International Journal of Management and Marketing Intelligence	International Journal of Management and Marketing Intelligence, 2(3), 11-20.	
	Volume: 2	http://ijmmi.com
	Issue: 3	ISSN: 3080-860X
	Received: July 01, 2025	Accepted: August 11, 2025
	Citation: Khawaldeh, B., García, A. M., & Faris, H. (2025) Evaluation of Convolutional Neural Networks for Predicting Steering Angles in Autonomous Driving Systems Using the Udacity Simulator. International Journal of Management and Marketing Intelligence 2(3), 11-20. DOI: https://doi.org/10.64251/ijmmi.86	

Evaluation of Convolutional Neural Networks for Predicting Steering Angles in Autonomous Driving Systems Using the Udacity Simulator

Bashar Khawaldeh[1*], Antonio Miguel Mora García[1], Hossam Faris[2]

[1]Signal Theory, Telematics and Communications Department (TSTC), University of Granada, Granada, Spain.
[2]Business Information Technology Department, The University of Jordan, Amman, Jordan.

ARTICLE DETAILS

Article History
Published Online: September 2025

Keywords
Convolutional Neural Networks
Steering Angles
Autonomous Driving Systems
Udacity Simulator

JEL Codes:
R41, R42, I18

Corresponding Author Email:
khawaldeh@correo.ugr.es

ABSTRACT

This research explores the use of convolutional neural networks (CNNs) to develop autonomous driving systems, focusing on predicting steering angles for autonomous vehicles. A custom dataset was created using the Udacity simulator, which provides a high-fidelity simulation environment with multiple lanes and three-angle cameras to collect driving data. This data encompasses, but is not limited to, vehicle speed, steering angle, and gear position. To achieve this objective, two CNN models were utilized: the simple Comma.ai model and the more complex NVIDIA model. These models were utilized to compare performance and examine their ability to predict steering angles accurately. The experimental framework entailed the training of models on data using mean loss evaluation (MSE) and the subsequent validation of their performance on independent validation data. The findings of the study demonstrated that both models exhibited the capacity to differentiate driving patterns from visual data. However, the NVIDIA model exhibited superior performance in complex environments, a feat attributable to its advanced architecture and its ability to extract more accurate features. Conversely, the Comma model demonstrated superior performance in less complex environments, making it a suitable option for simpler systems.

1. INTRODUCTION

The transportation sector is a fundamental component of modern society's development. In recent decades, it has undergone a qualitative shift, attributable to the integration of digital technologies and artificial intelligence into its systems. One of the most notable expressions of this evolution is the advent of autonomous vehicles, which signify a paradigm shift in mobility paradigms by leveraging sophisticated algorithms to process data and formulate decisions in dynamic and intricate driving scenarios. The accuracy of predicting steering angles is a critical factor in ensuring vehicle safety and efficiency, as it is directly related to the vehicle's ability to maintain lane stability, navigate curves, and avoid obstacles. Recent studies have demonstrated the efficacy of deep learning techniques, particularly convolutional neural networks (CNNs), in processing image data to extract driving patterns and predict steering angles with a high degree of accuracy. Accordingly, the objective of this research is to examine and appraise the efficacy of CNN models in predicting steering angles. This evaluation will be based on simulation data that has been collected using an advanced simulation environment. The research also focuses on comparing two different neural network models-the simple Comma.ai model and the more complex NVIDIA model—to explore their ability to adapt to varying driving environments and highlight their strengths and limitations. This paper presents a practical guide for evaluating convolutional neural networks (CNNs) in the context of autonomous driving. The present study focuses on the generation of simulation data, in conjunction with an examination of preprocessing techniques, training methods, and optimization methods.

2. STATE OF THE ART

The evolution of autonomous driving can be traced back to early inventions such as Leonardo da Vinci's self-propelled cart in the 15th century, Whitehead's torpedo in 1868, the aircraft autopilot in 1933, and automotive cruise control introduced in 1945. Notable milestones in the field of robotic transportation include Stanford's camera-guided vehicle, developed in 1961, and Japan's 1977 traffic-sign-recognizing car. The 1980s and 1990s witnessed additional breakthroughs, including Ernst Dickmanns' vision-based vehicle in 1987 and the Predator drone in 1995, thereby extending autonomy into the domain of aviation. From 2004 to 2013, the Defense Advanced Research Projects Agency (DARPA) initiated a series of challenges that significantly accelerated the development of vehicle automation

technologies. Initially, vehicles that had failed to complete a 150-mile course by 2007 were able to successfully navigate a 60-mile urban route shortly thereafter. This development laid the foundation for the development of contemporary self-driving technologies, as reported by WIRED Brand Lab (WIRED Brand Lab., 2016).

