Tomislav Volarić · Boris Crnokić · Daniel Vasić
Editors

Digital Transformation in Education and Artificial Intelligence Application

Second International Conference, MoStart 2024
Mostar, Bosnia and Herzegovina, April 24–26, 2024
Proceedings

Springer

Editors
Tomislav Volarić
Faculty of Science and Education
University of Mostar
Mostar, Bosnia and Herzegovina

Daniel Vasić
Faculty of Science and Education
University of Mostar
Mostar, Bosnia and Herzegovina

Boris Crnokić
Faculty of Mechanical Engineering,
Computing and Electrical Engineering
University of Mostar
Mostar, Bosnia and Herzegovina

ISSN 1865-0929 ISSN 1865-0937 (electronic)
Communications in Computer and Information Science
ISBN 978-3-031-62057-7 ISBN 978-3-031-62058-4 (eBook)
https://doi.org/10.1007/978-3-031-62058-4

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

We are proud to present the proceedings of MoStart 2024: the 2nd International Conference on Digital Transformation in Education and Artificial Intelligence Applications, held from April 24–26, 2024, in Mostar. This event, hosted by the University of Mostar, continued to foster international collaboration and showcase innovative research in AI and digital education.

The conference featured a broad range of topics including computer vision, natural language processing, and the latest advancements in the digital transformation of education. Notably, the application of artificial intelligence, the incorporation of gamification and robotics into learning processes, and innovative technologies such as IoT were thoroughly explored. The event attracted 32 submissions, with 17 high-quality papers selected for presentation after a meticulous single-blind peer review process.

The conference was highlighted by a keynote speech from Professor Koiti Hasida, which provided fascinating insights into practical applications of graphical methods in enhancing critical thinking skills. Additionally, a panel discussion on the integration of artificial intelligence in education offered a platform for in-depth discussions on current challenges and future directions in this field.

A workshop also underscored the importance of collaboration between the University of Mostar and the University of Split, reinforcing the conference's emphasis on academic partnerships.

We extend our heartfelt thanks to the program committee members and all contributors for their vital roles in the conference's success. Our sponsors are also gratefully acknowledged for their support.

Reflecting on MoStart 2024's achievements, we are encouraged by the productive discussions and new connections formed. We look forward to advancing these conversations in future editions of the conference.

Thank you for your participation in MoStart 2024.

April 2024

Daniel Vasić
Tomislav Volarić
Boris Crnokić

Organization

General Chairs

Tomislav Volarić	University of Mostar, Bosnia and Herzegovina
Boris Crnokić	University of Mostar, Bosnia and Herzegovina

Program Committee Chair

Daniel Vasić	University of Mostar, Bosnia and Herzegovina

Steering Committee

Boris Crnokić	University of Mostar, Bosnia and Herzegovina
Tomislav Volarić	University of Mostar, Bosnia and Herzegovina
Mirela Kundid Vasić	University of Mostar, Bosnia and Herzegovina
Krešimir Rakić	University of Mostar, Bosnia and Herzegovina
Davorka Topić Stipić	University of Mostar, Bosnia and Herzegovina
Daniel Vasić	University of Mostar, Bosnia and Herzegovina
Damir Vasilj	University of Mostar, Bosnia and Herzegovina
Marin Bošnjak	University of Mostar, Bosnia and Herzegovina
Tin Brdar	Ministry of Science, Education, Culture and Sports, The Herzeg-Bosnian Canton, Bosnia and Herzegovina
Emil Brajković	University of Mostar, Bosnia and Herzegovina
Krešimir Čavar	University of Mostar, Bosnia and Herzegovina
Željko Ćorić	Institute of Education, Mostar, Bosnia and Herzegovina
Snježana Damjanović	School Center Martin Nedić (OFM), Orašje, Bosnia and Herzegovina
Josip Doko	University of Mostar, Bosnia and Herzegovina
Goran Dujak	Ministry of Education, Science, Culture and Sports, The Posavina Canton, Bosnia and Herzegovina
Ana Kordić	Ministry of Education, Science, Culture and Sports, The West Herzegovina Canton, Bosnia and Herzegovina

Hrvoje Ljubić	University of Mostar, Bosnia and Herzegovina
Maja Marić	University of Mostar, Bosnia and Herzegovina
Anton Martinović	University of Mostar, Bosnia and Herzegovina
Petar Matić	University of Mostar, Bosnia and Herzegovina
Vedran Mihalj	University of Mostar, Bosnia and Herzegovina
Manlio Napoli	University of Mostar, Bosnia and Herzegovina
Ivan Ostojić	University of Mostar, Bosnia and Herzegovina
Tomislav Papac	University of Mostar, Bosnia and Herzegovina
Ana Pinjuh	University of Mostar, Bosnia and Herzegovina
Karlo Popović	University of Mostar, Bosnia and Herzegovina
Robert Rozić	University of Mostar, Bosnia and Herzegovina
Jelena Skoko	Institute for Upbringing and Education, Bosnia and Herzegovina
Robert Slišković	University of Mostar, Bosnia and Herzegovina
Goran Škvarč	Croatian Academic and Research Network – CARNET, Croatia
Valentina Vidović	Ministry of Science, Education, Youth, Culture and Sports, Bosnia and Herzegovina
Franjo Vučić	University of Mostar, Bosnia and Herzegovina
Selena Knežić Buhovac	University of Mostar, Bosnia and Herzegovina
Inja Stojkić	University of Mostar, Bosnia and Herzegovina

Program Committee

Sanja Bijakšić	University of Mostar, Bosnia and Herzegovina
Mirjana Bonković	University of Split, Croatia
Ivo Čolak	University of Mostar, Bosnia and Herzegovina
Juan Manuel Fernández Luna	University of Granada, Spain
Irena Galić	University of Osijek, Croatia
Sven Gotovac	University of Split, Croatia
Tamara Grujić	University of Split, Croatia
Rainer Herpers	Bonn-Rhein-Sieg University of Applied Sciences, Germany
Branko Katalinić	Vienna University of Technology, Austria
Zdenko Klepić	University of Mostar, Bosnia and Herzegovina
Goran Martinović	University of Osijek, Croatia
Pedro Miguel Moreira	Polytechnic Institute of Viana do Castelo, Portugal
Vladan Papić	University of Split, Croatia
Ljiljana Šerić	University of Split, Croatia
Maja Braović	University of Split, Croatia

Marko Rosić	University of Split, Croatia
Slavomir Stankov	University of Split, Croatia
Zoran Tomić	University of Mostar, Bosnia and Herzegovina
Drago Žagar	University of Osijek, Croatia
Boris Crnokić	University of Mostar, Bosnia and Herzegovina
Malik Čabaravdić	University of Zenica, Bosnia and Herzegovina
Ani Grubišić	University of Split, Croatia
Tonćo Marušić	University of Mostar, Bosnia and Herzegovina
Jonathan Schler	Holon Institute of Technology, Israel
Jan Snajder	University of Zagreb, Croatia
Hrvoje Novak	University of Zagreb, Croatia
Danijel Topić	University of Osijek, Croatia
Tomislav Volarić	University of Mostar, Bosnia and Herzegovina
Branko Žitko	University of Split, Croatia
Krunoslav Žubrinić	University of Dubrovnik, Croatia
Elisabete Cunha	Polytechnic Institute of Viana do Castelo, Portugal
Bárbara Cleto	Polytechnic Institute of Porto, Portugal
Angelina Gašpar	University of Split, Croatia
Janez Gotlih	University of Maribor, Slovenia
Miroslav Grubišić	University of Mostar, Bosnia and Herzegovina
Timi Karner	University of Maribor, Slovenia
Mirela Kundid Vasić	University of Mostar, Bosnia and Herzegovina
Nikola Ljubešić	University of Ljubljana, Slovenia
Željko Marušić	University of Mostar, Bosnia and Herzegovina
Mirza Oruč	University of Zenica, Bosnia and Herzegovina
Ivan Peko	University of Split, Croatia
Krešimir Rakić	University of Mostar, Bosnia and Herzegovina
Višnja Simić	University of Kragujevac, Serbia
Suzana Tomaš	University of Split, Croatia
Daniel Vasić	University of Mostar, Bosnia and Herzegovina
Josip Vasilj	University of Split, Croatia
Nikolina Maleta	University of Mostar, Bosnia and Herzegovina
Lada Maleš	University of Mostar, Bosnia and Herzegovina
Iva Klepić	University of Mostar, Bosnia and Herzegovina

Artificial intelligence-based control of autonomous vehicles in simulation: a CNN vs. RL case study

Ive Vasiljević[1][0009−0005−5270−2973], Josip Musić[1][0000−0002−9185−5762], and José Lima[2][0000−0001−7902−1207]

[1] Faculty of Electrical Engineering, Mechanical Engineering and Naval Architecture, University of Split, 21000 Split, Croatia
{ive.vasiljevic.00,jmusic}@fesb.hr

[2] Research Centre of Digitalization and Intelligent Robotics, Instituto Politécnico de Bragança, 5300-253 Bragança, Portugal
jllima@ipb.pt

Abstract. The article provides a comparison of Convolutional Neural Network (CNN) and Reinforcement Learning (RL) applied to the field of autonomous driving within the CARLA (CAr Learning to Act) simulator for training and evaluation. The analysis of results revealed CNNs better overall performance, as it demonstrated a more refined driving experience, shorter training durations, and a more straightforward learning curve and optimization process. However, it required data labelling. In contrast, RL relayed on an exhaustive (unsupervised) exploration of different models, ultimately selecting the model at timestep 600,000, which had the highest mean reward. Nevertheless, RL's approach revealed its susceptibility to excessive oscillations and inconsistencies, necessitating additional optimization and tuning of hyperparameters and reward functions. This conclusion is further substantiated by a range of used performance metrics (objective and subjective), designed to assess the performance of each approach.

