Advances in Robotics & Mechanical Engineering

DOI: 10.32474/ARME.2023.04.000181

ISSN: 2643-6736

Research Article

Deep CNN End-to-End Learning for Autonomous RC Cars

Muhammad Raheel Bhutta*

Department of Electrical and Computer Engineering, University of UTAH Asia Campus, Incheon, South Korea

***Corresponding author:** Muhammad Raheel Bhutta, Department of Electrical and Computer Engineering, University of UTAH Asia Campus, Incheon, South Korea

Received: August 04, 2023 **Published:** **August 26, 2023**

Abstract

The following paper describes the implementation of deep neural networks on a small-scale car platform. The car module which takes images from a single frontal camera is trained by deep convolutional neural networks (CNN) in an end-to-end manner to produce the final outputs in the form of steering angle and throttle. Both physical and virtual parts are created to check the real performance of the project and to compare the difference between them. Robotic Operating System (ROS) on Raspberry Pi 3+ made it possible to run and test keras models with the TensorFlow backend to check whether it is possible to use a single camera sensor to control and keep the car on the road (in our case on the track map).

Keywords: Artificial intelligence; deep learning; convolutional neural networks; end-to-end learning; self-driving car; computer vision

Introduction

Throughout history, humanity has tried to find better ways of transportation. A significant number of researchers with quick development of the technologies in computer vision and machine learning made major progress in achieving autonomous driving. Today, there is a high possibility to see how industry leaders are trying to make their cars fully automated. The statistics show that approximately 1.35 million people die each year because of road traffic crashes. While road traffic injuries are the leading cause of death for children and young adults aged from 5 to 29 years old [1]. A potential revolution of self-driving cars can decrease the number of fatalities caused by road accidents. With the help of modern computers which can store vast amounts of data and computer vision techniques which can process big data, researchers and industrial companies are achieving amazing results. One of the commonly used deep learning techniques known as Convolutional Neural Networks (CNNs) has already been implemented in many self-driving cars to check the robustness of the system. Convolutional Neural Networks are considered supervised learning as the neural nets are being fed a huge amount of data and being taught before taking any independent actions. It is almost like teaching your son or daughter to ride a bicycle, first you show them how to correctly ride it and later let them try to do it independently. However, with neural networks, it is a bit different. To train the neural nets you need to collect a big amount of quality data if you want to get better results. There are many ways of collecting such data. For instance, Tesla captured images and telemetry data from customer vehicles [2]. The data taken under different conditions can perform better results, because there is no guarantee that our driving will be on a smooth or straight way in a sunny bright day. The goal of our project is to collect low cost, high-quality data for inexpensive physical test platforms to get better control over our car model with robust system. The main input of our self-driving car will be real time video captured by a single frontal camera placed on the top, which later provides the system with real time data to feed our Convolutional layers to predict probable steering angle and throttle. All these data consumption, processing and finalizing the output will be conducted implemented end to end learning. To do so, we have implemented a virtual simulation of our car model as well as a virtual environment to derive the difference between the real world and simulated ones. However, the results can be slightly or totally different because of the environmental influence of the real world compared to the virtual. The major factors can be the lightning of the scene in our case the place where the track map is placed, also the flatness of the land, and other environmental factors, while in our virtual environment, all those factors will be set by default and remain the same during training and learning sessions.

Related Work

The first implementation of neural networks in an end-to-end manner was in 1980 [3]. ALVINN (Autonomous Land Vehicle in a Neural Network) used fully connected neural networks with 3 layers to keep itself on the tracking road. A single camera mounted on top of the car and a laser finder was taking images as input and produced the direction the vehicle should follow. In 2000, DARPA tested it on a small-scale 50cm off-road truck (DAVE) [4]. Two forward-pointing wireless color cameras fed the images to the 6-layered CNN. However, the system was remote, where a computer was processing the video and sending the generated output to the truck via radio. During their research on end-to-end learning for self-driving cars, a group of NVIDIA researchers trained Convolutional Neural Networks to get steering commands from a single front-facing camera [5]. With a minimum number of training data acquired from humans, the car could perform well in traffic on local roads with or without lane markings, in highways. Surprisingly, the system could make correct decisions even during unclear visual guidance like parking lots and unpaved roads. NVIDIA DevBox and Torch 7 were used for data training sessions. While the system runs on NVIDIATM PX self- driving car computer with 30 frames per second (FPS), which is quite high. Another amazing project made by Joshua E Siegel et al. [6], where Convolutional Neural Networks were implemented on a radio-controlled small-scale vehicle, which operates on the Robotic Operating System (ROS) [7]. To run the same model without modifications both virtual and physical environments were created resemble to each other. After the training process on simulated environment the model was successfully transferred to the physical environment without any changes to check the robustness of the model on gamified simulator and physical platform. Meanwhile, Qi Zhang et al. neglected to train their data in an end-to-end manner to control their self-driving car directly from raw sensory data but chose deep reinforcement learning instead to let the car take free decisions like a human [8]. During this process an agent (RC car) for taking correct decision gets a reward, while for negative one gets penalty. However, Deep Q-learning used for the project may cause damage to the car as it is taking free, uncontrolled decisions.

