2022 International Conference on Emerging Trends in Computing and Engineering Applications (ETCEA) | 978-1-6654-7709-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/ETCEA57049.2022.10009778

Hyperparameters Optimization of Convolutional Neural Networks using Evolutionary Algorithms

Abeer Al-Hyari
Electrical Engineering Department
Al-Balqa Applied University
As Salt 19117, Jordan
abeer.hyari@bau.edu.jo

Mua'ad Abu-Faraj
Department of Computer Information Systems
The University of Jordan
Aqaba 77110, Jordan
m.abufaraj@ju.edu.jo

***Abstract*—This paper presents an approach for tuning hyperparameters in Convolutional Neural Networks (CNNs) by adopting evolutionary algorithms, i.e., Genetic Algorithms (GAs). CNNs include abounding hyperparameters that must be adjusted cautiously to accomplish the highest classification accuracy in case of image classification tasks or other related tasks. GAs are used to effectively go across the enormous search space of CNNs hyperparameters and choose the finest CNN architecture that operates well on a given task. CNN architecture contains a sequence of several convolutional layers followed by some fully-connected layers, as the number of layers increases, the number of corresponding hyperparameters will exponentially increase, examples of hyperparameters include but not limited to: number of convolutional filters in each convolutional layer, the number of nodes in the fully-connected layers, and the type of activation function. The proposed tuning framework was tested and evaluated using the SVHN dataset for digit classification of printed digits cropped from pictures of house number plates. The proposed approach can generate a CNN architecture with a validation accuracy of 92.31%.**

***Index Terms*—Hyperparameter, genetic algorithm, search space, evolutionary, tuning, CNNs**

I. Introduction

Convolutional Neural Networks (CNNs) have demonstrated their superiority in various applications; they can recognize useful, valuable, and important features due to their application of nonlinear operations in a deep architecture. This makes CNNs a great option, especially in vision-related applications, including medical diagnosis, object classification, object tracking, and object detection. CNNs are being used in many vital applications daily. For instance, Tarawneh et al. [1] introduced a security system for biometrics and finger knuckle print recognition systems that fuse the feature extracted from different layers of the VGG-19 network. The obtained results demonstrate that the extracted features are promising and can be used efficiently in the finger knuckle print recognition system.

Although CNNs have achieved tremendous success in vision-related applications, there is only so much architecture that can fit all applications. Furthermore, no guidance manual can help the designers choose the best architecture for a given task. There is a exponentially growing relationship between the number of layers in a CNN and their corresponding parameters. Thus, systematically exploring the huge search space of parameters for the best CNN architecture is an important step, referred to as hyperparameters optimization.
CNNs' Hyperparameters are predefined prior to the training process of the CNN model. The most prevalent ways of selecting the values of the hyperparameters are based either on the designer's experience or previous outstanding architectures. Alternatively, the CNN designer predefined and determined their values prior to the training. The definition of hyperparameter optimization for CNNs is to select the hyperparameters, including the number of layers, their order, connectivity between them, and the type of each layer to obtain outstanding performance for the assigned task.
Searching for the best CNN architecture for a certain task can take a long time and consume much computational power. Therefore, an approach for systematically searching for the best architecture in a reasonable time is a must. One of the most important searching algorithms is GA. GAs are metaheuristic search methods that were influenced by the natural selection process. GAs depend on biology-influenced operators, including crossover, mutation, and selection. GAs were used as a design exploration tool for Artificial Neural Networks (ANNs) Moreover, CNNs, due to their capabilities in exploring and searching the enormous solution space.
In this paper, we introduce an elementary yet powerful approach to automate the design process of CNNs using GAs. Section II gives a short survey of the most classical work in this domain. The methodology section III provides technical details about the new framework and the dataset used. Section IV summarizes the results achieved. Conclusions and suggestions to expand the proposed work are presented in Section V.

II. Related Work

Lately, much research has shed light on implementing various approaches for hyperparameter optimization of CNNs. One of the most promising approaches in this field is GAs [2]–[15]. This section explores the most famous and representative literature that adopts GAs for optimization purposes in CNNs.

After the breakthrough of CNNs in 2012, attempts were made to extend some existing traditional evolutionary-based approaches. For instance, (Hypercube)-based NEAT (HyperNEAT) [2], which targets the regular shallow ANNs, was extended to be applied to modern deep architectures of neural networks that involve higher number of layers as in DeepNEAT [3], every gene encodes a convolutional layer instead of a single neuron, and it contains a hyperparameter table prone to a mutation in the course of the evolution process. Miikkulainen et al. [3] developed Cooperative DeepNEAT (CoDeepNEAT) framework that involves two sub-populations: one is for blueprints, and the other is for the modules.