The development of autonomous vehicles has been intricately linked to advancements in machine learning and deep learning (DL), which have led to substantial enhancements in the capacity of vehicles to discern, interpret, and respond to intricate environments. A seminal example of this pioneering work was the ALVINN project (1989), which employed a neural network to control a vehicle using camera input, thereby demonstrating the feasibility of real-time neural control despite initial skepticism (Pomerleau, 1988). In the early stages of research in this field, prominent figures such as Yann LeCun encountered challenges in implementing neural networks for computer vision purposes. These difficulties can be attributed to two main factors: first, the limited computational capabilities of the time, and second, the suboptimal training methodologies employed at the time (Krizhevsky et al., 2012). To add more, Convolutional neural networks (CNNs) have emerged as a significant advancement in the field of deep learning (DL), paving the way for the development of vehicles capable of detecting objects, recognizing traffic signs, and making navigation decisions using visual data. Bojarski and his coauthers advanced this approach by training CNNs directly on driving datasets to predict steering commands, demonstrating the practical effectiveness of deep learning (DL) in real-world driving scenarios (Bojarski et al., 2016). Recent advancements in autonomous driving have employed simulation platforms and sophisticated learning strategies to augment system capabilities. Simulators such as CARLA (Dosovitskiy et al., 2017) and VISTA (Amini et al., 2020) have played a pivotal role in accelerating research by providing safe, controlled environments for testing and training across a range of conditions. To establish a connection between simulated and real-world environments, domain adaptation methods such as domain randomization (Amini et al., 2020; Tobin et al., 2017). have been employed to enhance model robustness and generalization. Which demonstrate the efficacy of RL in optimizing driving policies in dynamic and unpredictable environments. To calrify, this study specifically examines and compares the Comma and NVIDIA CNN models for end-to-end autonomous driving. The performance of these systems was evaluated across a range of driving scenarios to assess their strengths and limitations in steering prediction, perception accuracy, and adaptability. The findings offer practical insights into the robustness and efficiency of CNN-based models, guiding the further development of reliable autonomous driving systems.

3. MATERIALS AND METHODS

3.1. Udacity Simulator

The Udacity self-driving car simulator is an open-source platform built on Unity (https://github.com/udacity/self-driving-car-sim) (Udacity2017). The system has been engineered to facilitate the training and evaluation of autonomous vehicle models. The simulator provides an interactive learning environment, enabling users to record driving data—including steering angles, speed, throttle, and reverse gear status—using a vehicle equipped with three cameras (center, left, and right) capturing at 24 frames per second (fps). The device offers two operational modes: a training mode for manual data collection and an autonomous mode for evaluating model performance. Video frames are stored as JPG images along with associated metadata, facilitating in-depth analysis and debugging. The simulator contains two tracks: the 1.1-kilometer Lake track, which is relatively elementary but incorporates curves in both directions, and the 1.5-kilometer Jungle track, which is more challenging due to its complex terrain and sharper turns. For the present study, the Lake track was selected for the initial data collection. The platform exhibits efficiency and user-friendliness, with the capacity to operate on computers with medium specifications, thereby ensuring accessibility to a broad user base. However, the system is not without its limitations. Foremost among these is its inability to automatically return the vehicle to the track after collisions. In addition, the lack of manual speed control may have a deleterious effect on the consistency of data collection. Notwithstanding these limitations, the simulator furnishes a realistic environment conducive to the generation of datasets and the evaluation of autonomous driving algorithms.

3.2. Dataset Collection and Processing

To this end, a custom dataset was created using the Udacity driving simulator to facilitate CNN-based prediction of steering angles. The vehicle was manually operated by a human driver, while three synchronized cameras (center, left, and right) captured multiple viewpoints at a rate of 24 frames per second (fps). Concurrently, the data set encompasses parameters such as steering angle, throttle position, vehicle speed, and the status of the reverse gear (Shafiee et al., 2017).. The dataset covers a series of driving sessions conducted under diverse conditions, initially comprising over 9,000 images. To mitigate redundancy and enhance model generalization, an undersampling approach was implemented, yielding 3,882 images. Subsequently, both datasets were meticulously partitioned into training (80%) and validation (20%) subsets, thereby ensuring a balanced distribution of steering angles. The preprocessing pipeline entailed the following steps: (1) cropping to focus on the road region, (2) converting images from RGB to YUV color space, (3) applying a Gaussian blur to reduce noise, and (4) resizing to 200 × 66 pixels to match the CNN input requirements. The combination of these steps resulted in enhanced feature representation, the elimination of irrelevant details, and the assurance of consistent, high-quality inputs. Consequently, the process yielded a diverse and optimized dataset that is well-suited for training and evaluating autonomous driving models.

3.3. CNN Models for Autonomous Driving

Significant advancements in artificial intelligence have been made, particularly in the field of computer vision, resulting in substantial progress in image recognition. The advent of deep learning (DL) models, particularly convolutional neural networks (CNNs), has been instrumental in this development. These networks demonstrate a high degree of proficiency in the identification of intricate patterns within visual data, while exhibiting a reduced need for parameters and enhanced efficiency in training processes, as evidenced by (Caffaratti et

al., 2019). In the domain of autonomous driving, convolutional neural networks (CNNs) have demonstrated remarkable proficiency in interpreting visual inputs from vehicle-mounted cameras, a capability that is indispensable for self-driving systems. A significant advantage of these models lies in their capability for automated feature extraction directly from raw images. By acquiring knowledge of hierarchical representations of visual data, CNNs are capable of predicting significant driving actions, such as steering angles. The capacity to process intricate visual data facilitates autonomous vehicles' ability to accurately perceive their surroundings and make real-time navigation decisions. This section underscores two prevalent CNN architectures in the domain of autonomous driving: the Comma model (Comma.ai., 2016; Santana et al., 2016) and the NVIDIA models (Bojarski et al., 2016; Lade et al., 2021; Smolyakov et al., 2018). The two architectures have been engineered to process camera inputs and predict steering angles, a pivotal component of self-driving technology. A detailed comparison of their structural differences, parameter sizes, and the ways each model is specifically optimized for steering angle prediction in autonomous vehicles will be presented.