Keywords: reinforcement learning · CNN · CARLA simulator.

1 Introduction

The autonomous driving phenomenon has significantly advanced over the years of its development [1]. The automotive tech industry has made momentous enhancements to the capability and reliability of sensors, cameras, and vehicle-to-everything (V2X) communication [2]. Collectively, these features generate many consumer benefits, but they also may produce additional value for the auto industry [3, 4]. To ensure that automotive companies are following a well-established standard, whilst announcing their progress on the state of their autonomous driving systems, the Society of Automotive Engineers (SAE), a standards-developing organization, has defined six levels of driving automation. Each level puts fewer constraints on the human driver, meaning less attention to the road and actions

are needed. Additionally, each level provides more features, like line centering and adaptive cruise control, automatic emergency break, etc.

The current state of the autonomous driving industry is monopolized by larger companies including Tesla, Ford, Mercedes, General Motors, and Kia / Hyundai [5]. Notably, these companies are considered to have the most reliable systems in place, however, they seem to differ in what SAE level they fall into. The best level most companies have to commercially offer, including Tesla, is only Level 2, as defined by the SAE standard [5]. As of today, Mercedes has rolled out a self-certified Level 3 system (Drive Pilot). However, this standard needs very specific conditions to work, including the feature requests, where a human driver needs to take over the control of the vehicle. The Drive Pilot, doesn't exceed speeds of 45 mph (cca. 72 km/h), in conditions of clear weather during the day, and only on roads mapped by the system [5].

These autonomous driving systems are made possible using numerous sensors attached to a car, giving meaningful information about the environment. Even with all the information available, research suggests that perhaps creating autonomous driving systems was more difficult than first predicted [6, 7]. In [7], it is stated that 80% of self-driving is relatively simple – making the car follow the line of the road, sticking to a certain side, and avoiding collisions. The next 10% involve more difficult situations such as roundabouts and complex junctions. The last 10% has proven to be arduous, which covers the problem of "edge cases". They are rare cases that can occur on the road, such as an animal being in the middle of the road, or a child running across the street. This final 20% is why the autonomous driving industry is still not making any significant progress.

While developing autonomous driving systems, there are numerous questions to be explored, including how many sensors to use, what information to use from those sensors, for what purpose to use them, and which algorithms to use alongside these sensors [8]. Sensors like Light Detection and Ranging (LiDAR), radar, and cameras are among a few that are routinely used to sense the world around vehicles. The hardware used in autonomous vehicles has remained relatively stable in recent years, while the software has undergone constant updates and changes. This dynamic nature of software development makes it crucial to demonstrate the capabilities of various learning algorithms especially in the domain of self-teaching cars where vehicles are equipped with advanced artificial intelligence systems that enable them to learn from experiences and adapt their behavior to improve their driving skills autonomously. Furthermore, with the rising popularity of end-to-end (E2E) learning, which is a method where a system learns to perform a task directly from input to output, without relying on many intermediate steps, Convolutional Neural Network (CNN) has proven to be an efficient tool for processing various types of raw data and generating meaningful predictions [9]. While CNNs are well-known for their proficiency in handling image and video data (e.g., for tasks like object detection [10], lane recognition [11], and traffic sign classification [12] in autonomous driving systems), they are also applicable and effective in other domains. For instance, they excel in tasks like speech recognition (audio data) [13], natural language processing

(text data) [14], and even certain medical diagnoses, where patterns and features can be automatically learned from the data [15]. Another approach that has shown remarkable success in autonomous driving is Reinforcement Learning (RL) [16–18]. Through RL, autonomous vehicles can learn complex driving behaviors and decision-making skills, such as handling various traffic scenarios, obeying traffic rules, and adapting to different road conditions. By iteratively improving through trial and error, RL algorithms can achieve acceptable levels of autonomy in driving tasks.

Obviously, due to safety concerns and ease of implementation (i.e. testing different scenarios, including low probability ones), it is not possible to initially test/develop new software for autonomous vehicles (including various artificial intelligence algorithms) in the real-world scenario [19]. Realistic computer simulations have thus emerged as a viable and practical solution to the problem. Much research [9, 17, 18] has been focused on it in recent years, with some interesting results. For example, in [18] end-to-end driving policies were developed in CARLA (CAr Learning to Act) simulator [20], and later deployed on a full-sized car (over 9 driving scenarios and more than 400 test drives). RL was used as a basis for developing different driving policies with the vehicle being controlled via throttle and steering variables and an RGB camera used as a main sensor. During testing, authors noted a simulation-to-reality gap which resulted in degraded performance in real-world scenarios. They also noted the positive influence on the performance of regularization and augmentation.

The lack of detail and information richness of computer-generated images compared to real-world images has also been noted in [21] in which increased usage of computer game engines for simulation has been described. For example, the LGSVL simulator (whose active development has now been stopped) is based on the Unity game engine [22], while AirSim uses Unreal Engine [23]. Research [24] has also gone into determining the deterministic level of simulators, namely the CARLA simulator. It was shown that it was non-deterministic, with actor collisions and system resource utilization being identified as the main sources of it. Authors suggest that these parameters need to be monitored for improved performance and that some commercially available simulators could be considered for more consistent performance. Extensive overviews of modeling and simulation approaches used in autonomous vehicles can be found in [25, 26].

AI-based algorithms can be found in many tasks associated with autonomous vehicles (in a simulation and real-world scenarios) [27], with (deep) neural networks usually used for perception tasks, decision-making and control, and RL for optimizing driving behavior (control). In [28], a CNN was used in an end-to-end fashion for inference of vehicle steering angle and throttle position within the simulation environment (Udacity open-source simulation platform). Data augmentation was used for generating training data, which resulted in collision-free driving within the tested examples. In [9] CNN-based end-to-end approach (using RBG road images) was also used, but with architecture (and thus computational complexity) optimization in mind. The approach was tested/implemented in the CARLA simulator and compared with nVIDIA's PilotNet. Obtained per-

formance results were comparable (with the difference in Mean Square Error - MSE - of 0.00004), but the resulting network was approximately 40x lighter than PilotNet i.e. it had 4x fewer parameters resulting in somewhat lower inference time. However, it should be noted that PilotNet was tested in real-world scenarios with good results [29], while the proposed architecture was tested in simulation only.

Two additional examples in which the CARLA simulator was used, but with a different artificial intelligence algorithm, RL, are [17, 30]. In [17] Deep-Q-Network (DQN) and Deep Deterministic Policy Gradient (DDPG) algorithms were implemented and compared within the simulator. Archived control was compared with traditional controllers like Linear Quadratic Regulator (LQR) with DDPG achieving better results (than DQN) scoring Root Mean Square Error (RMSE) of about 0.1 m (on tracks up to 700 m long). However, the authors used inserted waypoints (employing a path planning algorithm) to achieve desired results, and not, more standard, sensor output like an RGB camera or a LiDAR. The model-free RL algorithms were also used in [30] for controlling an autonomous vehicle in a complex urban scenario like a busy roundabout (with about 50 vehicles in its vicinity in the used map). Algorithms were given a raw bird-eye view of the map as their input (which is not a realistic scenario). All tested algorithms successfully entered the roundabout in 80% of cases but only the Soft Actor-Critic (SAC) algorithm managed to reach the desired goal point in 58% of test cases. It was found that almost all failure cases came from rear-ending the front vehicle.

Based on the presented brief literature review it can be seen that CNNs and RL are among the most widely used AI-based algorithms for autonomous vehicle control. However, comparative analysis of the two in the same scenario is largely missing from the available literature. Thus, this paper aims to develop and implement CNN and RL (with varying implementation hyperparameters) within the CARLA simulator to analyze and compare their results (on the same track) alongside observed implementation details.

2 Materials and methods

This section focuses on the implementation of CNN and RL in the CARLA simulator. Each method is explained clearly and concisely, highlighting the differences in their implementation.

2.1 Artificial intelligence-based algorithms

Artificial Neural Networks An Artificial Neural Network (ANN), also known simply as a NN, is a learning system that uses a network of interconnected neurons (nodes) to process inputs and generate desired outputs. The NN itself may be used as a part of other machine learning algorithms, to process complex data inputs into a space that computers can understand [31]. Additionally, it's important to note that there exist numerous types of ANNs, each tailored for specific purposes and accommodating different types of input data. For instance,

CNN excels in image analysis, making it ideal for processing two-dimensional grid-like data, such as images, while Recurrent Neural Network (RNN) is well-suited for sequential data processing, which includes time series data, natural language text, and speech.

NNs have a wide variety of applications, but recently they also faced some limitations that make them not universally applicable to every problem. First and foremost, NNs are hardware-dependent, requiring additional efforts to handle more complex problems. Also, due to their "black box" nature, it becomes challenging to determine the exact degree to which each independent variable affects the dependent variables [32]. Additionally, training data plays a crucial role in NNs, leading to issues like over-fitting and generalization problems, where the model may become too tailored to the specific training data. To perform effectively, NN demands a colossal volume of data, which can be particularly problematic in situations where data collection and labeling are arduous and expensive (of which autonomous driving is a good example). Gathering and annotating such extensive datasets can be resource-intensive, often necessitating substantial human effort and expertise. Despite their limitations, NNs offer numerous advantages that make them versatile [31]. For instance, their adaptability allows them to be applied to both regression and classification problems. Being mathematical models with approximation functions, they can handle any data that can be converted into a numeric format. Moreover, NNs excel at modeling nonlinear data with multiple inputs, such as images. Once trained, they provide fast predictions, and they can be designed with any number of inputs and layers.

