Application Based Self-Driving Car Simulator Using Modified CNN Nvidia Model

Dr. Poovarasan Selvaraj, Ajithkumar R
Assistant Professor, Department of CS (AI&DS), Sri Ramakrishna College of Arts & Science, Combatore-641 006
UG Student, Department of CS (AI&DS), Sri Ramakrishna College of Arts & Science, Combatore-641 006

Abstract - Self-driving car system using NVIDIA's Convolutional Neural Network (CNN) model, which maps raw pixel data from a single front-facing camera directly to steering commands. Leveraging deep learning, we trained the model on a diverse dataset encompassing city streets, highways, and off-road terrains. The simulation showcases the model's ability to autonomously navigate complex driving conditions without traditional components like lane detection, path planning, or control algorithms. Instead, the model learns to interpret and respond to road features and driving scenarios solely from human steering inputs during training. Our findings highlight the benefits of an end-to-end learning approach, where the CNN optimizes the entire driving task integrally, achieving robust performance across various driving contexts. This method potentially enhances efficiency and effectiveness over traditional autonomous driving systems, demonstrating the feasibility of streamlined, deep learning-based solutions for self-driving technology.

Keywords: DAVE-2 System, CNN, CNN Model, Autonomous Land Vehicle in a Neural Network (ALVINN) system.

I. INTRODUCTION

The emergence of Convolutional Neural Networks (CNNs) marks a significant advancement in pattern recognition, particularly within image analysis. Departing from conventional methods reliant on manual feature extraction and subsequent classification, CNNs have transformed the field by autonomously learning features directly from raw data. This paradigm shift has shown exceptional efficacy in tasks such as image recognition, leveraging the intrinsic capability of convolution operations to capture intricate spatial relationships within images [1]. Moreover, the availability of vast labeled datasets such as the Large-Scale Visual Recognition Challenge (ILSVRC) and the computational power of modern graphics processing units (GPUs) have propelled CNNs to the forefront of machine learning in recent years.

The capabilities of CNNs to address the complex challenge of autonomous driving. Our journey began over a decade ago with the DARPA Autonomous Vehicle (DAVE) project, an initial endeavor to train a scaled-down radio control car to navigate challenging environments based on human-driven data. This pioneering effort laid the groundwork for end-to-end learning in autonomous driving, drawing inspiration from works such as Pomerleau's Autonomous Land Vehicle in a Neural Network (ALVINN) system.

II. LITERATURE REVIEW

(N. Corporation, 2016) The rapid development of automotive technology is largely due to advances in deep learning, particularly through the use of Convolutional Neural Networks. NVIDIA, among the pioneers in this field, has successfully used CNNs for autonomous driving and embedded these models into their proprietary driving frameworks to improve perception, planning and control performance in autonomous vehicles. This review focuses on basics, development steps, and current techniques for driving using the NVIDIA model. The introduction of Convolutional Neural Networks heralded a revolutionary era in image analysis and pattern recognition. Traditional image processing methods relied heavily on manual extraction, requiring careful planning of individual elements and then insertion in stages. CNNs have revolutionized this paradigm by enabling automatic learning of sequences from pixel data. This unique ability to independently learn and extract features allows CNNs to detect complex

patterns and structures in images, making them highly effective at tasks such as image recognition and object recognition.

(Nicolas Gallardo,2017) The context of autonomous driving, these networks are essential for processing large amounts of sensor data, which form the underlying technology that allows the sensory system to recognize and respond to road users, obstacles and road signs. The path to autonomous driving using neural networks may begin with early initiatives such as DARPA's Autonomous Vehicles project. The project aimed to train radio-controlled vehicles to navigate the environment using data obtained from human-driven vehicles. These early efforts showed that neural networks could be taught to mimic human behavior, paving the way for advanced systems. Based on this basic project, NVIDIA launched, the latest model of the automotive industry and a leader in its field. NVIDIA Pilot Net architecture is considered a significant development in this field; It trains CNNs to map raw pixel data from camera input to control commands.