Simulation in Virtual Environment

Objects Creation

Firstly, to create the car object, we used modeling software Blender 3D [9] to model the different parts of the car (Figure 1). Then, we moved the modeled objects to a game engine environment. Using the virtual environment of Unity 3D platform [10], we created a terrain and a road for our modeled car (Figure 2). The physics of the car and the lighting conditions were adjusted. We defined the behavior and the dynamics of the car with scripts using C# programming language. Our Camera object has 3 sub objects as: Camera follow (the main camera) which follows and shows the car object from its rear, Camera helper for extra actions and Camera sensor for data generation. The car model includes car basic throttle, static speed and other physics, which includes Collision sensors as well. The car can be AI controlled (self-driving mode) or manually controlled with joystick (manual mode).

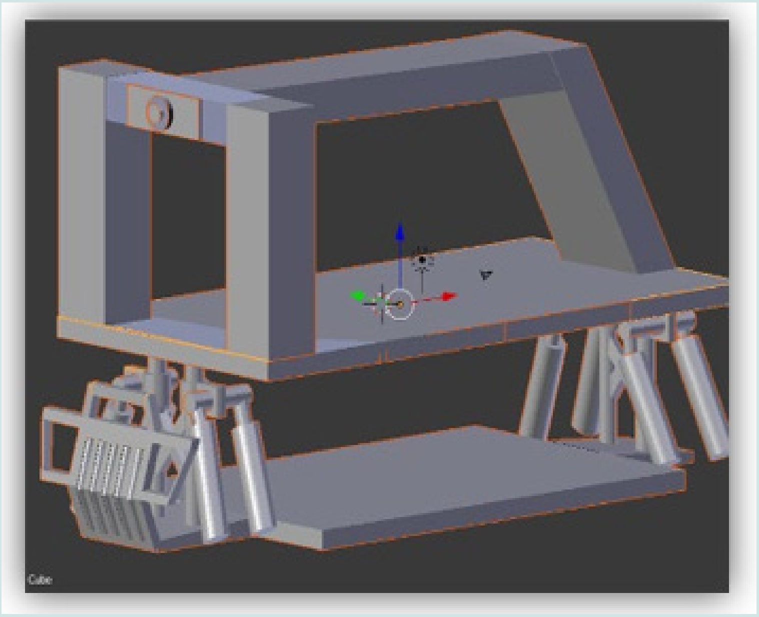

Figure 1: 3D object of a chassis.

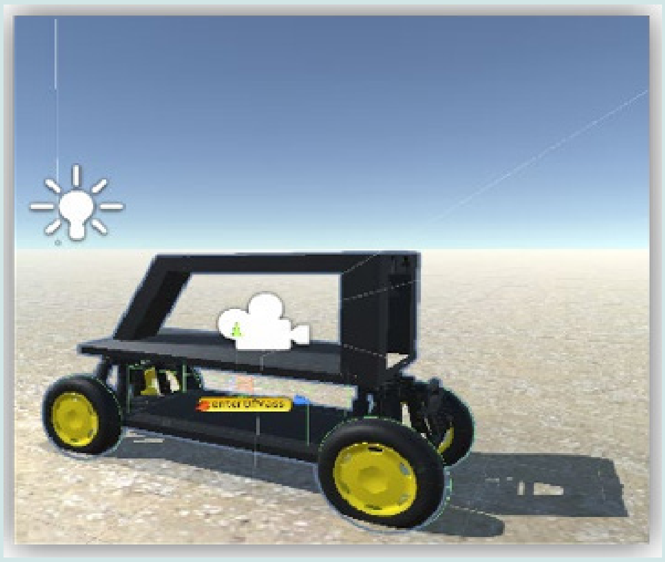

Figure 2: Final model (in Unity).

Data Generation

To generate the data which is vital for our training process we have switched to manual drive mode to collect images from our Camera sensor. During the manual drive our front camera (Camera sensor) took screenshots from the correct direction the car is following while we were controlling it by ourselves. In our C# script file, we have defined the directory of our generated data that should be saved. To avoid further problems, we have created the folder in the same directory as the source files located in. Totally, around 50 000 images were collected and ready to feed our CNN layers.

