Comparative Analysis of Neural Network Models for Autonomous Driving: A Case Study of the NVIDIA model and the DeepDriving Model

1st Sujan Pasumarti
Dept.of Electronics and Communication
Vellore Institute of Technology
Tamil Nadu, India
sujan.p2021a@vitstudent.ac.in

2nd Yaganti Pavan Sai
Dept.of Electronics and Communication
Vellore Institute of Technology
Tamil Nadu, India
yaganti.pavansai2021@vitstudent.ac.in

3rd Revathi V
Dept.of Electronics and Communication
Vellore Institute of Technology
Tamil Nadu, India
revathi.v2021@vitstudent.ac.in

4th Boopalan G
Dept.of Electronics and Communication
Vellore Institute of Technology
Tamil Nadu, India
boopalan@vit.ac.in

5th Shanmugasundaram S
Dept.of Electronics and Communication
Vellore Institute of Technology
Tamil Nadu, India
mshanmugasundaram@vit.ac.in

2024 3rd International Conference on Artificial Intelligence For Internet of Things (AIIoT) | 979-8-3503-7212-0/24/$31.00 ©2024 IEEE | DOI: 10.1109/AIIoT58432.2024.10574744

Abstract*—Artificial intelligence has significantly advanced technology, bringing once-imaginary concepts like self-driving cars into reality. This research focuses on leveraging AI, computer vision, and neural networks to develop autonomous vehicles capable of lane changes, parking, and navigating various traffic conditions. It compares NVIDIA-based models with deep driving models in simulated environments, highlighting their training methods and drawbacks. NVIDIA-based models, using convolutional neural networks, offer end-to-end learning with large datasets but lack interpretability. Deep driving models decompose tasks, requiring careful feature engineering but providing interpretability. Through simulations, both models are evaluated for accuracy, computational efficiency, and adaptability. This research provides insights into their strengths and limitations, contributing to discussions on improving autonomous driving systems

***Keywords*—Autonomous Vehicles, Deep Learning, Path Planning, NVIDIA Model, DeepDriving Model, Comparative Analysis.**

I. Introduction

The advent of autonomous driving technology has revolutionized the automotive industry, promising safer roads, improved transportation efficiency, and a new era of personalized mobility. Central to this technological leap are advanced algorithms that equip vehicles to perceive, interpret, and respond to their surroundings like human drivers. Two prominent methodologies have emerged for developing autonomous driving systems: the NVIDIA-based model and deep driving models. The NVIDIA approach, exemplified by frameworks like NVIDIA PilotNet, uses convolutional neural networks (CNNs) to directly map raw input images to steering commands. While effective and straightforward, it has limitations such as reliance on handcrafted features and susceptibility to overfitting. [1]

In contrast, deep driving models, employing deep reinforcement learning (DRL) techniques, take a holistic approach by learning driving policies through trial-and-error interactions with virtual or real-world environments. However, these models are criticized for their computational complexity, data inefficiency, and challenges in safe real-world exploration. [2]

Through experiments and theoretical analysis, we address critical research questions:
How do NVIDIA-based and deep driving models differ in framework, training paradigms, and computational requirements?

- How do NVIDIA based and DeepDriving models differ in their framework, training paradigms and computational requirements?
- What performance benchmarks assess the efficacy and safety of autonomous driving systems, and how do these metrics vary across different model architectures?
- What are the inherent limitations and challenges associated with these models, and how can they be mitigated?

Our research paper conducts a comprehensive comparative analysis of NVIDIA based and deep driving models for autonomous car simulation. We aim to evaluate their training methods, performance characteristics, and inherent challenges systematically, shedding light on their applicability, scalability, and potential avenues for improvement in building robust autonomous driving systems.

II. Methodology

A. Dataset Selection

The dataset utilised in this study was obtained from the Udacity self-driving car simulator. The simulator provides a virtual environment where you can drive a car autonomously using various control methods. To conduct our research, we opted for the manual driving mode, allowing us to directly control the car's steering, throttle and brake inputs. This mode provided us with the precise control over the car's movements.

B. Data Preprocessing

The dataset contains an excessive number of samples with a steering angle of 0 degrees since a large fraction of the driving data consists of cases where the automobile is traveling along the middle of the track. We exclude the high-frequency values that is, the values that correspond to a 0-

degree steering angle in order to lessen this bias. We do this to ensure that we have a more balanced distribution of the steering angles which reduces the probability that the model predicts 0-degree steering angle for cases where the car does not move on the center of the track. [3][4]

C. Lane Detection Process

In autonomous driving, tracking lane lines accurately is crucial for safe navigation. The method we use relies on the fact that lane lines change gradually as the vehicle moves. By comparing lane line slopes in nearby frames, we can predict their position in the current frame. This helps us focus our detection efforts on areas where lane lines are likely to be, reducing processing time. [5][6]. To address the challenges in the lane line detection process, a modified Hough transform approach is employed. This technique focuses on performing transformations specifically at the vanishing point and the limited surrounding pixels. By targeting these critical areas, the algorithm achieves improved real-time performance while maintaining accuracy in lane line detection. [7]

