Journal of Advanced Engineering Trends
ISSN : 2682 - 2091

Vol.43, No.1. January 2024

http://jaet.journals.ekb.eg

Genetic Optimization for Self-Driving Vehicles Using Automated Behavior Cloning based on Convolutional Neural Network

Ahmed Moaty[1,*)], Kamel Hussein Rahouma[2, 3], Ahmed Donkol[1]

[1]*CCE Dept., faculty of Engineering, Nahda University, Beni-Suef, Egypt*

[2]*Electrical Engineering Dept., Minia University, Minia, Egypt*

[3]*faculty of computer science, Nahda University, Beni-Suef, Egypt*

[*)] Email: *Ahmedmoaty4@gmail.com*

A R T I C L E I N F O

Article history:
Received:
Accepted:
Online:

Keywords:
Genetic
CNN
Self-driving
Behavior cloning

A B S T R A C T

Recently, autonomous driving technology has learned to drive safely and smoothly. Nonetheless, using convolutional neural networks (CNNs) has significantly influenced designs of autonomous driving technology. Most current designs of CNN-based autonomous driving technology are built manually by specialists in both CNNs and the investigated topics. Thus, finding the optimum CNN designs for learning safe driving behavior is complex. This study uses genetic algorithms to construct automated genetic behavior cloning (AGBC). The "Automated" features of the proposed algorithm require little knowledge of CNNs. Using a front-facing camera and an experienced driver's steering directions, researchers can acquire a strong CNNs architecture for learning safe driving behavior. Furthermore, it can be used to teach AGBC-CNNs to emulate human driving behavior. We compared our proposed technique AGBC-CNN with four manually adjustment behavior cloning CNNs (MABC-CNNs) and one automated genetic manually adjustment behavior cloning CNNs (AGMABC-CNNs) to validate its effectiveness and robustness. Studies show that AGBC-CNN outperforms existing architecture design techniques (MABC-CNNs and AGMABC-CNN) in terms of classification accuracy.

1. Introduction

The past decade has seen a rapid increase in the installation of driving assistance and vision systems in vehicles. The automotive market has begun to move toward "smart" cars, using the most recent technological advances in their high-end vehicles. Researchers developing self-driving vehicles may use the vehicle's built-in front-facing cameras to create the following features: (The road's lanes, identification of free parking spaces, recognition of traffic signs, detection and recognition of traffic signals, and detection and tracking of roadside objects). These features can be tracked and recognized at the same time. Consequently, the installation of a vision system in the vehicle (especially the front-facing camera) and possible self-driving car driving characteristics are designed to enhance the overall safety, [1].

Convolutional neural networks (CNNs) [2], the dominant approach in deep learning, have shown fantastic domination over alternative machine learning methods in many real-world applications. The performance of a CNN can be significantly impacted by its design. The most remarkable CNN designs, such as GoogleNet or ResNet, must be created by experts who thoroughly grasp data and CNNs. However, this degree of skill is unavailable to everyone. In both cases, users are not required to be familiar with CNN structures to understand the data they are working with. Users with no previous knowledge of CNNs may now witness the tuning abilities of CNN designs due to the growing interest in automating the building of CNN structures. However, CNN design algorithms may stimulate the widespread use of CNNs, increasing machine intelligence.

There are two categories of CNN architectural design algorithms: those that require domain knowledge and those that do not. Automated manually adjusted (AMA) requires human tuning in constructing CNN architectures. This category has been implemented in the AMA genetic CNN, hierarchical representation (hierarchical evolution), efficient architecture search method (EAS) [3], Block-QNN-S method [4], and advanced neural architecture search method (Analyst)., the other "automated" CNN architectural designs require no human intervention. Cartesian genetic programming, neural architecture search, and meta-modeling are examples of large-scale evolutionary methods. The AMA designs of CNN frequently exhibit superior performance than its "automatic" variants because of the additional advantages of manual skill. However, the critical advantage of the "automatic" designs is their lack of manual expertise, which is more appreciated by users who have no prior domain knowledge of CNN. Therefore, designing a CNN framework is required since NSANet only automates a portion of the framework's cells. The final CNN performance will underwhelm if the framework architecture is poorly designed. A conceivable CNN architecture can be quickly created using the available data and the large-scale evolution approach. Due to this lack of substantial CNN expertise, "automated" CNN architectural designs are appreciated more by CNN users [7].

CNN architectures can be classified into evolutionary algorithmic and reinforcement learning-based [8] designs, depending on the methods used. Evolutionary algorithms are used by genetic CNNs, large-scale evolution, hierarchical evolution, and CGP-CNN to identify the optimal answer. However, reward-penalty reinforcement learning is a method for finding the optimum solution similar to evolutionary algorithms but uses the reward-penalty concept of reinforcement learning. Experiments show that designs using reinforcement learning are more computationally intensive than those using evolutionary

Revised:14 June, 2022, Accepted:6 October , 2022

algorithms. For instance, using the NAS technique [9], it took 28 days and 800 GPUs to find a suitable CNN architecture on the CIFAR10 dataset. Evolutionary algorithm-based CNN designs are preferable since they require no high-end computational resources, which are inaccessible to everyone.