Training Process

We used Anaconda Jupiter notebooks to train our model externally. We have extracted useful data from the images in our data extraction python script to extract data arrays from the images and save it as .csv file in the same directory. In the .csv file there are arrays of pixel representations of our images. After, we defined our train python script and set the destination for our trained model. h5 file into our main directory to be saved. Later for the training session we defined CNN layers with the finisher which is fully connected layer. (Figures 3 & 4)

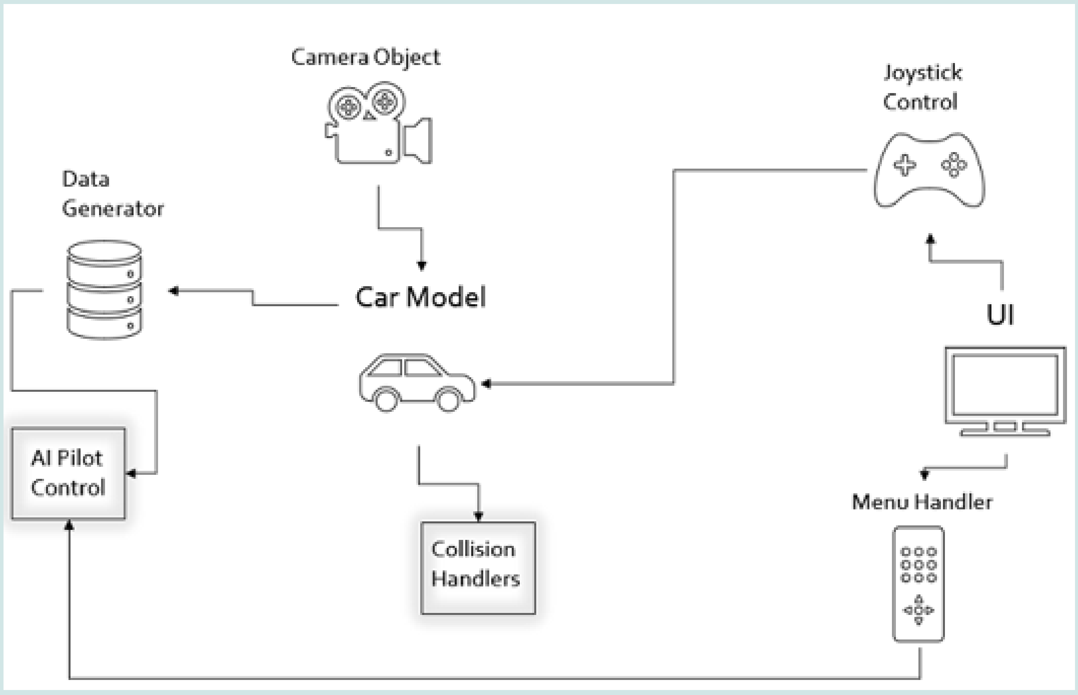

Figure 3: Design pattern of a virtual simulation.

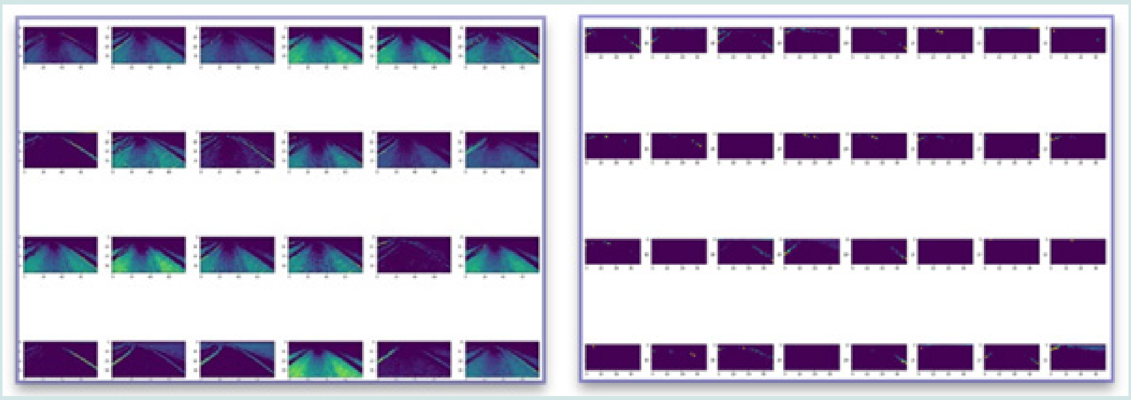

Figure 4: Internal state of CNN.

Hardware

Hardware Setup

Our RC car is attached to remote control which sends signals to steer the wheels depending on user's control via remote controller. In our case, our intention is to control the car purely in an auto pilot mode. To do so, we must buy several extra parts, rewire the vehicle's servo, and connect it to the onboard computer (Raspberry Pi3+). For capturing 120x160 RGB images we will need a mono frontal wide-angle camera and a motor controller to connect and control the steering servo and the electronic speed controller (ESC).

