AI

MDPI

Article

Optimizing Steering Angle Prediction in Self-Driving Vehicles Using Evolutionary Convolutional Neural Networks

Bashar Khawaldeh [1,*], Antonio M. Mora [1] and Hossam Faris [2]

[1] Department of Signal Theory, Telematics and Communications, University of Granada, ETSIIT and CITIC-UGR, C. Periodista Daniel Saucedo Aranda, 18014 Granada, Spain; amorag@ugr.es
[2] Information Technology Department, King Abdullah II School for Information Technology, The University of Jordan, Queen Rania St., Amman 11942, Jordan; hossam.faris@ju.edu.jo
* Correspondence: prof.khawaldeh@gmail.com or khawaldeh@correo.ugr.es

Abstract: The global community is awaiting the advent of a self-driving vehicle that is safe, reliable, and capable of navigating a diverse range of road conditions and terrains. This requires a lot of research, study, and optimization. Thus, this work focused on implementing, training, and optimizing a convolutional neural network (CNN) model, aiming to predict the steering angle during driving (one of the main issues). The considered dataset comprises images collected inside a car-driving simulator and further processed for augmentation and removal of unimportant details. In addition, an innovative data-balancing process was previously performed. A CNN model was trained with the dataset, conducting a comparison between several different standard optimizers. Moreover, evolutionary optimization was applied to optimize the model's weights as well as the optimizers themselves. Several experiments were performed considering different approaches of genetic algorithms (GAs) along with other optimizers from the state of the art. The obtained results demonstrate that the GA is an effective optimization tool for this problem.

Keywords: autonomous driving; artificial intelligence; convolutional neural network; evolutionary computing; genetic algorithm; machine learning; deep learning

Citation: Khawaldeh, B.; Mora, A.M.; Faris, H. Optimizing Steering Angle Prediction in Self-Driving Vehicles Using Evolutionary Convolutional Neural Networks. *AI* **2024**, *5*, 2147–2169. https://doi.org/10.3390/ai5040105

Academic Editors: Jeng-Shyang Pan, Junzo Watada, Vaclav Snasel and Pei Hu

Received: 13 September 2024
Revised: 21 October 2024
Accepted: 24 October 2024
Published: 30 October 2024

1. Introduction

Driving is a daily activity in which a person faces many challenges and situations, including environmental and road conditions, other drivers' behavior, and road signs. Thus, self-driving is a challenging, commonplace topic nowadays and a subject of research with many variables. The world, especially through the use of artificial intelligence (AI) techniques, is moving toward a rapid revolution in the development of robotic and self-driving systems [1].

A self-driving system must possess the capacity to adhere to a predefined route and to comply with established traffic regulations. Furthermore, it must be capable of navigating the surrounding traffic in a manner that is both safe for and considerate to other road users in addition to adhering to the established regulations. Roads are not normally straight; there can be curves and meanders along the way. Moreover, there could be different situations: a single-lane road in one direction, a two-lane road in two different directions, or a multi-lane road in two different directions, for instance.

Based on the literature [2–4], analysis, and practical applications, it is evident that controlling a vehicle's steering angle is crucial for ensuring its safety and performance. The ability to control a vehicle's direction significantly impacts its trajectory, stability, and responsiveness, whether navigating sharp turns on winding roads or facing challenges in tight urban environments. The primary factor to consider, regardless of the context, is the steering angle, which determines the vehicle's movement.

The vehicle is upright when it aligns completely with the axis of movement. Moving the steering wheel causes the vehicle to deviate either right or left by applying force to the

appropriate wheels, resulting in a change in the angle between the two axles. The car is directed in the desired direction based on this angle.

Choosing the optimal steering angle is a major challenge for the driver due to the importance of these angles as well as the challenges the driver may face in making appropriate decisions. Therefore, the focus of this work was to develop methods for accurately predicting the values of these angles and guiding the driver accordingly.

Developing a self-driving system requires a lot of effort, money, and risk. Therefore, the best approach is to start working and researching in a simulator and then transfer that experience to the real world. Several simulators were considered and evaluated, and finally, the Udacity simulator [5] was chosen for its 'simplicity', its precision, and the fact that it allows the collection of datasets while 'online' (i.e., during simulation). These can either be sets of images or raw information from the car, such as the steering angle, speed, camera position, etc. These values can be used to label data.

There have been many studies on the development of self-driving systems. Most of them have applied artificial neural networks (ANNs) to build and train a model [2,3,6–8]. In this study, we also considered an ANN approach, namely, a deep learning variation called a convolutional neural network (CNN) [9] in other domains; it can be also utilized in this problem, since we considered images for defining the dataset to be used.

In addition, a type of evolutionary algorithm (EA) [10,11] was applied to optimize the CNN configuration and performance. EAs are metaheuristics derived from Darwin's theory, which are based on the use of a population of potential solutions to a problem that are considered as individuals. These are each assigned a fitness value (depending on their quality), and then the best is normally (not always) combined with other solutions aiming to obtain better individuals in the next iteration (called a generation). EAs are very effective for solving optimization problems [12] and can also be used to solve complex problems [10]. These algorithms have shown remarkable effectiveness in tuning weights and probabilistic parameters, for instance, in gaming environments [13]. Thus, it would be interesting to explore their potential use in other contexts, such as improving CNN models in the autonomous driving context. Specifically, in this paper, the genetic algorithm (GA) [14] was applied to optimize the CNN models since it is one of the most widely used algorithms in the literature [15]. In addition, it is a pretty simple approach which has been considered in this preliminary study, making it easier (than using more complex algorithms) to analyze its potential impact or influence in the CNN's performance.

This combination (CNN + GA) was used to predict the optimal steering angle value of a self-driving car. Thus, this study includes the application of several different CNN approaches to predict the aforementioned angle for a self-driving system. The objective is to improve the performance and accuracy of the deep learning models by optimizing the weights they depend on.

Different operators and configurations of evolutionary algorithms were tested in order to find the best set of parameters to optimize the weights of the CNN models.

Moreover, several experiments were conducted, applying the GA to different variations of CNNs, in addition to other approaches using several state-of-the-art optimizers to improve the model performance. Then, all the optimized models were compared to see the impact of the optimization (GA) and to select the best overall model.

Given that a CNN is an algorithm to be applied on images, we composed a dataset using an equipped vehicle that recorded video while driving and stored it as images divided into samples. Each sample consisted of three images taken from different camera positions (right, center, and left) and labeled with the steering angle. This was conducted inside the Udacity simulator, which allows the manipulation of steering angle parameters. It also permits placing three cameras in the desired positions on the car, and then it records a video while the vehicle is moving on the simulated road. A series of images is captured within each frame, accompanied by supplementary data pertaining to the vehicle, including the steering angle.

This dataset, which is part of the contribution of this work, has been lately refined; we applied both image processing methods (to extract the most relevant information from each image), and data balancing [16] (namely, an undersampling approach). The dataset has been improved through the inclusion of thousands of additional labeled images aiming to cover many more driving situations in the training circuit than other existing approaches in this domain.

The rest of this paper is structured as follows: The following section presents the *state of the art* of some previous related studies and their results. The third section, *Materials and Methods*, describes the methodology used to develop our proposed autonomous driving model to predict the steering angle in a driving system. So, it describes the dataset extraction and preprocessing, the CNN model structure, and the proposed optimization methodology. Next, the *Experiments, Results, and Discussion* section presents the experiments conducted and the obtained results, as well as a deep analysis and comparison of the different approaches tested. Finally, the last section comments on the conclusions reached and includes some insights for future work.

2. State of the Art

The need for operational autonomous vehicles has motivated growing research to find solutions and algorithms that optimize various aspects of self-driving. The ultimate goal is to develop a fully integrated autonomous driving system.

Most of the studies have been conducted using vehicle driving simulators. Thus, there is research that has developed domain randomization to transfer from simulation to the real world based on deep neural networks [17], but in 2020, a group of researchers tried to train this type of neural network in CARLA (Car Learning to Act), an open-source simulator for autonomous driving research [18,19]. It did not work well when trained individually; it could not be transferred to the real world, so viewpoint augmentation (a popular preprocessing trick to improve generalization accuracy) was applied to it. It was called DR-AUG (Domain Randomization–Augmentation) [20]. In the same line, a research technique was also used to train an autonomous driving model on a simulator without using labels (objects with information) from the real world [21].

There are many simulators, including VISTA (Virtual Image Synthesis and Transformation for Autonomy), which is a data-driven simulator for end-to-end autonomous driving training [20]. However, maybe the most used simulator is Udacity [5], in which several different approaches have been tested, mainly convolutional neural network (CNN) models for autonomous driving [2,3,6,7].

The CNN training is performed on data, which could be a set of images collected while driving on different types of roads, residential streets (paved or unpaved), or highways, in different weather conditions. In one of the previous works [22], they collected data over 72 hours of driving in many different conditions in the real world and captured by one camera mounted on the center of the windshield of the car (later complemented with some 'transformed' images). These images were then labeled with some extra information about the car status. The CNN model is able to learn efficiently to drive in many different situations, but it seems to be not very robust (not completely reliable in all the situations).

Our study follows a similar methodology, but that dataset was not published. Moreover, since we used the Udacity simulator to collect the images, the precision of our samples is likely higher regarding the rest of the information about the car (the steering wheel, in our case). Moreover, we tested the CNN model with different optimization methods, aiming to obtain a definitive (and reliable) approach. Finally, study [22] used very powerful (and expensive) computational resources, while our approach was run on a much humbler setup.