Evolutionary algorithms are a type of metaheuristic optimization paradigm developed from biological evolution. For example, the genetic algorithm (GA) is the most widely used evolutionary algorithm because of its theoretical backing and promising performance in dealing with several optimization problems [10, 11]. GA can use bio-inspired operators like mutation, crossover, and selection to generate high-quality optimal solutions. Most operators are created from the ground up to deal with specific problems. Neural network topologies may be automatically optimized using CNN-GA [11], which employs evolutionary algorithms. This technique does not need to further alter the architecture of the neural network. Therefore, it is an algorithm for automatically creating a CNN architecture.

Humans use their eyes, hands, and brains mainly to drive any vehicle. Therefore, in an automated vehicle, the detection of road images using a camera instead of the eye, steering angles prediction instead of hands, and CCN architecture instead of the brain is paramount. Thus, the detection of road images and predicating the steering angle using CCN architecture is the dominant way to achieve a self-driving car. The suggested method automates everything without relying on human involvement to find the optimum CNN designs. Therefore, self-driving cars use the suggested CNN-GA algorithm that makes the following contributions:

1) Because the crossover operators are primitively built for persons of the same length [12], GAs adopt a fixed-length encoding scheme. Here, a predetermined length for the encoding is considered. Ideally, the length should be the ideal CNN depth, which is primarily unknown before the experiment due to the inaccurate estimation of the supplied quantity. Numerous studies have separately proposed this variable-length encoding approach. However, the final CNN architecture is unideal since the crossover operator does not have a similar redesign. Our research overcomes the above concerns using an efficient and effective variable-length encoding approach and a crossover operator combination.

2) The fundamental components or well-designed blocks of CNNs form the basis for most extant CNN architectural algorithms. Hence, both strategies often produce ineffective and complicated CNN architectures with low generalization abilities, respectively. Skip connections prevent vanishing gradient (VG) [13] difficulties in this study while dealing with complex data. However, this design decreases the search area, making it easier to identify the optimum performance. However, the designs of the suggested algorithm are substantially more straightforward than previous algorithms with comparable performance.

3) Prior algorithms required substantial processing resources to speed up CNN design and provide consumers with an ideal CNN architecture within a reasonable period. Particularly, these algorithms use an inefficient data parallelism method. The fundamental reason is that most designs are medium-sized and do not require much computing power. In this absence, communication costs take up most computing resources. Thus, an asynchronous computational component is constructed in the suggested approach to fully use the provided computing resources to expedite individual fitness assessment. Contrarily, a caching component minimizes population fitness evaluation time.

The rest of the paper is arranged as follows: Section 2 begins with a brief introduction to the subject matter and then discusses comparable works and their context. Section 3 discusses the proposed algorithm in all its nitty-gritty glory. Finally, Sections 4 and 5 detail the methods, data, and conclusions gleaned from the research. Section 6 presents conclusions and future projects.

2. Literature Review

2.1. similar work

Vanita Jain [14] used reinforcement learning. The author uses an iterative update that frequently changes the Q-values to lessen the correlation further. His methodology demonstrates that, after several rounds, the virtually produced vehicle creates collision-free travel and exhibits human-like driving behavior. The systematic review on AVS [15] using deep learning is divided into several modules, including tasks like decision-making, end-to-end controlling and prediction, path and motion planning, and augmented reality-based -head-up display while analyzing research works from 2011 to 2021, concentrating on RGB camera vision. M. R. Bachute [16] investigated variety of tasks, including Motion Planning, Vehicle Localization, Pedestrian Detection, Traffic Sign Detection, Road-marking Detection, Automated Parking, Vehicle Cybersecurity, and Fault Diagnosis, various Machine Learning and Deep Learning Algorithms are used in Autonomous Driving Architectures.

2.2. proposed algorithm background

This section focuses on the genetic CNN technique [17], which is the basis for the proposed algorithm. Generally, a CNN architecture can be encoded in a genetic CNN via a series of steps. To produce the final model, several layers of smaller designs are assembled. NSANet uses convolutional layers to build the final network. Additionally, the genetic CNN provides a framework for constructing a network with convolutional layers replaced with small designs. A “cell” in the genetic CNN is a cell in a specific stage of development. Each stage in a CNN formation is encoded with a series of specified building pieces and their associations. Building elements for the first and final layers are manually provided, and all subsequent convolutional layers retain the same parameters for the sequence they are placed. The linkages between these ordered construction parts are encoded using a binary string encoding technique as a binary string. A string serves as a connection point for all components of the building. As an option, the number of phases can be manually selected. Many human inputs are required to finish this encoding strategy, which can be observed by the number of construction elements and total stages. Because they are linked to the depth of the discovered CNN, prior knowledge of the CNN domain is necessary to predefine these values for acceptable CNN depths. Each level of the construction components of the genetic CNN cannot be changed. Thus, CNN’s connections can only be detected, and it does not impact the CNN depth using this encoding strategy.

3. The Proposed Technique

3.1. Genetic Algorithm

The genetic algorithm is divided into four steps as shown in Fig. 1.