Car Chassis

One of the most important parts of our project is our RC car. All the things must be well considered while choosing the car and making a final decision on which one to buy. The signal receiver needs to be separate from the car's ESC (electronic speed controller), so that we will be able to connect the steering servo and electronic speed controller to our motor controller to fully control both. The ESC also should allow for fine-tuning of the steering angle of the car. Having full control over steering is preferred, because some low-end small-scale vehicles give only discrete angle steps (left, right and center). Finally, the last thing to take into consideration is the volume of the car, in other words the scale of the model. The scale range can be between 1/32 to 1/8 or more, but the car should be big enough to locate all parts and batteries. So, our choice was High speed RTR Monster Truck (HSP No.94186) with 1/16 scale [11] (Figure 5).

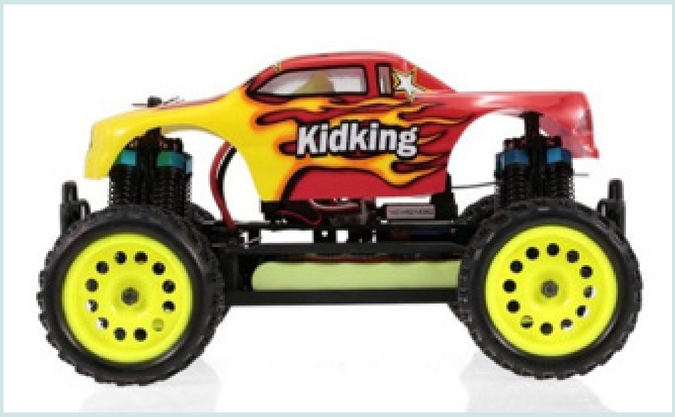

Figure 5: High speed RTR monster truck.

Motor and Servo

The main purpose of servo or actuator in a RC car is to convert the electrical commands received from radio control system into physical movement. In our case it will be plugged into a specific receiver channel and will be used to move specific parts of our RC car model. We have used the Adafruit PCA9685 16 Channel 12 Bit PWM Servo Driver, which has a 16-channel I2C-bus controlled LED controller [12]. The PCA9685 with 16-channel 12-bit PWM Servo Driver can control 16 servos with only 2 pins. So, to control the motor and servos, we will need to send specific PWM signals.

Camera

When it comes to choosing the camera, resolution is not important as we will be down sampling the input images during the training and testing time. So, we stopped by Sain Smart Wide-Angle Fish-Eye Camera Lenses [13], because wide angle camera will be good to capture and get bigger image of car's surroundings. Also, the price of the camera is good.

Raspberry Pi (Onboard Computer)

We had many different options when it came to choosing the onboard computer, but the important thing is the size of your vehicle and where the small-scale computer will be installed. We had a wide range of choices like NVIDIA DRIVE PX Pegasus [14], however it was a bit expensive and more useful for bigger projects. So, for smaller and cheaper projects we had options such as Arduino [15], Raspberry Pi 3b+ [16] and Orange Pi [17]. And finally, we chose Raspberry Pi 3b+ as it is easily possible to run keras models with TensorFlow backend on it and it includes wi-fi support.

Battery, Cables, and Screws

Except the ESC, which is powered from the vehicle's battery, other components, mainly the Raspberry Pi should be connected to an external source of power like power banks etc. The size of external batteries can depend on the scale of the RC car and how long it should stay active. To connect our bottom and upper part (3D Printed roll cage and top plate) we will need M2x6 screws and M3x10 screws. Finally, to connected Adafruit PCA9685 16 Channel 12 Bit PWM Servo Driver to our onboard computer Raspberry Pi we will use female to female jumper wires. (Figure 6)

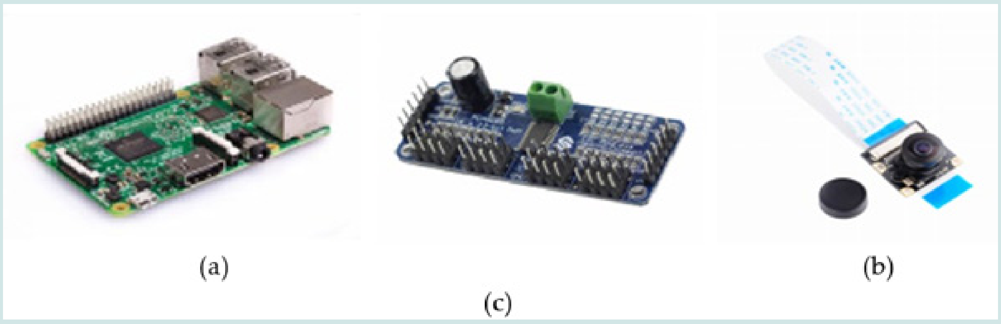

Figure 6: (a)Raspberry Pi 3 model B+ (b)Adafruit PCA9685 16 channel 12 bit PWM servo driver (c) SainSmart wide angle fish-eye camera lenses.