Other studies have used data from simulation and translated them to the real world [21]. The authors also applied deep learning approaches based on images, obtaining a feasible solution (as a proof of concept).

In the work [23], the authors created a 'simple' (smartphone-size) physical model of a car and applied different realistic scenarios where it should drive autonomously. They also used supervised training with a CNN as well as an implementation of deep Q-learning to train the car on recognizing some road signs.

A CNN was also used for object detection and lane tracking for autonomous driving using a real dataset collected in the Rancho Palos Verdes area around San Pedro [24].

Another proposal [4] used CNN, Long Short-Term Memory (LSTM), and Fully Connected (FC) layers to control the steering angle, also considering data gathered from Udacity, using two vehicles capturing images from different perspectives and combining them. In the present work, the Udacity simulator was also used.

Following other proposals, in this paper, a CNN model for autonomous driving was trained using image data samples. However, in our case, the samples were labeled with an accurate steering angle. Each sample consists of three images collected from three camera positions (right, center, and left) by driving the car in the Udacity simulator, recording a video, and then saving the data as images.

Furthermore, our study was conducted using a genetic algorithm to select the best learning scheme, which includes preprocessing, attribute selection, and learning algorithms. The data were analyzed, and a set of potential solutions was generated to evaluate their performance and effectiveness in achieving the identified learning objectives [25].

This research aims to develop convolutional neural network (CNN) models and justify the weights used in them to achieve maximum accuracy in predicting steering angles in autonomous driving systems. Although many studies have preceded us in using artificial neural networks to build and train models [2,3,6,7], current research has not only been limited to creating an effective predictive model for steering angles but also focused on combining a convolutional neural network (CNN) and genetic algorithms (GAs) to improve model performance in steering prediction. In addition, efforts have been made to develop and train a CNN model using our own data extracted from the Udacity simulator.

The present study represents an advancement over previous research in the domain of autonomous driving utilizing a convolutional neural network (CNN) in two main aspects: the generation of a more reliable and complete dataset and the application of evolutionary optimization techniques over the CNN models.

While numerous previous studies have concentrated on the utilization of simulation data or genuine data for the training of autonomous driving models, our approach has firstly focused on the generation of a much more complete (and accurately labeled) image-based dataset. Using the Udacity simulator tool to gather images from the car's perspective, we conducted a process of systematic experimentation and expert revision for the generation of thousands of images representing a wide variety of driving situations. Thus, this led us to complete a dataset from which the models can be better trained, being then more reliable and effective in the prediction of steering angles.

Moreover, our study is distinguished by its integration of genetic algorithms with CNNs, with the objective of improving their performance. While numerous previous works have primarily focused on the creation of 'pure' neural network models using standard optimizers, our research aims to provide a comprehensive comparison of the performance of the CNN model using a wide variety of optimizers, such as Nadam, Adam, RMSprop, Adamax, SGD, Adagrad, Adadelta, and FTRL, with the CNN + GA model (using a GA as the optimizer). In addition, the application of GAs as an enhancing tool to improve the performance of the best optimizers was also studied in this work.

3. Materials and Methods

This section introduces the considered simulator and describes the collected dataset (with the applied preprocessing and enhancing methods) as well as the deep learning model that was applied.

3.1. Udacity Simulator

The Udacity simulator [5] is an open-source self-driving car framework developed with the Unity Engine. It is available at https://github.com/udacity/self-driving-car-sim (accessed on 5 June 2023).

Udacity is designed for students and researchers, since it offers several utilities to train self-driving car models, mainly using deep learning techniques, which can be integrated in the framework. Among its tools, the possibility of recording the driving session is offered in order to create your own dataset to be later processed.

Moreover, it is not very demanding with regard to computing resources, so it does not require a high-level machine. Finally, Udacity is quite easy to use, and it also achieves the desired goals of precision and accuracy, since the simulation is very realistic in several aspects.

Figure 1 shows a screenshot of the Udacity simulator, where its realistic graphics appearance can be seen.

Figure 1. Udacity simulator.

It allows video to be captured at 24 frames per second (fps) from three cameras (right, center, and left) and later stores it as a set of JPG images. It also allows storing a text file containing additional information, such as image title, steering angle, throttle, reverse, and speed, among others.

This simulator provides two driving tracks (shown in Figure 2, right side): the Lake Track spans approximately 1.1 km of a quite 'simple' profile. However, it contains standard curves in both directions, which will be very useful for the controller to 'learn' driving. The other circuit is named the Jungle Track, with a length of 1.5 km and a much more complex profile (see Figure 2, left side) with a lot of curves.

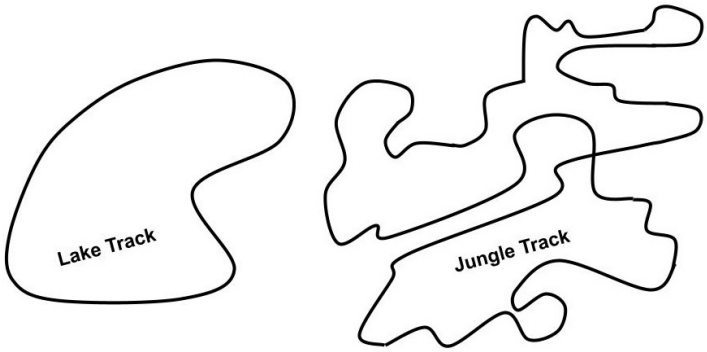

Figure 2. Udacity's Lake and Jungle Tracks.

The Lake Track was chosen as the training circuit for the model because of its diverse profile, which provides varied driving data for recording and training. Additionally, the track offers a simpler circuit for initial model training.

The Jungle Track was considered for verification and evaluation purposes. This is due to the circuit's complexity and challenges. This approach ensures a balance between diversity and result validation, ultimately improving the final model's quality and generalizability to various conditions and regions and, furthermore, ensuring the model's adaptability to real-world conditions and validating the results.

3.2. Dataset and Processing

Data for self-driving systems are normally very specific for a proposed model or even private [22]. For this reason, we decided to compose our own dataset. Following other approaches, we aimed to use an image-based dataset recorded from the car perspective. However, instead of just considering one frontal image, we placed three different cameras (with three different angles) on board the simulated car. This added a richer diversity of 'views' to the dataset. Figure 3 shows an example image from each camera angle.

Figure 3. Example of images of three camera positions.

The dataset was collected while a human expert controlled the car driving in the Udacity simulator. A 25 min recording session on the Lake Track was conducted utilizing three vehicle-mounted cameras located on the left, center, and right sides. The cameras captured an assortment of driving scenarios, such as varied curve entry speeds and steering angles. The recording was captured at 24 frames per second, and the average vehicle speed during the recording was 7.2 mph.

Thus, each sample in the dataset is composed of three images and the associated information at that specific moment in the simulator, such as the steering angle or the current speed [26].

Once the dataset was created, we conducted a preprocessing phase. Then, an undersampling process [27] was applied: firstly, identical images with the same steering angle were removed, since they do not add relevant information for the training of the self-driving model. Many of these images were stored in the recordings, since 24 frames are saved every second. Secondly, there was a large amount of samples with a steering angle equal to 0 (which means 'do not take any action or move straight ahead'), so these

samples were removed. After undersampling, the data distribution became more logical, as illustrated in Figure 4, which shows the distribution of the original dataset.

There are several justifications for the exclusion of samples with a steering angle of 0 (i.e., straight-line driving) from prediction models, with regard to both methodology and theory. Primarily, the exclusion of a considerable number of straight-line driving samples allows for the creation of a more balanced dataset, which is essential for the development of an accurate model. The incorporation of a substantial number of straight-line samples may result in the development of a model that is excessively simplistic, which would subsequently diminish its accuracy in anticipating curves or complex scenarios. Secondly, focusing on disparate steering angles enhances the model's predictive capacity, as it responds to data that necessitate a more intricate examination. Thirdly, this methodology minimizes noise and redundancy within the data, thereby enhancing the model's efficacy in processing diverse scenarios and optimizing the quality of the predictions. This, in turn, renders the model more balanced and effective in handling disparate driving scenarios.

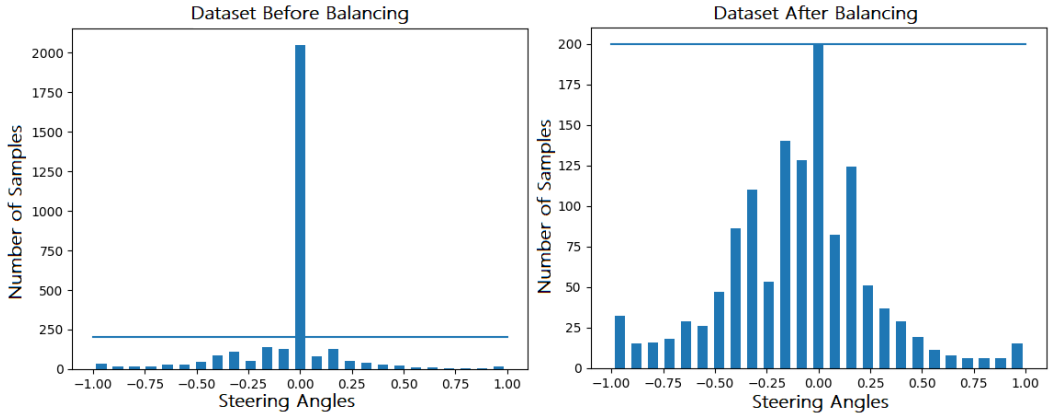

Figure 4. Distribution of dataset before and after data undersampling.