3.3.1. Comma Architecture

The Comma architecture is a deep convolutional neural network (CNN) designed for end-to-end learning in autonomous driving. This method enables the model to directly interpret camera images and generate control commands, such as steering angles, thereby eliminating the need for manual feature design. The CNN is composed of a series of layers arranged sequentially, with each layer extracting features, reducing dimensionality, and estimating steering angles based on the learned representations. This end-to-end learning approach has been demonstrated to optimize the training process while augmenting real-time performance in autonomous driving applications. The architecture of the Comma CNN comprises multiple layers, including convolutional layers that facilitate feature extraction, a flatten layer that enables data reshaping, and dense layers that execute the decision-making process. The arrangement of layers in the Comma model is illustrated in Figure 1 and detailed in Figure 2.

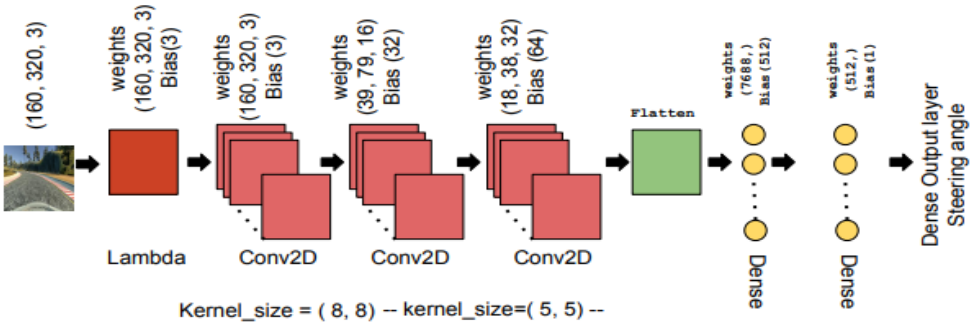

Figure 1: Overview of the Comma CNN architecture

Figure 1 is showcasing key layers for processing input images in tasks like steering angle prediction. The diagram features a Conv2D layer with specified kernels, Dense layers for classification or regression, and components like weights and biases to extract and process features efficiently. Input image sourced from the author's Udacity simulator dataset. The Comma architecture is designed to process image inputs with dimensions of 160, 320, or 3 by passing them through a series of convolutional layers. Each layer implements a series of filters on the input image, leading to a progressive reduction in spatial dimensions while concurrently augmenting the depth of the feature maps. The network's hierarchical feature extraction capability enables the identification of fundamental visual elements, including edges and textures. These elements are then integrated to generate more complex patterns at higher levels of abstraction, as previously documented by Jordan (Jordan, 2025). Also, the initial convolutional layer yields 16 feature maps, which are subsequently processed by a secondary convolutional layer comprising 32 filters. After this is a third convolutional layer comprising 64 filters. After the convolutional stages, the resulting output is flattened into a one-dimensional vector and passed through fully connected (dense) layers. These dense layers undergo a progressive decrease in size, ultimately resulting in a final layer comprising a single unit that serves to predict the steering angle. Addetionally, Figure 2 shows Comma CNN Architecture Overview. This detailed model structure highlights the layers involved in processing input images for steering angle prediction, including convolutional layers (Conv2D), flattening, and dense layers, along with the number of parameters for each.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================

 lambda (Lambda)             (None, 160, 320, 3)       0

 conv2d (Conv2D)             (None, 39, 79, 16)        3088
 conv2d_1 (Conv2D)           (None, 18, 38, 32)        12832
 conv2d_2 (Conv2D)           (None, 7, 17, 64)         51264

 flatten (Flatten)           (None, 7616)              0

 dense (Dense)               (None, 512)               3899904
 dense_1 (Dense)             (None, 1)                 513
=================================================================
Total params: 3967601 (15.14 MB)
Trainable params: 3967601 (15.14 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 2. Comma CNN Architecture Overview

The Comma model employs the Adam optimizer (Adam) to dynamically adjust the learning rate, thereby enhancing both convergence speed and overall performance. The model utilizes the mean-squared error (MSE) loss function, rendering it particularly well- suited for regression tasks, such as predicting continuous steering angles (Rao & Singer, 2017). The model, which encompasses 3.97 million parameters, is capable of capturing intricate relationships between input data, images, and the corresponding steering angle predictions. The architecture, implemented in Keras, commences with a Lambda layer that normalizes input images to the range [-1, 1]. This is followed by convolutional layers for feature extraction and fully connected layers for steering angle prediction. Figure 3 shows Python implementation of the Comma model using Keras for autonomous driving tasks. The model comprises multiple convolutional layers followed by dense layers to predict steering angles based on input images. The code normalizes the input, extracts features using convolutional layers, and uses fully connected layers for final prediction.