Another approach that is adopting GAs for the design exploration of CNNs is Multi-node Evolutionary Neural Networks for the Deep Learning (MENNDL) [4] where the design options are: convolutional filter size and number of them in every layer. However, the number of layers is shortened to three layers only. Additionally, the obtained networks are ready to deploy without further post-processing. Reducing the training time of CNNs was the main focus of Tirumala and his group in [5]; they managed to shorten the training time of CNN when tested on MNIST dataset after utilizing GAs for design optimization in replacement of traditional heuristic random initial architecture of CNN.

A set of innovative and intuitive mutation parameters are developed by Real et al. [6] to investigate their ability to generate ready-to-use CNNs with no additional human interventions.

Xie and Yuille [7] proposed a new encoding scheme that includes a binary fixed-length chromosome. Also, a set of basic evolutionary operations are executed to generate various CNN architectures and traverse the huge design space of CNNs in an efficient manner. They tested the transfer ability of the generated CNNs on basic datasets such as MNIST and CIFAR-10 to a more advanced dataset like ILSVRC2012. A Cartesian Genetic Programming (CGP) proved its potential in generating sophisticated CNN architectures that include highly functional residual-block in addition to traditional convolutional block [8]. The main drawback of this approach is the high computational cost.

Johnson et al. [9] presented a novel sequential crossover operator that allows more diversity in the early generations and evaluates them on a validation set with early stopping. The performance of the proposed approach was compared against the state-of-the-art methods on image classification benchmarks. In [10], a highly efficient network on the Intel Movidius Compute Stick with 5% accuracy improvement using genetic algorithms was implemented.

Liu et al. [11] accelerate the evolution process through transfer learning and experience-based greedy algorithm, and they test it on classical medical image analysis tasks. Xiao et al. [12] proposed a variable length genetic algorithm, and it can find a good model with time constraints. Aszemi and Dominic [13] compared grid and random search to the genetic algorithm in CNNs hyperparameter optimization, and they found that hybridizing the GA approach will enhance the obtained accuracy of the resulting optimized architecture. Three powerful evolutionary operators are used in [14] to accelerate the convergence process; the fine-tuned architectures were used to diagnose COVID-19 from chest x-ray images. On the other hand, GAs can be used to optimize the architecture of Recurrent Neural Networks (RNNs). In [15], AdacDeep is proposed to detect cyberattacks and overcome the common problems of RNNs learning, including vanishing and exploding gradients.

This work represents an extension to previous work by Al-Hyari and Areibi [16]; the improvements that were applied in this paper are: using a new and a more difficult dataset, which is Street View House Numbers (SVHN). Also, an early stopping technique is adopted in this work to avoid the long training time if the evolved CNN architecture is not improving over epochs. Furthermore, a new gene for the optimizer is introduced in the updated chromosome structure; the main aim is to examine the effect of various optimizers, especially in a more advanced dataset, and shorten the evolution time, which was the main issue in evolutionary approaches for hyperparameter optimization in CNNs.

III. Methodology

This section introduces the chromosome structure used This work is followed by the fitness function, dataset overview, and proposed framework.

A. Structure of the proposed chromosome

The Structure of the proposed chromosome is shown in Figure 1. The chromosome comprises two sections: the first section is for the corresponding parameters of a convolutional layer, while the other section is for the corresponding parameters of a fully-connected layer. The assumption of the proposed methods states that the convolutional layer in a generated CNN architecture can be up to six layers, and the fully-connected layers can be up to two layers, including the output layer. Furthermore, the activation function in the output layer is the softmax function since the generated CNN is used for the multi-class classification task. The proposed chromosome's length is fixed; it consists of six genes for the convolutional layer, five for the fully-connected layer, and one for the optimizer.

Therefore, each chromosome in the evolved population contains $(6 \times 6 + 5 \times 1 + 1 = 42)$ genes. Table I specifies the design alternatives and possible values for hyperparameters. The gene "Active" specifies whether a layer is existed in the phenotype of the corresponding chromosome or not. The gene "Batch normalization" determines the application of batch normalization or not. The application of batch normalization in vision or image related applications of CNNs is imperative due to performance enhancement and robustness that can be achieved by applying it.