At the end of this process, there were 3882 images left, each of which was taken from different positions (left, right, or center). In order to use the data to train the controllers, the original dataset was divided into 80% of samples for training and 20% for validation. These samples contain a wide variety of steering angles, as illustrated in Figure 5.

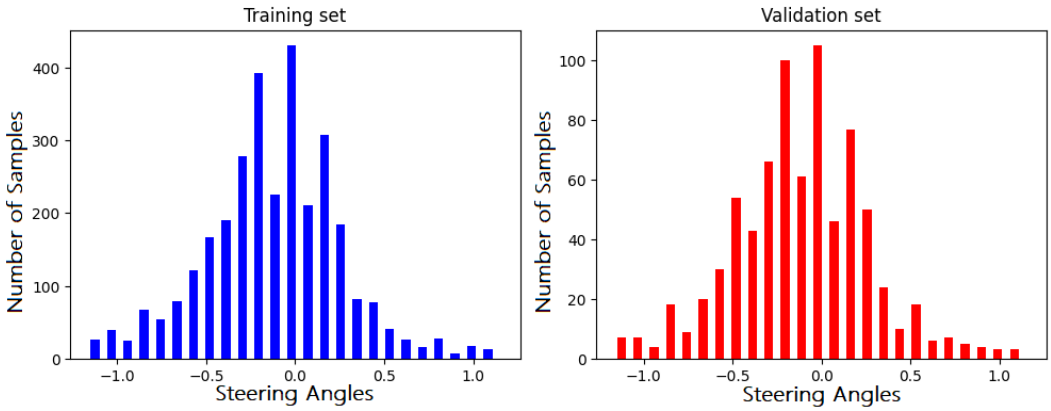

Figure 5. Original training and validation dataset distribution.

However, the original dataset was insufficient for training a model capable of handling all test conditions effectively. As a result, additional data were carefully added (by an expert) after thoroughly assessing the dataset's limitations, yielding a novel dataset with a total duration of approximately 61 min. Thus, it included 35,854 new images covering a much wider number of driving situations (steering angles with low representation in the

original dataset). The data recording session was conducted on the Lake Track with the use of three cameras positioned on the vehicle's left, middle, and right sides, respectively. The cameras documented a variety of driving scenarios, including different corner entry speeds and steering angles, with the recording captured at 24 frames per second. Following again the undersampling process, a total of 11,658 usable images were obtained. Furthermore, the novel dataset was divided into 80% of samples for training and 20% for validation. These samples contain a wide variety of steering angles, as illustrated in Figure 6.

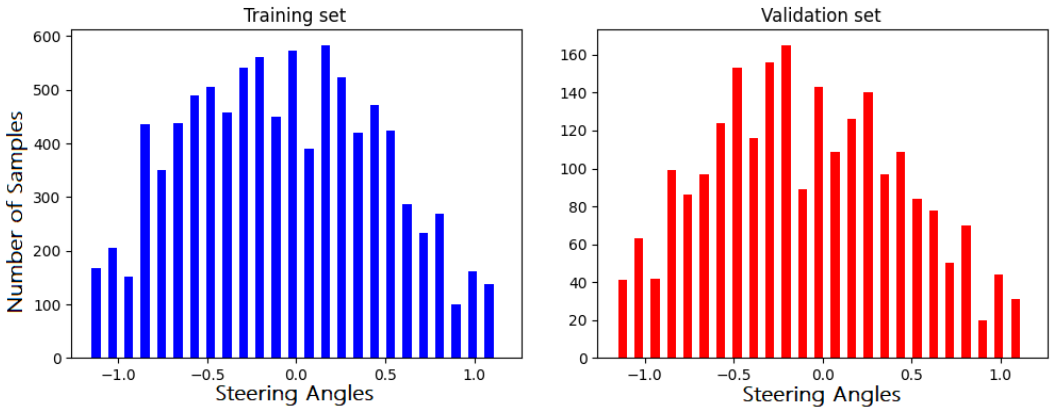

Figure 6. Novel training and validation dataset distribution.

Later, the dataset was augmented, aiming to create new images with new properties by changing some of the existing similar ones. This process helps to overcome underfitting, which occurs when a model is too simple to capture the underlying patterns in the data, especially when there is insufficient training data. This was performed using image augmentation functions (`imgaug` and `cv2` library) with randomly selected images from the dataset. Refer to the examples depicted in Figure 7. Namely, the applied processes were as follows:

- Zooming in: enlarging the dimensions by 130% on both the x and y axes to remove non-relevant information of the screen borders in order to enhance the extraction of features.
- Brightness change: multiplying all pixels by a random number in the range (0.2, 1.2); thus the image becomes darker if the random number is less than 1, and if the number is bigger than 1, it becomes brighter. This allows the model to extract image features across varying luminosity.
- Panning shift: moving from -10 to +10 pixels on each axis: x and y. This gives some variation to the images, aiming for the model to concentrate on the most critical aspects of the image.
- Flipping: mirroring the image horizontally and also changing the label of the steering angle to its complementary value (negative if it was positive and the other way round). This increases the chance of the model to extract features from the images in the opposite direction with a negative steering angle.

  After the augmentation process is complete, an example of this process can be observed in Figure 8, as well as in other works that utilize images for deep learning models [6]. All the images in the dataset were processed to remove unimportant details. This helps to speed up and refine the training stage. Thus, a combination of operations was conducted applying the CV2 library: (1) Cropping the image (to ensure the model focusing on the most important features in the image, such as road lines and road edges. (2) Converting the image from RGB to YUV, since YUV reduces the bandwidth in comparison to RGB, resulting in higher efficiency. (3) Using a Gaussian blur filter to smooth the image. (4) Resizing the image to 200 x 66 in order to better fit with the used CNN architecture (described in next subsection).

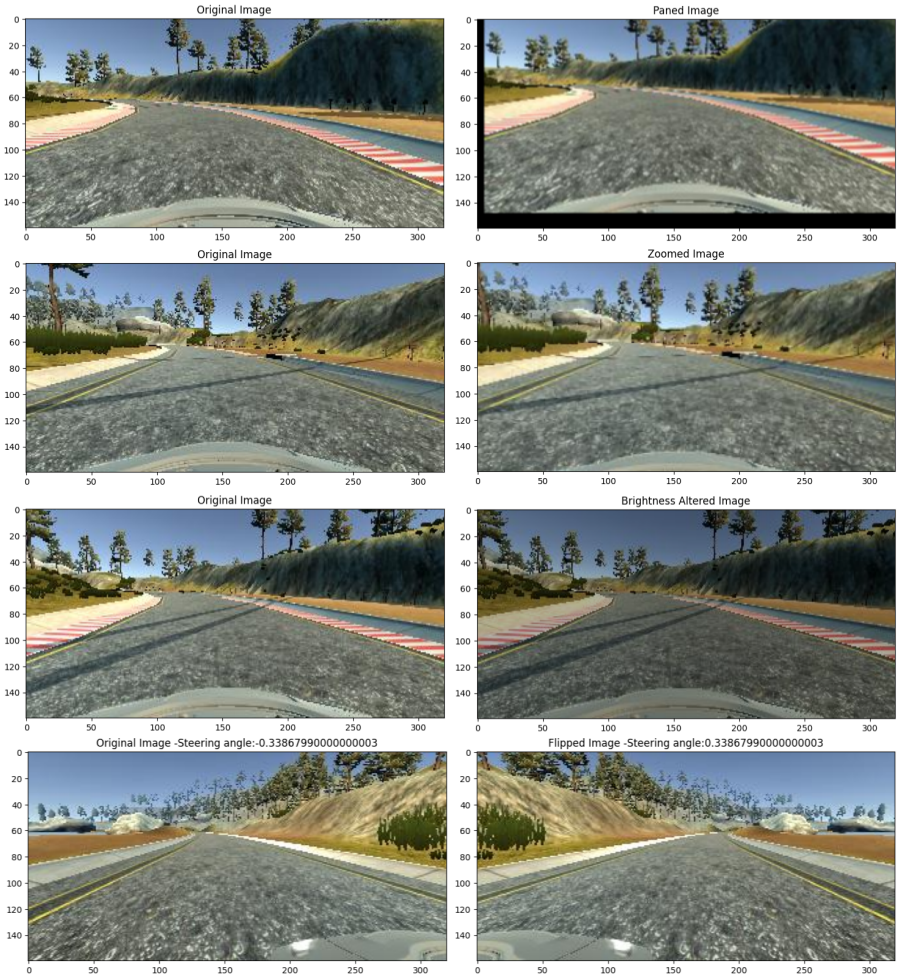

Figure 7. Augmentation examples.

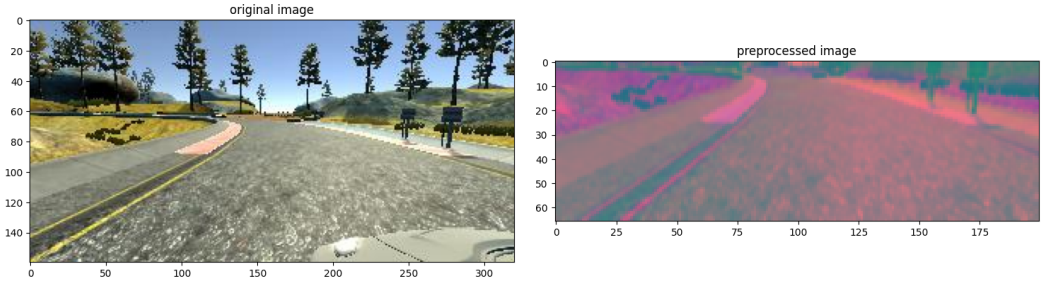

Figure 8. Original image and image preprocessed by cropping, RGB to YUV, Gaussian blur filter, and resizing.