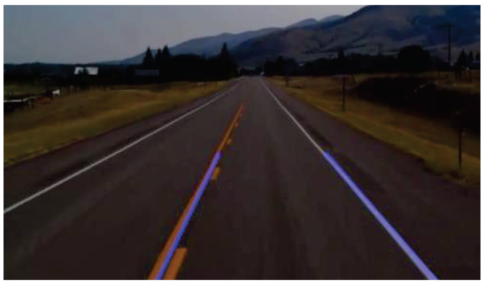


Fig. 1. Sample image of Lane Detection

This paper explores two prominent models used in the autonomous driving industry: the NVIDIA End-to-End model and the Deep Driving model. By analyzing their architectural designs, performance metrics, and practical applications, we aim to assess their effectiveness and accuracy with the given dataset. [8][9]

The NVIDIA model comprises five convolutional layers followed by rectified linear unit (ReLU) activation functions. The initial three layers use 5x5 filters with a (2, 2) stride, reducing spatial dimensions and increasing depth. The subsequent layers employ 3x3 filters without striding for detailed feature extraction. After convolution, the feature maps are flattened and fed into three fully connected layers with decreasing neuron counts. The model outputs a continuous steering angle prediction using mean squared error (MSE) loss and is optimized with the Adam optimizer at a learning rate of 0.0001. On the other hand, the Deep Driving model also employs convolutional layers with increasing filter sizes and depths, utilizing ReLU activation functions. The initial layers use 5x5 filters with a (2, 2) stride for spatial down sampling, followed by 3x3 filters for further feature refinement. After convolution, the feature maps are flattened and processed through fully connected layers. Notably, the model incorporates a Long Short-Term Memory (LSTM) layer to retain temporal information, followed by additional fully connected layers and a linear output layer for steering angle prediction.

Both models are designed for end-to-end learning in autonomous driving but differ in architecture, particularly with the Deep Driving model's inclusion of an LSTM layer for temporal information retention. These variations in design influence their respective training methodologies and performance characteristics, which we analyze to discern which model achieves superior accuracy and efficacy with the provided dataset. Through this investigation, we aim to contribute insights into the selection and optimization of model architectures for autonomous driving applications. Here are the key differences:

1) Architectural Differences:

The NVIDIA model architecture, proposed by NVIDIA in their paper "End to End Learning for Self-Driving Cars," consists of a series of convolutional layers followed by fully connected layers. It is designed to directly map raw pixel inputs from onboard cameras to steering commands. The Deep Driving model, on the other hand, typically includes convolutional layers for feature extraction, followed by recurrent layers (such as LSTM or GRU) to capture temporal dependencies in the data. This architecture is aimed at learning driving behaviors over time by considering sequential frames of input data. [1][2]

2) Feature Representation:

The NVIDIA model focuses on learning feature representations directly from raw image inputs, with convolutional layers extracting relevant visual features for steering prediction. The Deep Driving model may incorporate recurrent layers to capture temporal dependencies and dynamic changes in the driving environment over time. This allows the model to consider not only the current frame but also the context provided by previous frames.

3) Training Methodology:

The NVIDIA model is typically trained using supervised learning, where steering angles are directly predicted from image inputs. The model is optimized to minimize the error between predicted and ground truth steering angles. The DeepDriving model may involve more complex training methodologies, such as sequence prediction or reinforcement learning, depending on the specific architecture used. Recurrent layers enable the model to learn from sequential data and make predictions based on temporal context.

4) Model Complexity:

The NVIDIA model architecture is relatively straightforward, consisting of convolutional and fully connected layers. It is designed to be computationally efficient and scalable for deployment in real-world applications. The DeepDriving model may introduce additional complexity with the inclusion of recurrent layers, which require more computational resources and may be more challenging to train effectively.

III. LIMITATIONS

A. Computational issues:

Deep learning requires a large amount of computing power to manage the massive amounts of data it processes. These days, the most popular option is a graphical processing unit (GPU), which was first created for demanding image processing applications like gaming. It is still difficult to create inexpensive, power-efficient GPUs that work well in smart cars. Moreover, issues with synchronization and bandwidth continue to exist, which makes integration into the automobile ecosystem difficult. Here we used a RTX 3050 processor for training the model. [9]

B. Availability of Training Data:

The abundance of training data available in the current system is one of its main problems. Particularly, end-to-end learning systems need enormous volumes of heterogeneous data in order to efficiently predict different driving conditions and guarantee minimal safety requirements. According to experts, in order to make strong judgments regarding the safety of independent vehicles, at least one billion kilometers of real- world driving data must be collected. nonetheless, gathering and collecting such a wide range of datasets presents considerable logistical and technological challenges.