System Architecture

The architecture is based on distributed systems. ROS allows us to create multiple nodes which can exchange data via publisher/ subscriber methods. Each node operates individually. The data will be gathered by joystick and camera nodes and data generator combines them into a file and saves it for later training process. Once enough data is gathered, we deploy trained neural network in the AI Pilot Node. This node generates steering and throttle output based on the images taken from the camera. Then, it will be converted to PWM through Actuator Node and supplied to I2C Node (Figure 7).

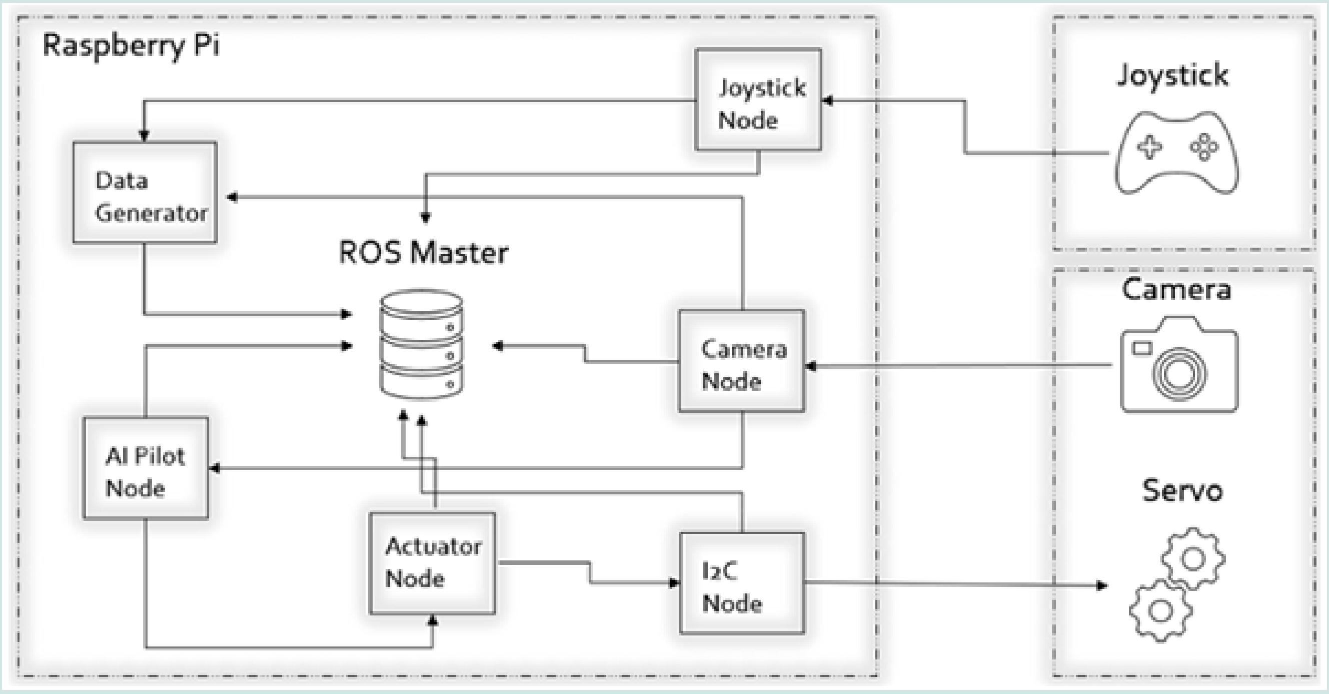

Figure 7: ROS system design.

Training Process

Background

Kunihiko Fukushima in 1983 introduced a noncognition (a hierarchical multilayered artificial neural network) which could recognize handwritten digits and characters with the help of neural networks [18]. Later in 1998, LeCun introduced the first model of convolutional neural networks with back propagation and gradient based learning [19]. The system could accurately classify handwritten digits and characters in 32x32 image data. The result has proved that CNN could be used to recognize visual patterns with minimum number of processes. However, with the research of NVIDIA the CNN Pilot Net Architecture could implement and test CNN in an end-to-end manner to self-driving cars [5].

CNN Network Architecture

In our project we had different generated datasets during the different conditions of the place where our track map is placed. To get better results we have reduced the FPS frame rates of video which were recorded from the frontal camera of the car during the data generation process, to avoid the repentance of the same image frame in a high FPS. Usually, feeding CNN with data starts with RGB images as inputs. Like other neural networks, CNN is made up of learnable weights, where the nodes receive inputs, calculate the weighted sum, implement some filter layers, pass it through an activation function and respond with a best fitting output. Briefly, CNN layers generate a feature map of an input image by iterating each specified filter over an image in a certain layer extracting specific features. The input RGB image with 3 channels is fed into Conv2D layer where specified filter will iterate over the feature map of an image and the output will be navigated to the next lawyer. Among widely used layers are Max Pooling and Min Pooling. In a Max Pooling the specific iterated part of an image array generates a new feature map with matrix values where the Max number will be selected and built a finalized feature map, but in Min Pooling it is the vice versa, the Min number will have higher priority to be selected. After, it will be possible to implement nonlinear activation function ReLU. The ReLU usually makes 0 outputs which are smaller than 0, briefly it sketches the graphs with positive outputs in it. The output from the last Convolution layer will be fed into 3 fully connected lawyers where later it will locate the output to the SoftMax function where the probable best fit answer will be generated. In our case it will generate the output in the form of steering angles and throttles. (Figure 8)

