

Using Deep Learning and Reinforcement Learning In Combination with Automatic Vehicles' Application of the Allee Effect

Author's Details:

Cao Thi Luyen, Faculty of Information Technology-University of Transport and Communications;
Email: luyenct@utc.edu.vn

Nguyen Quang Duc – Class 12A1 - Nguyen Sieu High School – Hanoi;
Email: quangduc0405@gmail.com

Nguyen Phuc Thanh K52-A7 Foreign Language Specialized School;
Email: flssk52thanh@gmail.com

Nguyen Minh Huy - Graduate student majoring in Industrial Automation - Hanoi University of Science and Technology;

Email: Huy.nm211231m@sis.hust.edu.vn

Abstract:

Convolutional Neural Network (CNN) can detect images from cameras installed on self-driving cars. First, we drove a car on a simulator and recorded frames from three cameras: left, right, and center. These frames were recorded at the rate of 30 frames per second. Additional data recorded were the distribution of steering angles, average velocity, etc. were passed through a CNN to train a self-driving system. CNN is like the eyes and visual area in the brain, so CNN's achievements in autonomous vehicle control are somewhat limited. Therefore, this paper proposes the use of algorithms based on Deep Learning (DL) combined with reinforcement learning (RL) in the control of autonomous vehicles. We call this algorithm Deep Reinforcement Learning (DRL) which can send control commands to the vehicle to navigate properly and efficiently along a defined route. CNN tracks multiple objects while RL predicts the environment or assesses the current condition of the vehicle to make the safest decision. DRL-based algorithms have been used to solve Markov Decision Processes (MDPs), where the scope of the algorithm is to compute the optimal policy of an autonomous vehicle for choosing actions in an environment to maximize a reward function.

Keywords: Convolutional Neural Network, self-driving cars, data processing.

1. Introduction

In recent years, the search for automatic guided vehicles (AGV) is becoming more and more prominent globally and locally in Vietnam; they are now being used extensively. Compares to traditional vehicles, AGVs bring various benefits. Using AGV lifts in delivering items, transferring between stages of an assembly line, and assembling and storing within a line to distinguish inbound and outbound items of a logistic chain can decrease the need for human intervention. Comparing the efficiency of an AGV robot that can work 24/7 (excluding charging time) on 3 shifts per day to that of a normal worker is an unfair comparison. No AGV will take “sick days” like humans unless it is completely disabled. Though AGVs require “rest” days for maintenance, those days are controlled by the vehicles' manager. Not only efficiency is greater, but AGVs can also reduce training time, internships, overtime bonuses, health bonuses, insurance, and retirement bonus for companies. The cost of running AGVs only requires maintenance costs.

An AGV that is trained and controlled by command lines or a future autonomous system will obey humans' instructions exclusively. AGV lifters will know which location to go to, lifting and transferring items without the need for time to search and familiarize themselves. It will know when there are obstructions to send off a warning signal, how to avoid that obstacle, and how to return to its predetermined path. Autonomous vehicles will not fall into a “tired” state or be distracted by the environment; the vehicle, therefore, will not be driven off the predetermined path, minimizing risks to itself, the items carried, and even humans.

In environments that have poor lighting or undesirable temperature, humidity, air quality, toxins, and working space, AGVs are an ideal replacement for human workers. AGVs are used in multiple workspaces, such as factories, assembly lines, logistic systems, hospitals, supermarkets, etc.

The technology used to control AGVs usually involves pre-programmed systems or recognizing simple images. In recent years, though, improvements made in Deep Learning along with advanced image and verbal processing allow neural systems to control the processes of pattern recognition, communication, and driving vehicles.

Processing images and detecting obstacles, traffic lights and signs, and dangerous situations are the core features to develop autonomous vehicles. It is a necessity to have a fundamentally sound mathematical model to process data for controlling the vehicle. In this paper, we will use a newer recognition system – Convolutional Neural Network – to control autonomous vehicles.

The efficiency of CNN when processing images is due to its adaptation and its 2-D network. CNN is inspired by the AI model and biological neural processes and is applied to a certain degree of success in recognizing models (LeCun et al.) [1]

CNN is one of the advanced Deep Learning models. It enables the building of high-precision smart systems that is common nowadays. Like most feedforward neural networks (FNNs), CNN is trained as a version of a back-propagation algorithm, with ReLU as the activating function. We will discuss the architecture of the CNN based on the definition of convolution and feature mapping in later sections.

CNN has changed pattern recognition. Before its widespread adoption, most recognition tasks were performed by extracting predetermined features, then passing through a classifier. A breakthrough feature of CNN is the ability to learn features automatically from examples [2]. CNN's approach is more advanced than other systems because the convolutional network can recognize the 2D features of the image. Furthermore, by scanning the entire image using the convolutional layer, the number of parameters required is fewer than in most systems. While CNN's features have been in use for more than twenty years [3], its adoption has only been on the rise in recent years due to the introduction of two features. First, large data sets e.g. the Large Scale Visual Recognition Challenge (ILSVRC) [4] have been providing data for system training and validation. Second, CNN's algorithms are now performed by more advanced graphics processing units (GPUs) which accelerate learning and inference processes.

Classic autonomous driving systems often use advanced sensors to perceive the environment and complex control algorithms to navigate safely in arbitrarily challenging situations. Typically, these frameworks use a modular architecture in which individual modules process information asynchronously. The perceptual layer captures information from surroundings using different sensors like cameras, LiDAR, RADAR, GNSS, IMU, and so on. Regarding the control layer, some of the most used control methods are the PID control method [9] and the quadratic Linear Regulator (LQR) algorithm [10].

However, despite their good performance, these controllers are often environment-dependent, so their respective hyperparameters must be appropriately tuned for each environment to obtain the expected behavior.

CNN navigates autonomous vehicles by following and mimicking an expert's system decisions. In that sense, an expert system (usually a human) provides a set of driving data, which is used to train (dealer) driving policy through learning. The main advantage of this approach is simplicity, as it achieves very good results in end-to-end applications (navigation from a current location to target a certain location as quickly as possible to avoid collisions and departures on the road in an arbitrarily complex dynamic environment). Its main drawback, however, is the difficulty of imitating any potential driving scenes, and the inability to reproduce unlearned behaviors. This limitation makes this approach potentially dangerous in some real-life driving situations that have not been observed before.

While reinforcement learning (RL) algorithms are learning dynamically with a trial and error method to maximize results, reward for a correct prediction, penalize for an incorrect prediction, and successfully tested to form the Markov Decision Process (MDP). The combination of Deep Learning techniques and Reinforcement Learning algorithms has proven capable of solving some of the most challenging tasks of autonomous driving.