(A. Pomerleau,1989) This approach is quite different from traditional modular networks, which consist of performing various functions such as sensing, planning and control. Instead, end-to-end learning facilitates the process of learning the direct relationship between cognitive processes and control processes, creating a more efficient and comprehensive decision-making process. NVIDIA's DRIVE is a complete platform designed to simplify the development and deployment of autonomous systems. The platform combines powerful computing hardware with software and hardware to create powerful automation solutions. Using deep learning and artificial intelligence, the platform processes input from a range of sensors including cameras, LiDAR and radar to create detailed information about vehicles and make driving decisions.

III. OVERVIEW OF THE DAVE-2 SYSTEM

A simplified block diagram of the DAVE-2 training data gathering system is depicted in Figure 1. The data acquisition vehicle is equipped with three cameras installed behind the windshield. Simultaneously, the system records the steering angle applied by the human driver along with timestamped video from the cameras. The steering command is obtained by connecting to the Controller Area Network (CAN) bus of the car. To ensure system independence from the car's geometry, the steering command is encoded as 1/r, Where represents the turning radius in meters. To prevent singularity issues when driving straight, 1/r is utilized instead of r, as the turning radius for driving straight is infinite. This encoding smoothly transitions from left turns to right turns through zero. The training data consists of a single dataset [2].

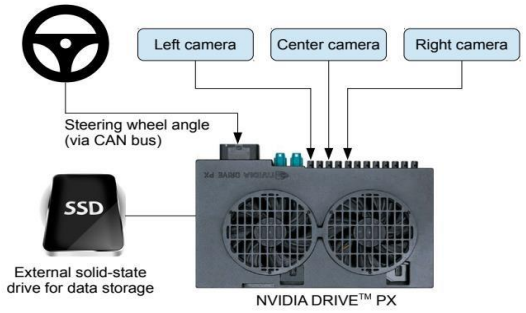

Figure 1: Data collection system.

The left and right cameras provide images of two specific features from the center. Changes between the camera and rotation are equivalent to changing the image from the nearby camera. Since we lack 3D modeling skills, we estimate the change by assuming all points below the horizontal line are on the ground and all points above the horizon are far away. This approach works well on flat ground but distorts ground-based objects like cars, poles, trees, and buildings. Fortunately, these distortions don't significantly impact network training. The vehicle's video controlled navigation system returns the car to its target position and orientation within two seconds. To achieve this, we employ a Convolutional Neural Network (CNN) in our autonomous vehicle training system, as shown in Figure 2. The CNN simulates the proposed rule by processing images and adjusting its weights to match the desired output. This is done using the backpropagation algorithm, implemented in the Torch 7 machine learning package.

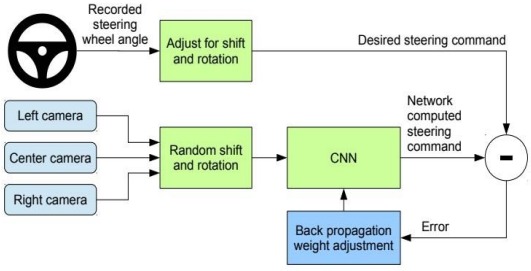

Figure 2: Training the neural network.

IV. NETWORK ARCHITECTURE

The train our network weights to minimize the maximum error between the control command executed by the network and the command issued by the human driver or the modified command for off-center and rotating images (see Section 5.2). Our storage network is illustrated in Figure 4. The network consists of nine components, including a standard level, five certified components, and three fully integrated components. The input image is segmented in the YUV plane and passed through the array. The first stage of the network is to create standard images. The standard is robust and not adapted to learning. Network normalization enables regular programs to be customized with network mode and prioritized via the GPU. The validation component is designed for performance and is selected through a series of multivariate tests. We employ an irregular convolutional neural network with a 2×2 pitch and 5×5 and 5×5 variable resolution in the first three layers, and a 3×3 size in the last two layers.

Follow a five-phase verification and three-phase integration process, leading to the radio variable output control value. All connected components are designed to work as the controller, but when training the system end-to-end, we find that it is challenging to make a clear distinction between the components of the network that essentially act as attractors to get the system to work, and those that act as controllers [3].