When considering the processing of images, a fundamental distinction arises between ANNs and CNNs [31]. ANNs tend to employ a comprehensive network of neurons and connections, resulting in a vast number of parameters that can lead to computational inefficiencies, particularly for larger images. In contrast, CNNs adopt a more efficient approach, leveraging local parameter sharing within convolutional layers to efficiently extract hierarchical features from images. As an example, in [33] comparison between ANN-based and CNN-based image classification was performed. In terms of the number of used parameters for particular networks, the deep ANN network (with five dense layers) had about 31.4 million trainable parameters, while the CNN network (with three convolutional layers and two dense layers) had about 0.4 million trainable parameters, which is a reduction of about 78x. In terms of achieved accuracy for the proposed networks, ANN achieved 90% while CNN achieved 97%.

Reinforcement learning In contrast to conventional NNs reliant on labeled datasets for supervised learning, in RL, agents acquire knowledge through dynamic interactions (in unsupirvised manner) with an environment, optimizing actions to maximize cumulative rewards. Unlike supervised learning's dependency on curated and labeled datasets, RL offers a framework where agents autonomously adapt and learn in intricate real-world settings, showcasing its potential for autonomous learning in complex scenarios. Therefore, RL consists of four main parts [16]:

- **Agent** – entity that perceives the environment and acts upon it.
- **Environment** – agent's world in which it interacts.
- **Policy** – a strategy used by the agent for the next action based on the current state.
- **Reward** – signal that the agent observes upon taking actions.

To formalize RL issues, the Markov Decision Process (MDP) is utilized. The agent in MDP continually interacts with the environment and performs actions; the environment responds to each action and develops a new state.

RL offers a range of learning algorithms. Broadly, they are categorized as model-free and model-based algorithms [35], with a simple differentiation: if an agent can predict the next state and reward before taking its next action after learning, it's a model-based RL algorithm; otherwise, it's model-free. Within the model-free category, there is a further division into on-policy and off-policy techniques. On-policy methods focus on improving the policy used to make decisions, while off-policy techniques employ a separate behavioral policy to explore the environment and gather samples. These samples influence both the agent's behavior and the learning of a refined policy called the target policy. In practice, model-free algorithms are more prevalent than model-based ones [35]. It's worth noting that the applicability of each algorithm depends on the agent's action space, whether deterministic or stochastic.

One of the key advantages of RL is its ability to tackle complex and challenging problems that are often beyond the reach of traditional methods. These traditional methods, such as rule-based systems or supervised learning, may struggle when confronted with tasks involving long-term planning, sequential decision-making, and intricate interactions with an environment. RL, on the other hand, excels in achieving long-term objectives and navigating intricate problem spaces. Furthermore, the learning paradigm is very comparable to human learning. But naturally, as with everything, there are limitations to RL, that make it unusable in some situations. For basic problems, RL is not the best option, it needs a large amount of data and processing, and last but not least for genuine physical systems, the curse of dimensionality limits RL. Most of the limitations are very specific and generally, its many advantages make it more desirable and hence this widely used [16, 35].

2.2 CARLA simulator

The CARLA Simulator [20] is a powerful open-source tool for autonomous driving research and development. It provides a realistic virtual environment where machine-learning models can be trained, tested, and evaluated without the need for real-world cars or physical environments. CARLA offers a range of functionalities to create immersive and challenging scenarios. CARLA simulates a variety of real-world elements, such as urban environments, roads, traffic, pedestrians, and dynamic objects as illustrated in Figure 1. By accurately modeling these components, CARLA provides a sense of realism that is essential for training and testing autonomous driving algorithms. The simulator allows users to

manipulate various parameters, such as weather conditions, lighting, and traffic density.

Fig. 1: Examples of default CARLA maps [20]

One of the key advantages of CARLA is its ability to generate annotated data for training machine learning models. By integrating with the simulator, researchers can collect vast amounts of labeled data, including sensor data (such as camera images, LiDAR point clouds, and radar readings) and ground truth information (such as object positions and semantic segmentation). This data can then be used to train and validate the performance of machine learning models in a controlled and reproducible manner. It provides an Application Programming Interface (API) and libraries, including Python and C++, for customization and integration with external tools and frameworks. Furthermore, CARLA supports the implementation of RL algorithms, where agents learn to navigate and make decisions in the simulated environment through trial and error. However, a persistent challenge is the simulation-reality gap [36], where simulations may not perfectly mirror real-world complexities.

CARLA simulator operates using a client-server architecture, where the server generates and maintains the simulation environment, and the client interacts with the server to control vehicles, access sensor data, and receive simulation updates. Clients connect to the server, which runs simulations using Unreal Engine 4 and CARLA Plugins. The server manages physics and rendering, whereas clients can retrieve data and issue commands via the Python or C++ CARLA API, with Python offering ease of use and C++ providing performance.

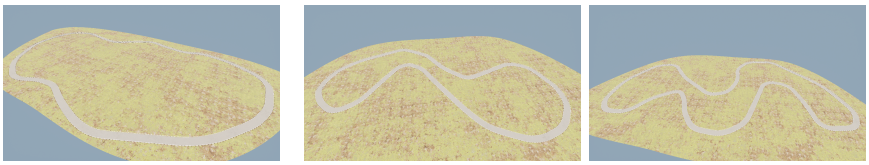
During the experiments, CARLA version 0.9.13 was utilized as the foundation for developing and testing the AI-based learning methods. Building CARLA from source allowed for the incorporation of additional features and capabilities, as well as the utilization of the Unreal Editor for asset creation and map manipulation. For creating custom roads and scenarios, a software called RoadRunner was used, courtesy of MathWorks [37]. RoadRunner is an interactive editor that enables designing 3D scenes for simulating and testing automated driving systems.

2.3 Experimental Setup

All the training, testing, and overall development was made on a machine with an NVIDIA GeForce GTX 1070 with 8 GB GDDR5 memory graphics card, 16 GB of DDR4-3200 MHz RAM, a 512 GB NVMe SSD, and Intel Core i5-11600K with 6 cores and 12 threads.

Convolutional Neural Network (CNN) The main emphasis of the CNN implementation lies in five fundamental elements: the setup of the environment, data collection, data preprocessing, data splitting, and an overview of the network architecture.

Enviroment setup To achieve diversity and data generalization, five distinct maps were generated and exported using RoadRunner. These maps shared a consistent environment and road style, differing primarily in their road layouts. Urban features such as houses, trees, bins, and hydrants were intentionally kept minimal to streamline image collection and reduce map size. Each map included both clockwise and anti-clockwise variants to capture data from both driving directions. Furthermore, these maps feature level road surfaces without significant inclines or declines. Four out of the five maps were allocated for training and validation, while the remaining map was dedicated to testing the trained model. Figure 2 depicts all five maps, and how each of them had a different road shape. Upon the completion of map creation in RoadRunner, the maps were exported, and the final enhancements were applied within the Unreal Editor. These enhancements included the addition of imperceptible features, such as specific spawn points on the map and autopilot waypoints.



(a) First map (b) Second map (c) Third map

(d) Fourth map (e) Test map

Fig. 2: Training and testing maps used during the experiments

Data Collection To collect data for training, first, a single camera sensor was attached to the vehicle, designed for collecting RGB images. The camera

was placed at the front of the car, a bit below the vehicle's hood. The vehicle that was used during training and testing was a Tesla Model 3. The images were saved to a specific folder using the built-in static function, which saves an image after every frame of the simulation running. Using a specific property in CARLA's world settings, the FPS during the data collection process was fixed at 33.33.

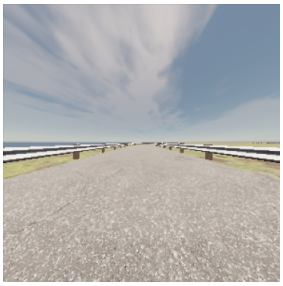
One lap was made around the map in each direction, providing a thorough exploration of the entire road, and making a closing loop. On average, each lap took 4 minutes and 36 seconds to complete, therefore for each map around eight to nine minutes, depending on the size of the map and the intensity of the curvatures. The size of the saved images was $300 \times 300 \times 3$. Along with the images, steering angle and throttle values were also recorded. Steering values went from -1 to +1, and throttle values go from 0 to +1. Since CARLA functions in a client-server principle, the simulator was running in synchronous mode, which means that the client and server were running at the same time.

For the best possible quality of the data collected, a Tesla Model 3 with CARLA autopilot was used. The autopilot followed carefully positioned waypoints, and in doing so, it collected steering angle values that contribute to the feature of lane centering. Additionally, by adding invisible features, such as speed limiters, the vehicle knew to maintain a reasonable speed before a curve. The entire behavior replicates human-like driving patterns. This choice was made to address the issue of not controlling a vehicle with a basic keyboard or joystick, which tends to generate noisy data, leading to model instability.

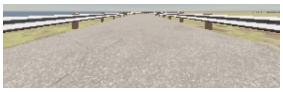
Furthermore, the data collection process was fully automated, eliminating the need for human supervision and reducing the demands on human attention. After collecting data from training maps, in total 29,606 instances of data were saved and consequentially used for training. The captured images took just under 6 GB of disk memory.