2. Convolutional Neural Network (CNN)

CNNs are commonly used to detect objects in an image. This section will discuss the algorithms involved in detecting patterns.

The fundamental layers of a CNN are:

2.1. The Convolutional layer

This is the most important layer of a CNN; it performs mathematical operations. Key features of this layer are stride, padding, filter map, and feature map.

- CNN applies a filter to regions of the image. These filter maps are 3D matrixes that consist of numbers known as parameters.
- Stride is a parameter of the network that modifies how much movement should be performed on the image or video.
- Padding is the process of adding layers of zeroes to the input
- A feature map is the result of each filter map scan over the input. Mathematical operations are performed after each scan.

The convolutional layer can be considered as a sliding window on matrixes as described below.

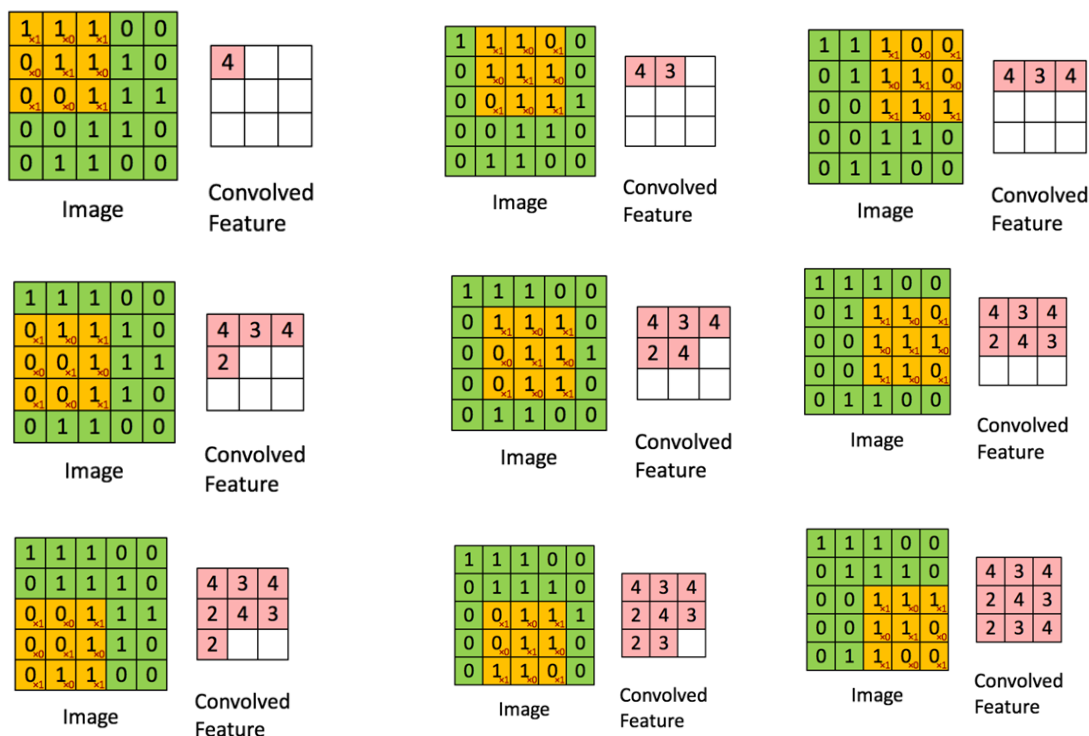


Figure 1. Convolutional layer to extract key features of an image
(Source: <https://nttuan8.com/bai-6-convolutional-neural-network/>)

Each convolutional layer contains a parameter known as a kernel that was trained to extract key features of an image without selection. In the example, the left triangular matrix was a black-and-white image that was digitalized. The dimension of the matrix was 5×5 and the intersection of every column and row has the value of 0 or 1.

Convolution is a 3×3 matrix. The sliding window (kernel, filter, or feature detect) is also a small matrix (in the example, it was a 3×3 matrix.). The convolved feature is the product of the filter matrix and the 5×5 matrix.