V. DATA COLLECTION

The training data was gathered by driving an electric vehicle on various roads and in different weather conditions. Most of the data was collected in central New Jersey, with additional data from Illinois, Michigan, Pennsylvania, and New York. The dataset includes a range of roads, such as freeways (paved and unpaved), residential streets with parking, tunnels, and unpaved freeways.

Data was collected in various weather conditions, including fair weather, cloudy, snow, and rain, during both day and night. In some cases, the low-lying sun caused light to reflect off the road and scatter through the glass.

Data was obtained using wireline measurement instruments, specifically a 2016 Lincoln MKZ or a 2013 Ford Focus, with cameras mounted in the same location.

The system is car-agnostic, allowing it to be used with any vehicle make or model. Drivers were instructed to exercise caution when necessary, driving normally otherwise. By March 28, 2016, approximately 72 hours of driving data had been collected.

The function takes in a dataset data and an optional parameter display. It creates a histogram of the steering angles in the dataset using np. histogram, and then plots the histogram and a line indicating the target number of samples per bin if display is True.

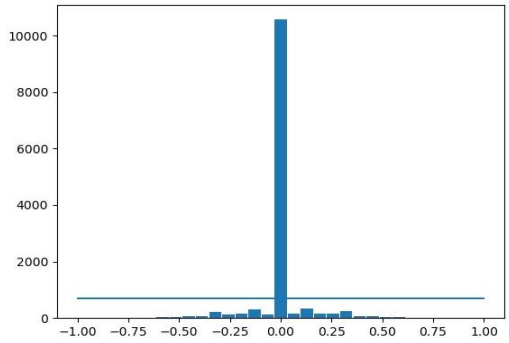

Figure 3: Data collection

The code is designed to balance a dataset by removing excess images from each bin of steering angles. The balance data function is called again at the end to ensure the dataset is balanced after removing images [4].

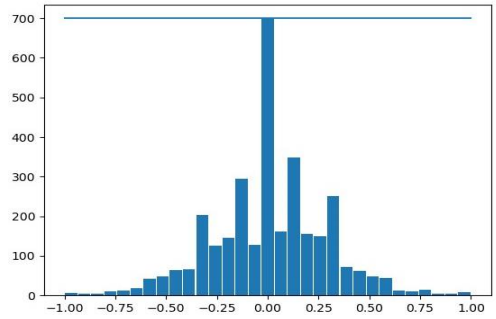

Figure 4: Remove data

Reprocessing data:

The augment image function applies random image augmentations to an input image, including panning, zooming, brightness adjustment, and flipping, and returns the augmented image along with the modified steering angle. The function reads an image from a file path, then randomly applies transformations with a 50% chance of execution for each. These transformations include panning by up to 10% in both

x and y directions, zooming by up to 20%, adjusting brightness by a random factor between 0.2 and 1.2, and flipping the image horizontally. The steering angle is also modified by negating its value when the image is flipped. The function returns the augmented image and the modified steering angle, which can be used to increase the diversity of a dataset and improve the robustness of machine learning models.

Figure 5: CROPPED IMAGE

The preprocess function takes an image as input and applies a series of transformations to prepare it for use in a machine learning model. First, it crops the image to remove the top 60 rows and bottom 25 rows, focusing on the central region of the image. Next, it converts the image from RGB to YUV color space, which is more suitable for image processing tasks. The image is then blurred using a Gaussian filter to reduce noise and smooth out the features. After that, the image is resized to a fixed size of 200x66 pixels to ensure consistency across all images. Finally, the pixel values are normalized by dividing by 255, which helps to prevent features with large ranges from dominating the model. The preprocessed image is then returned, ready for use in training or testing a machine learning model [5].

Figure 6: PREPROCESSED IMAGE

VI. MODEL

The first select relevant data from our annotated dataset, which includes road type, weather conditions, and driver activities. The only use data where the driver is staying in a lane and discard the rest. The video is then sampled at 10 frames per second (FPS). A higher sampling rate would result in redundant, similar images, providing little useful information [6].