3.3. CNN Model Architecture

A convolutional neural network (CNN) is a deep learning (DL) method designed to process images.

Prior researchers have achieved considerable success in employing this deep learning (DL) technique for pattern recognition in video [26]. Furthermore, the extraction of more useful and powerful features is feasible through the exploitation of advances in machine learning, particularly those pertaining to deep learning [28]. Another study focused on plant disease classification, and a convolutional neural network (CNN) was applied over the features extracted from plant leaf images. This convolution process is summarized in the inputs, filters, and outputs, which include multiple layers and neurons [29].

In this study, the NVIDIA architecture [22] was considered to build the CNN based on previous research findings [2,6].

In a previous study [6], the NVIDIA architecture was compared to a less complex structure. NVIDIA demonstrated superior simulation performance, achieving an accuracy rate of 96.83% compared to the other architecture's accuracy rate of 76.67%. These results suggest that the NVIDIA CNN architecture outperforms other architectures in terms of performance and accuracy, making it an attractive option, particularly in areas such as artificial intelligence and efficient image processing.

The considered configuration for the CNN utilizes the parameters defined in different works of the literature [6,22]. Thus, the kernel size is set to 5 by 2 in the first three layers and 3 by 3 in the following two. Figure 9 shows the considered CNN architecture, which uses images extracted from the simulator.

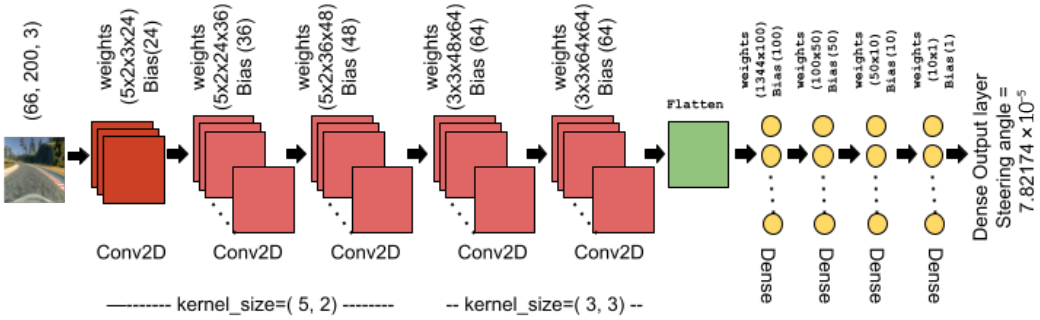

Figure 9. Convolutional neural network (CNN) for steering angle prediction based on NVIDIA architecture.

As can be seen in Figure 9, the architecture consists of eleven layers; the first is the input layer containing the car trajectory samples (images). Each sample is labeled with a steering angle value in the range $[-1, 1]$, where a negative angle indicates a left turn and a positive angle a right turn. This is followed by five layers with different filters. Then comes a layer to flatten all the inputs, followed by four dense layers. Finally, the output layer yields the predicted steering angle [6,22].

An exponential linear unit (ELU) was used as the activation function in all the layers, since ELUs accelerate learning in deep neural networks and lead to reach a higher accuracy. An ELU also improves learning compared to units with other activation functions [30].

3.4. Evolutionary Optimization by Means of Genetic Algorithm

Evolutionary algorithms (EAs) are inspired by Darwin's theory in which individuals adapt to survive and eliminate traits that make them weaker [10].

EAs have many approaches, such as evolutionary programming, evolutionary strategies, genetic programming, and genetic algorithms [10]. In this study, the genetic algorithm (GA) was chosen because it is one of the most applied algorithms for optimization in the literature [12,15], and it is quite simple to implement and combine with other approaches. The GA displays clear maps of the natural evolutionary process on the computer, where it is used in machine learning, pattern recognition, and optimization problems [10].

The genetic algorithm (GA) is a population-based method, in which every individual (or chromosome) is a possible solution for the optimization problem to solve. An individual is composed of a set of values that are called 'genes'. The GA applies four main operators:

- Selection: a set of individuals (normally the best) is chosen to be the parents of the following generation (iteration).
- Crossover: a function that combines the material of two individuals (parents) to create some descendants (a mixture of the combined ones).
- Mutation: a random variation in the individuals' genes.

- Replacement: a substitution of some individuals (normally the worst) by the new generated ones.

A GA strongly depends on the proper definition of an Evaluation Function, normally denoted as a fitness function.

In the context of evolutionary algorithms, each solution is typically evaluated by calculating its fitness value, which facilitates the subsequent selection process. This allows the algorithm to identify the solution that most closely approximates the desired outcome. In the context of our study, we employed a generational genetic algorithm [31], whereby the selected solutions were subjected to a process of mutual recombination. This operator enables the exchange of characteristics between two or more parental solutions, thereby producing new offspring. The objective of this approach is to facilitate the transmission of beneficial traits, thereby enhancing the quality of the population in subsequent generations.

It is important to note, however, that not all evolutionary algorithms rely on combination techniques. To illustrate, a steady-state algorithm [32] may prioritize selection and replacement strategies, thereby circumventing the necessity for a crossover process. Moreover, there are sophisticated evolutionary algorithms, such as CMA-ES [33], xNES [34], and OpenAI-ES [35], which prioritize the enhancement of a single individual rather than the population as a whole. These algorithms employ techniques such as self-adaptation and evolutionary strategies. To provide a comprehensive overview of evolutionary algorithms, we expand our discussion to include these alternative approaches, providing relevant citations to enhance our understanding of evolutionary algorithms.

We considered the GA for the so-called metaoptimization of the CNN model. So, the GA was applied to optimize the weights of the model.The algorithm follows the loop presented in Figure 10.

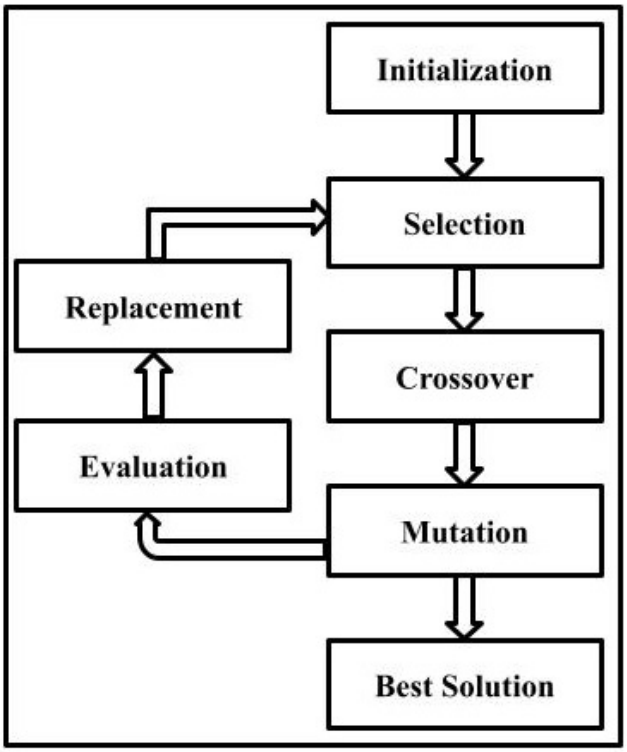

Figure 10. Genetic algorithm procedure.

Thus, in our approach, CNN models were considered as individuals, and their weights were the genes. A generational approach was considered, so half of the population are selected as the parents of the following generation. Two different selection operators were used: best individuals of the population and random selection.

Regarding the crossover operation, a series of weights in a layer is transferred from one model/individual to another, and the layer is chosen randomly. As Figure 11 shows, a sequence of 24 weights in one layer of the model is transferred to the same layer of the other model, with the position of the sequence in both parents chosen randomly.

For the mutation, the weights within the model are modified in a random manner. This is performed with the intention of enhancing diversity and exploring new solutions. This is achieved by replacing the specified weight with a random value drawn from a uniform distribution between 0 and 1.

This research examined two types of population replacement, either replacing completely the previous population with the best new individuals or replacing the population with a mix of the selected parents and new children.

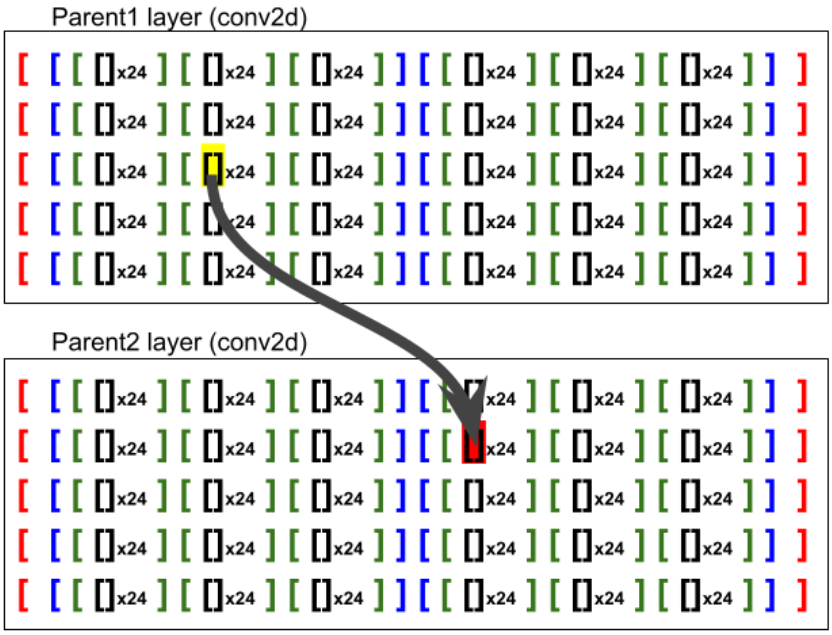

Figure 11. Example of crossover process.