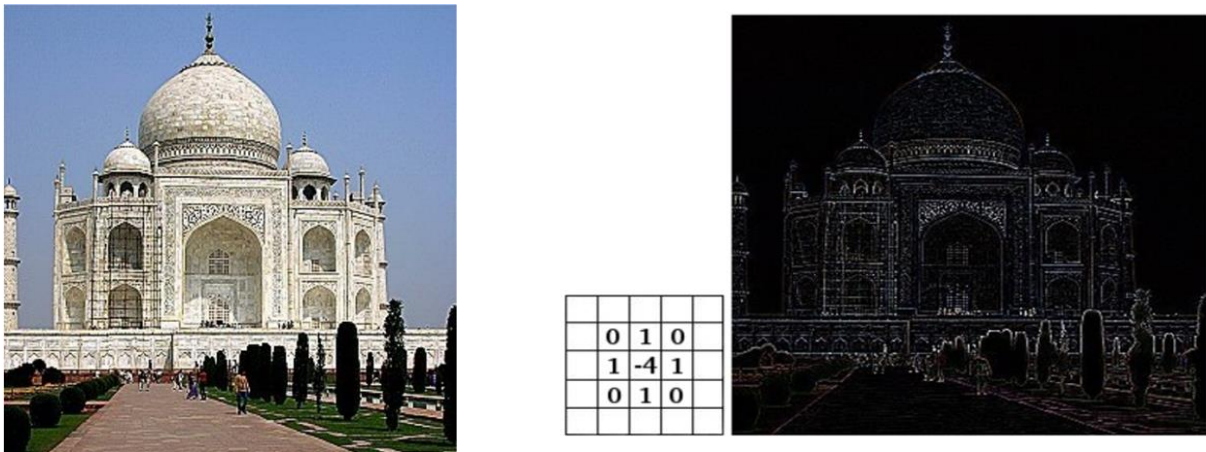


Figure 2. An image that has its feature extracted by a left triangular matrix

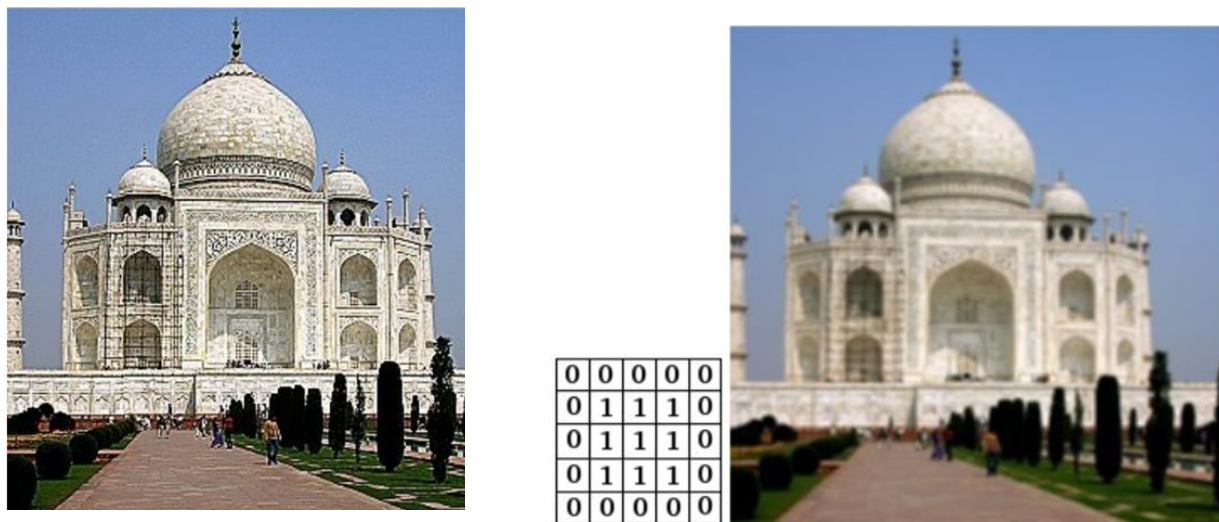


Figure 3. The same image that has its feature extracted by a different left triangular matrix

2.2. Activating function (Rectifier layer)

This layer is also known as the ReLU (Rectified Linear Unit) Layer. The purpose of this layer is to replicate the movement of electrical impulses of nerve cells through the axon. The activation layer consists of basic functions such as sigmoid, hyperbolic tangent (tanh), ReLU, Leaky ReLU, and Maxout. ReLU is the more common function; it is preferred over the other functions because it can perform calculations faster, making it more suitable for network training. When using ReLU, great care should be taken when altering the Learning Rate or observing the Dead Unit. This layer is used after the Filter Map was produced and the ReLU function was applied to all functions on the filter map.

2.3. Pooling layer

If the input is too large, the pooling layer will be placed between convolutional layers to decrease the parameter. There are two types of pooling layers: max pooling and average pooling

If max pooling is used, the number of parameters decreases. Therefore, CNN will introduce multiple Filter Maps; each Filter Map will then produce a unique max pooling layer.

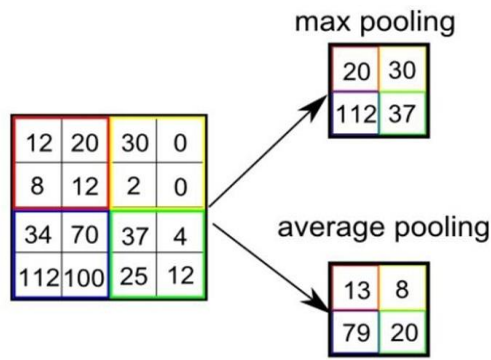


Figure 4. The pooling operation

2.4. Fully connected layer

This layer produces the result after the image was passed through the convolutional and pooling layer. The result is a model that can extract features of the image. To increase the number of outputs, the fully connected layer will then be used. Furthermore, if the fully connected layer retains the quality of the image, it will then pixelate that image. The layer will then select the best image from the pool of pixelated images.

3. Mathematical model of a CNN

The convolutional neural network is a feedforward neural network that has little weighting (weighting is near zero). With less weighting, the training process is streamlined and simpler than other fully connected networks.

CNN has shown excellent efficiency when processing 2D images. In this scenario, the input image is an RGB-colored image that can be considered as a $r \times c \times 3$ tensor, where r is the number of rows and c is the number of columns. Three $r \times c$ channels correspond to the three primary colors.

Each color channel can be expanded to the nuclear level. However, each three-color channel can be grouped into groups of three nuclei of the same type. In this situation, the convolution function is still the same. Kernels will “stack” onto the image at the coordinate (0,0,0); the sum of the products of each data input will then be obtained as the first output. Then, the nucleus will be shifted one unit in any direction to complete the process.

Regarding a stationary nucleus, each object in the tensor $X^{(l)}$ can be solved by inputting the latent variable $X_{ijk}^{(l)}$, where $1 \leq i \leq r^{(l)}, 1 \leq j \leq c^{(l)}$ and $1 \leq k \leq 3$ (see figure 5.) A mapping for the k th layer and the k th channel with the corresponding $w^{(l)}$ nucleus and the bias $b^{(l)}$ can be calculated as:

$$X_{ijk}^{(l)} = \phi \left(\sum_s \sum_p X_{i+p, j+r, k}^{(l-1)} w_{prk}^{(l)} + b^{(l)} \right)$$

where the activation function ϕ (usually ReLU) is used to avoid gradient loss.

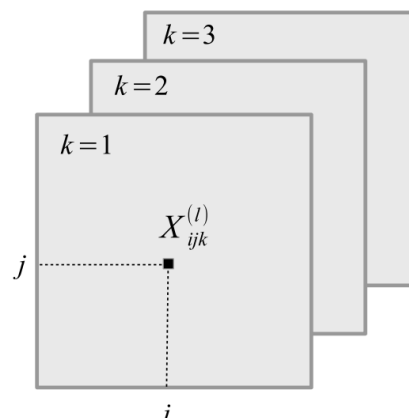


Figure 5. Objects of the tensor $X^{(l)} \in R^{r^{(l)} \times c^{(l)} \times 3}$. The input $X_{ijk}^{(l)}$ indicates the activation of coordinate (i, j) in the k th layer.