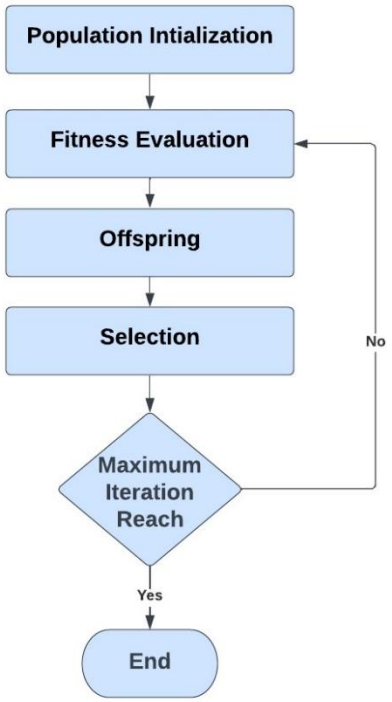

Figure 1: Flowchart of the genetic algorithm

1. Population initialization

The Population initialization consists of two convolutional layers and a skip connection to make a skip layer, as shown in Fig. 2. An influential CNN depends heavily on its depth, and the skip connections allow it to more efficiently achieve this depth level. Only the skip and pooling layers are employed to build a CNN in the proposed encoding approach. The code representing the whole CNN is the sequential string connection of codes representing each layer. Thus, a random value, r, is produced from (0,1) to represent the pooling type for each node during setup. The feature map numbers for convolutional layers are used as a string to represent the feature code of a single skip layer [18].

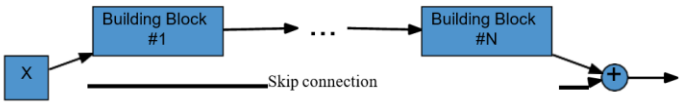

Figure 2: Example of a skip connection

2. Fitness evaluation

The fitness evaluation explains how the population is evaluated for fitness. Each person's fitness is directly derived from the cache if the individual is on it. Alternatively, the person is put asynchronously on an accessible GPU for a fitness assessment. Theoretically, irregular names are used when they do not confuse persons encoding various architectures. When decoding a CNN, the output of the convolutional layer is supplemented with a rectifier activation function followed by a batch normalization procedure, which follows the standards of current CNNs [19].

Next, the stochastic gradient descent (SGD) technique was used to train the CNN on the training data on the provided GPU, and the classification accuracy was determined by the evaluation data of fitness. The model-parallel pipeline divides a large model into several smaller ones, each loaded into a GPU. The memory of one GPU is too little to adequately process all the data immediately. Thus, the overall processing time is reduced by completing these sub problems on various platforms. Our suggested approach has a similar asynchronous component.

For each entry in the cache component, an individual CNN's identification and fitness value are combined into one string similar to a map data structure. A CNN architecture algorithm, including the suggested method, only evaluates thousands of CNNs in a real-world application. We need not worry about the cache component size since it takes very little space on our hard disks. The cache component, for starters, resembles a map data structure in that each record is a string that combines the identification and the fitness value of a CNN. A record like "identifier1=97.23," for instance, indicates that the identifier is "identifier1" and its fitness value is "97.23". Second, as we have already mentioned, the 224-hash code [20] is used to calculate the identification, which can provide 2224 distinct identifiers. The suggested approach and other CNN architecture algorithms only analyses a small number of CNNs in practice. It goes without saying that we do not need to take the competing situation into account since it will not arise. Thirdly, the proposed algorithm's 224-hash code implementation will produce an identifier with a length of 32, and the fitness value is the classification accuracy, which is represented by a string with a length of 4. A surrogate model is frequently used to calculate the fitness of computationally demanding issues, hence avoiding the direct fitness evaluation. Each record in the cache component is a string with a length of 37 and a file encoding of UTF-8 that takes up 37 bytes. Even if the cache file contains thousands of items, it will obviously only take up a very little amount of disc space. Consequently, we do not require.