Algorithm 1 also describes the GA process. The algorithm is based on a genetic algorithm, which is used to optimize a population of individuals with the objective of minimizing the loss function and achieving enhanced performance. The algorithm commences with the delineation of a set of parameters, including the mutation probability (p_m) and a generation comprising 50% of the population (n = 16) with an initial population of 32 individuals (L). The process commences with the iteration of several generations (MAX_GENS), during which the parents are selected either based on the least loss or randomly to generate new individuals. Crossings are performed between individuals to generate new offspring, and mutations are performed on the resulting offspring according to the mutation probability in order to enhance diversity and explore new solutions. Subsequently, the new individuals are evaluated, and the least efficient individuals in the population are replaced in accordance with the specified scenarios. This may be accomplished by comparing the fitness of the new individuals (Indivi_i2) with that of existing individuals or by combining the performance of the parents with that of the offspring. The objective is to optimize the performance of the population over generations. This is accomplished through the implementation of selection, crossing, and mutation, which collectively facilitate the attainment of the optimal solution with minimal loss and maximal efficiency.

Algorithm 1: Genetic algorithm optimizer.

Parameters: Mutation probability $P_{mut} = 0.2$ (or 1)
Parameters: Population size $N = 32$ (16 parents per generation)
Parameters: Generations MAX_GENS
Input: Initial population of individuals $P = \{Ind_1, Ind_2, \ldots, Ind_N\}$
Output: Optimized population

for $g = 1$ **to** MAX_GENS **do**
  `// Selection of Parents`
  $ParentPool \leftarrow []$
  for $i = 1$ **to** $\frac{N}{2}$ **do**
    if *Selection criterion (e.g., minimum loss or random)* **then**
      $ParentPool$.append(selected_parent)
    end
  end
  `// Crossover and Mutation to Generate Offspring`
  $Offspring \leftarrow []$
  for $i = 1$ **to** $\frac{N}{2}$ **do**
    $Parent_1, Parent_2 \leftarrow$ Randomly select two parents from $ParentPool$
    $Child \leftarrow$ Crossover($Parent_1, Parent_2$)
    for $j = 1$ **to** *genes_count* **do**
      $r \leftarrow$ random number between $[0, 1]$
      if $r < P_{mut}$ **then**
        Mutate gene j of $Child$
      end
    end
    $Offspring$.append($Child$)
  end
  `// Replacement Strategy`
  Evaluate fitness of all individuals in P and $Offspring$
  foreach *individual child in* $Offspring$ **do**
    $parent\_fitness \leftarrow$ fitness of corresponding parent
    $child\_fitness \leftarrow$ fitness of $child$
    `// Scenario 1: Replace Parent if Offspring is Better`
    if $child\_fitness < parent\_fitness$ **then**
      Replace parent with child in population P
    end
    `// Scenario 2: Combine Parent and Offspring Fitness`
    $combined\_fitness \leftarrow parent\_fitness + child\_fitness$
    Use combined fitness to adjust the probability of replacement
    if $combined\_fitness$ *meets threshold* **then**
      Replace parent with child
    end
  end
end
return *Optimized population* P

4. Experiments, Results, and Discussion

Three experiments were conducted in this study. Firstly, we trained different approaches of convolutional neural network (CNN) models combined with different optimizers from the literature, which were analyzed. Secondly, a GA was tested as an optimizer for a CNN, evaluating and comparing eight different configurations. Finally, the GA was also applied on a fully trained model in order to fine-tune its already optimized weights (with a standard optimizer). All the results of these models were compared. In the first

two experiments, the original dataset was considered, while the third experiment was conducted also using the novel dataset to compare its influence on the results.

The Python programming language and Google Collaboratory (one machine and one GPU) were utilized to write and compile the code, with various libraries such as KERAS, TensorFlow, Scikit-learn, Pandas, NumPy, CV2, and Imgaug integrated for added functionality.

4.1. Experimental Setup

The models were evaluated considering the mean squared error (MSE) as a loss function. It calculates the mean of the squared difference between observed and predicted values [36]. See the formula in Equation (1):

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - y_i')^2 \quad (1)$$

where n is the number of observations/samples, y_i is the value of the i-th observation, and y_i' is the i-th predicted value. The learning rate was set to the default value in the CNNs.

The GA was executed for 50 generations (iterations) with a population of 32 individuals (models), using the aforementioned selection, crossover, mutation, and replacement approaches. Fitness was calculated by determining the mean squared error (MSE) for each individual when running the CNN model with its respective weights.

A total of 30 runs were performed for each non-deterministic optimizer and 10 for each GA approach (due to the high demand on computing time of this algorithm).

4.2. Training CNN Model Using Different Optimizers

In the first experiment, eight existing optimizers were studied, namely, Nadam, Adam, RMSprop, Adamax, SGD, Adagrad, Adadelta, and Ftrl [37].

The *loss value* was calculated using the mean square error (MSE) on the training data (TD) and also on the validation or test data (VD). The average loss for the 30 models trained by each optimizer is shown in Figure 12.

Table 1 also presents the results but in a numerical way. Training the models by using the Nadam optimizer had the lowest average loss based on the training dataset, while the highest mean loss was obtained using Ftrl. In addition, RMSprop and Adam obtained a very good result, as RMSprop had the lowest loss based on the validation dataset. Therefore, we used the models trained with this optimizer in the third set of experiments, which is explained in Section 4.4.

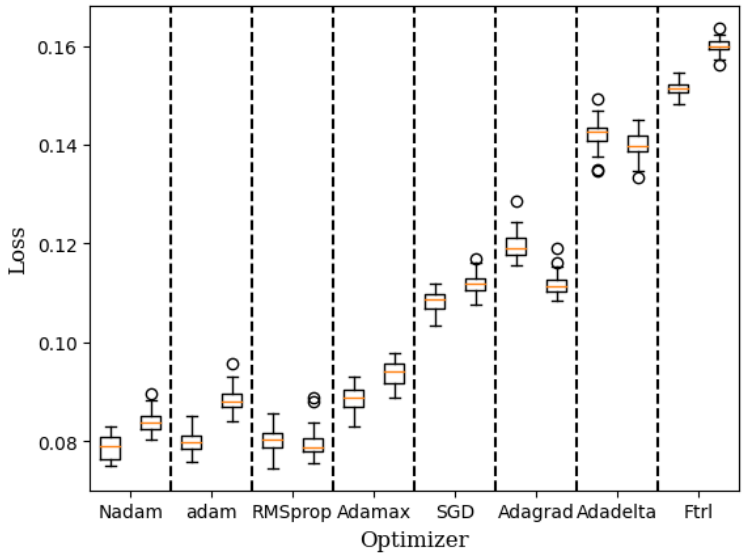

Figure 12. Loss (MSE) boxplots for the 30 trained models for each optimizer, based on the training dataset (left boxplot) and validation dataset (right boxplot).

Table 1. Loss (MSE) results: minimum, average (Avg), and standard deviation (SD) of 30 runs on the training dataset (TD) and the validation dataset (VD) for model training using different optimizers.

Optimizer	Data	Min Loss	Avg	SD
Nadam	TD	**0.0750**	0.0789	±0.0024
	VD	0.0802	0.0838	±0.0022
Adam	TD	**0.0758**	0.0799	±0.0022
	VD	0.0839	0.0884	±0.0024
RMSprop	TD	**0.0745**	0.0802	±0.0025
	VD	0.0755	0.0796	±0.0030
Adamax	TD	0.0829	0.0885	±0.0026
	VD	0.0888	0.0937	±0.0026
SGD	TD	0.1033	0.1081	±0.0019
	VD	0.1075	0.1117	±0.0019
Adagrad	TD	0.1155	0.1195	±0.0026
	VD	0.1084	0.1116	±0.0022
Adadelta	TD	0.1346	0.1422	±0.0030
	VD	0.1332	0.1398	±0.0028
Ftrl	TD	0.1480	0.1515	±0.0015
	VD	0.1562	0.1600	±0.0015

4.3. Optimizing CNN Model by Means of GA Using Different Configurations

As stated before, in the second experiment, an optimization method based on a GA was implemented and applied to a CNN to train a self-driving model. The considered evaluation function (fitness) was the loss value, since a smaller loss means a smaller error and therefore a better optimization result.

This section also studies the impact of the different configurations for the genetic operators (selection, mutation, and replacement), as described in Section 3.4.