The output can be written as a matrix $Y = \sigma(WX + B)$,

where $X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_6 \end{bmatrix}$, $B = \begin{bmatrix} b \\ b \\ b \end{bmatrix}$, $Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_5 \end{bmatrix}$. The weighting can be written as a 5×6 matrix:

$$W = \begin{pmatrix} w_1 & w_2 & 0 & 0 & 0 & 0 \\ 0 & w_1 & w_2 & 0 & 0 & 0 \\ 0 & 0 & w_1 & w_2 & 0 & 0 \\ 0 & 0 & 0 & w_1 & w_2 & 0 \\ 0 & 0 & 0 & 0 & w_1 & w_2 \end{pmatrix}$$

Therefore, $WX + B$ can be expanded as

$$WX + B = \begin{bmatrix} w_1 & w_2 & 0 & 0 & 0 & 0 \\ 0 & w_1 & w_2 & 0 & 0 & 0 \\ 0 & 0 & w_1 & w_2 & 0 & 0 \\ 0 & 0 & 0 & w_1 & w_2 & 0 \\ 0 & 0 & 0 & 0 & w_1 & w_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} + \begin{bmatrix} b \\ b \\ b \end{bmatrix} = \begin{bmatrix} w_1x_1 + w_2x_2 + b \\ w_1x_2 + w_2x_3 + b \\ w_1x_3 + w_2x_4 + b \\ w_1x_4 + w_2x_5 + b \\ w_1x_5 + w_2x_6 + b \end{bmatrix}$$

and Y can be written as

$$Y = \sigma(WX + B) = \begin{bmatrix} \sigma(w_1x_1 + w_2x_2 + b) \\ \sigma(w_1x_2 + w_2x_3 + b) \\ \sigma(w_1x_3 + w_2x_4 + b) \\ \sigma(w_1x_4 + w_2x_5 + b) \\ \sigma(w_1x_5 + w_2x_6 + b) \end{bmatrix}$$

4. Structure of a CNN

CNN is a “stack” of convolution layers that uses non-linear functions such as ReLU or Hyperbolic tangent (*tanh*) to activate the weighting of each node. After the activating function is applied to the layer, the output data will be more abstract for the next layer to process. In a feedforward neural network, the output of each node will then directly feed the next layer. These layers are known as the fully connected layers or affine layers. In a CNN, however, these layers are connected by convolution.

The input of a layer is the convolved data from the previous layer; therefore, CNN is locally receptive. Each neuron of the next layer is the result of applying a filter on a local region of the previous neuron. Each layer consists of combinations of filters and their results. Moreover, other layers (such as the pooling layer or the subsampling layer) can be used to filter the data i.e. removing useless features)

During the training period, CNN will automatically learn the parameters of any filter layer based on the task given. For example, when processing images, CNNs will automatically find the suitable parameter for each filter layer in the order of raw pixels, edges, shapes, facial, and high-level features. The final layer is used to process the image.

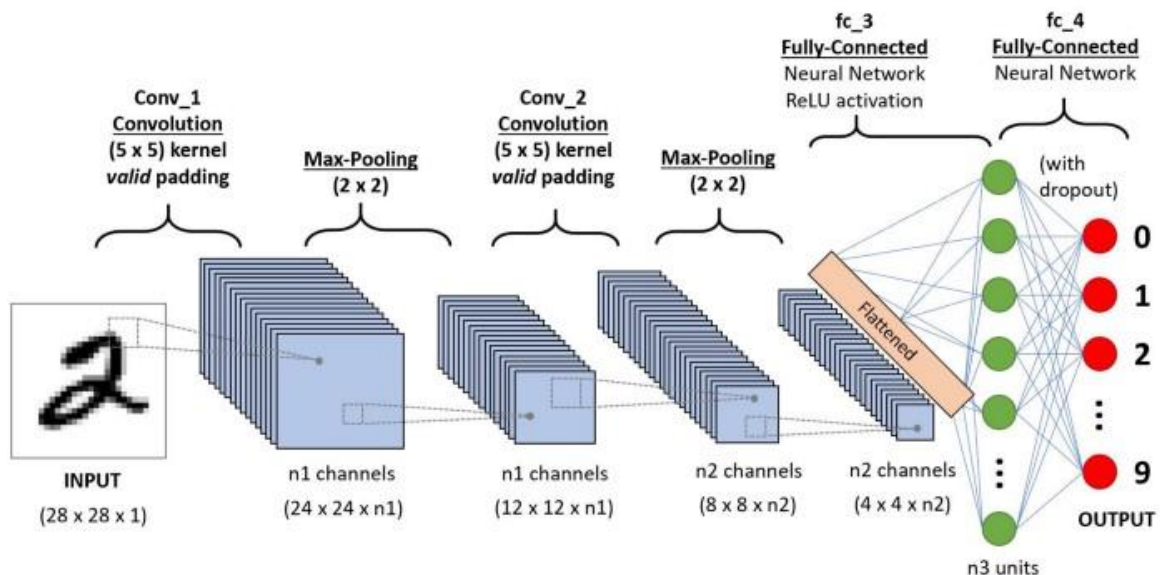


Figure 6. Diagram of a CNN

(Source: <https://nttuan8.com/bai-6-convolutional-neural-network/>)

The CNN model has two notable features: Location Invariance and Compositionality. The same object can be perceived differently if its projection was altered by translating, rotating, and scaling. The pooling layer will provide invariance in the degree of translation, rotation, and scaling. The compositionality of a CNN will then allow the data to be presented increasing in value and more abstract for the next convolution from the filter.

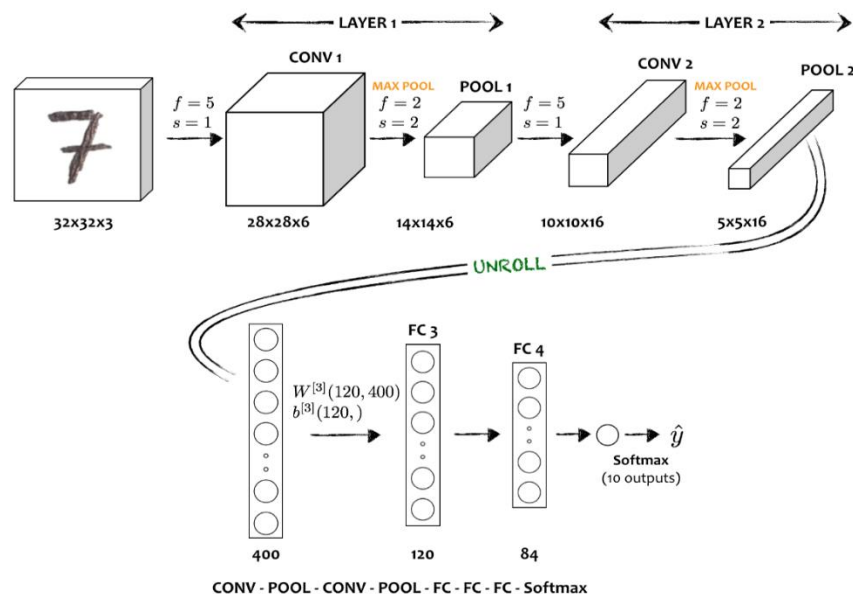


Figure 7. A CNN for recognizing handwriting.

(Source: <https://nttuan8.com/bai-6-convolutional-neural-network/>)

5. Reinforcement learning

Deep reinforcement learning combined with neural networks underpins a learning architecture that allows software-defined agents to learn the best possible actions in a virtual environment to achieve optimal goals. That is, it merges functional approximation and objective optimization, mapping state-action pairs with reward expectations. These algorithms consider the agent's behavior at the time of learning with his action reward structure, rewarding the agent when the chosen action is good, and penalizing otherwise.