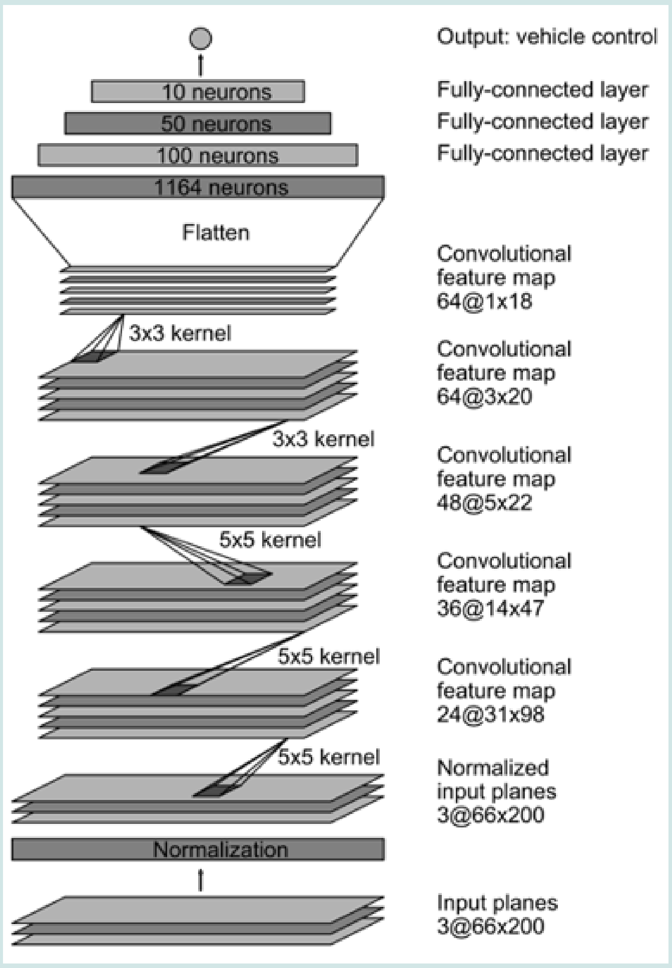

Figure 8: PilotNet Architecture [5].

Generated Dataset

The dataset was recorded through the car's data generator node. We have extracted around 17 000 images from our recorded videos during the training process through the manual drive. As was mentioned before, the dataset was recorded in a lower FPS (around 10 frames per second) to avoid any resample images which would not grant any additional information. To guarantee the robustness of the acquired dataset we have driven the car in different lightning conditions, with low or high brightness of the place with track map, in a flat or not a flat surface, with or without surroundings. Before images are fed into the network it was cropped to make sure that only the relevant part of the image is present to the system. Then the images were downscaled to the system architecture and later initialized to have RGB pixel intensities between 0 and 1, instead of 0 to 255 to help the system to show a better performance. (Figure 9)

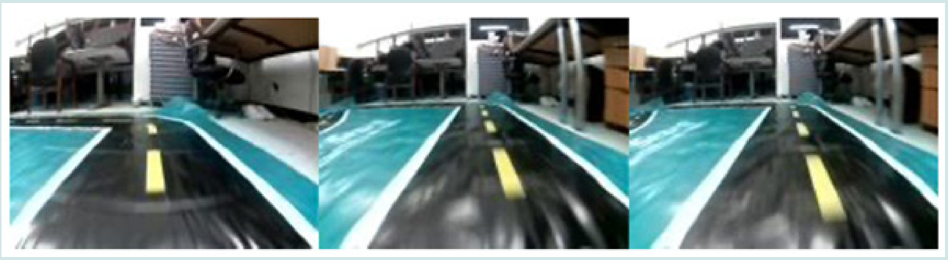

Figure 9: Recorded images from Pi camera.