C. Safety Issues:

In independent driving systems, safety is still the top precedence, especially when it comes to deep neural network adaptability. These systems are vulnerable to inimical disquiet, in which small changes to the attributes of the images can beget misclassification and occasionally dangerous results. likewise, not enough exploration has been done on safety assurance and verification ways for machine literacy systems, particularly deep literacy algorithms. Standardization sweats are hampered by the current auto safety standard, ISO26262, which doesn't address the special difficulties presented by tone- learning algorithms. [10]

IV. TRAINING EVALUATIONS AND TESTING

A. Data Loading and Exploration

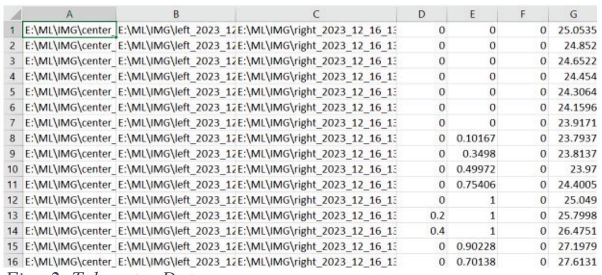
The code begins by loading driving data from a CSV file named driving_log.csv (which is the data that is obtained from Udacity) using the pandas library. This file contains information about the paths to center, left, and right camera images, along with corresponding steering angles, throttle and speed. The data is organized into a Data Frame with specified column names to facilitate further processing and analysis. To ensure there the data representation was clear we adjusted the display settings to show the complete names of the column without truncation by using the pd_set_option function.

B. Data Analysis and Visualization

The code then proceeds to check the distribution of steering angles in the dataset. It divides the steering angle data into a specified number of bins and plots a histogram to visualize the distribution. The purpose of this visualization is to gain insights into the steering behaviour of the vehicle during data collection. Understanding the steering angles helped us to identify imbalances in the dataset.

C. Data Preprocessing

Several preprocessing steps were performed to prepare the dataset for training which includes defining a function to extract the filename from the complete image path, making it easier to work with image filenames. Additionally, the steering angles for left and right camera images are adjusted to simulate the effect of the vehicle being off center. This adjustment helped us to obtain a more diverse training dataset and improve the model's generalization capabilities.



	A	B	C	D	E	F	G
1	E:\ML\IMG\center_	E:\ML\IMG\left_2023_12	E:\ML\IMG\right_2023_12_16_1	0	0	0	25.0535
2	E:\ML\IMG\center_	E:\ML\IMG\left_2023_12	E:\ML\IMG\right_2023_12_16_1	0	0	0	24.852
3	E:\ML\IMG\center_	E:\ML\IMG\left_2023_12	E:\ML\IMG\right_2023_12_16_1	0	0	0	24.6522
4	E:\ML\IMG\center_	E:\ML\IMG\left_2023_12	E:\ML\IMG\right_2023_12_16_1	0	0	0	24.454
5	E:\ML\IMG\center_	E:\ML\IMG\left_2023_12	E:\ML\IMG\right_2023_12_16_1	0	0	0	24.3064
6	E:\ML\IMG\center_	E:\ML\IMG\left_2023_12	E:\ML\IMG\right_2023_12_16_1	0	0	0	24.1596
7	E:\ML\IMG\center_	E:\ML\IMG\left_2023_12	E:\ML\IMG\right_2023_12_16_1	0	0	0	23.9171
8	E:\ML\IMG\center_	E:\ML\IMG\left_2023_12	E:\ML\IMG\right_2023_12_16_1	0	0.10167	0	23.7937
9	E:\ML\IMG\center_	E:\ML\IMG\left_2023_12	E:\ML\IMG\right_2023_12_16_1	0	0.3498	0	23.8137
10	E:\ML\IMG\center_	E:\ML\IMG\left_2023_12	E:\ML\IMG\right_2023_12_16_1	0	0.49972	0	23.97
11	E:\ML\IMG\center_	E:\ML\IMG\left_2023_12	E:\ML\IMG\right_2023_12_16_1	0	0.75406	0	24.4005
12	E:\ML\IMG\center_	E:\ML\IMG\left_2023_12	E:\ML\IMG\right_2023_12_16_1	0	1	0	25.049
13	E:\ML\IMG\center_	E:\ML\IMG\left_2023_12	E:\ML\IMG\right_2023_12_16_1	0.2	1	0	25.7998
14	E:\ML\IMG\center_	E:\ML\IMG\left_2023_12	E:\ML\IMG\right_2023_12_16_1	0.4	1	0	26.4751
15	E:\ML\IMG\center_	E:\ML\IMG\left_2023_12	E:\ML\IMG\right_2023_12_16_1	0	0.90228	0	27.1979
16	E:\ML\IMG\center_	E:\ML\IMG\left_2023_12	E:\ML\IMG\right_2023_12_16_1	0	0.70138	0	27.6131

Fig. 2. Telemetry Data

Fig. 3. Camera Data

D. Image Preprocessing

A function is defined to preprocess images before feeding them into the neural network model. Preprocessing steps include cropping, color space conversion from RGB to YUV, Gaussian blur, resizing, and normalization. These preprocessing steps were done to enhance the quality of the input images which makes it easier for the model to extract the desired features for steering prediction.