5.1. Q-Learning

The Q-Learning algorithm [11] generates an exact matrix so that the agent can maximize its reward in the long run. This approach is only practical for limited environments, with limited space for observations, since an increase in the number of states or actions causes an algorithm to misbehave. Q-Learning is a policy-free, model-free RL based on the Bellman Equation, where v refers to its optimal value:

$$v(s) = E[R_{t+1} + \lambda v(S_{t+1}) | S_t = s]$$

E refers to the expectation, while λ refers to the discount factor for the forward rewards, and rewrites it as Q-value:

$$Q^\pi(s, a) = E[r_{t+1} + \lambda r_{t+2} + \lambda^2 r_{t+3} + \dots | s, a] = E_{s'}[r + \lambda Q^\pi(s', a') | s, a]$$

Where the optimal Q value Q^* can be expressed as follows:

$$Q^*(s, a) = E_{s'}[r + \lambda \max_{a'} Q^*(s', a') | s, a]$$

The objective of Q-Learning is to maximize the Q-value repeat policy, which corrects the loop between policy evaluation and policy improvement.

5.2. Deep Q-Learning

As we pointed out above, Q-learning lacks generality as the observation space increases. Imagine a situation with 10 possible states and 10 possible actions, we have a 10x10 matrix, but if the number of states increases to 1000, the Q matrix increases significantly, making it difficult to manage manually. To solve this problem, Deep Q-Learning (DQN) manages two-dimensional arrays by introducing Neural Networks. DQN estimates the Q-Value using it in a learning process, where the state is the input of the Network and the output is the corresponding Q-value for each action. The difference between D-Learning and Deep Q-Learning is represented in the equation of y :

$$y_j = r_j + \gamma \max_a Q(s_{j+1}, a'; \theta -)$$

where θ stands for the parameters in the Neural Network.

6. Using CNNs and DRL for self-driving vehicles

6.1. Formulating MDP

In recent years, a large number of reinforcement learning algorithms have been developed to solve the Markov decision process (MDP). MDP is defined by a tuple (S, A, P, R) , where:

- The state, S is the set of states that define the current status of the agent,
- The action, A is the set of actions that the agent takes at the current instance,
- The policy, P is how the agent process as the action is taking place. In practice, the policy can be regarded as a probability distribution. Therefore, $P: S \times A \rightarrow P(S)$ is the Markov transition probability matrix,
- The reward function, R , assigns a score based on the state of the environment. Thus, $R: S \times A \rightarrow R(\cdot | s, a)$ is the reward distribution.

Taking any action $a \in A$ at any state $s \in S$, $P(\cdot | s, a)$ determines the probability of the next state, and $R(\cdot | s, a)$ is reward distribution. A function $\pi: S \rightarrow P(A)$ maps every state $s \in S$ to a probability distribution $\pi(\cdot | s)$ over A .

For any Markov state S_t , the probability of the next state S_{t+1} (denoted s') is dependent on the current state $S_t = s_t$, and independent of any past state. In mathematical terms,

$$\mathbb{P}[S_{t+1} | S_t] = \mathbb{P}[S_{t+1} | S_1, \dots, S_t],$$

where \mathbb{P} is the state transition probability.

Markov Process is a sequence of Markov states e.g. S_1, S_2 . It is defined by (S, \mathbb{P}) . Denoted as $P_{ss'}$, it is the probability of jumping from the current state s to the state s' . $P_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s]$. Therefore, the Markov Reward Process $R_s = E[R_{t+1} | S_t = s]$ is the expected reward over any state possible. It is defined by (S, P, R, γ) , where γ is the discount factor. The Markov Decision Process is essentially Markov Reward Process with action, which is defined by (S, A, P, R, γ) . The reward and the next state now depend on both

the current state and action, hence the notation $R_s^a = \mathbb{E}[R_{t+1}|S_t = s, A_t = a]$. The cumulative reward is called the Return.

The discount, γ , is used to factor out the uncertainties regarding the future rewards and the contribution of future rewards to the Return.

The policy, π , is the distribution of action at a known state. It is defined as the probability distribution over the set of actions a given the state s ; $\pi(a|s) = \mathbb{P}[A_t = a|S_t = s]$.

Considering the general MDP explanation above, we use MDP to solve the autonomous navigation task, which involves an agent that observes the state s_t of the vehicle (the state of the environment) and generates an action a_t . This causes the car to transition to a new state s_{t+1} , generating a reward $r_t = R(s_t, a_t)$ based on the new observation. The Markov Decision Making Process is a process of (S, A, P_a, R_a) where the goal is to create a good product "Policy", that is, the function(s) that the decision maker will select when in state s_t

a) State space S : This term refers to the information received from the environment during each step of the algorithm. In our case, we model s_t as a tuple $s_t = (vf_t, df_t)$ where vf_t is a vector visual feature associated with the image I_t or a set of visual features extracted from the figure image, usually a set of waypoints w_t obtained using the model-based path planner $vf_t = f(I_t, w_t)$. df_t is a driving feature vector consisting of an estimate of the number of vehicles with speed v_t , the distance to the center of the lane d_t , and the angle between the vehicle and the center of the lane ϕ_t $df_t = (v_t, d_t, \phi_t)$

b) Action space A : To interact with the vehicle available in the simulator, commands for throttle, steering, and brakes must be provided continuously. The throttle and brake range are $[0,1]$ and the driving range is $[-1,1]$. Therefore, at each step, the DRL agent must publish an action $a_t = (\text{acc}_t, \text{steer}_t, \text{brake}_t)$ with commands in their scope.

c) The state transition function P_a is the probability that action a in state s at time t will lead to state s_{t+1} at time $t + 1$: $P_a = \text{Prob}(s_{t+1}|s_t, a_t)$

d) Reward function $R(s_{t+1}, s_t, a_t)$: This function generates an immediate reward when moving the agent from s_t to s_{t+1} . The goal in Markov's decision-making process is to generate a "Policy" $\pi(s)$ when that would choose an action for a state. This function will maximize the expectation of future cumulative rewards

$$E = \sum_{t=0}^{\infty} \gamma^t R(s_t, s_{t+1})$$

6.2. CNN-DRL agent

The next step is trying to obtain road features from a vehicle with a front-facing camera via CNN as shown in Figure 8, and from these features to determine the action taken by the vehicle according to the procedure end-to-end and in online mode. To do this, two parts are proposed to set the state vector S , the first part extracts the road features through the CNN, and the second part is formed by two identical Fully Connected layers used in the previous cases.