Thus, 10 runs were performed with each of the following configurations:

- BBH: selects the best individuals, replaces the population with the best individuals, and has high mutation probability.
- BBL: selects best individuals, replaces population with best individuals, and has low mutation probability.
- BNH: selects the best individuals, replaces the population with parents and children, and has high mutation probability.
- BNL: selects the best individuals, replaces the population with the parents and children, and has low mutation probability.
- RBH: randomly selects individuals, replaces the population with the best individual, and has high mutation probability.
- RBL: randomly selects individuals, replaces the population with the best individual, and has low mutation probability.
- RNH: randomly selects individuals, replaces population with parents and children, and has high mutation probability.
- RNL: randomly selects individuals, replaces population with parents and children, and has low mutation probability.

Tuning a model such as a CNN with a huge amount of weights would be an extremely demanding task if we aimed to test all the possibilities, given that there could be millions of

possible combinations. Thus, applying a GA can perform this task in a reasonable amount of time.

In this study, the GA optimizer starts from a random distribution of weights in the initial population, and it is able to evolve to a competitive configuration for the CNN, as shown in the results presented in Table 2 and Figure 13 after 50 generations. To enhance clarity and effectively emphasize the performance of the optimal optimizer, we decided to limit the range of the curves (graphs) presented in our analysis, since showing the complete ranges would have made it harder to see the essential details necessary for a clear understanding of the key findings. In particular, the RNH and RNL results are cut, because these approaches are based on random selection methods, which lead to considerable variability in the results.

Table 2. Loss (MSE) results of 10 runs applying GA on the CNN model with different configurations for selection function, replacement policy, and mutation operator. Minimum, average (Avg), and standard deviation (SD) are included.

	Min Loss	**Avg**	**SD**
BBH	0.1445	0.1471	±0.0016
BBL	0.1417	**0.1451**	±0.0020
BNH	0.1445	0.1472	±0.0019
BNL	0.1433	0.1459	±0.0015
RBH	0.1443	0.1472	±0.0013
RBL	0.1436	0.1457	±0.0016
RNH	0.4200	8.4456	±7.1396
RNL	0.1498	0.2496	±0.2076

As can be seen in the table and figure, all the approaches yielded similar results, excepting RNH and RNL, which are the configurations that are more focused on exploration, i.e., they add more random components to the search than the rest of the approaches: random selection of parents and replacement with children even if they are worse than previous individuals in the population. This can be also seen in the high value of the standard deviation in both cases.

The best results were obtained by the BBL configuration, which is the approach that most focuses on exploitation, i.e., selecting the best parents overall, replacing with the best individuals, and applying a low mutation factor.

In comparing the optimizer based on a GA and the rest of the optimizers from the literature, we see the best solution (0.1417) was obtained by BBL. It is better than the best solution obtained by the Ftrl optimizer (0.1480).

Given that this proposal is a proof of concept conducted using relatively low computational power (one machine with one GPU), we can consider these results to be promising. This also leads us to think that obtaining a better loss than the best optimizers is doable, but it would be a very demanding task using a GA, because this method is, at the end of the day, a search algorithm inside a space of possible solutions, and the number of combinations of weight values is gargantuan.

However, considering the great power of GAs, they can obtain the optimal combination of weights if we give them enough time/computational power, as they are able to move to different parts of the search space, avoiding local optima, which is something that most of the standard optimizers cannot achieve.

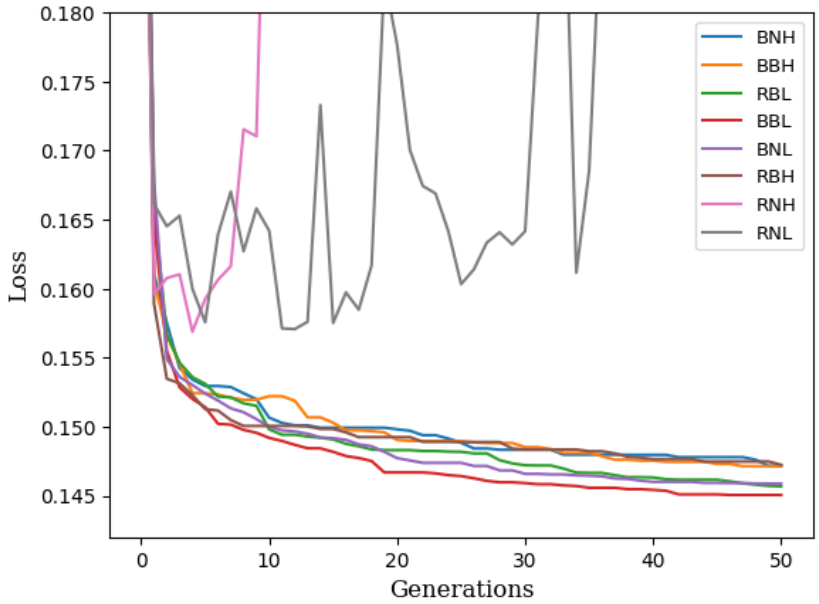

Figure 13. Average loss (MSE) of 10 runs for each of the different configurations for the GA optimizer over the CNN model.

4.4. GA Optimization of a Fully Trained (and Optimized) Model

Finally, the last experiment assessed the value of the GA for the enhancement of already optimized models. To this end, we considered the most effective approaches outlined in Section 4.2, namely, CNN models with RMSprop, Adam, or Nadam optimizers. Thereafter, a GA with the BBL configuration (the optimal outcome of the second experiment) commenced with the set of weights defined by that optimizer (of 32 models trained by RMSpro, Adam, or Nadam as population) and conducted an additional evolutionary process.

Table 3 presents a comparative analysis of three optimizers—namely, Adam & GA, Nadam & GA, and RMSprop & GA—on two distinct datasets. The original and novel datasets were employed to assess the performance of the optimizers. The results demonstrated that RMSprop & GA achieved the lowest minimum loss on the original dataset, indicating superior training efficiency. Additionally, Adam & GA exhibited consistent performance across the two datasets, achieving the lowest minimum loss on the original dataset compared to the other optimizers. Overall, RMSprop & GA demonstrated superior performance in reducing losses, particularly during training. However, in the simulation, Adam & GA was the most effective optimizer, a finding that is discussed in the subsequent section.

As can be seen, an improvement is obtained in all cases, which proves again the value of GAs. It also reinforces our hypothesis regarding the application of a GA as the optimizer in order to obtain smaller loss than the rest of optimizers.

Although the difference in the optimization is not big, it was consistent with the application of a GA for only 50 generations (which will be bigger with more generations). Moreover, we can consider it as relevant due to the sensitivity of the field in which this study is enclosed. Thus, any error in the self-driving vehicle is a potential threat to human life or to material losses. Furthermore, the aforementioned optimization yielded favorable outcomes in terms of driver performance as assessed through simulation-based evaluations.

In this research, execution speed and optimization time were not discussed because all the tested optimizers were integrated in Keras [37], so they can be run on clusters of GPUs or TPUs. Furthermore, Keras is built on the TensorFlow platform, which provides high-speed computation [38]. The tested GA is a preliminary approach, which would be improved if it could be implemented (or integrated) also in Keras.

Table 3. Comparison of enhanced optimizers (by means of GA) on the two datasets. Loss (MSE) value is presented at the beginning (initial population loss) and after the GA optimization process (min loss).

Optimizer	Dataset	Init Populat Loss	Min Loss
Adam & GA	Original	0.076	0.0752
	Novel	0.123	0.1199
Nadam & GA	Original	0.075	0.0747
	Novel	0.130	0.127
RMSprop & GA	Original	0.075	0.0709
	Novel	0.129	0.122

4.5. Autonomous Drivers Performance Tests

Performance tests for autonomous drivers represent a crucial component of the evaluation process; while theoretical outcomes may exhibit promising results, actual driving performance may diverge from these results. The drivers were subjected to testing in a simulator on two distinct tracks, the Jungle and Lake Tracks. The assumed checking distance was set to be three laps of each track.

Table 4 presents the simulation performance of models representing autonomous controllers driving through the Jungle and Lake tracks. The table provides the obtained results of the CNN models combined with the standard optimizers considered in this study, as well as those obtained after the improvement of the best optimizers by means of a GA, as stated previously in Section 4.4. The table shows the 'theoretical' evaluation based on the loss (MSE) value for the predicted steering angle, together with some performance metrics reached at the circuit, namely, max and average speed, covered distance, and time driving. In each case, the first row depicts the results achieved by training the model on the original dataset, while the next row shows the results obtained after training the model on the new dataset.

Looking at the results in the training track (Lake), it can be noticed that almost all the controllers successfully completed the three-lap test distance. Those that were not able to do this could not be considered as reliable controllers for autonomous driving following the presented approach. Focusing on the best ones, the fact that the loss in the new dataset was always worse could be remarkable. This could be interpreted as a poorer performance for the models; however, this happens due to the 'harder' dataset used. In spite of this, the controllers trained using the new dataset behaved much better (than those trained using the original one) when we tested them on the circuit. This reinforces the quality of the enhanced dataset we composed.

With regard to the results obtained on the Jungle Track (unknown for the controllers), it can be seen that, again, models trained using the new dataset obtain a much better performance on the circuit, while their loss is a bit worse. Actually, only models trained with the new dataset were able to finish the three laps on this track, since it is much more difficult than the training one (see Figure 2). Specifically, just Adam variations could reach the end of the test without experiencing any crashes or deviations from the specified path. Only "Adam" and "Adam&GA" made it to the end of the test circuit. Although "Adam" ended with a driver going off the track before getting back on, "Adam&GA" finished the track without any derailments. The rest of the controllers had incidents that caused them to avoid finishing (such as crashing out of the bounds of the lane). This performance is particularly impressive on the Jungle Track, given that the model had not undergone any pretraining and given its difficulty.

