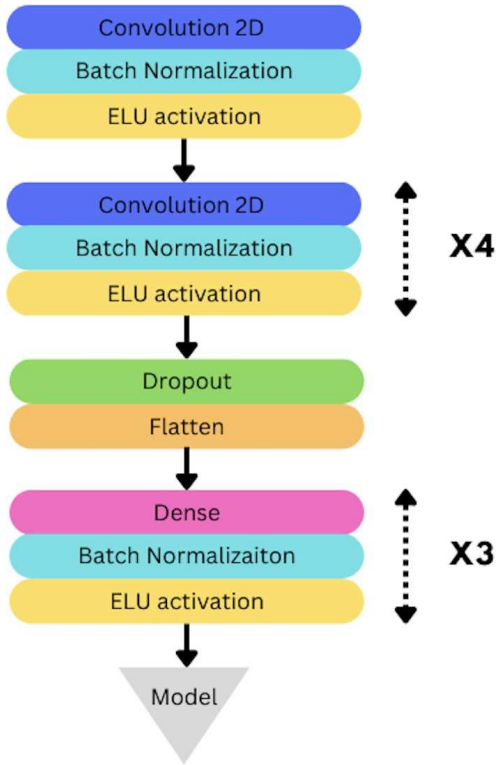

Fig. 15. Proposed CNN Architecture

separates the color information from the brightness information in the image.

3) Apply a Gaussian blur filter with a kernel size of 3x3 to the image to reduce noise and smooth the image.
4) Resize the image to a shape of 200x66 to fit the inputsize of the model.
5) Normalize the pixel values of the image to be in the range of 0 to 1.

```
Epoch 1/10
300/300 [==============================] - 111s 369ms/step - loss: 0.0800 - val_loss: 0.0595
Epoch 2/10
300/300 [==============================] - 117s 389ms/step - loss: 0.0534 - val_loss: 0.0446
Epoch 3/10
300/300 [==============================] - 129s 431ms/step - loss: 0.0483 - val_loss: 0.0531
Epoch 4/10
300/300 [==============================] - 119s 396ms/step - loss: 0.0461 - val_loss: 0.0341
Epoch 5/10
300/300 [==============================] - 126s 420ms/step - loss: 0.0437 - val_loss: 0.0368
Epoch 6/10
300/300 [==============================] - 142s 474ms/step - loss: 0.0424 - val_loss: 0.0300
Epoch 7/10
300/300 [==============================] - 143s 477ms/step - loss: 0.0406 - val_loss: 0.0357
Epoch 8/10
300/300 [==============================] - 136s 454ms/step - loss: 0.0406 - val_loss: 0.0295
Epoch 9/10
300/300 [==============================] - 135s 450ms/step - loss: 0.0397 - val_loss: 0.0306
Epoch 10/10
300/300 [==============================] - 138s 459ms/step - loss: 0.0393 - val_loss: 0.0336
```

Fig. 16. Loss over epochs for Model Architecture

These procedures produce a pre-processed image that only contains the pertinent data required for the self-driving car model to produce precise predictions. This method helps in identifying road aspects and make suitable driving judgements more accurately by deleting extraneous data and decreasing noise.

E. Network Architecture

The BCNet Model [1] and Nvidia Model [17], which have been successful in end-to-end self-driving tests, are the models that served as the basis for this project's network architecture. It is a deep convolutional neural network designed for supervised image classification. Convolutional layers, batch normalization, ELU [5] activation, dropout, and dense layers are all used in the architecture to extract key characteristics from input pictures and learn the intricate mapping to the steering angle.

Convolutional, activation, batch normalization, and dropout layers are among the 27 layers in the model, which is based on Nvidia's network architecture for self-driving cars. It consists

of 1 flatten layer, 4 fully connected layers, 5 convolutional layers, 8 activation layers, 8 batch normalization layers, and1 dropout layer. The model utilizes convolutional layers along with strides to downsample the data. These layers utilize learnable filters to perform convolutions on the input images, allowing the network to capture local spatial information and identify relevant features. Following each convolutional layer, batch normalization layers are added for training stability, and ELU activation functions add non-linearity to capture complicated patterns. Dropout regularization is used to mitigate overfitting, and a Flatten layer is applied to reshape the output of the convolutional layers.