An RGB image as shown in Figure 7, where the controllable region is marked, of size $[640 \times 480]$ is used as input to the CNN region. CNN consists of three complex layers with 64 pieces of size $[7 \times 7]$, $[5 \times 5]$, and $[3 \times 3]$ respectively, using all of them RELU as activation function and followed by average polling layer. The output of this CNN is fattened and combined with the driving features and the whole state vector is used to provide 2-layers Fully Connected that decide the final action to be taken.

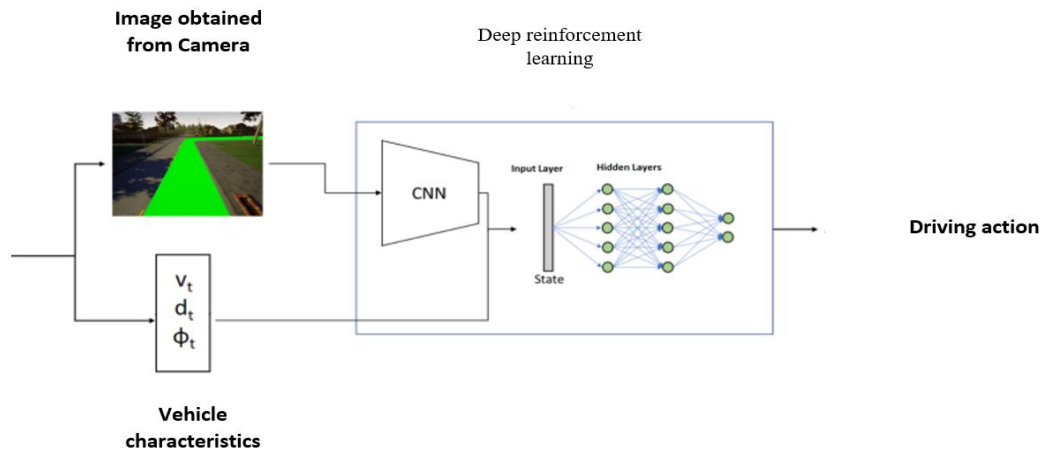


Figure 8. Combining CNN and reinforcement learning in autonomous vehicle navigation

7. The experiment

7.1. Procedure

Three cameras were installed on the autonomous car to collect data. Along with the time-stamped video, the car's telemetry, i.e. the steering command and brake/throttle inputs, were also collected. The steering command was collected via the Controller Area Network (CAN) and displayed in the form $\frac{1}{r}$ (where r is the turning radius in meters). $\frac{1}{r}$ was used over r to avoid singularity on a straight-line drive (the turning radius is now infinity). $\frac{1}{r}$ smoothly transfer from zero to a positive value on a right turn and a negative value on a left turn.

The training data consists of a frame taken from the video, combined with its corresponding driving command ($\frac{1}{r}$). Training the network solely on captured data is inadequate, though. The car needs to learn how to recover from its mistakes; otherwise, the car will slowly drift off the course. Therefore, the training data also contains images that have the car off-center from the lane and on the side of the road.

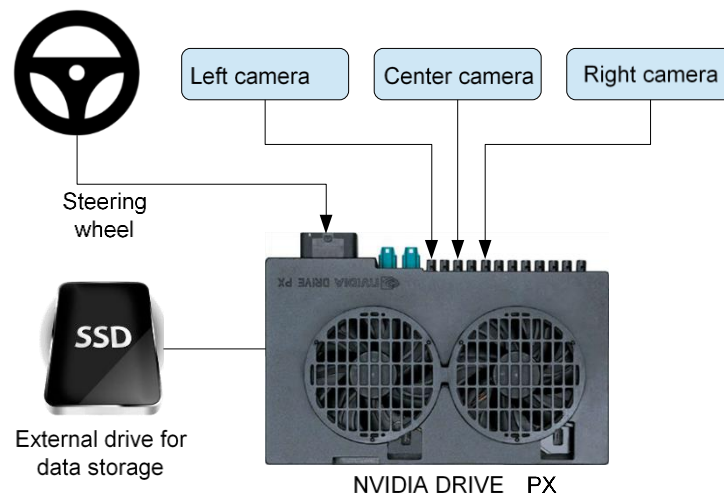


Figure 9. Overview of the data collection system

Images for off-center shifts can be obtained from the left and right cameras. Additional shifts between cameras can also be simulated by shifting the perspective of the nearest camera. Precision in this process requires 3D knowledge, which we don't have. Therefore, we estimated the transformation by assuming that all points below the horizon are on the same surface and all points on the horizon are infinitely far away. This estimate works fine for flat terrains but will cause distortions to other objects e.g. parked cars, lamp posts, plants, and buildings. Fortunately, these distortions do not affect the training process. The transformed

image produced a drive command that would bring the car back to the preferred course in less than two seconds.

A diagram of the training process is provided in figure 9. The image is fed to the CNN to calculate the steering angle. The computed angle is then compared to the required angle so that the weighting can be adjusted to have the output as close as possible to the requirement. The weighting adjustment is performed using back-propagation by the machine learning package.



Figure 10. Network training

After the training process, the network can use a single image from the camera. The process is illustrated in figure 11.

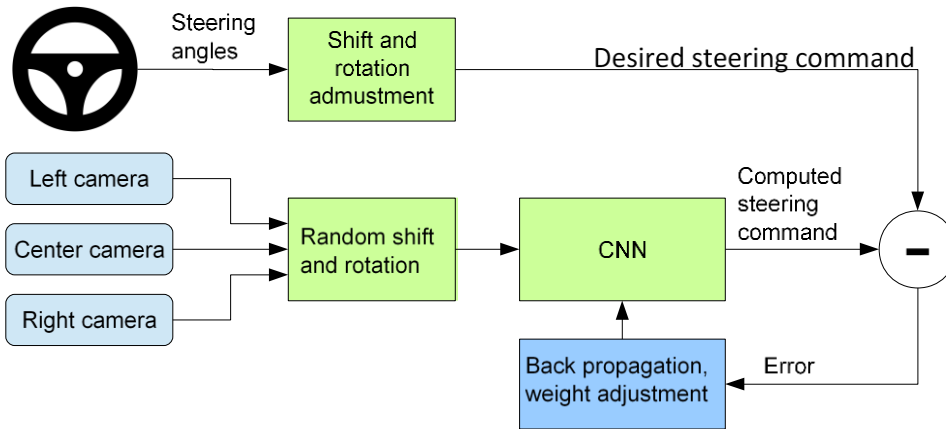


Figure 11. The system used to compute steering commands from a center camera

7.2. Results

The training data was collected from a simulator with the camera running at 30 frames per second. We collected images from the left, right, and center cameras, steering angle, throttle, and brake.