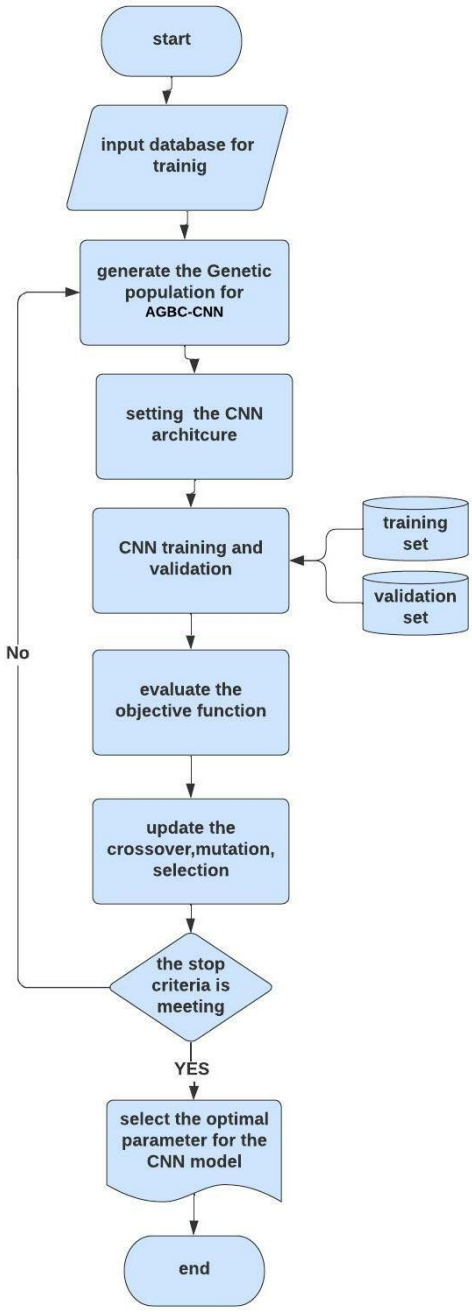

Figure 3: Proposed CNN architecture

3. Offspring

The offspring where crossover and mutation are the two aspects of the proposed method. Because of the crossover procedure, new offspring will be created and measured using the collection size. Randomly selecting an individual from the current population and executing a specific mutation operation depending on the probabilities. The crossover operator is based on the one-point crossover used in GAs. Our proposed approach includes four distinct mutation operators [21]. Because a deeper CNN has greater power, this design was born out of a desire to make a more powerful CNN. The input data is halved when just one pooling layer is used, making the newly found CNN unavailable.

4. Selection

Individuals are selected from the current population using the binary tournament selection and placed into the next population. The best individual is selected and placed into the parent population. Otherwise, it replaces the worst individual in the next parent population [22].

3.2. Proposed CNN architecture

A collection of preconfigured CNN building blocks, population size, maximum number of GA, and image classification dataset generations are provided. On the supplied dataset, we access the fitness of each individual, which encodes a specific CNN architecture. Then, a population of individuals living into the next generation is picked using the environmental selection from the existing population. Generally, the convolutional, pooling, and sometimes fully connected layers make up a CNN as demonstrated in Fig. 3.

3.3. Proposed steering angle prediction

When developing an autonomous driving system, we used end-to-end learning. Our autonomous driving system used raw pixels from the image as its input and the steering angle as its output to manage the vehicle. The trained network will only learn how to manage a car based on an input signal from the camera during real-time inference, according to the end-to-end learning method used as shown in Fig. 4.

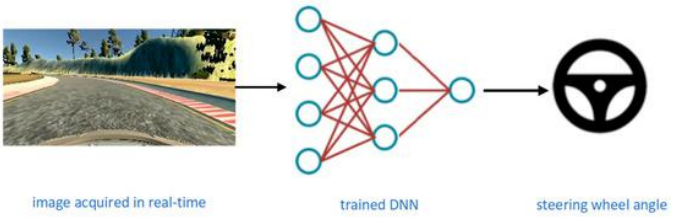

Figure 4: steering angle prediction

4. Design of the Experiment

We performed several picture classification tasks to assess the performance of our proposed algorithm. This section is divided into three subsections: the first subsection introduces state-of-the-art (SOTA) algorithms to be compared with the proposed approach. The second subsection lists the benchmark datasets used in this study. Finally, we present the parameter settings for the proposed algorithm in the third subsection.

4.1. The Challengers

To demonstrate the effectiveness of AGBC-CNN, we compared it with the current SOTA algorithms. These algorithms were selected from three separate categories. For example, ResNet [23], DenseNet [24], and VGGNet [25] are examples of MABC-CNN. We compared two ResNet models with depths of

110 and 1,202, respectively. For simplicity, they are referred to as ResNet (110 depth) and ResNet (1,202 depth). The results showed that ResNet (110 depth) has the best classification accuracy on Udacity Supplied data [26]. Among the DenseNet versions, we chose the DenseNet-BC version since it realized the best classification accuracy with the fewest parameters, followed by VGGNet.

Furthermore, we considered AGMABC-CNN, such as Block-QNN-S in the second section. Finally, the proposed algorithm AGBC-CNN, such as CNN-GA + cutout [27] was used in the third section. The "cutout" refers to a regularization strategy employed in training CNNs, which potentially increases the overall performance. The proposed approach provides an "automated" way to construct promising CNN structures for those without deep expertise in CNN tuning.

4.2. Datasets

Two primary sources of datasets were used:

To train the proposed and SOTA algorithms, the training dataset must first be constructed. The first dataset (Udacity data) was provided by Udacity, containing unzipped files with a size of 365 MB. The files include 24,108 photos evenly distributed among photographs taken with the center, left, and right front cameras, as shown in Fig. 5. Each picture has a 160 × 320 pixel resolution and three channels of RGB color. A CSV file containing 8,036 entries serves as the index for the data.

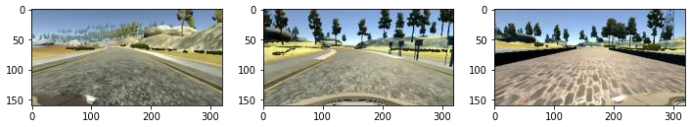

Figure 5: Right, center, and left camera

The second dataset (collected data) was generated by the free source Udacity driving simulator [28]. The unzipped 1,616 MB data collection contains 99,702 photos from the center, left, and right front-facing cameras. Additionally, each picture has a 160 × 320 pixel resolution and three channels of RGB color. Also, the data index was saved in a CSV file with 33,234 lines of entries. We obtained the data by driving the automobile manually around Track 1 in the simulator multiple times (about 10) with the safest driving style feasible. The inclusion of the "recovery" data was especially suggested during training. A good beginning point for capturing data is when the car is about to go off track and the vehicle is guided in the other direction. This will allow the model to learn how to recover from a loss of control situation.