Data Preprocessing After collecting the images, one of the essential parts of the whole process was preprocessing the images before using them for the training. Figure 3 shows the progressive changes made to an example image by applying all the preprocessing steps. The same steps were applied to the whole dataset of images during training and testing. Note that the images were captured with a specific Field-Of-View (FOV) attribute set at 130^o.

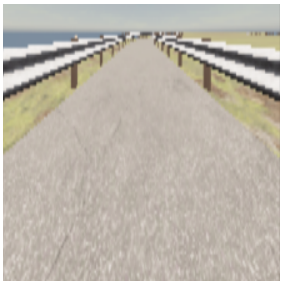
The original image is presented in Figure 3a. The initial step in data preprocessing involved cropping the images to extract the Region of Interest (ROI) focused on the road ahead, resulting in new image dimensions of $90 \times 300 \times 3$, as demonstrated in Figure 3b. This step primarily reduced the image's height to exclude the sky. Subsequently, the images were resized to $224 \times 224 \times 3$ to maintain a square aspect ratio, as depicted in Figure 3c. Following resizing, the images were converted to grayscale, reducing the color channels to a single channel, as shown in Figure 3d. Additionally, Gaussian blur was applied to minimize image noise and enhance overall smoothness. Although this step is not visually apparent, it holds practical significance. Lastly, the images underwent normalization using the mean and the standard deviation of the training dataset, which



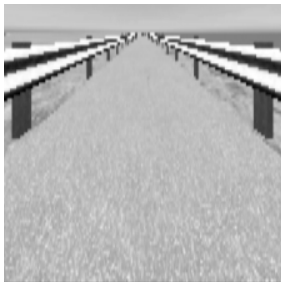
(a) Example of original image



(b) Example of cropped image



(c) Example of resized image



(d) Example of grayscaled image

Fig. 3: Examples of image during various stages of the data preprocessing stage

contributes to stabilizing the gradient descent process. This stabilization enables the utilization of larger learning rates and expedites model convergence, as discussed in [38, 39]. Figure 4 illustrates the final image appearance after applying all augmentation techniques, which serves as the basis for training.

Fig. 4: Example image after applying all the preprocessing techniques including normalization

The images were normalized by subtracting the mean (μ) of each feature and a division by the standard deviation (σ). This way, each feature has a mean of 0 and a standard deviation of 1 [40]. The normalization follows the Equation 1.

$$x = \frac{x - \mu}{\sigma} \tag{1}$$

Before normalization, the mean and standard deviation of the entire image dataset were calculated. The mean was calculated by summing the pixel values divided by the total number of pixel values. After that, the standard deviation was calculated with the Equation 2.

$$\sigma = \sqrt{E[X^2] - (E[X])^2} \tag{2}$$

where $E[X^2]$ represents the expectation of the mean of the squared data, while $(E[X])^2$ represents the expectation of the square of the mean of the data [40].

Data Splitting When the data was collected, the first four maps were used. Furthermore, that data was then split into data for training and data for validation with the ratio of 80:20, thus out of a total number of 29,606 instances of data, for training, 23,685 instances of data were used, and 5,921 instances for validation.

Additionally, every time the data was split, the images were shuffled when choosing which images would be used for training and which for validation. Notably, the usage of shuffling distorts the temporal order of the dataset. In some cases where a dataset suffers from sequential dependencies, such as trajectory prediction in the context of complex traffic situations, maintaining that order can be important. However, in situations where temporal order is less critical,

such as object detection in individual frames, shuffling can be advantageous to prevent the model from learning unintended patterns. Assuming the dataset used in this work is simple enough to not be affected by the distortion of the temporal order (in part enabled by choosing simple maps, as described previously), the option of not shuffling the dataset was not explored, as the results were already satisfactory.

Network Architecture The proposed architecture draws inspiration from a well-established architecture developed by Lee, as documented in the GitHub repository [41]. The network uses a lightweight approach in the sense of how many convolutional layers were used, while at the same time maintaining high accuracy. During the implementation of the network, there were additional experiments performed regarding some parameters, such as the dimensions of the images, batch sizes, color model for the images, and number of features for some layers. After experiments, the following architecture was chosen, as depicted in Figure 5.

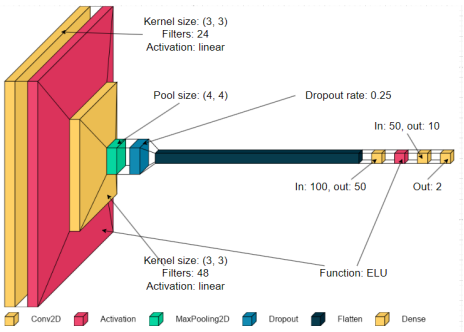


Fig. 5: Simplified architecture of the used CNN

The network was divided into two distinct sections. The initial segment focuses on convolution and comprises two convolutional layers, one max-pooling layer, a dropout layer with a 25% dropout rate, and employing the Exponential Linear Unit (ELU) activation function. The first convolutional layer employs a 3×3 kernel with a stride of 2, followed by ELU activation. Subsequently, the second convolutional layer utilizes a similar 3×3 kernel with a stride of 2. A max-pooling layer with a 4×4 kernel and a stride of 4 follows these convolutional layers.

The second section marks the transition from the convolutional layers to the linear NN by flattening the pooled feature map. This transformation retains spatial information while converting it into a format suitable for fully connected layers. The first linear layer, which follows, consists of 8,112 input neurons and 50 output neurons, a direct consequence of the preceding convolutional layers and their operations on the input image. Specifically, the initial 48×13×13 dimension resulting from the convolutional layers transforms into 8,112 neurons

after the flattening process. An additional ELU activation function enhances convergence before the second linear layer, which has 50 input neurons and 10 output neurons. The final linear layer comprises 10 input neurons and concludes with 2 output neurons responsible for predicting steering angles and throttle values (i.e. regression-type output). The transition from a 48×13×13 dimension in the convolutional layers to a single vector of 8,112 neurons enables seamless progression to the fully connected layers. Model training employed the Adam optimization algorithm with a learning rate set at 0.0001. To monitor network performance during training, MSE loss serves as the criterion.

In the end, the model ended up with 416,838 trainable parameters, with the first part containing 10,656 parameters and the second (flattened) part 406,182 parameters.

Reinforcement Learning (RL) RL implementation involves several key steps. Firstly, setting up the RL environment is crucial, providing a platform for the agent to interact and learn. Next, defining the action and observation spaces allows the agent to explore and perceive the environment effectively. Establishing a reward function guides the agent toward its objectives. Lastly, choosing the appropriate RL algorithm enables the agent to iteratively learn and improve its decision-making process.

Enviroment setup As the basis for the implementation of the RL, a package called Stable-Baseline3 (SB3) was used. It was introduced in [42] as a collection of dependable RL algorithms implemented in PyTorch. To use these RL algorithms, a custom environment had to be implemented to work seamlessly with SB3. Custom environment in this case is a simple class that inherits from an OpenAI Gym Class [43], which means it has to implement five basic methods: *init*, *step*, *reset*, *render*, and *close*. The rendered screen obtained using Pygame (as an output from the render step) is illustrated in Figure 6. Please note the upper left corner of the figure in which important RL and vehicle-related data was presented.

Fig. 6: Rendered screen during RL exploration with all the relevant information

Action and Observation Spaces In the context of vehicle control, values for steering angle and throttle belong in the continuous space. Whilst dealing with such continuous control variables, it is more natural and efficient to represent the action space as a continuous domain. This allows the agent to learn policies that can handle different values within the specified ranges. Moreover, it enables the agent to explore a wide range of actions during training, making it more adaptable to various driving scenarios and ensuring it can handle subtle variations in steering and throttle requirements. Space domain for the steering angle and throttle are defined as shown in Table 1.

Table 1: Action space minimum and maximum values

	Min	Max
Steering ang.	-1	1
Throttle	0	1

Furthermore, the observation space consists of an image and speed component. By combining the processed image and the speed value, the observation space provides the necessary information to the agent so that it can make informed decisions about how to control the vehicle based on what it "sees" and how fast it's moving. Please note that the images were processed the same way as for CNN-based implementation.

Reward Function The reward function is crucial as it serves as the guiding signal for the agent's learning process. When the agent takes an action that leads to a positive outcome or brings the agent closer to achieving its goal, it receives a higher reward. Conversely, if the action results in a negative consequence or moves the agent away from its objective, it receives a negative reward. Therefore, it was essential to design a well-structured reward function that encourages optimal behavior. In this context, the principles for reward assignment were as follows:

$$\textit{If the car has collided with the road barrier}$$
$$R = -200 \tag{3}$$
$$\textit{If the car has not collided and speed is 0}$$
$$R = -5 \tag{4}$$
$$\textit{If the car has not collided and speed is } s$$
$$R = \min\left(r, \frac{s'}{s} \times r\right) \tag{5}$$

The Equation 3 describes a scenario in which the vehicle collides with the road barrier, resulting in a penalty of -200. In contrast, Equation 4 addresses a scenario in which the vehicle remains stationary, attempting to exploit the fact that by remaining stationary it will avoid the higher penalty of colliding and forcing it to move around the environment, incurring a smaller penalty of -5.

The Equation 5 employed in the scenario where the vehicle neither collides nor remains stationary relies on three variables. The current speed, denoted as *s'*, is compared to the minimum desirable speed, denoted as *s*. As *s'* approaches *s*, the function increases linearly towards the maximum reward, *r*. Once the current speed equals or exceeds *s*, the function consistently returns the maximum reward. This design choice effectively encourages the model to strive for the highest possible speed without imposing an artificial upper limit on the speed itself. The decision was made to omit the imposition of a maximum speed threshold in the reward function, as it did not present a significant concern or challenge in the context of this thesis. In this instance, the reward r can vary between 0.033 and 5, with the minimal desired speed s set at 15 km/h.