	center	left	right	steering	throttle	reverse	speed
0	center_2022_04_26_15_12_23_403.jpg	left_2022_04_26_15_12_23_403.jpg	right_2022_04_26_15_12_23_403.jpg	0.0	0.0	0.0	0.000079
1	center_2022_04_26_15_12_23_717.jpg	left_2022_04_26_15_12_23_717.jpg	right_2022_04_26_15_12_23_717.jpg	0.0	0.0	0.0	0.000078
2	center_2022_04_26_15_12_23_868.jpg	left_2022_04_26_15_12_23_868.jpg	right_2022_04_26_15_12_23_868.jpg	0.0	0.0	0.0	0.000078
3	center_2022_04_26_15_12_24_009.jpg	left_2022_04_26_15_12_24_009.jpg	right_2022_04_26_15_12_24_009.jpg	0.0	0.0	0.0	0.000083
4	center_2022_04_26_15_12_24_120.jpg	left_2022_04_26_15_12_24_120.jpg	right_2022_04_26_15_12_24_120.jpg	0.0	0.0	0.0	0.000080
...
2634	center_2022_04_26_15_17_59_336.jpg	left_2022_04_26_15_17_59_336.jpg	right_2022_04_26_15_17_59_336.jpg	0.0	0.0	0.0	0.000817
2635	center_2022_04_26_15_17_59_480.jpg	left_2022_04_26_15_17_59_480.jpg	right_2022_04_26_15_17_59_480.jpg	0.0	0.0	0.0	0.000520
2636	center_2022_04_26_15_17_59_618.jpg	left_2022_04_26_15_17_59_618.jpg	right_2022_04_26_15_17_59_618.jpg	0.0	0.0	0.0	0.000596
2637	center_2022_04_26_15_17_59_749.jpg	left_2022_04_26_15_17_59_749.jpg	right_2022_04_26_15_17_59_749.jpg	0.0	0.0	0.0	0.000457
2638	center_2022_04_26_15_17_59_874.jpg	left_2022_04_26_15_17_59_874.jpg	right_2022_04_26_15_17_59_874.jpg	0.0	0.0	0.0	0.000265

2639 rows × 7 columns

Figure 9. The collected data from approximately two minutes of driving

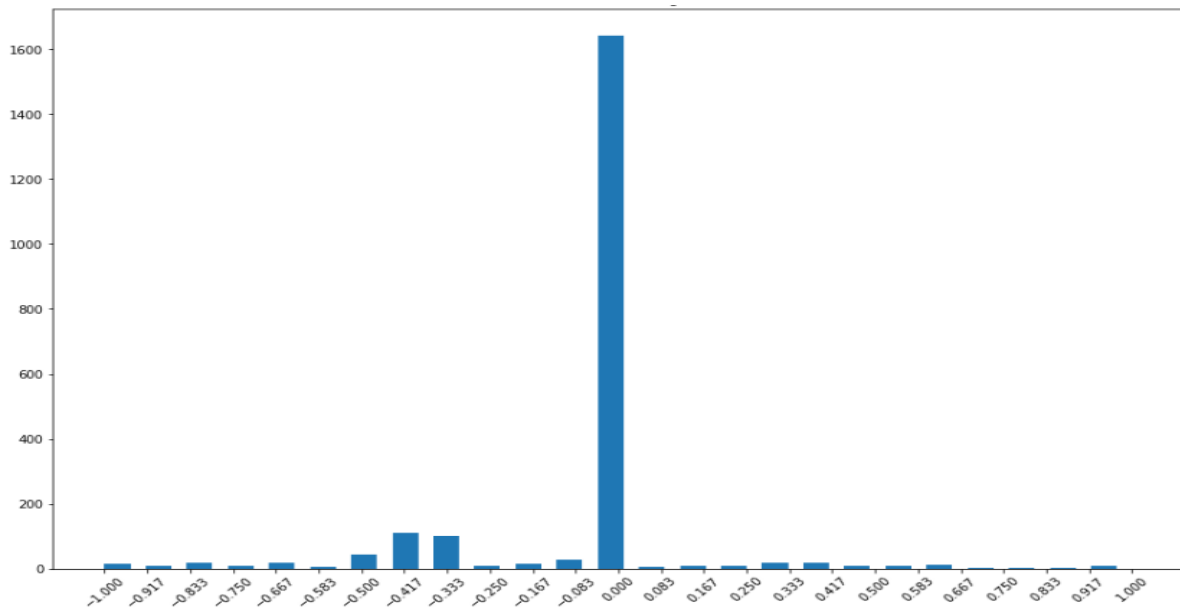


Figure 10. Frequency of the steering angles

We trained the weight of the network to decrease inaccuracies between the computed steering angle and the human steering angle or the steering angle that was altered for the transformations.

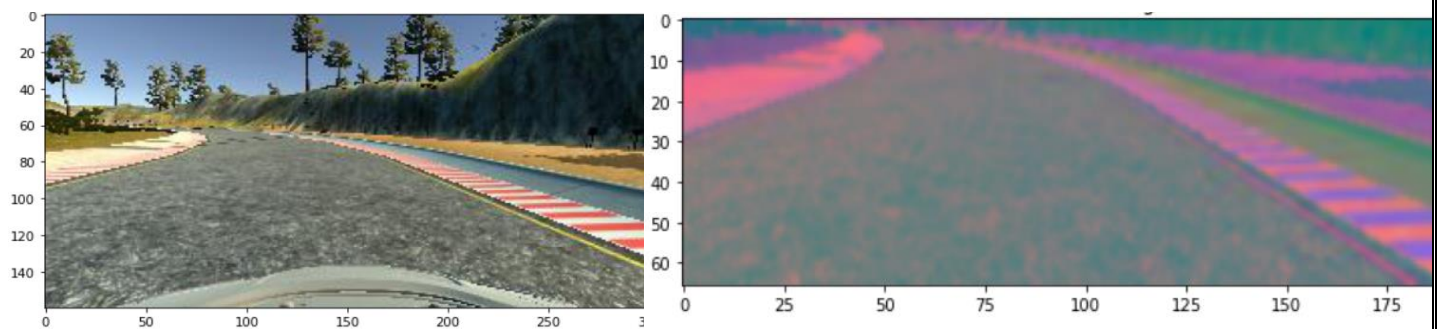


Figure 11. The original image (left) and the processed image (right)

The total number of parameters in the model is estimated to be 252219; therefore, a significant amount of data is required to effectively train the network.