Focusing on the optimizers, the results also indicate that models trained with Adam outperform those trained with RMSprop, as measured by the loss (MSE). Vehicles trained using a CNN and RMSprop were only able to cover short distances before deviating from

the track during the simulation. Models trained with RMSprop & GA (RMSprop and genetic algorithm) outperformed those trained with RMSprop alone on both tracks.

Furthermore, the effectiveness of models trained with SGD, Adagrad, and Adadelta was limited as they could only cover short distances on the training and validation tracks before derailing. The case of Nadam is also remarkable, but in the bad sense, since the results from driving on the Jungle Track are poorer using the novel dataset than the original one.

Additionally, we could argue that the CNN model utilizing the Ftrl optimizer was not appropriate for autonomous driving as it did not produce angle changes, even when traveling only a short distance before veering off the track.

The results of this study offer insight into model performance, including adaptability to various environments and the impact of algorithms on final outcomes. Based on these results, it can be concluded that the models trained using Adam with genetic algorithms were the most effective.

The optimal model was Adam & GA, trained on the novel dataset, which demonstrated remarkable performance in the simulation. The model demonstrated the capacity to complete the entire circuit with a top speed of 18.9 m/s, signifying a notable enhancement in performance compared to previous trials. In comparison, Adam also completed the stages but deviated from the road on the Jungle Track and then recovered and continued.

Table 4. Performance metrics for simulation on Jungle and Lake Tracks along 3 laps: Comparison of performance of the different CNN controllers, both original ones with standard optimizers and those improved applying GA. The results show the following: the used optimizer, the epoch (G is for considered generations in the genetic algorithm), and the obtained loss in the steering angle prediction. First line refers to original dataset and second to the new dataset. Together with the loss, the controller's performance in each track is presented as follows: driving speed (maximum, average, and standard deviation), distance covered out of the 3 laps, and the time driving (in seconds) to cover that distance.

Track	Optimizer	Epochs	Loss	Speed (m/s)			Dist. (m)	Dur. (s)
				Max	Avg	SD		
Lake (Training Track)	Adam & GA	50 G	0.0752	6.19	5.44	±0.04	3 Laps	1400
		50 G	0.1199	7.92	7.48	±0.059	3 Laps	1024
	Adam	10 E	0.0839	5.48	5.44	±0.008	3 Laps	1400
		50 E	0.1229	8.03	7.48	±0.060	3 Laps	1025
	Nadam & GA	50 G	0.0747	8.56	7.49	±3.17	3 Laps	1032
		50 G	0.127	7.64	7.48	±0.039	3 Laps	1021
	Nadam	10 E	0.0802	5.71	5.44	±0.020	3 Laps	1400
		34 E	0.1298	7.64	7.48	±0.027	3 Laps	1019
	RMSprop & GA	50 G	0.0709	6.31	5.41	±0.35	863.122	357
		50 G	0.122	7.87	7.48	±0.05	3 Laps	1023
	RMSprop	10 E	0.0755	7.68	5.42	±0.40	737.042	304
		38 E	0.1291	7.79	7.48	±0.05	3 Laps	1024
	Adadelta	10 E	0.1333	6.85	5.39	±0.55	260.088	108
		50 E	0.2197	11.35	7.5	±1.13	217.88	65
	Adamax	10 E	0.0889	6.32	5.37	±0.49	384.058	160
		40 E	0.1339	7.68	7.48	±0.04	3 Laps	1024
	SGD	10 E	0.1076	7.54	5.39	±0.57	349.073	145
		31 E	0.1712	7.82	7.49	±0.024	3 Laps	1016
	Adagrad	10 E	0.1084	6.76	5.37	±0.62	218.067	91
		24 E	0.1908	11.77	7.51	±0.89	339.08	101
	Ftrl	10 E	0.1563	7.08	5.35	±0.58	265.392	111
		14 E	0.3008	9.94	7.4	±0.81	254.91	77

Table 4. *Cont.*

Track	Optimizer	Epochs	Loss	Speed (m/s)			Dist. (m)	Dur. (s)
				Max	Avg	SD		
Jungle (Testing Track)	Adam & GA	50 G	0.0752	7.19	4.87	±0.64	308.864	142
		50 G	0.1199	18.9	6.9	±1.31	3 Laps	635
	Adam	10 E	0.0839	7.16	4.88	±0.94	117.789	54
		50 E	0.1229	18.49	5.45	±2.94	3 Laps	814
	Nadam & GA	50 G	0.0747	10.67	7.11	±0.9	330.408	104
		50 G	0.127	9.66	6.89	±1.57	110.844	36
	Nadam	10 E	0.0802	7.09	4.72	±1.05	80.162	38
		34 E	0.1298	9.1651	6.63	±2.12	26.694	9
	RMSprop & GA	50 G	0.0709	6.67	4.9	±0.66	249.661	114
		50 G	0.122	18.21	7.3	±1.86	777.966	238
	RMSprop	10 E	0.0755	7.12	4.89	±0.67	238.48	109
		38 E	0.1291	10.3	6.98	±1.18	287.352	92
	Adadelta	10 E	0.1333	9.38	4.89	±2.94	21.85	10
		50 E	0.2197	10.08	6.74	±2.37	7.528	9
	Adamax	10 E	0.0889	6.86	4.68	±1.02	79.479	38
		40 E	0.134	10.15	7.12	±0.61	141.3	160
	SGD	10 E	0.1076	10.82	4.98	±1.43	80.204	36
		31 E	0.1712	9.896	7.12	±0.88	216.49	68
	Adagrad	10 E	0.1084	7.25	4.69	±1.19	62.94	30
		24 E	0.1908	9.5	7.1	±0.96	171.264	54
	Ftrl	10 E	0.1563	7.01	4.45	±1.89	19.916	10
		14 E	0.3008	9.93	6.69	±2.6	20.937	7

5. Conclusions and Future Work

In this work, we present an approach for a module to be used in a future autonomous driving agent, aiming to infer the steering angle in a self-driving car based on a set of images captured while driving and received as input.

To this end, we created two of our own image-based datasets collected while driving in the Udacity simulator. The simulator allowed us to gathering images from three on-board camera locations (right, center, and left). Then, each group of images were preprocessed (to enhance their utility) and labeled considering the current steering angle.

Convolutional neural network (CNN) models of NVIDIA architecture with eight different optimizers (Nadam, Adam, RMSprop, Adamax, SGD, Adagrad, Adadelta, Ftrl) were trained on these data to analyze the best approach. Training the CNN model using the Nadam optimizer had the lowest/best average loss, while the highest/worst average loss was obtained using the Ftrl optimizer.

In addition, a genetic algorithm (GA) approach was proposed and tested, aiming to optimize the set of weights on which the CNN depends. Thus, additional CNN models based on the same architecture were then built starting from random weights. The GA was applied to optimize the weights of these models, as well as other previously trained and optimized models, using some of the aforementioned optimizers from the literature.

Several experiments were performed with different configurations for the GA, and the obtained models were then compared with previous optimizers. According to the results of these experiments, we concluded that the GA is a valid and promising tool for optimizing CNN weights, starting from both random values and optimized ones.

This study presents simulation results on the performance of CNN models using different optimization tools, such as Nadam, Adam, and Adam & GA (Adam with genetic algorithms). The results indicate that using Adam & GA was the most effective, while models using RMSprop were less effective than those using the genetic algorithm alone. The effectiveness of models using SGD, Adagrad, and Adadelta was found

to be weak. Furthermore, it was observed that models using Ftrl are not suitable for autonomous driving.

Thus, this tool will be deeply studied in future works, aiming to optimize not only weights of the models but also their internal structure (layer composition) and their configuration parameters (algorithm metaoptimization). In this line, the GA will be implemented using other tools and integrated into Keras, if possible, to take advantage of the computing resources that can be used in this framework. In addition, more sophisticated EA approaches (such as OpenAI-ES [35]) or even specialized ones will be applied to the CNN models, aiming to reach a better performance in autonomous driving. Another line to explore will be the consideration of Transformers [39], a more advanced deep learning technique that could be useful in this domain.

Author Contributions: Conceptualization, B.K. and A.M.M.; Methodology, B.K.; Software, B.K.; Validation, B.K. and A.M.M.; Formal Analysis, B.K.; Investigation, B.K.; Resources, B.K.; Data Curation, B.K.; Writing—Original Draft Preparation, B.K.; Writing—Review and Editing, A.M.M. and H.F.; Visualization, B.K.; Supervision, A.M.M. and H.F.; Project Administration, A.M.M.; Funding Acquisition, A.M.M. All authors have read and agreed to the published version of the manuscript.