Algorithm and Parameters In the process of selecting a suitable RL algorithm, a thorough evaluation was crucial. Several established algorithms, including Proximal Policy Optimization (PPO), DDPG, Twin Delayed DDPG (TD3), and SAC, were tested. The SAC algorithm ultimately emerged as the preferred choice due to its consistent utility during testing. This is in line with observations of other, similar research [30]. To expedite experimentation and given the constraints, certain hyperparameters such as the number of timesteps (set to two million), learning rate (0.0001), and buffer size (1000) were chosen, reflecting a pragmatic approach rather than exhaustive fine-tuning effort.

Model Traning In the RL approach, computational efficiency in model training was aimed for. It deviated from the CNN-based approach by exclusively employing data from a single training map and restricting training to a single driving direction (due to high computational cost and requied time). While the streamlined training methodology may initially appear restrictive, its outcomes were satisfactory. The RL model not only demonstrated the capability to learn collision avoidance but also exhibited adaptability to different environments. This adaptability was substantiated through evaluation on an entirely unseen (testing) map. The model's ability to perform effectively on the test map underscored its capacity to generalize knowledge effectively, even when trained solely on a single map

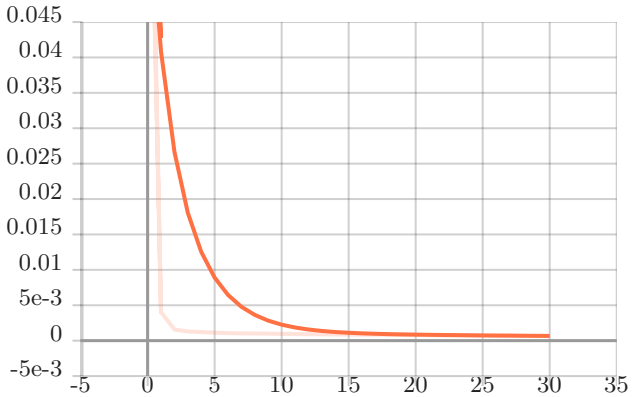
3 Results and discussion

3.1 CNN

The proposed CNN architecture was built, trained, and tested using Python programming language (v. 3.8.10), PyTorch (v. 1.13.0+cu117) and PyGame (v. 2.1.2), along with some additional PyTorch utilities (Dataset, DataLoader, SubsetRandomSampler and Torchvision), as well as other Python libraries, such as NumPy (v. 1.23.5) and Pandas (v. 1.5.2).

The training was conducted over 30 epochs using a batch size of 64, leveraging both training and validation data. The training time was three hours and five minutes. It's worth noting that the choice of 30 epochs was determined to be sufficient through repeated training sessions. This approach aimed to strike a

balance between training efficiency and model generalization. Throughout the training process, the model exhibited substantial progress in minimizing the MSE loss. As seen in Figure 7 the MSE loss consistently decreased during each epoch, indicating the model's ability to capture intricate patterns and relationships within the dataset.



(a) Training loss

(b) Validation loss

Fig. 7: Training and validation loss – Smoothed line is visible and unsmoothed is presented with a higher level of transparency

Batch size is a crucial NN training parameter that defines the number of training examples in each iteration. During training, data was grouped into batches, and model parameters were updated using the average gradient from batch loss values. Batch size impacts training speed, generalization, and resource use. Selecting an optimal size depends on the dataset, model complexity, and resources. Past research suggests smaller batch sizes may better avoid local min-

ima and uncover global minima [44–46]. Therefore, the model underwent training and testing with a range of batch sizes, spanning from an initial 16 and incrementing successively to 32, 64, 128, and up to 1024 [47]. Table 2 shows the result over each batch size. The results were taken from an epoch that gave the best validation loss. The number in the brackets represents the epoch at which the results were measured.

Table 2: Model MSE results in respect to used training batch size

Batch size (Epoch)	**Training Loss**	**Validation Loss**	**Testing Loss**
16 (16)	0.000604	0.000789	0.000568
32 (24)	0.000569	0.000799	0.000578
64 (27)	**0.000690**	**0.000798**	**0.000554**
128 (29)	0.000856	0.000879	0.000630
256 (27)	0.000926	0.000886	0.000623
512 (28)	0.001158	0.000970	0.000663
1024 (30)	0.001586	0.001130	0.000788

After evaluating various models trained with different batch sizes, the model utilizing a batch size of 64 stands out. Its strong test performance, along with solid training and validation results, make it a promising choice. These factors highlight its effectiveness for real-world applications. Comparing two different batch sizes, as seen in Figure 8, 64 and 1024, during the training process reveals a notable observation: the smaller batch size of 64 demonstrates a faster rate of convergence compared to the larger batch size of 1024.

When the selected model was tested on an unseen road, the model exhibited challenges in reacting to significant road curves. This raised questions about the impact of vehicle speed on the model's response accuracy. Therefore, an investigation to isolate the variable of vehicle speed was conducted. The vehicle's speed was held at a constant 8 km/h and examined the model's predictions under this controlled setting. However, an unexpected observation emerged over time: the model's inference time, i.e., the time it required to generate predictions from a single image, became apparent. By measuring how much time a model needs to output a prediction for a single image, it averaged 0.025 seconds for each prediction. This insight held significance as it enabled the determination of an optimal FPS value, indicating the rate at which the model could function seamlessly. Notably, before this discovery, the FPS was at 80. The inference time translated to an effective FPS of approximately 40, meaning the model could process up to 40 images per second. This understanding not only addressed the model's earlier struggles with road curves but also set the stage for a fluid and real-time performance. As a result, the simulation was restricted to 30 FPS to ensure both timely model reactions and the smooth operation of the simulation.

The vehicle followed the road (without collisions), whilst keeping the throttle and speed according to the curvature of the road. The bigger the curvature,

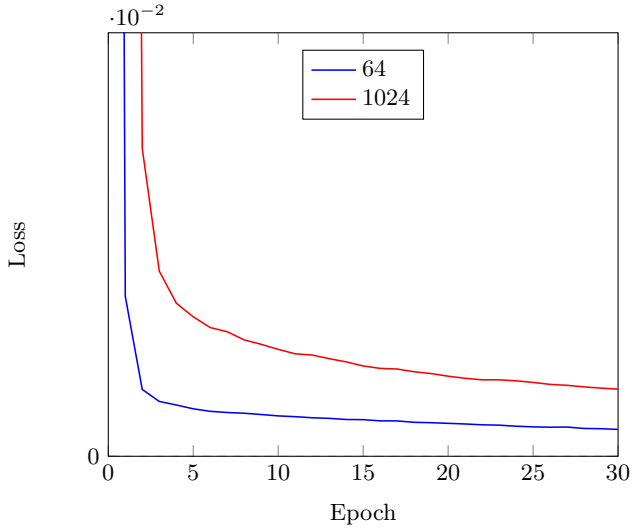


Fig. 8: CNN training MSE difference between two batch sizes – 64 and 1024

the lower the vehicle's speed, and the smaller the curvature, the bigger the speed. The model also recognized how to keep the vehicle in the center of the road, enabling a smooth ride. The maximum speed was set to 30 km/h, and the minimum speed depended on the curvature. On straight sections of the road, the model achieved a maximum speed of 30 km/h, while on curves, the minimum speed depended on the curvature, typically around 21 km/h. While a model's ability to control the vehicle and follow the road is crucial, achieving a smooth ride involves a higher level of finesse and attention to detail. It encompasses factors such as minimizing unnecessary steering corrections, avoiding sudden accelerations or decelerations, maintaining a consistent trajectory, and overall providing a comfortable experience for passengers or users.

This behavior is illustrated in Figure 9, where all three values of interest are adjusted according to the curvature of the road. When comparing the three plots, it becomes evident that as the steering angle increases, the throttle decreases, resulting in lower speed. Conversely, when the steering angle decreases, the throttle increases, leading to higher speed. A smooth ride showcases the model's proficiency in not just basic control but also in creating coherent, natural, and safe driving behavior. In the case of starting a vehicle from an unnatural position, or making a mistake during the predictions, the model was still able to correct the vehicle onto a correct trajectory.

3.2 RL

The RL model was built, trained and tested using Python programming language (v. 3.8.10), PyTorch (v. 1.13.0+cu117), Stable-Baselines3 (v. 2.0.0), Gymnasium

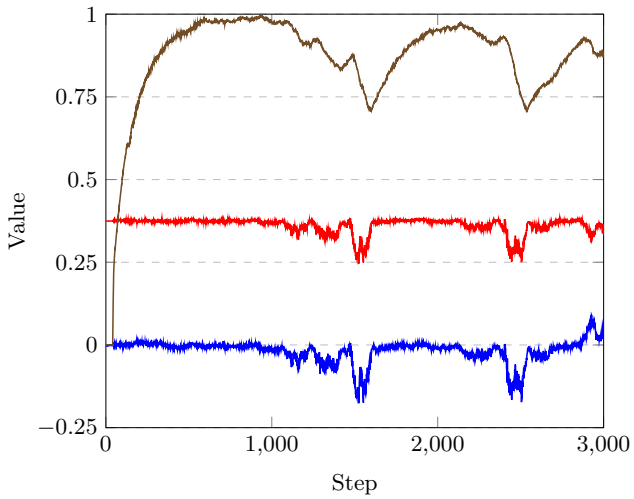


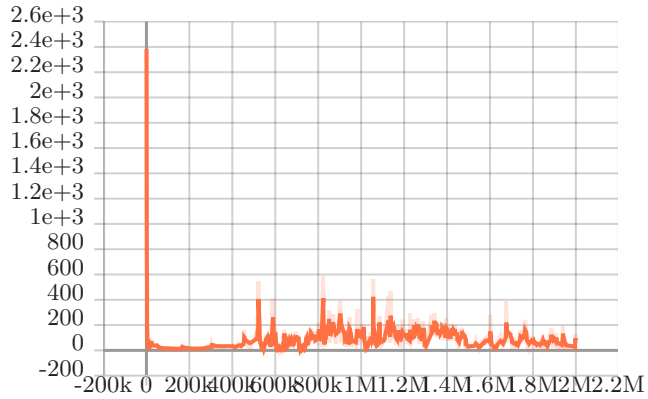
Fig. 9: Values of steering angle (blue), throttle (red), and speed (brown) during testing of the CNN model

(0.28.1), Baselines3 Zoo (2.0.0) and PyGame (v. 2.1.2), along with an additional PyTorch utility Torchvision (v. 0.15.1+cu118), as well as other Python libraries, such as NumPy (v. 1.23.5) and Pandas (v. 1.5.2).