Python Code: Comma Model

```python
def Comma_model():
    model = Sequential([
        # Normalize input data to [-1, 1]
        Lambda(lambda x: x / 127.5 - 1.0, input_shape=(160, 320, 3)),
        # Convolutional layers for feature extraction
        Conv2D(16, (8, 8), strides=(4, 4), activation='relu'),
        Conv2D(32, (5, 5), strides=(2, 2), activation='relu'),
        Conv2D(64, (5, 5), strides=(2, 2), activation='relu'),
        # Flatten and fully connected layers
        Flatten(),
        # Hidden layer with 512 units and ReLU activation
        Dense(512, activation='relu'),
        Dense(1)  # Output layer for prediction
    ])
    # Compile the model with Adam optimizer and MSE loss
    model.compile(optimizer='adam', loss='mse')
    return model
```

Figure 3. Python implementation of the Comma model using Keras for autonomous driving tasks

3.3.2. NVIDIA Architecture

The NVIDIA architecture is a state-of-the-art deep convolutional neural network (CNN) that has been meticulously engineered for autonomous driving applications. In comparison to the Comma model, it exhibits a more intricate network configuration, comprising five convolutional layers for feature extraction, succeeded by three fully connected (dense) layers for regression. The incorporation of additional convolutional layers into the network facilitates the capture of more abstract, high-level features from raw camera images, thereby enhancing its capacity to accurately predict driving behaviors, such as steering angles, across a diverse range of scenarios. This renders the model especially efficacious in autonomous driving tasks that necessitate robustness under variable environmental conditions. The convolutional layers fulfill two primary functions: (1) progressively reducing the spatial resolution of input images, thereby enabling the network to prioritize the most salient features, and (2) increasing the depth of feature maps to capture complex and abstract patterns as processing progresses deeper into the network. Initial layers are responsible for detecting basic visual elements, such as edges and textures. In contrast, subsequent layers extract more intricate structures, including object shapes and spatial arrangements. These advanced structures are essential for accurately predicting steering angles. The final convolutional layer produces a feature map of dimensions $1 \times 21 \times 64$, which is subsequently flattened into a one-dimensional vector comprising 1,344 elements, thereby preparing it for integration into the subsequent fully connected layers. The fully connected layers are responsible for the regression task, processing the flattened feature vector through three dense layers with decreasing numbers of units: 100 units in the first layer, 50 in the second, and 10 in the third. ReLU

activation functions are applied throughout to enable the network to learn complex nonlinear mappings. The final dense layer contains a single neuron that outputs a scalar value representing the predicted steering angle, thereby guiding the vehicle's direction during autonomous operation. As demonstrated in Figures 4 and 6, and elaborated in Figure 5, the network commences with a convolutional layer comprising 24 filters with a 5 × 2 kernel. This layer is engineered to capture low-level features across a broad receptive field. Subsequent convolutional layers progressively reduce spatial dimensions by adjusting stride sizes while concurrently increasing the number of filters to enhance the richness of feature representation. This hierarchical design enables the network to efficiently extract meaningful features from input images. The NVIDIA architecture establishes a robust mapping from high-dimensional image inputs to low-dimensional steering outputs by combining convolutional layers for feature extraction with fully connected layers for regression. This design ensures strong generalization across diverse driving scenarios, supporting safe and reliable autonomous navigation. The NVIDIA architecture is characterized by 231,459 parameters, offering a relatively modest yet potent configuration for deep convolutional networks. The device's compact design ensures efficient training and deployment, even in settings with limited resources, while maintaining the ability to capture intricate and highly detailed features from input data.

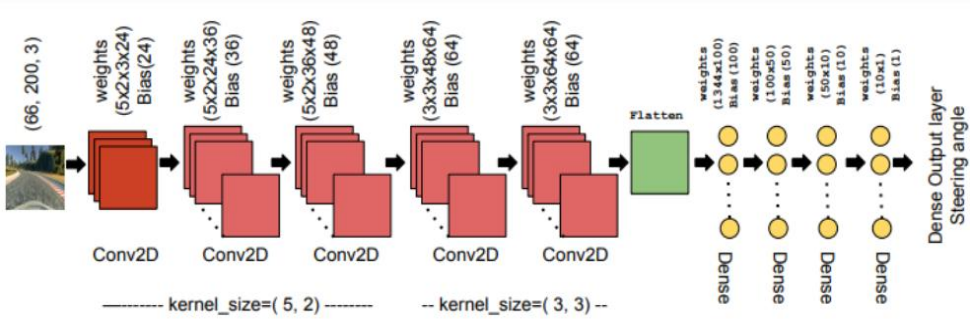

Figure 4. The NVIDIA CNN architecture for autonomous driving uses Conv2D and Dense layers to predict steering angles from input images