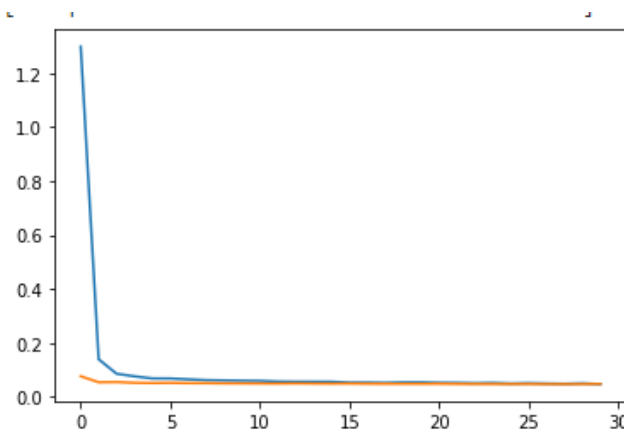


Figure 12. The loss function after training

After 30 epochs, the value of the loss function is near zero so the training process stopped. We have proven that DQN-CNN can recognize the entire road and lane without identifying markings, or signs, or using a pre-programmed route. DQN gets the best results as the number of sets increases, whereas DQN-CNN gets the best models in the early sets and this is why the number of training sessions is larger in DQN. DQN needs more sets to train because its learning uses a decay parameter in the reward chain.

The figure below is a screen capture of the simulator that runs itself with the model stored in a *.h5 file from training. The commands and vehicle's velocity are on the left-hand side, while the vehicle driving is on the right.

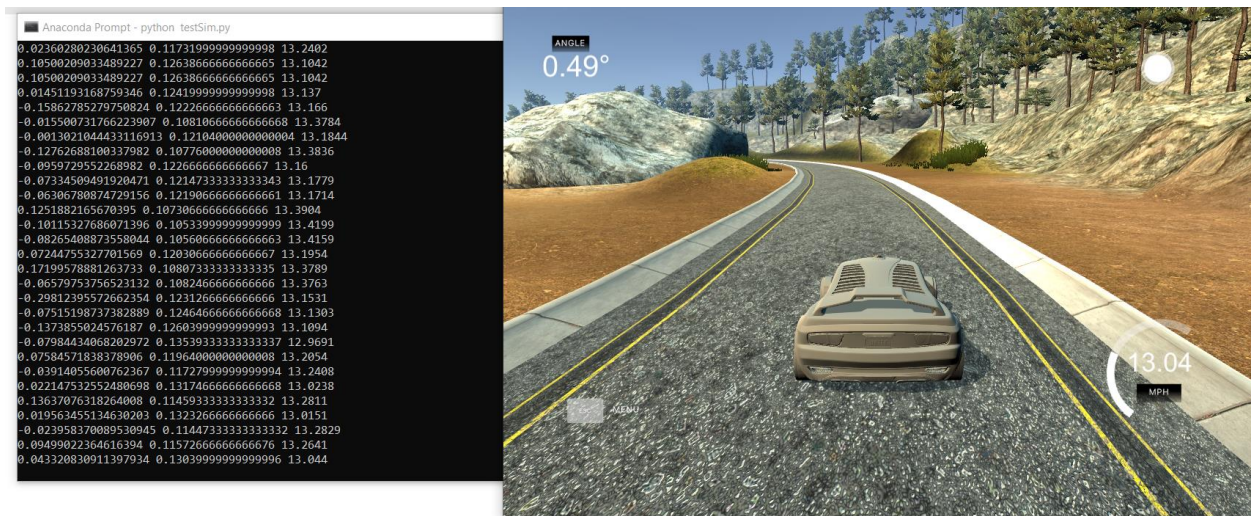


Figure 14. Simulation of the autonomous vehicle.

8. Conclusion

The results reported in this study show how the navigation pattern can be processed in autonomous vehicles using new techniques based on Deep Learning. The DQN-CNN is capable of achieving its goals by controlling the trajectory, and the driving is similar to that of a human driver as it exercises constant control over both speed and steering. We hope that our proposed architecture, based on the DRL control layer, will serve as a solid baseline in modern Autonomous technology Navigator tested in a simulated environment reality.

9. References

- i. Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. *Backpropagation applied to handwritten zip code recognition*. *Neural Computation*, 1(4):541–551, Winter 1989. URL: <http://yann.lecun.org/exdb/publis/pdf/lecun-89e.pdf>.
- ii. Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. *Imagenet classification with deep convolutional neural networks*. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL: <http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>.
- iii. L. D. Jackel, D. Sharman, Stenard C. E., Strom B. I., , and D Zuckert. *Optical character recognition for self-service banking*. *AT&T Technical Journal*, 74(1):16–24, 1995.
- iv. *Large scale visual recognition challenge (ILSVRC)*. URL:<http://www.image-net.org/challenges/LSVRC/>.
- v. *Net-Scale Technologies, Inc. Autonomous off-road vehicle control using end-to-end learning*, July 2004. *Final technical report*. URL: <http://net-scale.com/doc/net-scale-dave-report.pdf>.
- vi. Dean A. Pomerleau. *ALVINN, an autonomous land vehicle in a neural network*. *Technical report*, Carnegie Mellon University, 1989. URL: <http://repository.cmu.edu/cgi/viewcontent.cgi?article=2874&context=compsci>.
- vii. *Wikipedia.org. DARPA LAGR program*. http://en.wikipedia.org/wiki/DARPA_LAGR_Program.
- viii. Danwei Wang and Feng Qi. *Trajectory planning for a four-wheel-steering vehicle*. In *Proceedings of the 2001 IEEE International Conference on Robotics & Automation*, May 21–26 2001. URL: <http://www.ntu.edu.sg/home/edwwang/confpapers/wdwicar01.pdf>.

- ix. *DAVE 2 driving a Lincoln.* URL:<https://drive.google.com/open?id=0B9raQzOpizn1TkRla241ZnBEcjQ>
- x. *Cheein FAA, De La Cruz C, Bastos TF, Carelli R (2010) Slam-based cross-a-door solution approach for a robotic wheelchair. Int J Adv Robot Syst 155–164*
- xi. *Gutiérrez R, López-Guillén E, Bergasa LM, Barea R, Pérez Ó, Gómez-Huélamo C, Arango F, Del Egido J, López-Fernández J (2020) A waypoint tracking controller for autonomous road vehicles using ros framework. Sensors 20(14):4062*
- xii. *Fan J, Wang Z, Xie Y, Yang Z (2020) A theoretical analysis of deep Q-learning. In: Learning for Dynamics and Control. PMLR, pp 486–489*