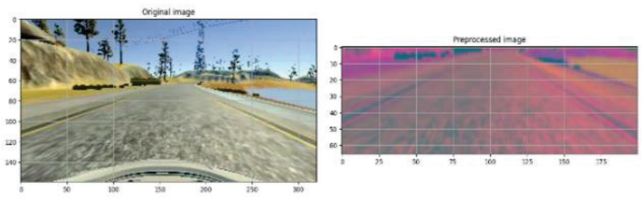


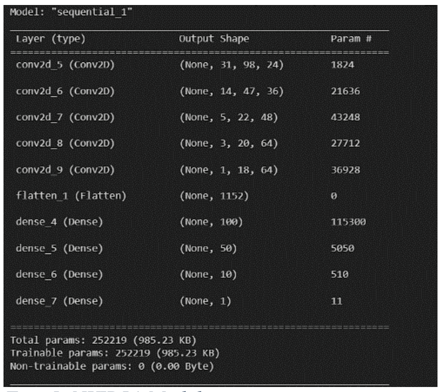
Fig. 4. Preprocessed Image Sample

E. Batch Data Generation

A generator function is defined to generate batches of training data with a specified batch size. This function yields batches of preprocessed images and corresponding steering angles, optionally applying augmentation techniques. Batch data generation helps in the efficient training of neural network models by loading and processing data in smaller batches, reducing memory requirements and facilitating parallel processing.

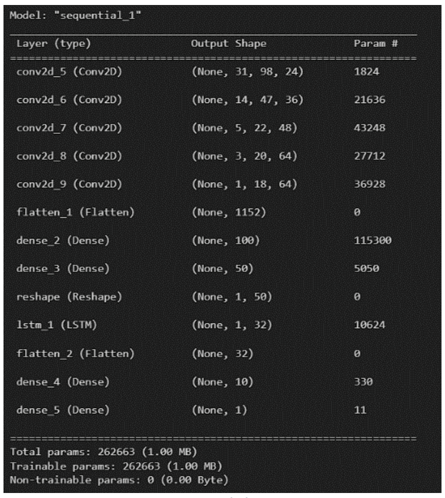
F. Model Definition

We defined both the models using the designs proposed, which consists of several convolutional layers followed by fully connected layers which are the dense layers. These models were defined using the Sequential API from TensorFlow Keras. We specifically constructed a Convolution Neural Network (CNN) model, which is a deep learning architecture mostly effective for image related tasks.



```
Model: "sequential_1"
_________________________________________________________________
 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

 flatten_1 (Flatten)         (None, 1152)              0

 dense_4 (Dense)             (None, 100)               115300

 dense_5 (Dense)             (None, 50)                5050

 dense_6 (Dense)             (None, 10)                510

 dense_7 (Dense)             (None, 1)                 11

=================================================================
Total params: 252219 (985.23 KB)
Trainable params: 252219 (985.23 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Fig. 5. NVIDIA Model

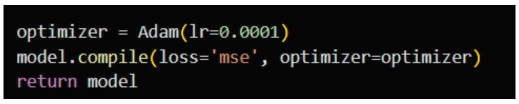


```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_5 (Conv2D)           (None, 31, 98, 24)        1824

 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

 flatten_1 (Flatten)         (None, 1152)              0

 dense_2 (Dense)             (None, 100)               115300

 dense_3 (Dense)             (None, 50)                5050

 reshape (Reshape)           (None, 1, 50)             0

 lstm_1 (LSTM)               (None, 1, 32)             10624

 flatten_2 (Flatten)         (None, 32)                0

 dense_4 (Dense)             (None, 10)                330

 dense_5 (Dense)             (None, 1)                 11

=================================================================
Total params: 262663 (1.00 MB)
Trainable params: 262663 (1.00 MB)
Non-trainable params: 0 (0.00 Byte)
```

Fig. 6. DeepDriving Model

G. Training Configuration

Before training, the model architecture is compiled with key settings: the Mean Squared Error (MSE) loss function for regression tasks like steering angle prediction, and the Adam optimizer with a learning rate of 0.0001 for efficient deep neural network training. These specifications prepare the model for the training process.

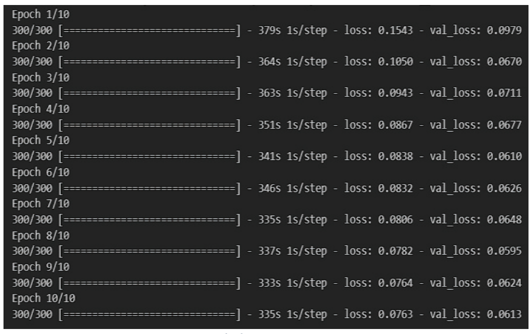


```
optimizer = Adam(lr=0.0001)
model.compile(loss='mse', optimizer=optimizer)
return model
```

Fig. 7. Adan Optimizer Function

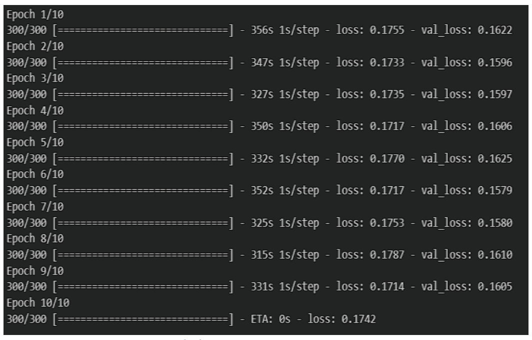
H. Generating the model

Once the model is compiled and the data is loaded, the training process initiates. This pivotal phase involves invoking the fit method of the model, which orchestrates the training of the neural network. Throughout this iterative process, the model dynamically adjusts its parameters, including weights and biases, leveraging the training data to minimize the specified loss function. Each iteration, known as an epoch, is a critical step in refining the model's predictive capabilities. Within each epoch, the entirety of the dataset is systematically propagated through the neural network, enabling the model to glean insights and refine its understanding of the underlying patterns within the data. The efficacy of the training loop is governed by parameters such as steps_per_epoch, which dictates the number of batches processed per epoch, and epochs, determining the total number of training iterations. Through this meticulously orchestrated process, the neural network iteratively hones its predictive prowess, gradually converging towards optimal performance.