Key parameters (Figure 4) like kernel sizes, weights, and biases are applied in each layer. The input image is from the author's dataset, collected using the Udacity simulator. A substantial corpus of research underscores the striking parallels between the NVIDIA model and the Comma model, most notably their shared utilization of the Adam optimizer and the MSE loss function. The primary distinction, however, lies in the NVIDIA model's greater complexity, which is achieved through the inclusion of additional layers. These layers have been shown to enhance the system's capacity to discern and interpret intricate patterns within the input data. Consequently, the NVIDIA model demonstrates superior proficiency in the extraction of fine-grained features, while its comparatively efficient parameter count renders it well-suited for real-time operations, such as autonomous driving. The NVIDIA model is implemented using Keras, consistent with the framework employed in the Comma architecture. The model's architecture comprises five convolutional layers, succeeded by fully connected layers. This configuration processes input images, such as those captured by a vehicle-mounted camera, to generate steering angle predictions. The convolutional layers are responsible for gradually extracting higher-level features from raw images, while the fully connected layers combine this extracted information to generate accurate control predictions. Beyond its architecture, the NVIDIA model incorporates rigorous training practices to ensure effective learning and generalization. The model is trained on an extensive dataset comprising camera images captured under diverse driving conditions to minimize the mean squared error (MSE) between the predicted and actual steering angles. Figure 5 shows the NVIDIA model architecture for steering angle prediction includes Conv2D layers for feature extraction and Dense layers for regression. It has 231,459 trainable parameters (904.14 KB).

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================

 conv2d (Conv2D)             (None, 31, 100, 24)       744
 conv2d_1 (Conv2D)           (None, 14, 50, 36)        8676
 conv2d_2 (Conv2D)           (None, 5, 25, 48)         17328
 conv2d_3 (Conv2D)           (None, 3, 23, 64)         27712
 conv2d_4 (Conv2D)           (None, 1, 21, 64)         36928

 flatten (Flatten)           (None, 1344)              0

 dense (Dense)               (None, 100)               134500
 dense_1 (Dense)             (None, 50)                5050
 dense_2 (Dense)             (None, 10)                510
 dense_3 (Dense)             (None, 1)                 11
=================================================================
Total params: 231459 (904.14 KB)
Trainable params: 231459 (904.14 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 5. The NVIDIA model architecture for steering angle prediction

Figure 6 presents the Python implementation showcases the end-to-end DL architecture used by NVIDIA for autonomous driving. The model consists of multiple convolutional layers that perform feature extraction from input images, followed by fully connected layers that predict the steering angle of the vehicle. It is compiled with the Adam optimizer and the MSE loss function, which are key components for efficient model training.

Python Code: NVIDIA Architecture

```
def NVIDIA_Architecture():
    model = Sequential()

    # Convolutional layers for feature extraction
    model.add(Convolution2D(24, kernel_size=(5, 2), strides=(2, 2),
    ↪  input_shape=(66, 200, 3), activation='elu'))
    model.add(Convolution2D(36, kernel_size=(5, 2), strides=(2, 2),
    ↪  activation='elu'))
    model.add(Convolution2D(48, kernel_size=(5, 2), strides=(2, 2),
    ↪  activation='elu'))
    model.add(Convolution2D(64, kernel_size=(3, 3), activation='elu'))
    model.add(Convolution2D(64, kernel_size=(3, 3), activation='elu'))

    # Flatten the output for the fully connected layers
    model.add(Flatten())

    # Fully connected layers for prediction
    model.add(Dense(100, activation='elu'))
    model.add(Dense(50, activation='elu'))
    model.add(Dense(10, activation='elu'))

    # Output layer for prediction
    model.add(Dense(1))

    # Compile the model with Adam optimizer and MSE loss
    model.compile(optimizer='adam', loss='mse')
    return model
```

Figure 6. The Python implementation showcases the end-to-end DL architecture used by NVIDIA for autonomous driving

3.3.3. Comparative Analysis

Table 1 outlines key characteristics such as depth, number of parameters, activation functions, training efficiency, and primary strengths, highlighting the differences in their design and functionality for autonomous vehicle applications.

Table 1. A comparative analysis of the Comma and NVIDIA neural network architectures used in autonomous driving

Aspects	Comma	NVIDIA
Depth	3 convolutional layers	5 convolutional layers
Parameters	3.97 million	231,459
Activation Function	ReLU	ELU
Training Efficiency	Moderate	High
Key Strengths	End-to-end learning; robust design	Abstract feature extraction; compact design

The proposed study effects among Both architectures have been meticulously engineered to estimate steering angles in autonomous driving. However, a comparative analysis reveals that the NVIDIA model introduces a higher degree of complexity. This augmented intricacy is attributable to the incorporation of extra convolutional layers, which facilitate the capture of more nuanced visual motifs. Conversely, the Comma model employs a more streamlined and efficient approach, particularly well-suited for less demanding driving scenarios. The selection of one model over another is contingent upon a balanced consideration of model complexity and the requirements of the driving environment. the independent variable and dependent variable are seen in the study model, as shown in Figure 1.

4. EXPERIMENTAL RESULTS

This section presents the evaluation of two convolutional neural network (CNN) architectures: the Comma.ai model and the NVIDIA model, with a focus on their overall performance. All experiments were conducted in Python using Google Colaboratory with GPU acceleration. The hardware setup included either an NVIDIA Tesla T4 (2560 CUDA cores, 16GB VRAM) or an NVIDIA Tesla K80 (2496 CUDA cores, 12GB VRAM), verified via the nvidia-smi command. For computationally intensive tasks, the Tesla T4 was preferred. The implementation relied on several key libraries: Keras for model construction (Keras, 2023) , TensorFlow for training and deployment (Abadi et al., 2016), Scikit-learn for preprocessing and machine learning tasks, Pandas for structured data management, and NumPy for numerical computations. Also, Models were trained with a standard learning rate to ensure both stability and convergence (LeCun et al., 2015). The dataset was split into 80% for training and 20% for validation. Performance was assessed using the Mean Squared Error (MSE)

(Wikipedia contributors, 2023), which quantifies the average squared difference between predicted and actual values, serving as an objective basis for comparing model accuracy.