Funding: This work has been partially funded by the Spanish Ministry of Science, Innovation and Universities MICIU/AEI/10.13039/501100011033 under project PID2023-147409NB-C21, as well as by the European Union NextGenerationEU/PRTR, under projects TED2021-131699B-I00 and TED2021-129938B-I00. It has also been funded by projects PID2020-113462RB-I00 and PID2020-115570GB-C22 of the Spanish Ministry of Economy and Competitiveness; project C-ING-179-UGR23 financed by the "Consejería de Universidades, Investigación e Innovación" (Andalusian Government, FEDER Program 2021-2027); and project PPJIA2023-031 (Plan Propio de Investigación y Transferencia UGR).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data supporting the reported results can be obtained upon reasonable request from the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Hagos, D.H.; Rawat, D.B. Recent Advances in Artificial Intelligence and Tactical Autonomy: Current Status, Challenges, and Perspectives. *Sensors* **2022**, *22*, 9916. [CrossRef] [PubMed]
2. Smolyakov, M.V.; Frolov, A.I.; Volkov, V.N.; Stelmashchuk, I.V. Self-Driving Car Steering Angle Prediction Based on Deep Neural Network: An Example of CarND Udacity Simulator. In Proceedings of the 2018 IEEE 12th International Conference on Application of Information and Communication Technologies (AICT), Baku, Azerbaijan, 17–19 October 2018; pp. 1–5. [CrossRef]
3. Khanum, A.; Lee, C.-Y.; Yang, C.-S. End-to-End Deep Learning Model for Steering Angle Control of Autonomous Vehicles. In Proceedings of the 2020 International Symposium on Computer, Consumer and Control (IS3C), Taichung, Taiwan, 13–16 November 2020; pp. 189–192. [CrossRef]
4. Valiente, R.; Zaman, M.; Ozer, S.; Fallah, Y.P. Controlling Steering Angle for Cooperative Self-Driving Vehicles Utilizing CNN and LSTM-Based Deep Networks. In Proceedings of the 2019 IEEE Intelligent Vehicles Symposium (IV), Paris, France, 9–12 June 2019; pp. 2423–2428. [CrossRef]
5. Udacity. An Open Source Self-Driving Car. 2017. Available online: https://udacity.com/self-driving-car (accessed on 28 September 2021).
6. Lade, S.; Shrivastav, P.; Waghmare, S.; Hon, S.; Waghmode, S.; Teli, S. Simulation of Self-Driving Car Using Deep Learning. In Proceedings of the 2021 International Conference on Emerging Smart Computing and Informatics (ESCI), Pune, India, 5–7 March 2021; pp. 175–180. [CrossRef]
7. Setta, I.G.A.; Shehata, O.M.; Awad, M.A. Multivariate Prediction of Correct Lane for Autonomous Electric Vehicle Using Deep Learning Models. In Proceedings of the 2020 8th International Conference on Control, Mechatronics and Automation (ICCMA), Cairo, Egypt, 6–8 November 2020; pp. 127–130. [CrossRef]
8. Broome, M.; Gadd, M.; De Martini, D.; Newman, P. On the Road: Route Proposal from Radar Self-Supervised by Fuzzy LiDAR Traversability. *AI* **2020**, *1*, 558–585. [CrossRef]
9. Li, Z.; Liu, F.; Yang, W.; Peng, S.; Zhou, J. A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects. *IEEE Trans. Neural Netw. Learn. Syst.* **2022**, *33*, 6999–7019. [CrossRef] [PubMed]

10. Coello, C.A.C. An Introduction to Evolutionary Algorithms and Their Applications. In *Advanced Distributed Systems*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2005; Volume 3563, pp. 425–442.
11. Bäck, T. *Evolutionary Algorithms in Theory and Practice*; Oxford University Press: Oxford, UK, 1996.
12. Vikhar, P.A. Evolutionary Algorithms: A Critical Review and Its Future Prospects. In Proceedings of the 2016 International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC), Jalgaon, India, 22–24 December 2016; pp. 261–265.
13. Mora, A.M.; Fernández-Ares, A.; Merelo, J.J.; García-Sánchez, P.; Ferndes, C.M. Effect of Noisy Fitness in Real-Time Strategy Games Player Behaviour Optimisation Using Evolutionary Algorithms. *J. Comput. Sci. Technol.* **2012**, *27*, 1007–1023. [CrossRef]
14. Goldberg, D.E. *Genetic Algorithms in Search, Optimization and Machine Learning*; Addison Wesley: Reading, MA, USA, 1989.
15. Mirjalili, S.; Song Dong, J.; Sadiq, A.S.; Faris, H. Genetic Algorithm: Theory, Literature Review, and Application in Image Reconstruction. In *Nature-Inspired Optimizers*; Mirjalili, S., Song Dong, J., Lewis, A., Eds.; Springer: Cham, Switzerland, 2020; Volume 811; p. 5._5. [CrossRef]
16. Tampuu, A.; Matiisen, T.; Semikin, M.; Fishman, D.; Muhammad, N. A Survey of End-to-End Driving: Architectures and Training Methods. *IEEE Trans. Neural Netw. Learn. Syst.* **2022**, *33*, 1364–1384. [CrossRef] [PubMed]
17. Tobin, J.; Fong, R.; Ray, A.; Schneider, J.; Zaremba, W.; Abbeel, P. Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World. In Proceedings of the 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vancouver, Canada, 24–28 September 2017; pp. 23–30.
18. Dosovitskiy, A.; Ros, G.; Codevilla, F.; Lopez, A.; Koltun, V. CARLA: An Open Urban Driving Simulator. In Proceedings of the 1st Annual Conference on Robot Learning, Mountain View, CA, USA, 13–15 November 2017; pp. 1–16.
19. Malik, S.; Khan, M.A.; El-Sayed, H. CARLA: Car Learning to Act—An Inside Out. *Procedia Comput. Sci.* **2022**, *198*, 742–749. [CrossRef]
20. Amini, A.; Gilitschenski, I.; Phillips, J.; Moseyko, J.; Banerjee, R.; Karaman, S.; Rus, D. Learning Robust Control Policies for End-to-End Autonomous Driving from Data-Driven Simulation. *IEEE Robot. Autom. Lett.* **2020**, *5*, 1. [CrossRef]
21. Bewley, A.; Rigley, J.; Liu, Y.; Hawke, J.; Shen, R.; Lam, V.D.; Kendall, A. Learning to Drive from Simulation without Real World Labels. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Montreal, Canada, 20–24 May 2019; Volume 456, pp. 4818–4824.
22. Bojarski, M. End to End Learning for Self-Driving Cars. *arXiv* **2016**, arXiv:1604.07316.
23. Chishti, S.O.A.; Riaz, S.; Zaib, M.B.; Nauman, M. Self-Driving Cars Using CNN and Q-Learning. In Proceedings of the 2018 IEEE 21st International Multi-Topic Conference (INMIC), Karachi, Pakistan, 1–2 November 2018; pp. 1–7. [CrossRef]
24. Saranya, M.; Archana, N.; Reshma, J.; Sangeetha, S.; Varalakshmi, M. Object Detection and Lane Changing for Self-Driving Car Using CNN. In Proceedings of the 2022 International Conference on Communication, Computing and Internet of Things (IC3IoT), Chennai, India, 10–11 March 2022; pp. 1–7. [CrossRef]
25. Murillo-Morera, J.; Castro-Herrera, C.; Arroyo, J.; Fuentes-Fernandez, R. An Automated Defect Prediction Framework Using Genetic Algorithms: A Validation of Empirical Studies. *Intel. Artif.* **2016**, *19*, 114–137. Available online: https://journal.iberamia.org/index.php/intartif/article/view/48 (accessed on 6 August 2024). [CrossRef]
26. Shafiee, M.J.; Chywl, B.; Li, F. Fast YOLO: A Fast You Only Look Once System for Real-Time Embedded Object Detection in Video. *arXiv* **2017**, arXiv:1709.00724. [CrossRef]
27. Buda, M.; Maki, A.; Mazurowski, M.A. A Systematic Study of the Class Imbalance Problem in Convolutional Neural Networks. *Neural Netw.* **2018**, *106*, 249–259. [CrossRef] [PubMed]
28. Fawole, O.A.; Rawat, D.B. Recent Advances in 3D Object Detection for Self-Driving Vehicles: A Survey. *AI* **2024**, *5*, 1255–1285. [CrossRef]
29. Lu, J.; Tan, L.; Jiang, H. Review on Convolutional Neural Network (CNN) Applied to Plant Leaf Disease Classification. *Agriculture* **2021**, *11*, 707. [CrossRef]
30. Clevert, D.A.; Unterthiner, T.; Hochreiter, S. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). *arXiv* **2015**, arXiv:1511.07289.
31. Pagliuca, P.; Inglese, D.; Vitanza, A. Measuring emergent behaviors in a mixed competitive-cooperative environment. *Int. J. Comput. Inf. Syst. Ind. Manag. Appl.* **2023**, *15*, 69–86.
32. Whitley, D.; Kauth, J. GENITOR: A different genetic algorithm. In Proceedings of the 4th Rocky Mountain Conference on Artificial Intelligence, Denver, CO, USA, 13–15 June 1988.
33. Hansen, N.; Ostermeier, A. Completely derandomized self-adaptation in evolution strategies. *Evol. Comput.* **2001**, *9*, 159–195. [CrossRef] [PubMed]
34. Wierstra, D.; Schaul, T.; Glasmachers, T.; Sun, Y.; Peters, J.; Schmidhuber, J. Natural evolution strategies. *J. Mach. Learn. Res.* **2014**, *15*, 949–980.
35. Salimans, T.; Ho, J.; Chen, X.; Sidor, S.; Sutskever, I. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv* **2017**, arXiv:1703.03864.
36. Mean Squared Error. Wikipedia. Available online: https://en.wikipedia.org/wiki/Mean_squared_error (accessed on 22 October 2023).
37. Keras. Available online: https://keras.io (accessed on 5 June 2023).

38. Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G.S.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015. Available online: https://tensorflow.org (accessed on 5 June 2023).
39. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.; Kaiser, Ł; Polosukhin, I. Attention is all you need. *Advances in Neural Information Processing Systems*; 2017; Curran Associates, Inc.: Red Hook, NY, USA.