Inaccurate data will probably cause model malfunction, resulting in subpar results. Thus, several data visualization and analysis subroutines have been created. Fig 6. shows that the output of these subroutines can be seen in the produced training collected data. The histogram of the steering angle values acquired while driving collected data.

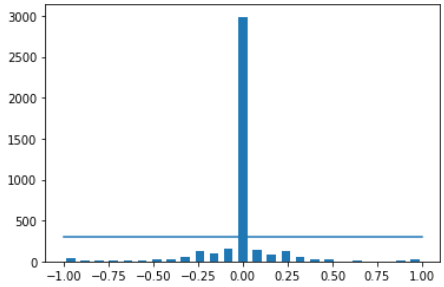

Figure 6: Histogram of the steering angle consisting of per 25 bins.

The pictures from the front camera must be preprocessed to make them more usable and handier during the learning process before using the training/validation datasets. The preprocessing stages were designed to enhance training outcomes and minimize computation. In the sequence of execution, below are the preprocessing stages that were implemented:

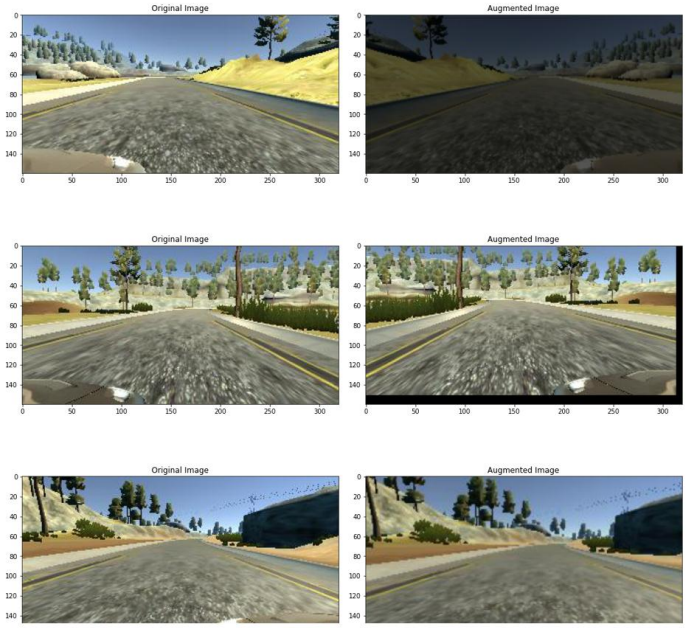

Figure 7: Examples of augmented images

1) Image normalization: In Keras, the "Lambda function" may normalize color photographs by simply implementing a min-max scaling. Instead, of 0–255, the RGB pixels' values are centered at zero and scaled from −1 to 1.

2) Region of interest concentration: By removing 70 and 25 pixels from each picture, we can concentrate on the area of interest and limit the number of inputs needed for our analysis (faster learning process). The photos were reduced to a 65 × 320 × 3 pixel size.

3) Augmentation: All photos (around the y-axis) were flipped, and the matching steering angles were reversed to double (enlarge) data. Accordingly, the source-1 and source-2 data contained 48,216 and 199,404 samples, respectively. For each CSV line record, six training samples can be generated (center, left, right, flipped-center, flipped-left, and flipped-right), changing brightness and shifting in images, as shown in Fig.7.

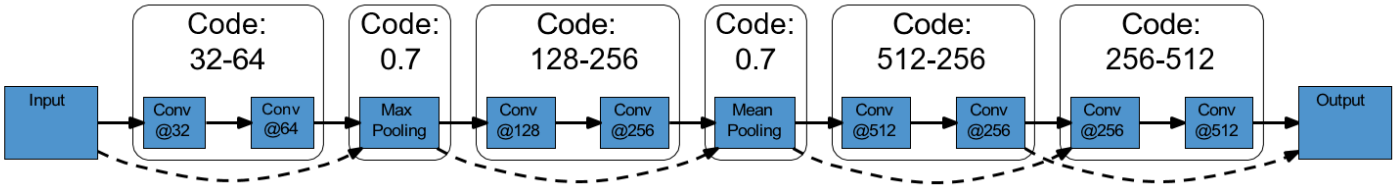

Figure 9 : the code representing the whole CNN which is the sequential string connection of codes representing each layer is "32-64-0.7-64-256-0.7-512-256-256-512".

4) Jittering: Images were "jittered" before being supplied to the AGBC-CNN and SOTA algorithms to decrease the model's potential to overfit the test track circumstances. The jittering consist of two adjustments the shadows are randomly adjusted, and the horizontal shift is random. For the shadow effect, we only require a random rectangular area of the picture that darkens and extends upwards from the left or right border. An image's horizon may be shifted up to 1/8 of the image's height using a perspective transform that begins at the horizon line (approximately 2/5 of the image's total height). Because the test track has a specific topology, a horizon shifts mimics this.