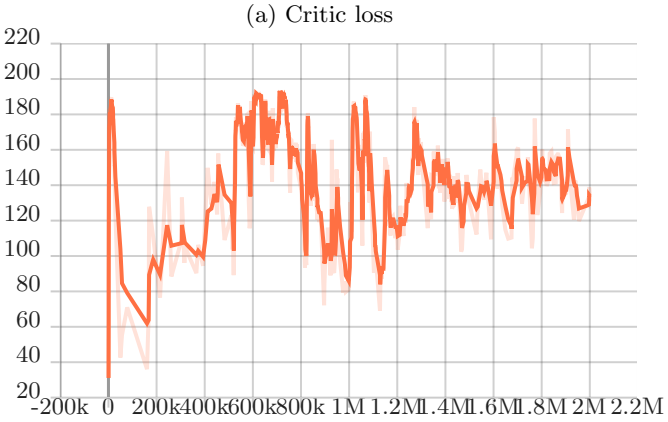
To leverage SAC's capabilities, an extensive training regimen was pursued. Utilizing two million timesteps, the model's learning trajectory was carefully shaped. This process spanned about 7 days (on used hardware), with each iteration enhancing the model's competence. Over these two million timesteps, a special callback class was implemented that saved the model every 100,000 timesteps.

After the training finished, the next step was to figure out which one of the models was the right one. In off-policy RL algorithms like SAC, the primary metric for assessing the quality of learning is often the critic loss. The critic loss represents how well the value function (critic network) is approximating the true value of states or state-action pairs. A lower critic loss indicates that the value function is improving its estimation accuracy, leading to more effective learning and decision-making by the agent. While the actor loss is also important as it reflects how well the policy (actor-network) is improving, the critic loss provides a more direct measure of the algorithm's ability to estimate the value of different states and actions.

Figure 10 depicts how the critic and actor losses range throughout the training process. Both critic and actor loss values are desirable in lower values. During training, the critic loss remains stable at around 50, reflecting progress in approximating state-action values. However, the actor loss, averaging around 150, suggests an ongoing refinement of the policy's action-selection strategy. This divergence between critic and actor losses could be due to task complexity, the



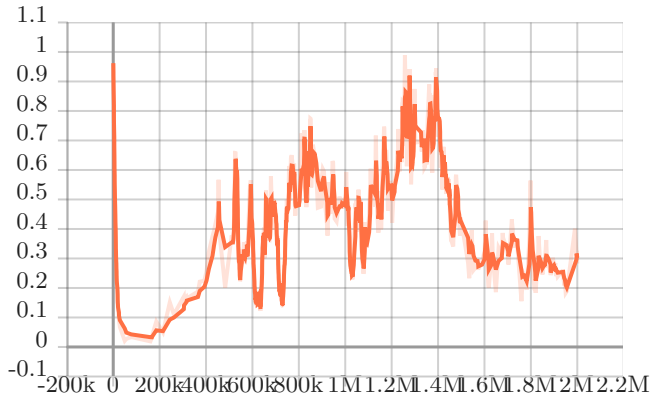
(a) Critic loss



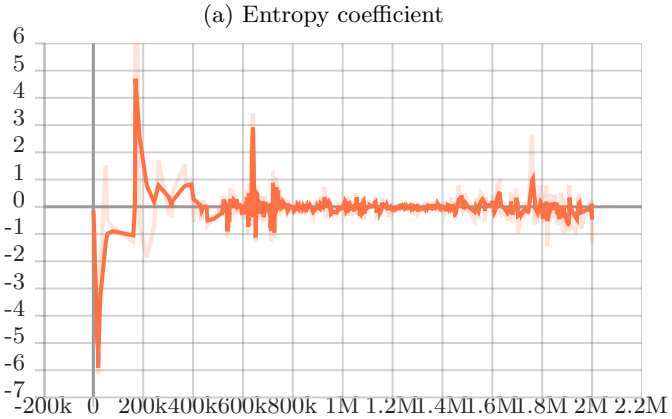
(b) Actor loss

Fig. 10: Critic and actor loss during RL training – Smoothed line is visible and unsmoothed is presented with a higher level of transparency

balance between exploration and exploitation, or the interaction between value estimation and policy improvement.



(a) Entropy coefficient



(b) Entropy coefficient loss

Fig. 11: Entropy coefficient and its loss during training – Smoothed line is visible and unsmoothed is presented with a higher level of transparency

Moreover, what is also interesting is the entropy coefficient and its corresponding loss. Figure 11 shows both of these values throughout the training process. Tuning the entropy coefficient helps balance exploration and exploitation during learning. A higher value promotes more exploration, while a lower value prioritizes exploiting the learned policy. The entropy coefficient loss is part of the optimization process in SAC. It is the loss associated with the entropy coefficient and is optimized alongside the actor and critic losses. Optimizing the entropy coefficient helps in dynamically adjusting the exploration level throughout training. The changing entropy coefficient during training is interesting. It

starts variable but becomes steadier by the end, indicating an improved balance between exploration and exploitation as the model learns. From this brief discussion, it can be seen that within the RL framework, several (hyper)parameters need to be chosen and adjusted (optimized). This in turn increases the complexity of developing RL-based algorithms compared to CNN-based ones.

Each of the saved models, from 100,000 to 2,000,000, was evaluated to obtain a mean reward and a standard deviation of the mean reward. A high mean reward coupled with a low standard deviation signifies that the agent is consistently achieving high rewards and maintaining stability in its behavior. Across all models, initial steps (100,000–500,000) exhibited lower speeds, often resulting in collisions with road barriers. Consequently, steering angle and throttle predictions during this phase tend to be inaccurate. Around step 600,000, the vehicle experiences a sudden acceleration to approximately 16 km/h, maximizing speed-based rewards. However, this acceleration introduces oscillations in steering angles, affecting ride smoothness. Between steps 700,000 and 1,300,000, the vehicle gradually reduces speed while stabilizing steering angles. From step 1,300,000 to 2,000,000, the vehicle maintains a slow pace but encounters road barrier collisions once more. Figure 12 shows how the mean reward and standard deviation of the reward move throughout timesteps, while Figure 13 depicts steering angle and throttle values for the selected/best step.

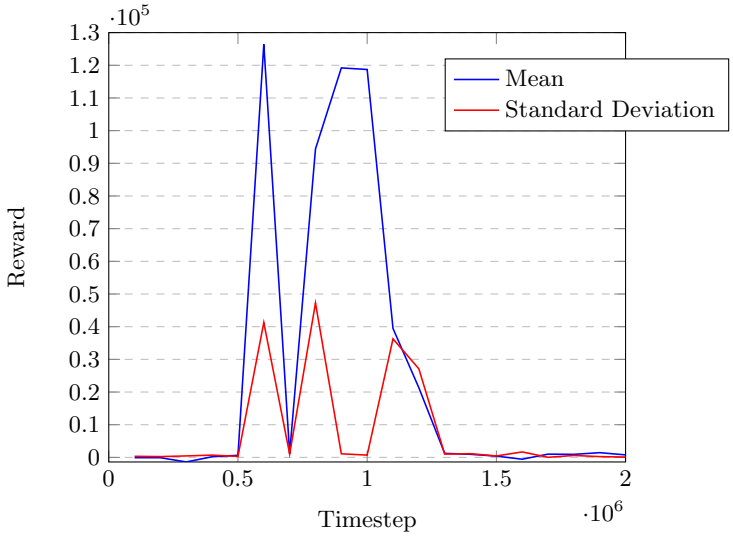


Fig. 12: Mean and standard deviation of rewards

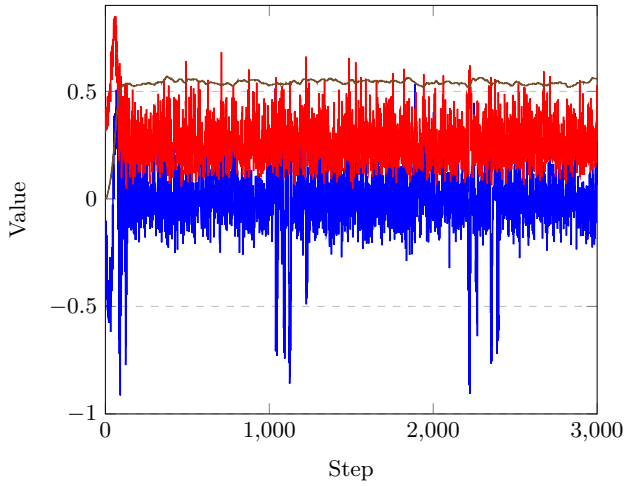


Fig. 13: Values of steering angle (blue), throttle (red), and speed (brown) during testing of the RL model

3.3 CNN vs. RL comparison

As it may be obvious, CNN and RL are two distinct approaches within the realm of artificial intelligence, each serving unique roles in addressing diverse problems. CNNs excel in extracting and understanding visual patterns from data, particularly images, while RL empowers agents to make informed decisions through interactions with their environments (through user-defined reward function). Several general insights can be obtained from the comparison of these two methodologies within this specific context.