4.1. Training the CNN Model with Comma Architecture

The CNN model was trained using the Comma architecture on a designated dataset over the course of ten consecutive epochs. The objective of this training was to assess the model's learning capability and its overall effectiveness. The mean duration of each epoch was approximately 460 seconds, indicative of the model's efficacy in managing the dataset. This relatively brief epoch duration exemplifies the optimized design of the training procedure, thereby enabling effective learning while conserving computational resources. The recorded training time facilitated a comprehensive examination of the model's capacity to learn and adapt across epochs. Additionally, it enabled close monitoring of its progress in tuning parameters and enhancing performance. Further information, including detailed evaluation metrics and results, can be found in Figure 7, which provides a more in-depth analysis of the model's development during training. In the initial epochs, a discernible discrepancy was noted between the training loss and the validation loss, underscoring the potential for enhancement in the model's learning process. Specifically, the validation loss exceeded the training loss, indicating that the model had not yet demonstrated effective generalization to unseen data. As the training program progressed, however, the model demonstrated a significant enhancement in its performance. This enhancement was evident in the progressive decline of both training and validation losses, suggesting that the model was becoming more adept at discerning patterns and generalizing from the dataset over time. Despite minor fluctuations in the loss values across epochs, the overall trend exhibited a consistent downward trajectory. By the tenth epoch, the training loss had reached its minimum value of 0.1487, while the validation loss was close behind at 0.1545, as illustrated in Figure 8. Training results of a CNN model using the Comma architecture for autonomous driving showed in Figure 7. The figure displays the loss and validation loss metrics over ten epochs of training, showing the model's progress in reducing error during the optimization process. The experiment was conducted using Python and is based on research experiments for autonomous driving applications.

```
Epoch 1/10
300/300 [==============================] - 503s 2s/step - loss: 0.1854 - val_loss: 0.1597
Epoch 2/10
300/300 [==============================] - 454s 2s/step - loss: 0.1522 - val_loss: 0.1581
Epoch 3/10
300/300 [==============================] - 505s 2s/step - loss: 0.1543 - val_loss: 0.1589
Epoch 4/10
300/300 [==============================] - 491s 2s/step - loss: 0.1536 - val_loss: 0.1567
Epoch 5/10
300/300 [==============================] - 449s 1s/step - loss: 0.1529 - val_loss: 0.1538
Epoch 6/10
300/300 [==============================] - 447s 1s/step - loss: 0.1529 - val_loss: 0.1577
Epoch 7/10
300/300 [==============================] - 427s 1s/step - loss: 0.1548 - val_loss: 0.1514
Epoch 8/10
300/300 [==============================] - 456s 2s/step - loss: 0.1525 - val_loss: 0.1561
Epoch 9/10
300/300 [==============================] - 461s 2s/step - loss: 0.1518 - val_loss: 0.1559
Epoch 10/10
300/300 [==============================] - 451s 2s/step - loss: 0.1487 - val_loss: 0.1545
```

Figure 7. Training results of a CNN model using the Comma architecture for autonomous driving

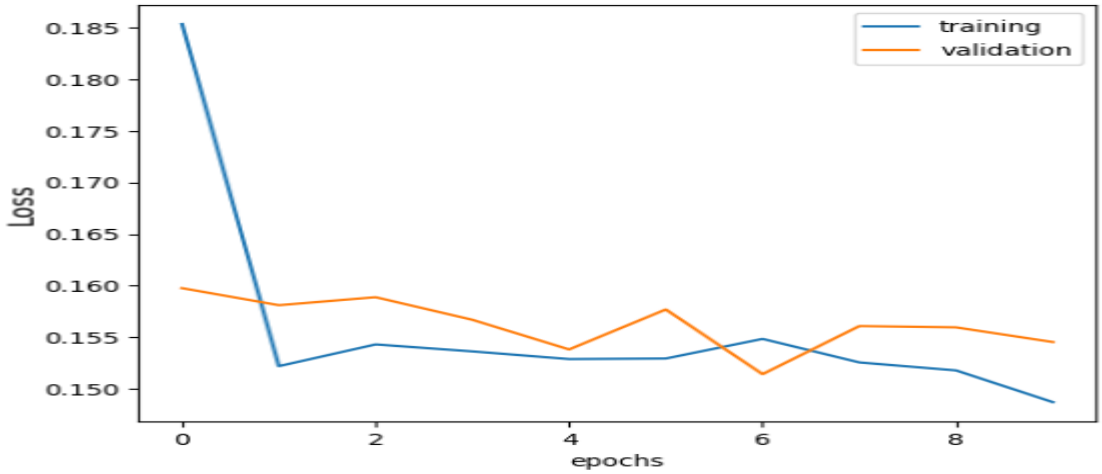

Figure 8. Training performance of the CNN designed to predict steering angles in autonomous driving systems, based on the Comma architecture

Figure 8 showcases the model's progress throughout the training process, highlighting how the CNN learns to optimize predictions for steering angles from input images. The experiment was conducted using Python, based on research aimed at improving autonomous driving models, and is based on the original dataset. The findings indicate a substantial decline in error rates, suggesting that the model exhibits satisfactory performance on the validation set. However, it must be acknowledged that this may not fully capture the model's potential performance, as training a CNN on NVIDIA GPUs could potentially yield more favorable results.