5) Flattening: Flattening the distribution of data gathered from the test track is substantially biased toward low and zero turning angles because of the track length. As a result, the neural network is biased toward straight route driving and is easily confused by sudden curves. Fig. 6 illustrates the distribution of the input data. A histogram of the turning angles was constructed to limit the incidence of low and zero-angle data points, and we determined the average number of samples per bin. Next, a "keep probability" for each sample of a bin was calculated. For bins with fewer samples than the average determined for each bin, the retain probability was set at 1.0. For all other bins, the keep probability equals the number of samples divided by the average number of samples per bin. Finally, random data points from the dataset were deleted with a rate of (1– "keep probability"). The resultant data distribution is shown in Fig. 8. Overall, the distribution is not uniform, but it is significantly closer to uniform at lower and zero turning angles. This strategy speeds up the training process since reduced-size data were used with more outstanding quality.

6) Decontaminating the dataset: It was determined that the model performed very severely on several data points and subsequently discovered that those data points had been mislabeled in numerous instances. The dataset with the lowest model performance was used to build a subroutine that displays frames from that dataset. We manually modified the steering angles to compensate for the mislabeled frames. Although this strategy is tiresome, it enhanced the training outcomes.

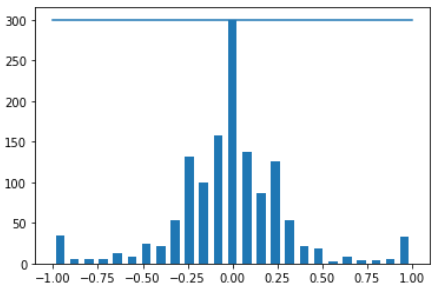

Figure 8: Histogram of the steering angle for 300 samples per 25 bins after removing zeros

7) Rearranging images: We shuffled the training data after each training epoch as a preventative measure against pattern memorizing and the resulting entrapment in local minima.

8) Data generator: It is necessary to use the generator function to load data into memory to smooth out the training process. It was

impossible to store all the data in the computer memory (or impractical). Thus, the data generator is created and loaded in memory for each patch (size = 128 pictures and angles). Keras's fit generator() function oversees the whole procedure.

4.3. Parameter Setting

The primary goal of this study is to develop AGBC-CNN to automatically identify new architectures. According to [7], crossover and mutation probability was set at 0.9 and 0.2, respectively, to train 350 epochs of SGD with a momentum of 0.9, learning rate decreasing by 0.1 at the 1st, 99th, and 199th epochs. Finally, we chose one SOTA algorithm and trained it for 300 epochs on the original training dataset. Classification accuracy was calculated and compared with peer rivals at the end of the test set. Furthermore, feature maps were limited to (32, 64, 128, 256, and 512) according to the parameters used by SOTA CNNs. We set the normalized chance of increasing the depth to 0.7 for four mutation processes while the other probabilities remained equal as shown in Fig. 9. Mutation probabilities may theoretically be set at any desired level if they are more significant than the probabilities of other mutations. The population and generation count were set to 20 in our method and those of its rivals, as shown in Fig. 10.

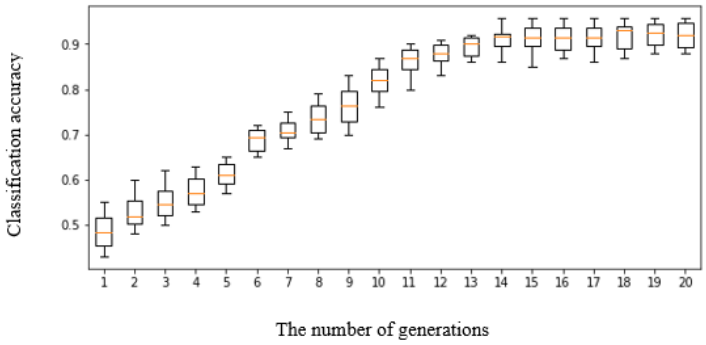

Figure 10: Best CNN architecture on collected data using the proposed algorithm AGBC-CNN

5. Overall Results

Table 1 shows the results of the comparison of the proposed algorithm with SOTA algorithms. It shows the category titles in the first column, with the SOTA algorithms organized into three

blocks depending on the categories. Additionally, the final column gives information on how much human help the associated CNN requires when finding the CNN architectures. Furthermore, the SOAT algorithms were listed in the second column. On the Udacity and collected datasets, the third and fourth columns relate to their classification accuracy, respectively. The best CNN found by CNN-GA + cutout in the previous generation follows from the population and was trained five times using our proposed technique. The cutout regularization of CNN-GA as "CNN-GA + cutout." On the training set, we first choose the best retrained CNN-GA architectures with the cutoff from Udacity and collected datasets, respectively, and then report the classification accuracy on the testing set.

The SGD with moment technique trains the AGBC-CNN model using the parameters provided in the previous section. The following observations were made throughout the training process. The network was trained using the Udacity dataset multiple times, but satisfactory results were not produced. The Udacity driving simulator gathered data by driving the automobile using a keyboard or joystick. Looping the automobile around, for example, "Track 1" a few times (more than 10) provides essential data for training. The transfer learning strategy was used in the first category, such as in ResNet (depth = 110), ResNet (depth = 1202), DenseNet-BC, and VGGNet. The final model was evaluated on "Track 1" in the simulator and provided acceptable results (no unsafe or sudden maneuvering).