Environment In the CNN approach, this environment setup serves as the foundation for data collection. The focus revolves around gathering pertinent data from the established environment. Conversely, the RL approach mandates an additional layer of complexity. In this case, the establishment of the environment marked only the initial phase. A subsequent step entailed the implementation of a specialized interface, extending the capabilities of the CARLA simulator environment. This augmentation was necessary to accommodate the unique requirements of the RL approach. Notably, the integration demanded the formulation of specialized callback functions. The inclusion of these functions underscores an augmented level of preparatory work requisite for the successful deployment of the RL methodology.

Data Within the CNN approach, the workflow entails processes of data acquisition, preprocessing, and subsequent partitioning into distinct subsets designated for training, validation, and testing purposes. This method necessitates meticulous manual intervention to scrutinize and manipulate the collected data, a critical step that significantly influences CNN's efficacy. Notably, the degree of refinement in data tailoring bears a direct correlation with the model's training

quality and ensuing performance. Conversely, the RL approach presents a more streamlined trajectory. It involves the seamless integration of the methodology within a predefined environment, characterized by carefully delineated action and observation spaces. This approach distinctly diverges from the CNN process, as it obviates the need for prior data collection. Instead, RL engages directly with the environment, iteratively interacting to learn optimal strategies. Important to note is that, RL deep in its implementation uses a CNN architecture to leverage the images used to learn. In this way, RL can extract important visual features from the images, and make a connection between the image and the actions needed to be applied for an accurate prediction.

Training and Loss The CNN training, primarily oriented towards predictive accuracy, employed the MSE loss metric. This metric illuminates the model's capability to forecast values within the given dataset. Additional insights are gained from variations such as the utilization of a validation set, which gauges the presence of overfitting, as well as test loss, which conveys how well a model deals with new unseen data. On contrary, central to RL's training process are metrics encompassing the critic and actor loss. These metrics gauge the convergence of the value estimation and policy networks, integral to RL's decision-making process. Additionally, the entropy coefficient and entropy coefficient loss measurements further accentuate RL's exploration-exploitation balance. The entropy coefficient regulates the level of exploration, influencing the policy's stochasticity. The corresponding loss metric evaluates the alignment of the policy's entropy with the desired exploration strategy. The temporal dimension further underscores the divergence.

CNN training exhibits a notable contrast in time expenditure, typically spanning hours. Empirical analysis, encompassing varied batch sizes, resulted in an average training duration of just over three hours. Conversely, the RL training horizon unfolds across multiple days, marked by considerable temporal investment. It's worth noting that variations exist within the RL domain, with algorithms like PPO showcasing notably reduced training periods, on the order of hours, typically around 12 hours. This temporal variation can be attributed to the inherent disparity in the learning dynamics. RL's iterative, trial-and-error-driven learning mechanism inherently demands longer exposure to the environment to discern optimal strategies. Conversely, CNN's supervised learning structure can achieve substantial progress within shorter timeframes due to the availability of labeled data (in simulation). In essence, we can say that CNN reduces computational complexity (in the form of required computational time for training) for the computer/machine by mitigating it to human operators or simulation (through data labeling and image processing). RL on the other hand frees the human operator or simulation of that burden by increasing computational time for the computer/machine in the form of environmental interaction.

Model Selection In the CNN paradigm, model selection revolved around an exhaustive exploration of varying batch sizes and their corresponding loss outcomes. By assessing the interplay between validation and testing losses, a judicious choice emerged—namely, a batch size of 64. Conversely, the RL method-

ology undertook a comprehensive model selection approach. This entailed evaluating the mean and standard deviation rewards across all timesteps, in addition to scrutinizing the vehicle's velocity and its propensity for collisions during evaluation. This holistic evaluation framework captures not only the model's reward attainment capabilities but also its proficiency in maintaining optimal speed and navigating the environment without collisions — a fundamental attribute for successful track completion.

Performance The CNN approach, although resilient, exhibited sensitivity to the simulation's FPS parameter, necessitating careful calibration. Notwithstanding this, its overall performance was commendable, marked by noteworthy attributes. In the CNN paradigm, the vehicle adeptly modulated its throttle and speed according to the road's curvature. It demonstrated a commendable aversion to collisions, showcased an ability to rectify its course in case of errors, and consistently maintained its position at the road's center. This amalgamation of actions facilitated a seamless journey characterized by stability and precision.

Contrastingly, the RL approach, while not achieving the full spectrum of accomplishments exhibited by CNN, garnered satisfactory results. It consistently adhered to a minimum speed threshold of 15 km/h, with minimal fluctuations hovering around 16 km/h, ensuring steadfast performance. Negotiating curves with finesse, it adeptly avoided collisions, reaffirming its competence. However, in comparison to CNN, a notable discrepancy emerged. The RL-driven vehicle displayed pronounced steering angle oscillations, manifesting as significant lateral deviations. This manifested in a somewhat erratic trajectory, characterized by continuous lateral shifts. Additionally, when considering real-world data collection in autonomous driving scenarios, RL is not easily implementable due to safety issues related to its trial-and-error approach.

4 Conclusion

Both the CNN and RL approaches need comprehensive optimizations. The forthcoming paths for enhancement hold distinct focal points, as articulated in the previous section. Within the CNN methodology, a crucial avenue for refinement involves paring down the number of trainable parameters embedded within the model's architecture (decreasing inference time and increasing possible simulation FPS). This strategic streamlining not only alleviates computational overhead but augments the model's adaptability to varying data distributions. Furthermore, architectural enhancements beckon, necessitating the exploration of configurations that harmonize with the task's intricacies. Hyperparameter optimization emerged as another pivotal domain, encompassing quintessential variables like learning rate, optimization algorithm, and batch size. Concurrently, considerations about data preprocessing—ranging from greyscale transformations to cropping strategies and image dimensions—imprint a substantial footprint on performance.

The terrain of RL optimization unfurls as a multidimensional landscape, heavily punctuated by hyperparameters. This emphasis stems from RL's in-

trinsic reliance on these tunable variables. Among the critical determinants are batch size, action noise, learning rate, and the count of sequential steps, forming a fraction of the parameters meriting meticulous calibration (SB3 has 22 parameters at its disposal to tune). In the same trajectory, the evolution of the reward function constitutes an indispensable facet. The future improvements beckon the construction of a more intricate and nuanced reward function, capably encapsulating the complex dynamics of the environment. Augmenting the sensor suite by introducing additional sensors to monitor the vehicle's relative position concerning the road's orientation emerges as a strategy to inculcate the vehicle with the skill to self-centered on the road.

Additionally, the "Optuna" hyperparameter optimization framework [48] emerges as a potent resource. This tool holds the potential to systematically guide the intricate process of parameter optimization, significantly expediting the exploration of optimal configurations. Finally, extensions of RL, like curiosity-driven exploration [49] could be explored, and their performance analyzed and compared to the standard approach.

References

1. D. Parekh, N. Poddar, A. Rajpurkar, M. Chahal, N. Kumar, G.P. Joshi, W. Cho, "A Review on Autonomous Vehicles: Progress, Methods and Challenges," Electronics, Vol. 11, No. 14: 2162, 2022.
2. J. Wang, Y. Shao, Y. Ge, R. Yu, "A Survey of Vehicle to Everything (V2X) Testing," Sensors, Vol. 19, No. 2:334, 2019.
3. W. Brenner, A. Herrmann, "An Overview of Technology, Benefits and Impact of Automated and Autonomous Driving on the Automotive Industry,", In: C. Linnhoff-Popien, R. Schneider, M. Zaddach (eds) Digital Marketplaces Unleashed, Springer, Berlin, Heidelberg, 2018.
4. SAE J3016:2021 standard, "Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems", The Society of Automotive Engineers, 2021. https://www.sae.org/standards/content/j3016_202104/
5. D. Croutch, "The State of Self-Driving Cars: Autonomous Advances," https://www.techspot.com/article/2644-the-state-of-self-driving-cars/, Accessed: 4 June, 2023.
6. R. Baldwin, "Self-Driving Cars Are Taking Longer to Build Than Everyone Thought," Car and Driver, https://www.caranddriver.com/features/a32266303/self-driving-cars-are-taking-longer-to-build-than-everyone-thought/, 2020., Accessed: January 15, 2023.
7. L. Clarke, "How Self-Driving Cars got Stuck in the Slow Lane," The Guardian, https://www.theguardian.com/technology/2022/mar/27/how-self-driving-cars-got-stuck-in-the-slow-lane, 2022., Accessed: January 15, 2023.
8. D. J. Yeong, G. Velasco-Hernandez, J. Barry, J. Walsh, "Sensor and Sensor Fusion Technology in Autonomous Vehicles: A Review," Sensors, Vol. 21, No.6:2140, 2021.
9. I. U. Hassan, H. Zia, H. S. Fatima, S. A. Yusuf, M. Khurram, "A Lightweight Convolutional Neural Network to Predict Steering Angle for Autonomous Driving Using CARLA Simulator, " Modelling and Simulation in Engineering, Vol. 2022:5716820, 2022.