```
Epoch 1/10
300/300 [==============================] - 379s 1s/step - loss: 0.1543 - val_loss: 0.0979
Epoch 2/10
300/300 [==============================] - 364s 1s/step - loss: 0.1050 - val_loss: 0.0670
Epoch 3/10
300/300 [==============================] - 363s 1s/step - loss: 0.0943 - val_loss: 0.0711
Epoch 4/10
300/300 [==============================] - 351s 1s/step - loss: 0.0867 - val_loss: 0.0677
Epoch 5/10
300/300 [==============================] - 341s 1s/step - loss: 0.0838 - val_loss: 0.0610
Epoch 6/10
300/300 [==============================] - 346s 1s/step - loss: 0.0832 - val_loss: 0.0626
Epoch 7/10
300/300 [==============================] - 335s 1s/step - loss: 0.0806 - val_loss: 0.0648
Epoch 8/10
300/300 [==============================] - 337s 1s/step - loss: 0.0782 - val_loss: 0.0595
Epoch 9/10
300/300 [==============================] - 333s 1s/step - loss: 0.0764 - val_loss: 0.0624
Epoch 10/10
300/300 [==============================] - 335s 1s/step - loss: 0.0763 - val_loss: 0.0613
```

Fig. 8. DeepDriving Model Training

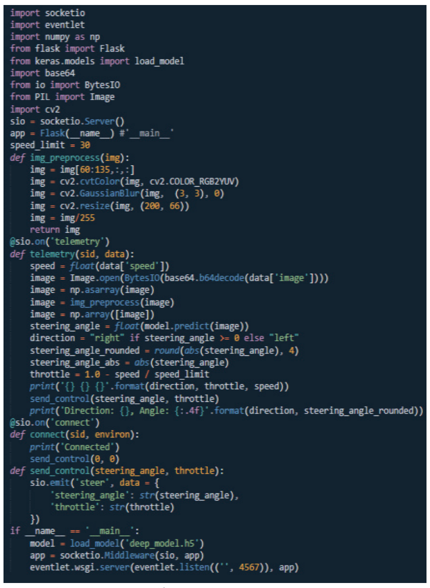


```
Epoch 1/10
300/300 [==============================] - 356s 1s/step - loss: 0.1755 - val_loss: 0.1622
Epoch 2/10
300/300 [==============================] - 347s 1s/step - loss: 0.1733 - val_loss: 0.1596
Epoch 3/10
300/300 [==============================] - 327s 1s/step - loss: 0.1735 - val_loss: 0.1597
Epoch 4/10
300/300 [==============================] - 350s 1s/step - loss: 0.1717 - val_loss: 0.1606
Epoch 5/10
300/300 [==============================] - 332s 1s/step - loss: 0.1770 - val_loss: 0.1625
Epoch 6/10
300/300 [==============================] - 352s 1s/step - loss: 0.1717 - val_loss: 0.1579
Epoch 7/10
300/300 [==============================] - 325s 1s/step - loss: 0.1753 - val_loss: 0.1580
Epoch 8/10
300/300 [==============================] - 315s 1s/step - loss: 0.1787 - val_loss: 0.1610
Epoch 9/10
300/300 [==============================] - 331s 1s/step - loss: 0.1714 - val_loss: 0.1605
Epoch 10/10
300/300 [==============================] - ETA: 0s - loss: 0.1742
```

Fig. 9. NVIDIA Model Training

I. Testing

A server-client system for a simulated self-driving car scenario is established where it loads a pre-trained deep learning model for steering angle prediction. When the simulated car sends telemetry data, including speed and camera images, the server preprocesses the images, predicts steering angles using the model, calculates throttle based on speed, and sends control commands back to the car. It communicates with the car using SocketIO and Flask, handling connections, and control commands.