Results and Discussion

We conducted experiments on our virtual model, and it gave the accuracy of around 95%. We used the early stop technique to stop the training process when there was no improvement over the specified period of epochs. (Figure 10) Similarly, we trained our model on real datasets. This time, the early stopping technique was utilized. However, compared to the model in virtual environment, it took less epochs to get converged at optimum (Figure 11). At the 15th epoch the model seemed to be overfit, therefore we stopped the training process further. After deploying a trained model on a car, we launched the AI mode. (Figure 11)

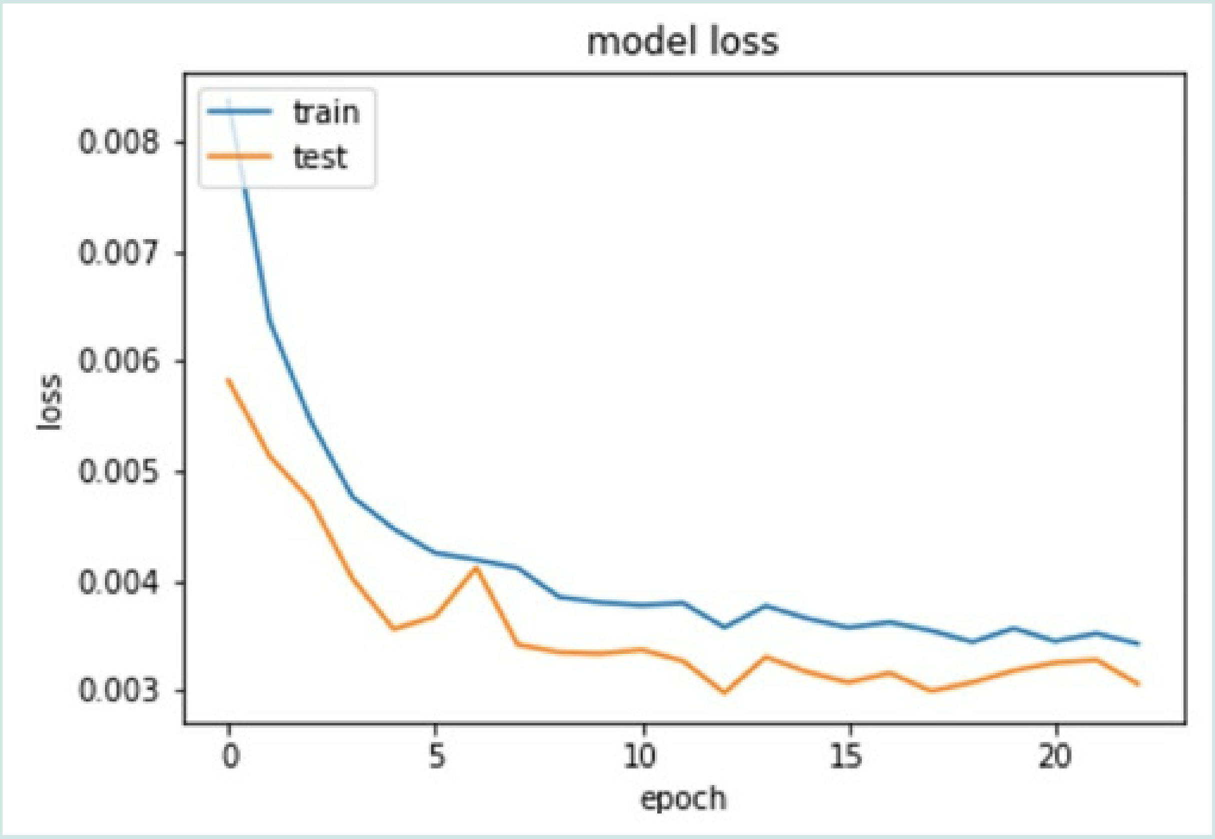

Figure 10: Model Loss for CNN in Simulated Environment.

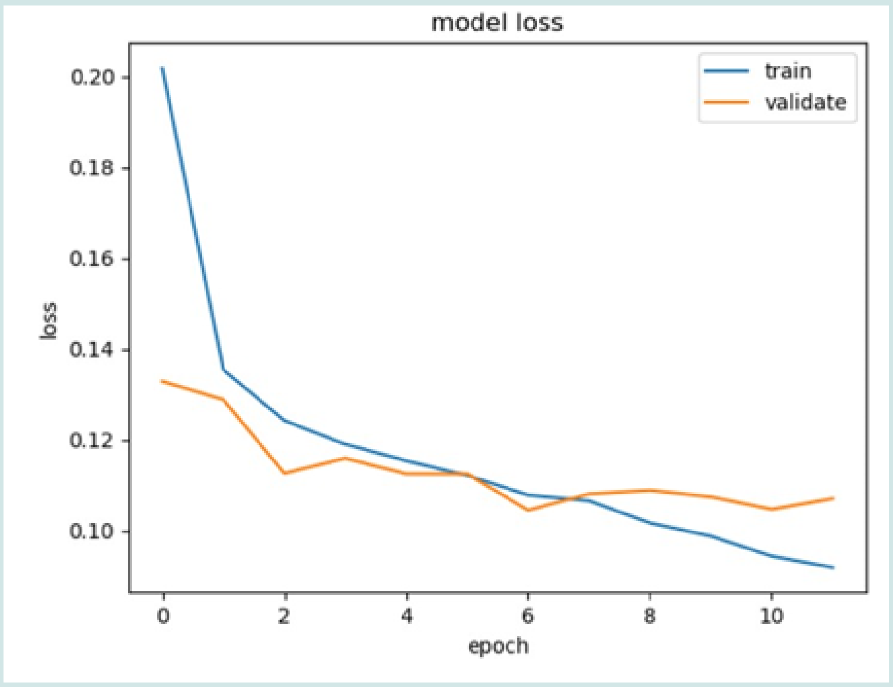

Figure 11: Model Loss for Real Car Model.