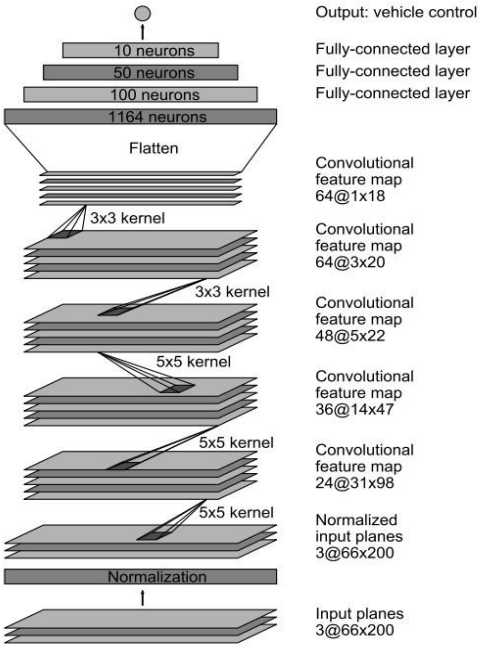

Figure 7: CNN architecture.

The network has about 27 million connections and 250 thousand parameters.

Normalization Layer: The first step is to normalize the input data. This is done by dividing the input values by 127.5 and then subtracting 1. This normalization step helps to scale the input values to a range that is suitable for the model to process.

Convolutional Layers: The next step is to apply three convolutional layers with the following specifications:

- Layer 1: 24 filters, 5x5 kernel, stride 2 • Layer 2: 36 filters, 5x5 kernel, stride 2
- Layer 3: 48 filters, 5x5 kernel, stride 2

These convolutional layers are designed to extract features from the input data. The 5x5 kernel size is used to capture local patterns in the data, and the stride of 2 is used to down sample the feature maps.

- Layer 4: 64 filters, 3x3 kernel, stride 1
- Layer 5: 64 filters, 3x3 kernel, stride 1

These convolutional layers are designed to extract more complex features from the data. The 3x3 kernel size is used to capture smaller patterns in the data, and the stride of 1 is used to preserve the spatial resolution of the feature maps.

Flatten Layer: The next step is to flatten the output of the convolutional layers into a 1D feature vector. This is done to prepare the data for the fully connected layers.

Fully Connected Layers: The next step is to apply three fully connected layers with the following specifications:

- Layer 6: 100 neurons
- Layer 7: 50 neurons
- Layer 8: 10 neurons

These fully connected layers are designed to learn complex relationships between the input features and the output steering angle.

Final Output Layer: The final step is to apply a single neuron output layer that produces the predicted steering angle. This layer takes the output of the previous fully connected layer and produces a single value that represents the predicted steering angle [7].

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 31, 98, 24)        1824
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 14, 47, 36)        21636
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 5, 22, 48)         43248
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 3, 20, 64)         27712
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 1, 18, 64)         36928
_________________________________________________________________
flatten (Flatten)            (None, 1152)              0
_________________________________________________________________
dense (Dense)                (None, 100)               115300
_________________________________________________________________
dense_1 (Dense)              (None, 50)                5050
_________________________________________________________________
dense_2 (Dense)              (None, 10)                510
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 11
=================================================================
Total params: 252,219
Trainable params: 252,219
Non-trainable params: 0
_________________________________________________________________
```

Figure 8: MODEL SEQUENCE

The training process and validate by using the epochs. Here we perform cycle of 30 epochs and each batch contains 100 elements which helps in decreasing the loss which is shown in fig.