```python
import socketio
import eventlet
import numpy as np
from flask import Flask
from keras.models import load_model
import base64
from io import BytesIO
from PIL import Image
import cv2
sio = socketio.Server()
app = Flask(__name__) #'__main__'
speed_limit = 30
def img_preprocess(img):
    img = img[60:135,:,:]
    img = cv2.cvtColor(img, cv2.COLOR_RGB2YUV)
    img = cv2.GaussianBlur(img,  (3, 3), 0)
    img = cv2.resize(img, (200, 66))
    img = img/255
    return img
@sio.on('telemetry')
def telemetry(sid, data):
    speed = float(data['speed'])
    image = Image.open(BytesIO(base64.b64decode(data['image'])))
    image = np.asarray(image)
    image = img_preprocess(image)
    image = np.array([image])
    steering_angle = float(model.predict(image))
    direction = "right" if steering_angle >= 0 else "left"
    steering_angle_rounded = round(abs(steering_angle), 4)
    steering_angle_abs = abs(steering_angle)
    throttle = 1.0 - speed / speed_limit
    print('{} {} {}'.format(direction, throttle, speed))
    send_control(steering_angle, throttle)
    print('Direction: {}, Angle: {:.4f}'.format(direction, steering_angle_rounded))
@sio.on('connect')
def connect(sid, environ):
    print('Connected')
    send_control(0, 0)
def send_control(steering_angle, throttle):
    sio.emit('steer', data = {
        'steering_angle': str(steering_angle),
        'throttle': str(throttle)
    })
if __name__ == '__main__':
    model = load_model('deep_model.h5')
    app = socketio.Middleware(sio, app)
    eventlet.wsgi.server(eventlet.listen(('', 4567)), app)
```

Fig. 10. Server-Client System Setup

V. RESULTS

Upon analyzing the training process of both the NVIDIA model and the Deep Driving model, a consistent decrease in both training and validation loss was observed with increasing epochs. This trend suggests effective learning from the training data and improvement in performance over time for both models. However, it is noteworthy that the Deep Driving model exhibited a lower loss compared to the NVIDIA model by the end of 10 epochs. Specifically, the training and validation loss values for the NVIDIA model at the 10th epoch were 0.1714 and 0.1605 respectively, while for the Deep Driving model, they were 0.0763 and 0.0612. These results indicate a potentially stronger performance of the Deep Driving model in capturing underlying patterns in the training data.

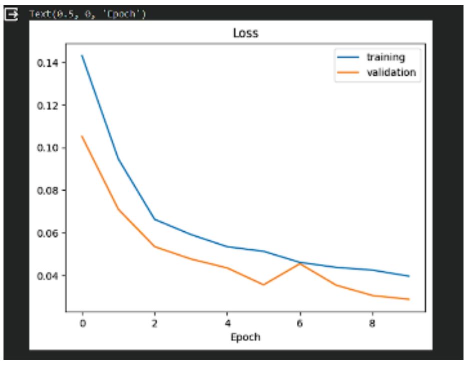


Fig. 11. Training vs Validation Loss (NVIDIA)

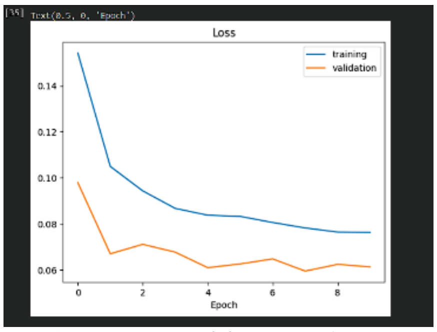
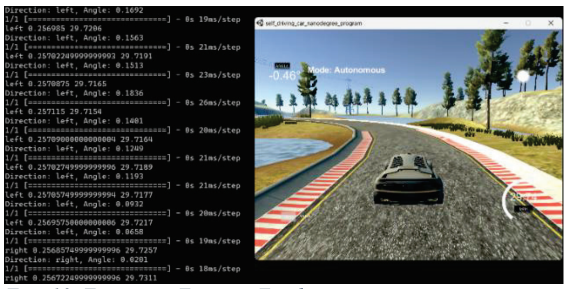


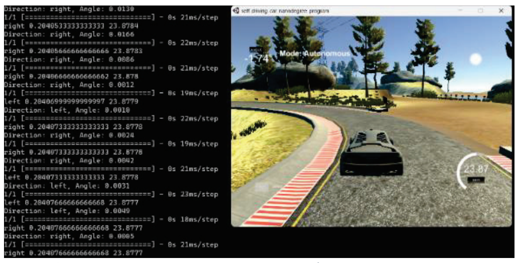
Fig. 12. Training vs Validation Loss (DeepDriving)

Considering these loss values, it is plausible to test our model on a different track than the one used for training. Real-time steering angle predictions were observed during testing, with negative angles indicating left steering and positive angles indicating right steering. The direction and angle information were displayed on the command window interface. Both cars operated in autonomous mode at maximum speed throughout the testing process.



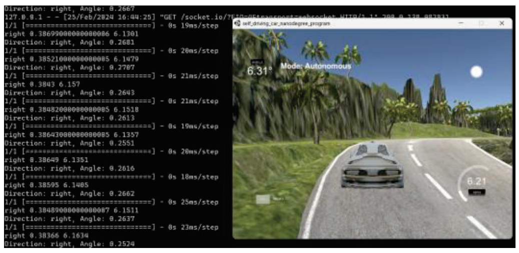
```
Direction: left, Angle: 0.1692
1/1 [==============================] - 0s 19ms/step
left 0.256985 29.7206
Direction: left, Angle: 0.1563
1/1 [==============================] - 0s 21ms/step
left 0.25702249999999993 29.7191
Direction: left, Angle: 0.1513
1/1 [==============================] - 0s 23ms/step
left 0.2570875 29.7165
Direction: left, Angle: 0.1836
1/1 [==============================] - 0s 26ms/step
left 0.257115 29.7154
Direction: left, Angle: 0.1401
1/1 [==============================] - 0s 20ms/step
left 0.25709000000000004 29.7164
Direction: left, Angle: 0.1249
1/1 [==============================] - 0s 21ms/step
left 0.25702749999999996 29.7189
Direction: left, Angle: 0.1193
1/1 [==============================] - 0s 21ms/step
left 0.25705749999999994 29.7177
Direction: left, Angle: 0.0932
1/1 [==============================] - 0s 20ms/step
left 0.25695750000000006 29.7217
Direction: left, Angle: 0.0658
1/1 [==============================] - 0s 19ms/step
right 0.25685749999999996 29.7257
Direction: right, Angle: 0.0201
1/1 [==============================] - 0s 18ms/step
right 0.25672249999999996 29.7311
```

Fig. 13. Testing on Training Track



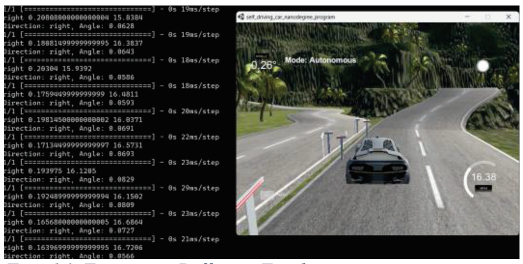
```
Direction: right, Angle: 0.0130
1/1 [==============================] - 0s 21ms/step
right 0.20405333333333331 23.8784
Direction: right, Angle: 0.0166
1/1 [==============================] - 0s 22ms/step
right 0.2040566666666666 23.8783
Direction: right, Angle: 0.0086
1/1 [==============================] - 0s 21ms/step
right 0.20406666666666662 23.878
Direction: right, Angle: 0.0012
1/1 [==============================] - 0s 19ms/step
left 0.20406999999999997 23.8779
Direction: left, Angle: 0.0010
1/1 [==============================] - 0s 22ms/step
right 0.20407333333333333 23.8778
Direction: right, Angle: 0.0024
1/1 [==============================] - 0s 19ms/step
right 0.20407333333333333 23.8778
Direction: right, Angle: 0.0042
1/1 [==============================] - 0s 21ms/step
left 0.20407333333333333 23.8778
Direction: left, Angle: 0.0031
1/1 [==============================] - 0s 23ms/step
left 0.2040766666666668 23.8777
Direction: left, Angle: 0.0049
1/1 [==============================] - 0s 18ms/step
right 0.2040766666666668 23.8777
Direction: right, Angle: 0.0085
1/1 [==============================] - 0s 21ms/step
right 0.2040766666666668 23.8777
```

Fig. 14. Testing on a Training Track