Previous work made by J.E. Siegel et al. showed a difference in the results, where simulated vehicle showed a better performance compared to the physical one and as it mentioned the reasons for that could possibly be the different lightning conditions on virtual versus physical environments or different camera lenses along with slightly change the scene of the environments [6]. The model initially was trained in a simulated environment and transferred to the physical vehicle without retraining it. However, our models which were separately trained for virtual and physical environment shower almost same good results as we have calibrated the camera for those special environments along with different camera lenses with different capabilities, while the models were trained on the same computer with same functionality. In comparison to a Deep Reinforcement Learning conducted by Q. Zhang et al. our CNN based end to end algorithm took less training time and better results with a single camera on the real and simulated road [8].

As Q. Zhang says in their results that the training times took very long to process the images and the learned strategy was unstable, especially when car had to take deal with turns to generate a robust steering angle. From the testing part it was obvious that the leaning policy was also less stable, and the car wriggled frequently while making turns. However, our robust system generated well designed steering angles acquired from our single frontal camera. NVIDIA's Pilot Net had a stronger architecture and tools if we compare our work with theirs [5]. The system had a single front-facing camera like ours but with much bigger volume and processing rate. And the work was conducted in a real car while our end-to-end approach was on small scale RC toy car with portable computer (Raspberry pi B+ type). When we switched our track map from one place to another one, the car did not perform as good as we expected and the reason for that could be different lightning conditions and background surroundings of the new place. Also, when we finally set our track map outside to test our vehicle on a clear weather day, we bumped into the worst-case scenario as the sun was too bright and the sky was too clean. Unfortunately, the environment influenced our robust system too badly. Meanwhile, Pilot Net with minimum trained data could achieve well performing cars with or without lane markings. And NVIDIA's system could take the right decisions even during unclear weather conditions and while running on unpaved roads. The reason for the robustness of the system probably could be high processing base computer (NVIDIATM PX self-driving car computer) with 30 frames per seconds (FPS), which is overweighs ours. Also, the GPU power with NVIDIA Dev Box and Torch 7 for data training sessions was much higher than ours.

Conclusion

In this paper, we described how to use deep convolutional neural networks in an end-to-end manner to control small scaled self-driving RC car. The car was built from scratch using custom hardware parts and trained in a manual driving mode on track map. We designed a virtual environment to check the performance of virtually autopiloted car models and later compare the results and changes with the physical model. We have derived important changes through a virtually simulated environment, which could possibly be added to our physical car to get vital improvements. After the training process we fully granted the CNN to generate independent steering angles and throttle controls for our car. Using proper forms of regularization and quality datasets may turn out to yield the best results. The system learned to steer autonomously on simplified flat roads using only a single camera to generate its input. The car was able to drive indefinitely successfully autonomously on a circular loop of the track map it was trained on before. The biggest challenges we had to face were lack of data acquired from different environments in different lighting conditions, where the car sometimes could not perform well when the track map was moved to a new unknown environment with different surroundings.

References

1. World Health Organization.
2. Los Angeles magazine.
3. Pomerleau DA (1988) ALVINN: An Autonomous land vehicle in a neural network.
4. LeCun Y, Muller U, Ben J, Cosatto E, Flepp B (2005) Off-Road Obstacle Avoidance through End-to-End Learning.
5. Bojarski M, Testa DD, Dworakowski D, Firner B, Flepp B, et al. (2016) End to End Learning for Self-Driving Cars.
6. Siegel JE, Pappas G, Sun Y (2009) A gamified simulator and physical platform for self-driving algorithm training and validation.
7. (2020) Robotic Operating System.
8. Zhang Q, Du T, Ch Tia (2019) Self-driving car trained by Deep reinforcement learning.
9. (2020) Blender.
10. (2020) Unity.
11. (2020) Amazon.
12. (2020) Adafruit.
13. (2020) SainSmart Wide Angle Fish-Eye Camera Lenses.
14. (2020) NVIDIA DRIVE PX Pegasus.
15. (2020) Arduino.
16. (2020) Raspberry Pi 3b+.
17. (2020) Orange Pi.
18. Fukushima k (1980) Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position. In Biological Cybernetics 36(1980): 193-202.
19. Lecun Y, Bottou L, Bengio Y, Haffner P (1998) Gradient-based learning applied to document recognition. In 88(11): 2278-2324.

DOI: 10.32474/ARME.2023.04.000181