4.2. Training the CNN Model with NVIDIA Architecture

Subsequent to the preliminary training using the Comma architecture, the CNN model was trained on the same dataset with the NVIDIA architecture for an additional ten epochs. The results of this stage, along with the detailed loss values, are presented in Figure 9 and Figure 10. During the initial epoch, the training loss exhibited an initial value of 0.1790 and demonstrated a substantial decrease by the tenth epoch, reaching 0.0815. This reduction is indicative of the model's progressive enhancement in minimizing the training loss as it learns from the dataset.

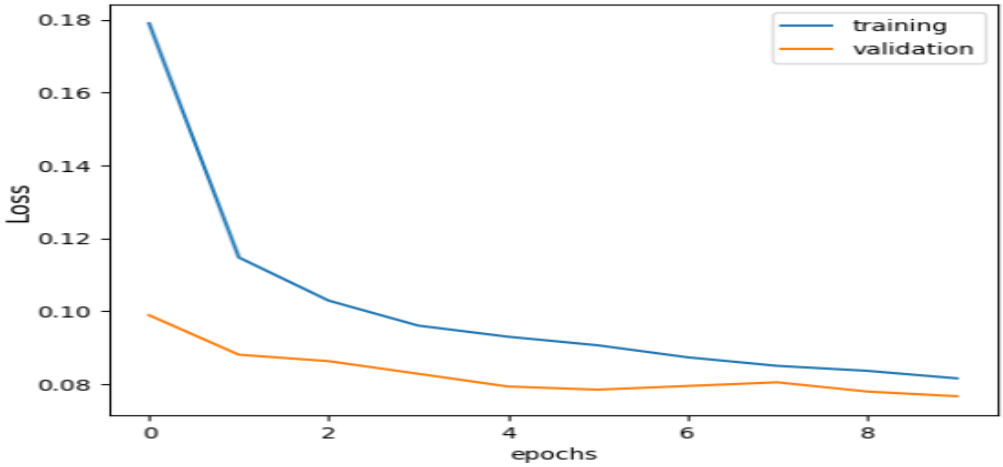

Figure 9. Training performance of the CNN designed to predict steering angles for autonomous driving, based on the NVIDIA architecture.

Figure 9 demonstrates the model's learning process and its ability to optimize predictions for steering angles using CNN layers. The experiment was conducted using Python, as part of research focused on advancing autonomous driving systems, and is based on the original dataset. The validation loss, which ranged from 0.0766 to 0.0989 during the training process, suggests that the model exhibited effectiveness in enhancing its performance across epochs. These results suggest that the model not only improved steadily during training but also successfully minimized loss, thereby demonstrating the effectiveness of the training process. The observed fluctuations in validation loss are indicative of a well-balanced generalization, thereby underscoring the model's increasing capacity to adapt to the dataset over time. Moreover, the findings from this stage of the dissertation provide valuable insights into CNN model training, particularly when leveraging NVIDIA architecture. The model underwent training over the course of ten iterations, with each iteration corresponding to a distinct phase of learning. Its performance was monitored through training accuracy and loss, offering a detailed view of the model's learning progression. Although the model was not strictly constrained to ten epochs, performance evaluation was predominantly informed by the training loss metric (see Figure 9). A thorough examination of the findings necessitates a comparison of the training and validation losses across epochs. At the initial epoch, a significant disparity was observed between the training loss (0.1790) and the validation loss (0.0989). This finding indicates that while the model was beginning to learn the training data, it had not yet demonstrated effective generalization capabilities to the validation set. To add more, in the subsequent epochs, both training and validation losses exhibited a consistent decline, indicative of enhanced model learning capabilities. By epoch 2, the training loss decreased to 0.1147, and the validation loss decreased to 0.0880. By epoch 3, the training loss was 0.1029, and the validation loss reached 0.0863, indicating a gradual yet consistent reduction. This trend of progressive improvement persisted through subsequent epochs. As the training process unfolded, a notable trend emerged: the discrepancy between the training and validation losses began to diminish and stabilize over time. By the tenth epoch, the training loss had decreased to 0.0815, while the validation loss reached 0.0766. This convergence suggests that the model effectively generalized to the validation data, a hallmark of a balanced model that circumvents both overfitting and underfitting. Overall, the model exhibited consistent enhancement in both training and validation performance metrics. The negligible discrepancy between the two losses in subsequent epochs suggests that the model learned efficiently without overfitting. In summary, the experiment demonstrated that the model's capacity to generalize to unseen data increased progressively during the training process, culminating in a robust and effective final model that is well-suited for the designated task. Figure 10 presents the model's loss and validation loss metrics over ten epochs, illustrating how the model improves its performance with each iteration. This experiment was conducted using Python as part of the research focused on optimizing CNN models for autonomous driving tasks.

```
Epoch 1/10
300/300 [====================] - 309s 1s/step    - loss: 0.1790 - val_loss: 0.0989
Epoch 2/10
300/300 [====================] - 295s 986ms/step - loss: 0.1147 - val_loss: 0.0880
Epoch 3/10
300/300 [====================] - 295s 984ms/step - loss: 0.1029 - val_loss: 0.0863
Epoch 4/10
300/300 [====================] - 296s 987ms/step - loss: 0.0960 - val_loss: 0.0828
Epoch 5/10
300/300 [====================] - 286s 956ms/step - loss: 0.0930 - val_loss: 0.0793
Epoch 6/10
300/300 [====================] - 279s 930ms/step - loss: 0.0906 - val_loss: 0.0784
Epoch 7/10
300/300 [====================] - 333s 1s/step    - loss: 0.0873 - val_loss: 0.0794
Epoch 8/10
300/300 [====================] - 251s 837ms/step - loss: 0.0850 - val_loss: 0.0804
Epoch 9/10
300/300 [====================] - 316s 1s/step    - loss: 0.0836 - val_loss: 0.0779
Epoch 10/10
300/300 [====================] - 245s 816ms/step - loss: 0.0815 - val_loss: 0.0766
```