```
Direction: right, Angle: 0.2667
127.0.0.1 - - [25/Feb/2024 16:44:25] "GET /socket.io/?E
1/1 [==============================] - 0s 19ms/step
right 0.38699000000000006 6.1301
Direction: right, Angle: 0.2681
1/1 [==============================] - 0s 20ms/step
right 0.38521000000000005 6.1479
Direction: right, Angle: 0.2707
1/1 [==============================] - 0s 21ms/step
right 0.3843 6.157
Direction: right, Angle: 0.2643
1/1 [==============================] - 0s 21ms/step
right 0.38482000000000005 6.1518
Direction: right, Angle: 0.2613
1/1 [==============================] - 0s 19ms/step
right 0.38643000000000005 6.1357
Direction: right, Angle: 0.2551
1/1 [==============================] - 0s 20ms/step
right 0.38649 6.1351
Direction: right, Angle: 0.2616
1/1 [==============================] - 0s 18ms/step
right 0.38595 6.1405
Direction: right, Angle: 0.2662
1/1 [==============================] - 0s 25ms/step
right 0.38489000000000007 6.1511
Direction: right, Angle: 0.2637
1/1 [==============================] - 0s 23ms/step
right 0.38366 6.1634
Direction: right, Angle: 0.2524
```

Fig. 15. Testing on Different Track