Table 1: Comparisons of the proposed algorithm with the SOTA algorithms on the Udacity and collected datasets in terms of classification accuracy.

Algorithms	Challengers	Udacity accuracy %	Collected accuracy %	Manual assistance
MABC-CNN	ResNet (depth =110)	84.98	91.89	Needed
	ResNet (depth =1202)	83.65	90.06	Needed
	DenseNet-BC	89.53	93.42	Needed
	VGGNet	81.64	91.73	Needed
AGMABC-CNN	Block-QNN-S	90.04	93.67	Semi-Needed
AGBC-CNN	**CNN-GA + cutout**	**91.68**	**95.87**	**Not Needed**

Bold numbers specify best results.

From Table 1, the Udacity data accuracy of the proposed scheme (AGBC-CNN) is 91.68 % which is greater than the overall challengers of MABC-CNN (84.98%, 83.65%, 89.53%, and 81.64% respectively according to Table 1) and AGMABC-CNN (90.04%). Furthermore, in collected data accuracy, the proposed scheme attains higher accuracy (95.87 %) than AGMABC-CNN (93.67 %) and all challengers of MABC-CNN. From the above comparison, the proposed technique behaves very well in dealing with self-driving vehicles using behavior cloning.

6. Conclusions

In this work, we proposed an evolutionary algorithm to design CNNs for self-driving cars for optimization. The essential contribution was discovering the optimal CNN design; the number of convolutional layers, skip layer, and pooling layer parameters required and included in the suggestions. Testing and analysis using G-CNN optimization approaches have shown that the recognition rate rose across all case studies, resulting in a solid performance with the fewest possible inputs to the algorithm. The collected data had the best accuracy rate of 95.87%, for the Udacity database had accuracy of 91.68%. A comparison with other current SOTA algorithms on self-driving cars (Udacity and collected datasets) confirms that the optimization methodologies used in this study provide comparable results. We worked to improve the number of convolutional layers, thickness of those layers, number of skip layers, and parameters of the pooling layers. Optimizing CNN topologies using optimization techniques is critical, as shown by these findings. Other CNN hyper parameters, a new version of the GA, or various evolutionary computing approaches can be employed in a future study to develop more resilient CNN designs for usage in diverse self-driving vehicle datasets. However, we are investigating working with input photographs in real-time or video-capturing them instead of static images for future experiments.

References

[1] T. Elsken, J.-H. Metzen, F. Hutter, "Simple and efficient architecture Search for convolutional neural networks" arXiv:1711.04528 [cs, stat], 2017, Accessed: Mar. 21, 2022. [Online]. https://arxiv.org/abs/1711.04528

[2] Y. Sun, B. Xue, M. Zhang, G. G. Yen, "Completely automated CNN architecture design based on blocks" IEEE Trans. Neural Netw. Learn. Syst., **31**(4), 1242–1254, 2020. https://doi.org/10.1109/TNNLS.2019.2919608.

[3] W. Farag, Z. Saleh, "Behavior cloning for autonomous driving using convolutional neural networks" IEEE Xplore, 2018. https://ieeexplore.ieee.org/document/8855753 (accessed Mar. 21, 2022).

[4] H.-C. Shin et al., "Deep convolutional neural networks for computer-aided detection: CNN architectures, dataset characteristics and transfer learning" EEE Trans. Med. Imaging, **35**(5), 1285–1298, 2016. https://doi.org/10.1109/tmi.2016.2528162.

[5] D. Wang, "Hierarchical representation learning with connectionist models" rc.library.uta.edu, Mar. 01, 2018. https://rc.library.uta.edu/uta-ir/handle/10106/27354 (accessed Mar. 21, 2022).

[6] L. Zhao, W. Fang, "An efficient and flexible automatic search algorithm for convolution network architectures" IEEE Xplore, Jun. 01, 2021. https://ieeexplore.ieee.org/abstract/document/9504945 (accessed Mar. 21, 2022).

[7] Y. Sun, B. Xue, M. Zhang, G. G. Yen, J. Lv, "Automatically designing CNN architectures using the genetic algorithm for image classification" IEEE Trans. Cybern., **50**(9), 3840–3854, 2020. https://doi.org/10.1109/TCYB.2020.2983860.

[8] Y. Yang, Z. Gao, Y. Li, H. Wang, "A CNN identified by reinforcement learning-based optimization framework for EEG-based state evaluation" J. Neural Eng., **18**(4), 046059, 2021. https://doi.org/10.1088/1741-2552/abfa71.

[9] A.-C. Cheng et al., "Searching toward pareto-optimal device-aware neural architectures" Proceedings of the International Conference on Computer-Aided Design, Nov. 2018. https://doi.org/10.1145/3240765.3243494.

[10] A. A. Ahmed, S. M. Darwish, "A meta-heuristic automatic CNN architecture design approach based on ensemble learning" IEEE Access, **9**, 16975–16987, 2021. https://doi.org/10.1109/access.2021.3054117.