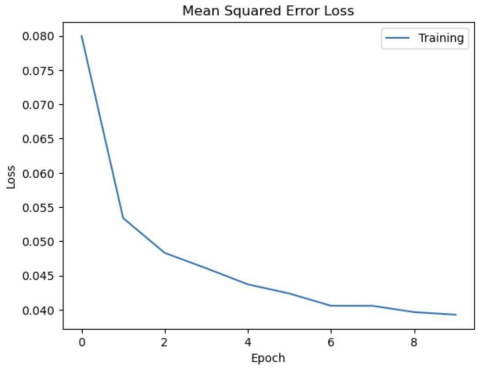

Fig. 17. Training Loss

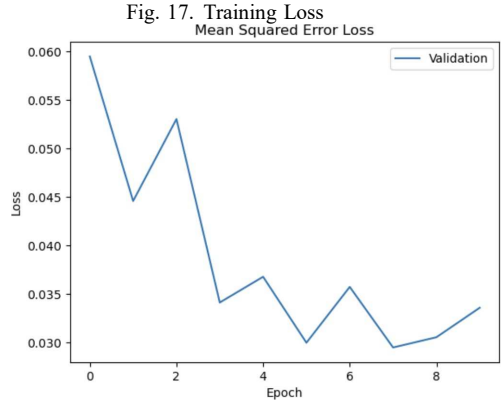

Fig. 18. Validation Loss

Four dense layers follow the convolutional layers to produce the steering angles. Additional dropout layers are added to prevent overfitting, and the fully connected layers are enlarged for improved performance. The model is optimized using the RMSprop optimizer, which adapts the learning rate automatically. Once the model is trained, it can be deployed and connected to the Udacity simulator for testing. This model is tested in a driving simulator to evaluate its performance.

V. Results

The model's training progress over a period of 10 epochs. A whole trip through the entire training dataset is represented by one epoch. By assessing the loss, more precisely the mean squared error (MSE), on both the training data and the validation data, the performance of the model is assessed.

1) Loss of value (calculated during the training period)
2) Model's generalization in driving abilities

In order to calculate value loss over each epoch, the Keras library provides val loss, which is the average loss following each epoch. The screenshots Fig. 16, Fig. 17, and Fig. 18, which depict the run of Architecture during the training phase, demonstrate how, despite the early epochs of the training phase recording a significant loss, that loss steadily decreases. The model is successfully learning the underlying patterns and relationships in the training data, enabling it to make better predictions on unseen validation data.

The model's ability to accurately mimic human driving behavior through behavioral cloning is evident in its smooth and precise steering responses. This means that the model can navigate the road with a level of familiarity and intuition similar to that of an experienced human driver. The model's exceptional performance, accurate steering angle predictions, adaptability to various driving scenarios, and robustness in challenging conditions make it a promising solution for real- world autonomous driving applications.

VI. Conclusion

Our proposed architecture has surpassed all previous archi- tectures and yielded significantly improved loss values. Behav- ioral cloning using deep learning techniques has proven to be highly effective in achieving a high-performing autonomous driving system. The model's remarkable ability to accurately mimic human driving behavior through behavioral cloning is evident in its smooth and precise steering responses. It exhibits a level of familiarity and intuition on par with experienced human drivers, allowing it to navigate the road with confi- dence. It showcases remarkable adaptability to various driving scenarios, effortlessly tackling different road conditions and challenges. Moreover, the model exhibits robustness, main- taining its superior performance even in demanding situations. The behavioral cloning model is a viable option for practical autonomous driving applications because of these characteris- tics.

Its success in replicating human driving behavior provides a strong foundation for developing safe and reliable autonomous vehicles. With its impressive performance and ability to navi- gate the road with precision, the model opens up new possibili- ties for the future of autonomous driving technology. However, we believe that there is still room for further improvement. Furthermore, we believe that our findings can be useful for future research aimed at improving the accuracy and reliabilityof self-driving applications.

References

[1] Farag, Wael, and Zakaria Saleh. "Behavior cloning for autonomous driving using convolutional neural networks." 2018 International Con- ference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT). IEEE, 2018.