```
1/1 [==============================] - 0s 19ms/step
right 0.20808000000000004 15.8384
Direction: right, Angle: 0.0628
1/1 [==============================] - 0s 19ms/step
right 0.18881499999999995 16.3837
Direction: right, Angle: 0.0643
1/1 [==============================] - 0s 18ms/step
right 0.20304 15.9392
Direction: right, Angle: 0.0586
1/1 [==============================] - 0s 18ms/step
right 0.17594499999999999 16.4811
Direction: right, Angle: 0.0593
1/1 [==============================] - 0s 20ms/step
right 0.19814500000000002 16.0371
Direction: right, Angle: 0.0691
1/1 [==============================] - 0s 22ms/step
right 0.17134499999999997 16.5731
Direction: right, Angle: 0.0693
1/1 [==============================] - 0s 23ms/step
right 0.193975 16.1285
Direction: right, Angle: 0.0829
1/1 [==============================] - 0s 29ms/step
right 0.19248999999999994 16.1502
Direction: right, Angle: 0.0809
1/1 [==============================] - 0s 23ms/step
right 0.16568000000000005 16.6864
Direction: right, Angle: 0.0727
1/1 [==============================] - 0s 21ms/step
right 0.16396999999999995 16.7206
Direction: right, Angle: 0.0566
```

Fig. 16. Testing on Different Track

VI. CONCLUSION

Our comparative analysis of NVIDIA based and deep driving machine learning models for auto-driving car simulation has provided valuable information on the strengths, weaknesses and potential areas for improvement in autonomous driving technology. We have explored the complexities of each model structure, examined its performance characteristics and underlined their specific weaknesses by means of practical experiments and theoretical analyses.

Our findings indicate that NVIDIA-based models offer a pragmatic solution for autonomous driving, leveraging CNNs to directly map input images to steering commands. On the other hand, deep driving models, driven by DRL techniques, offer a more adaptive and autonomous approach to driving, albeit with higher computational demands and challenges in real-world implementation. Comparing the training aspects, the values of loss and the validation loss for the NVIDIA-based model have lesser values than the deep driving model. Additionally, the aspect of speed has been a very important factor in this comparison. We have trained the models to attain a maximum speed of 30km/hr. The NVIDIA-based model resulted in a speed of approximately 30km/hr, whereas the deep driving model achieved a speed of 24km/hr. This suggests that the NVIDIA model is the most efficient speed training model for the data used in this research. The efficient steering angles produced by the NVIDIA model have also led to more effective path planning for the autonomous vehicle. Analyzing the research work, the NVIDIA model has been the most effective and efficient model for predicting the accurate path for the given dataset. In summary, the quest for autonomous driving represents a multifaceted challenge fraught with technical, societal, and regulatory barriers. By leveraging the complementary strengths of NVIDIA-based and deep driving models, and by addressing their respective weaknesses in a holistic manner, we can pave the way for a future where autonomous vehicles revolutionize transportation, enhance safety, and empower individuals with newfound mobility opportunities.

VII. FUTURE WORK

A large portion of the issue stems from the lack of sufficient training material. Providing billions of hours of real-world driving footage to a self-driving car could be the best method to teach it acceptable driving behaviour. When presented with an adequate amount of data, advanced framework consideration models perform extraordinarily well; nevertheless, when only a little amount of data is available, they perform astonishingly poorly.

To advance the state-of-the-art in autonomous driving, future research efforts should focus on several key areas of improvement: Hybrid Approaches: Integrating the strengths of NVidia-based and deep driving models could yield more robust and efficient autonomous driving systems.

Data Efficiency: Addressing the data inefficiency inherent in deep driving models is crucial for real-world deployment. Improving autonomous driving systems' safety and robustness is still a top priority. When presented with an adequate amount of data, advanced framework consideration models perform extremely well; nevertheless, when only a little amount of data is available, they perform astonishingly poorly to assure the dependability of autonomous vehicles in a variety of dynamic and varied contexts, future research should give priority to developing algorithms and procedures for safe exploration, risk assessment, and fault tolerance.

Future research should prioritize the development of algorithms and methodologies for safe exploration, risk assessment, and fault tolerance to ensure the reliability of autonomous vehicles in diverse and dynamic environments. Future research should focus on developing transparent and interpretable models that can provide insights into decision-making processes and facilitate human trust and collaboration. By addressing these challenges and leveraging the insights gleaned from our comparative analysis, future research endeavors can propel the field of autonomous driving towards unprecedented levels of innovation, safety, and societal impact.

REFERENCES

[1] M. Bojarski et al., “End to end learning for self-driving cars,” arXiv.org, https://arxiv.org/abs/1604.07316 (accessed Apr. 10, 2024).

[2] C. Chen, A. Seff, A. Kornhauser and J. Xiao, "DeepDriving: Learning Affordance for Direct Perception in Autonomous Driving," 2015 IEEE International Conference on Computer Vision (ICCV), Santiago, Chile, 2015, pp. 2722-2730, doi: 10.1109/ICCV.2015.312.

[3] S. Sun et al., "Scalability in Perception for Autonomous Driving: Waymo Open Dataset," arXiv preprint arXiv:1912.04838, 2019.

[4] C. Wu et al., "Uncertainty Estimation in Deep Neural Networks for Autonomous Driving," IEEE Transactions on Intelligent Vehicles, vol. 5, no. 3, pp. 546-556, 2020.

[5] Muhammad, Khan & Ullah, Amin & Lloret, Jaime & Del Ser, Javier & Albuquerque, V.H.C.. (2020). Deep Learning for Safe Autonomous Driving: Current Challenges and Future Directions. IEEE Transactions on Intelligent Transportation Systems. PP. 1-21. 10.1109/TITS.2020.3032227.

[6] J. Redmon et al., "You Only Look Once: Unified, Real-Time Object Detection," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 779-788.

[7] Grigorescu, Sorin & Trasnea, Bogdan & Cocias, Tiberiu & Macesanu, Gigel. (2019). A survey of deep learning techniques for autonomous driving. Journal of Field Robotics. 37. 10.1002/rob.21918.

[8] Ragheb, Noureldin & Mahmoud, Mervat. (2024). Implementing Deep Reinforcement Learning in Autonomous Control Systems. Journal of Advanced Research in Applied Sciences and Engineering Technology. 41. 168–178. 10.37934/araset.41.1.168178.

[9] Hasan, Fatima. (2022). Self-Driving Car to Drive Autonomously using Image Processing and Deep Learning. 10.13140/RG.2.2.16212.88963/1.

[10] Fathy, Mahmoud & Ashraf, Nada & Ismail, Omar & Fouad, Sarah & Shaheen, Lobna & Hamdy, Alaa. (2020). Design and implementation of self-driving car. Procedia Computer Science. 175. 165-172. 10.1016/j.procs.2020.07.026.