```
Epoch 1/10
300/300 [==============================] - 125s 417ms/step - loss: 0.0639 - val_loss: 0.0373
Epoch 2/10
300/300 [==============================] - 132s 439ms/step - loss: 0.0501 - val_loss: 0.0342
Epoch 3/10
300/300 [==============================] - 120s 400ms/step - loss: 0.0467 - val_loss: 0.0328
Epoch 4/10
300/300 [==============================] - 119s 398ms/step - loss: 0.0427 - val_loss: 0.0337
Epoch 5/10
300/300 [==============================] - 124s 415ms/step - loss: 0.0409 - val_loss: 0.0295
Epoch 6/10
300/300 [==============================] - 128s 427ms/step - loss: 0.0382 - val_loss: 0.0296
Epoch 7/10
300/300 [==============================] - 119s 398ms/step - loss: 0.0378 - val_loss: 0.0280
Epoch 8/10
300/300 [==============================] - 120s 402ms/step - loss: 0.0376 - val_loss: 0.0308
Epoch 9/10
300/300 [==============================] - 120s 399ms/step - loss: 0.0354 - val_loss: 0.0267
Epoch 10/10
300/300 [==============================] - 121s 402ms/step - loss: 0.0357 - val_loss: 0.0267
Model Saved
```

Figure 9: EPOCH WITH LOSS VALUE

VII. RESULT

Simulate driving scenarios using a Convolutional Neural Network (CNN) architecture. The CNN model employs the Exponential Linear Unit (ELU) activation function, which is known for its ability to introduce non-linearity into the model. The mean squared error (MSE) loss function is used to validate the data. As expected, the performance of the model improves after training the data. Notably, the values of the trained and tested data are very close, leading us to conclude that the model generalizes well and captures the underlying patterns in the data.

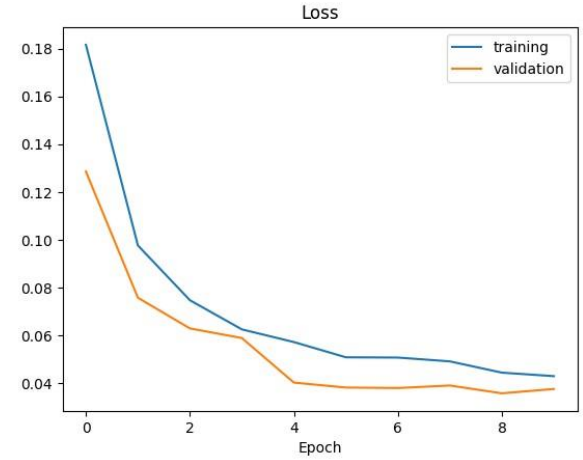

Figure 10: TRAINING AND VALIDATION

The results show that the car is successfully running on its track, with a speedometer visible in the right-hand side corner. The trained model is able to control the car effectively in different, unknown tracks, consistently completing laps without failing. Furthermore, with a larger training dataset that includes a variety of scenarios, the model's ability to remain in autonomous mode is expected to increase [8].

Figure 11: DRIVING IN AUTONOMOUS MODE

VIII. CONCLUSION

This paper presents a methodology for self-governing driving under stimulated conditions, leveraging deep learning strategies and end-to-end learning to achieve vehicle cloning. The Nvidia neural network serves as the core framework for the driver cloning algorithm, comprising five convolutional layers, one normalization layer, and four fully connected layers. The output of the model is the steering angle. The results demonstrate successful autonomous driving along a predefined stimulated path, using smaller datasets for training. Notably, all the data required to train the system are independently created in manual mode, thereby generating their own databases. To improve our method, we can focus on enhancing stimulus generalization. The limited generalizability of our approach is attributed to the small database, which restricts its applicability to real-world scenarios. Nevertheless, the car is currently performing well in autonomous mode along a predefined stimulated route.

REFERENCE

[1] Nicolas Gallardo, “Autonomous Decision Making for a Driver-less Car”, 2017.
[2] Naveen S Yeshodara, 2Nikhitha Kishore, “Cloud Based Self Driving Cars”, 2014.
[3] Joshi and M. R. James, “Generation of accurate lane-level maps from coarse prior maps and lidar,” 2014.
[4] Qudsia Memon, Shahzeb Ali, Wajiha Shah, “Self-Driving and DriverRelaxing Vehicle”, 2016.
[5] Dean A. Pomerleau. ALVINN, an autonomous land vehicle in a neural network. Technical report, Carnegie Mellon University, 1989.
[6] A. S. I. R. Jean-Francois Bonnefon, "The social dilemma of autonomous vehicles”, 2015.
[7] U. o. M. Centre for Sustainable Systems, "Autonomous Vehicles Factsheets”, 2018.
[8] N. Corporation, "End to End Learning of Self-Driving Cars”, 2016.