[2] Khan, Muhammad Ammar, et al. "Mitigating the Zero Biased Steering Angles in Self-driving Simulator Datasets." VISIGRAPP (4: VISAPP). 2022.

[3] Ly, Abdoulaye O., and Moulay Akhloufi. "Learning to drive by imita- tion: An overview of deep behavior cloning methods." IEEE Transac- tions on Intelligent Vehicles 6.2 (2020): 195-209.

[4] Tippannavar, Sanjay S., S. D. Yashwanth, and K. M. Puneeth. "SDR–Self Driving Car Implemented using Reinforcement Learning and Be- havioural Cloning." 2023 International Conference on Recent Trends in Electronics and Communication (ICRTEC). IEEE, 2023.

[5] Khanum, , Abida, Chao-Yang Lee, and Chu-Sing Yang. "Deep-Learning-Based Network for Lane Following in Autonomous Vehicles." Electron- ics 11.19 (2022): 3084

[6] Le Mero, Luc, et al. ”A survey on imitation learning techniques for end-to-end autonomous vehicles.” IEEE Transactions on Intelligent Transportation Systems (2022).

[7] Ferencz, Csana´d, and Ma´te´ Zo¨ldy. ”End-to-end autonomous vehicle lateral control with deep learning.”(2021) 12th IEEE International Con- ference on Cognitive Infocommunications

[8] Bojarski, Mariusz, et al. ”End to end learning for self-driving cars.” arXiv preprint arXiv:1604.07316 (2016).

[9] M Shvejan Shashank; Saikrishna Prathapaneni; M. Anil; N Thanuja Sri; Mohammed Owais; B Saikumar, ”Behavior Cloning for Self Driving Cars using Attention Models”, International Journal of Innovative Sci- ence and Research Technology (IJISRT), 2021, pp. 612-616.

[10] Coelingh, E., Nilsson, E., and Buffum, J. (2018). Driving tests for self- driving cars. In IEEE Spectrum. IEEE.

[11] Farag, W. and Saleh, Z. (2017). Safe-driving cloning by deep learning for autonomous cars. In International Journal of Advanced Mechatronic Systems, page 390.

[12] Codevilla, Felipe, et al. ”Exploring the limitations of behavior cloning for autonomous driving.” Proceedings of the IEEE/CVF International Conference on Computer Vision. 2019.

[13] Wael Farag, Zakaria Saleh, ”Traffic Signs Identification by Deep Learn- ing for Autonomous Driving”, Smart Cities Symposium (SCS’18), Bahrain, 22-23 April 2018.

[14] Paul, M., and Thomas, B. (2016). A Review on Google Self Driving Car Technology. International Journal of Engineering Research and Technology (IJERT), 4(6).

[15] Wael Farag, “CANTrack: Enhancing automotive CAN bus security using intuitive encryption algorithms”, 7th Inter. Conf. on Modeling, Simulation, and Applied Optimization (ICMSAO), UAE, March 2017.

[16] Ivanovs, M., Ozols, K., Dobrajs A and Kadikis, R, “Improving seman- tic segmentation of urban scenes for self-driving cars with synthetic images”. Sensors, 2022, 2252.

[17] Bhujbal, Kunal, and Mahendra Pawar. ”Deep Learning Model for Simulating Self Driving Car.” 2023 International Conference on Com- munication System, Computing and IT Applications (CSCITA). IEEE, 2023.

[18] Kesava, Prasad SA, and K. P. Peeyush. ”Autonomous robot to detect diseased leaves in plants using convolutional neural networks.” 2019 3rd international conference on trends in electronics and informatics (ICOEI). IEEE, 2019.

[19] Sreenivas, K. Vandith, M. Ganesan, and R. Lavanya. ”Classification of arrhythmia in time series ecg signals using image encoding and convolutional neural networks.” 2021 Seventh International conference on Bio Signals, Images, and Instrumentation (ICBSII). IEEE, 2021.

[20] Natarajan, B., et al. ”Creating Alert messages based on Wild Animal Activity Detection using Hybrid Deep Neural Networks.” IEEE Access (2023).

Dharshana, D., et al. ”A Novel Approach for Detection and Classification of Fish Species.” 2023 Second International Conference on Electrical, Electronics, Information and Communication Technologies (ICEE ICT). IEEE, 2023.