Figure 10. Training results of a CNN model with the NVIDIA architecture for predicting steering angles in autonomous driving systems

5. CONCLUSION

This study examines advanced CNN-based approaches for autonomous driving using simulators such as Udacity. Emphasis is placed on data quality, preprocessing, and augmentation, which are shown to significantly enhance model performance. Networks trained on diverse, high-quality datasets demonstrate strong generalization across varied driving scenarios. A comparative analysis of Comma and NVIDIA architectures reveals distinct strengths: NVIDIA excels at capturing complex image patterns, while Comma is more efficient in simpler environments and requires less training time. The study also highlights the critical role of optimization algorithms in balancing accuracy and computational efficiency. Results indicate that combining high-fidelity data, effective preprocessing, optimal architecture selection, and suitable optimization strategies yields robust and reliable performance in autonomous driving simulations. These findings provide actionable insights for developing safe, adaptable, and efficient autonomous vehicle systems, emphasizing the importance of ongoing research in data quality, architectural design, and optimization methods.

REFERENCES

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... & Zheng, X. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:1603.04467. 1-19.

Amini, A., Gilitschenski, I., Phillips, J., Moseyko, J., Banerjee, R., Karaman, S., & Rus, D. (2020). Learning robust control policies for end-to-end autonomous driving from data-driven simulation. IEEE Robotics and Automation Letters, 5(2), 1143-1150.

Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., ... & Zieba, K. (2016). End to end learning for self-driving cars. arXiv preprint arXiv:1604.07316. 1-9.

Caffaratti, G. D., Marchetta, M. G., & Forradellas, R. Q. (2019). Stereo matching through squeeze deep neural networks. Inteligencia Artificial, 22(63), 16-38.

Comma.ai. (2016). *research* [Source code]. GitHub. https://github.com/commaai/research. Accessed 13 March 2025.
Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., & Koltun, V. (2017). CARLA: An open urban driving simulator. In Conference on robot learning (pp. 1-16). PMLR.

Jordan, J. (2025). Convolutional neural network architectures. *Jeremy Jordan*. Retrieved January 15, 2025, from https://www.jeremyjordan.me/convnet-architectures/

Keras. (2023). *Keras.* https://keras.io

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems, 25, 1-9.

Lade, S., Shrivastav, P., Waghmare, S., Hon, S., Waghmode, S., & Teli, S. (2021, March). Simulation of self driving car using deep learning. In 2021 international conference on emerging smart computing and informatics (ESCI) (pp. 175-180). IEEE.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. nature, 521(7553), 436-444.

Pomerleau, D. A. (1988). Alvinn: An autonomous land vehicle in a neural network. Advances in neural information processing systems, 1, 305-313.

Rao, C. R., & Singer, B. (2017). *Regression analysis and its applications: A data-oriented approach* (2nd ed.). Springer. https://doi.org/10.1007/978-3-319-70952-1

Santana, E., & Hotz, G. (2016). Learning a driving simulator. arXiv preprint arXiv:1608.01230. 1-8.

Shafiee, M. J., Chywl, B., Li, F., & Wong, A. (2017). Fast YOLO: A fast you only look once system for real-time embedded object detection in video. arXiv preprint arXiv:1709.05943.

Smolyakov, M. V., Frolov, A. I., Volkov, V. N., & Stelmashchuk, I. V. (2018). Self-driving car steering angle prediction based on deep neural network an example of CarND udacity simulator. In 2018 IEEE 12th international conference on application of information and communication technologies (AICT) (pp. 1-5). IEEE.

Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., & Abbeel, P. (2017). Domain randomization for transferring deep neural networks from simulation to the real world. In 2017 IEEE/RSJ international conference on intelligent robots and systems (IROS) (pp. 23-30). IEEE.

Udacity (2017). *An open source self-driving car*. https://www.udacity.com/self-driving-car. Accessed 15 Feburary 2025.

Wikipedia contributors. (2023). *Mean squared error*. In *Wikipedia*. https://en.wikipedia.org/wiki/Mean_squared_error

WIRED Brand Lab. (2016, March). A brief history of autonomous vehicle technology. *WIRED*. https://www.wired.com/brandlab/2016/03/a-brief-history-of-autonomous-vehicle-technology/

.