[11] A. Bakhshi, N. Noman, Z. Chen, M. Zamani, S. Chalup, "Fast automatic optimisation of CNN architectures for image classification using genetic algorithm" IEEE Xplore, 2019.

https://ieeexplore.ieee.org/abstract/document/8790197 (accessed Mar. 21, 2022).

[12] D. Chu and J. E. Rowe, "Crossover operators to control size growth in linear GP and variable length GAs" 2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence), Jun. 2008, https://doi.org/10.1109/cec.2008.4630819.

[13] H. Tian, S.-C. Chen, M.-L. Shyu, "Genetic Algorithm Based Deep Learning Model Selection for Visual Data Classification" 2019 IEEE 20th International Conference on Information Reuse and Integration for Data Science (IRI), 2019. https://doi.org/10.1109/iri.2019.00032.

[14] Jain, V., Chaudhry, A., Batra, M. and Gupta, P., 2020. Virtual Autonomous Vehicle Using Deep Reinforcement Learning. SSRN Electronic Journal.

[15] M. I. Pavel, S. Y. Tan, and A. Abdullah, "Vision-Based Autonomous Vehicle Systems Based on Deep Learning: A Systematic Literature Review," Applied Sciences, vol. 12, no. 14, p. 6831, Jul. 2022, doi: 10.3390/app12146831.

[16] M. R. Bachute and J. M. Subhedar, "Autonomous Driving Architectures: Insights of Machine Learning and Deep Learning Algorithms," Machine Learning with Applications, p. 100164, Sep. 2021, doi: 10.1016/j.mlwa.2021.100164.

[17] F. Johnson, A. Valderrama, C. Valle, B. Crawford, R. Soto, R. Nanculef, "Automating configuration of convolutional neural network hyperparameters using genetic algorithm" IEEE Access, **8**, 156139–156152, 2020. https://doi.org/10.1109/access.2020.3019245.

[18] T. Kozek, T. Roska, L. O. Chua, "Genetic algorithm for CNN template learning" IEEE Trans. Circuits Syst. I Fundam. Theory Appl., **40**(6), 392–402, 1993. https://doi.org/10.1109/81.238343.

[19] Md. J. Raihan, M. A. R. Ahad, A.-A. Nahid, "Automated Rehabilitation Exercise Assessment by Genetic Algorithm-optimized CNN" 2021 Joint 10th International Conference on Informatics, Electronics & Vision (ICIEV) and 2021 5th International Conference on Imaging, Vision & Pattern Recognition (icIVPR), 2021.https://doi.org/10.1109/icievicivpr52578.2021.9564240.

[20] R. Housley, "A 224-bit One-way Hash Function: SHA-224," www.rfc-editor.org, Sep. 2004, doi: 10.17487/RFC3874.

[21] S. Gibb, H. M. La, S. Louis, "A Genetic Algorithm for Convolutional Network Structure Optimization for Concrete Crack Detection" 2018 IEEE Congress on Evolutionary Computation (CEC), 2018. https://doi.org/10.1109/cec.2018.8477790.

[22] S. Lee, J. Kim, H. Kang, D.-Y. Kang, J. Park, "Genetic algorithm based deep learning neural network structure and hyperparameter optimization" Appl. Sci., **11**(2), 744, 2021. https://doi.org/10.3390/app11020744.

[23] P. K. Shukla, J. K. Sandhu, A. Ahirwar, D. Ghai, P. Maheshwary, P. K. Shukla, "Multiobjective genetic algorithm and convolutional neural network based COVID-19 identification in chest X-ray images" Math. Probl. Eng., **2021**, 1–9. 2021.https://doi.org/10.1155/2021/7804540.

[24] A. Garcia-Diaz and H. Bersini, "DensEMANN: Building A DenseNet From Scratch, Layer by Layer and Kernel by Kernel" 2021 International Joint Conference on Neural Networks (IJCNN), 2021, https://doi.org/10.1109/ijcnn52387.2021.9533783.

[25] A. A. Ahmed, S. M. S. Darwish, M. M. El-Sherbiny, "A novel automatic CNN architecture design approach based on genetic algorithm" Advances in Intelligent Systems and Computing, 473–482, 2019, https://doi.org/10.1007/978-3-030-31129-2_43.

[26] Udacity Sample Training Data, https://d17h27t6h515a5.cloudfront.net/topher/2016/December/584f6edd_data/data.zip

[27] T. DeVries and G. W. Taylor, "Improved Regularization of Convolutional Neural Networks with Cutout" arXiv:1708.04552 [cs], 2017, [Online]. Available: https://arxiv.org/abs/1708.04552.

[28] Udacity Simulator, https://github.com/udacity/self-driving-car-sim

Abbreviation and symbols

CNNs	**Convolutional Neural Networks**
AGBC	**Automated Genetic Behavior Cloning**
MABC-CNNs	**Manually Adjustment Behavior Cloning CNNs**
AGMABC-CNNs	**Automated Genetic Manually Adjustment Behavior Cloning CNNs**
AMA	**Automated + Manually Adjusted**
EAS	**Efficient Architecture Search Method**
GA	**Genetic Algorithm**
SGD	**Stochastic Gradient Descent**
GA	**Genetic Algorithm**
PI	**Proportional–Integral**