10. J. G. Song, J. W. Lee, "CNN-Based Object Detection and Distance Prediction for Autonomous Driving Using Stereo Images," International Journal of Automotive Technology, Vol. 24, pp. 773—786, 2023.
11. A. Al Mamun, E. P. Ping, J. Hossen, A. Tahabilder, B. Jahan, "A Comprehensive Review on Lane Marking Detection Using Deep Neural Networks," Sensors, Vol. 22, No. 19:7682, 2022.
12. N. Youssouf, "Traffic sign classification using CNN and detection using faster-RCNN and YOLOV4," Heliyon, Vol. 8, No. 12, 2022.
13. A. Mehrish, N. Majumder, R. Bharadwaj, R. Mihalcea, S. Poria, "A review of deep learning techniques for speech processing," Information Fusion, Vol. 99:101869, 2023.
14. B. Alshemali, J. Kalita, "Improving the Reliability of Deep Neural Networks in NLP: A Review," Knowledge-Based Systems, Vol. 191:105210, 2020.
15. H. Yu, L. T. Yang, Q. Zhang, D. Armstrong, M. J. Deen, "Convolutional neural networks for medical image analysis: State-of-the-art, comparisons, improvement and perspectives," Neurocomputing, Vol. 444, pp. 92–110, 2021.
16. R. S. Sutton, A. G. Barto, "Reinforcement Learning: An Introduction," 2nd edition, The MIT Press, 2018.
17. Ó. Pérez-Gil, R. Barea, E. López-Guillén, L. M. Bergasa, C. Gómez-Huélamo, R. Gutiérrez, A. Díaz-Díaz, "Deep reinforcement learning based control for Autonomous Vehicles in CARLA," Multimedia Tools and Applications, Vol. 81, pp. 3553—3576, 2022.
18. B. Osiński, A. Jakubowski, P. Ziecina, P. Miloš, C. Galias, S. Homoceanu, H. Michalewski, "Simulation-Based Reinforcement Learning for Real-World Autonomous Driving," In: Proceedings of 2020 IEEE International Conference on Robotics and Automation (ICRA), pp. 6411–6418, 2020.
19. H. P. Schöner, "Simulation in development and testing of autonomous vehicles, " In Proceedings of 18. Internationales Stuttgarter Symposium, pp. 1083–1095, 2018.
20. A. Dosovitskiy, G. Ros, F. Codevilla, A. López, V. Koltun, "CARLA: An Open Urban Driving Simulator," Proceedings of the 1st Annual Conference on Robot Learning, pp. 1–16, 2017.
21. W. Li, C. W. Pan, R. Zhang, J. P. Ren, Y. X. Ma, J. Fang, F. L. Yan, Q. C. Geng, X. Y. Huang, H. J. Gong, W. W. Xu, G. P. Wang, D. Manocha, R. G. Yang, "AADS: Augmented autonomous driving simulation using data-driven algorithms", Science Robotics, 4, eaaw0863, 2019.
22. G. Rong, B. H. Shin, H. Tabatabaee, Q. Lu, S. Lemke, M. Možeiko, E. Boise, G. Uhm, M. Gerow, S. Mehta, E. Agafonov, T. H. Kim, E. Sterner, K. Ushiroda, M. Reyes, D. Zelenkovsky, S. Kim, "LGSVL Simulator: A High Fidelity Simulator for Autonomous Driving," In: Proceedings of 2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC), pp. 1–6, 2020.
23. S. Shah, D. Dey, C. Lovett, A. Kapoor, "AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles". In: Hutter, M., Siegwart, R. (eds) Field and Service Robotics. Springer Proceedings in Advanced Robotics, Vol. 5, 2018.
24. G. Chance, A. Ghobrial, K. McAreavey, S. Lemaignan, T. Pipe and K. Eder, "On Determinism of Game Engines Used for Simulation-Based Autonomous Vehicle Verification," IEEE Transactions on Intelligent Transportation Systems, Vol. 23, No. 11, pp. 20538–20552, 2022.
25. Q. Chao, H. Bi, W. Li, T. Mao, Z. Wang, M. C. Lin, Z. Deng, "A Survey on Visual Traffic Simulation: Models, Evaluations, and Applications in Autonomous Driving, " Computer Graphics Forum, Vol. 39, No. 1, pp. 287–308, 2020.

26. H. Alghodhaifi, S. Lakshmanan, "Autonomous Vehicle Evaluation: A Comprehensive Survey on Modeling and Simulation Approaches," IEEE Access, Vol. 9, pp. 151531–151566, 2021.
27. E. Yurtsever, J. Lambert, A. Carballo, Takeda, "A Survey of Autonomous Driving: Common Practices and Emerging Technologies," IEEE Access, Vol. 8, pp. 58443–58469, 2020.
28. J. Zhang, H. Huang, Y. Zhang, "A Convolutional Neural Network Method for Self-Driving Cars," In: Proceedings of 2020 Australian and New Zealand Control Conference (ANZCC), pp. 184–187, 2020.
29. M. Bojarski, C. Chen, J. Daw, A. Degirmenci, J. Deri, B. Firner, B. Flepp, S. Gogri, J. Hong, L. Jackel, Z, Jiam B.J. Lee, B. Liu, F. Liu, U. Muller, S. Payne N. Kota Nagendra Prasad, A. Provodin, J. Roach, T. Rvachov, N. Tadimeti, J. van Engelen, H. Wen, E. Yang, Z. Yang, "The NVIDIA PilotNet Experiments, " arXiv:2010.08776, 2020.
30. J. Chen, B. Yuan, M. Tomizuka, "Model-free Deep Reinforcement Learning for Urban Autonomous Driving," In: Proceedings of 2019 IEEE Intelligent Transportation Systems Conference (ITSC), pp. 2765–2771, 2019.
31. C. C. Aggarwal, "Neural Networks and Deep Learning: A Textbook, ", Springer Cham, 2018.
32. Baeldung, "Advantages and disadvantages of Neural Networks," Baeldung on Computer Science, https://www.baeldung.com/cs/neural-net-advantages-disadvantages, Accessed: 19 April, 2023.
33. M. A. Pratama, "Comparing Image Classification with Dense Neural Network and Convolutional Neural Network, ", Medium, https://medium.com/analytics-vidhya/comparing-image-classification-with-dense-neural-network-and-convolutional-neural-network-5f376582a695, Accessed: 25 Januray, 2024.
34. P. Antoniadis, "What is end-to-end deep learning?," Baeldung on Computer Science, https://www.baeldung.com/cs/end-to-end-deep-learning, Accessed: 8 September 2023.
35. H. Zhang, T. Yu, "Taxonomy of Reinforcement Learning Algorithms, " In: Dong, H., Ding, Z., Zhang, S. (eds) Deep Reinforcement Learning, Springer, 2020.
36. F. Muratore, F. Ramos, G. Turk, W. Yu, M. Gienger, J. Peters, "Robot Learning From Randomized Simulations: A Review, " Frontiers in Robotics and AI, Vol. 11, No.9:799893, 2022.
37. MatWorks, "RoadRunner - Design 3D scenes for automated driving simulation," https://www.mathworks.com/products/roadrunner.html, Accessed: 18 July 2023.
38. S. Ioffe, C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, " arXiv: 1502.03167, 2015.
39. C. Garbin, X. Zhu, O. Marques, "Dropout vs. batch normalization: an empirical study of their impact to deep learning, " Multimedia Tools and Applications, Vol. 79, pp. 12777—12815, 2020.
40. J. Willaert, "How To Calculate the Mean and Standard Deviation — Normalizing Datasets in Pytorch," Towards Data Science, https://towardsdatascience.com/how-to-calculate-the-mean-and-standard-deviation-normalizing-datasets-in-pytorch-704bd7d05f4c, Accessed: 18 July 2023.
41. H. M. Le, "Car behavioral cloning using Pytorch, " Github, https://github.com/hminle/car-behavioral-cloning-withpytorch/, Accessed: 15 March 2023.
42. A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, N. Dormann, "Stable-Baselines3: Reliable Reinforcement Learning Implementations," Journal of Machine Learning Research, Vol. 22, No. 268, pp.1–8, 2021.

43. G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, W. Zaremba, "OpenAI Gym, " arXiv:1606.01540, 2016.
44. N. S. Keksar, D. Mudigere, J. Nocedal, M. Smelyanskiy, P. T. P. Tang, "On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima,", arXiv:1609.04836, 2016.
45. L. Dinh, R. Pascanu, S. Bengio, Y. Bengio, "Sharp Minima Can Generalize For Deep Nets, " In: Proceedings of the 34th International Conference on Machine Learning, 2017.
46. I. Kandel, M. Castelli, "The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset, " ICT Express, Vol. 6, No. 4, pp. 312–315, 2020.
47. I. Vasiljević, J. Musić, J. Mendes, J. Lima, "Adaptive Convolutional Neural Network for Predicting Steering Angle and Acceleration on Autonomous Driving Scenario, " Optimization, Learning Algorithms and Applications, Third International Conference, OL2A 2023. 2023.
48. T. Akiba, S. Sano, T. Yanase, T. Ohta, M. Koyama, "Optuna: A Next-generation Hyperparameter Optimization Framework, " In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2019.
49. D. Pathak, P. Agrawal, A. A. Efros, T. Darrell, "Curiosity-driven Exploration by Self-supervised Prediction, " In: Proceedings of the 34th International Conference on Machine Learning, 2017.