Additionally, the dropout rate is considered because of its positive impact on preventing overfitting, a common issue in training CNNs. The gene "Dropout rate" suggested percentages are in the range of 10% up to 50%. The gene "Max

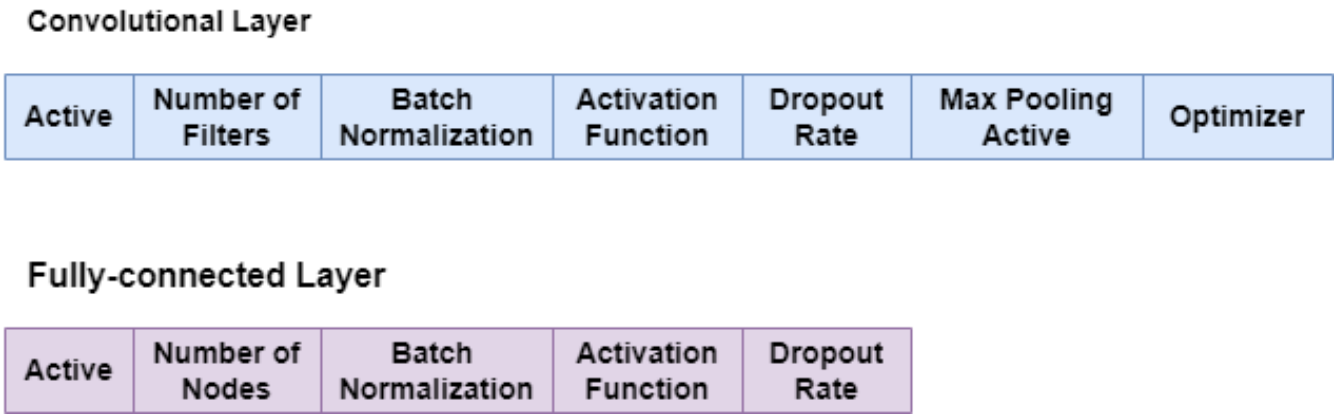

Fig. 1. Structure of the proposed chromosome

TABLE I
DESIGN ALTERNATIVES EMBEDDED IN THE PROPOSED CHROMOSOME STRUCTURE

Convolutional layer genes	
Gene	**Possible alternatives**
Active	{yes, no}
Convolutional filters count	{8, 16, 32, 64, 128, 256}
Batch normalization	{yes, no}
Type of activation function	{Relu, Sigmoid}
Dropout rate	{10%, 20%, 30%, 40%, 50%}
Max-pooling active	{yes, no}
Optimizer	{adam, rmsprop, adagrad, adadelta}
Fully-connected layer genes	
Gene	**Possible alternatives**
Active	{yes, no}
Nodes count	{16, 32, 64, 128, 256, 512, 1024}
Batch normalization	{yes, no}
Type of activation function	{Relu, Sigmoid}
Dropout rate	{10%, 20%, 30%, 40%, 50%}

pooling active" specifies whether max pooling is applied in a convolutional l layer or not; the main goal of pooling, in general, is to reduce the spatial dimensions of a feature map to half. Lastly, for the optimizer gene, there are four possible alternatives: adam, rmsprop, adagrad, and adadelta [17].

The second part of the proposed chromosome structure, which is in charge of encoding the fully-connected layer, consists of five genes. The gene "Active" determines the existence of its corresponding layer in the phenotype of the chromosome. The gene "Number of nodes" determines the count of nodes in a fully-connected layer; the possible range is between 16 to 1024. Further, the gene "Activation function" specifies the type of activation function, either sigmoid or Relu; lastly, the gene "Dropout rate" is similar to the one in the convolutional part of the chromosome. It is important to note that the design scenario in Table I was selected based on previous experience and common practice in CNNs design for vision-related applications.

B. Fitness Function

A single-objective fitness function is used to evaluate the evolved chromosomes across generations. The validation accuracy of each CNN architecture is regarded as a fitness score to each evolved chromosome. The objective of the fitness function is to select the fittest chromosomes based on their validation accuracy.

C. Dataset Used

Street View House Numbers (SVHN) [18] is a digit classification benchmark dataset that contains 600,000 32×32 RGB images of printed digits (from 0 to 9) cropped from pictures of house number plates. SVHN has three sets: training, testing sets, and an extra set with 530,000 images that are less difficult and can be used to help with the training process. Figure 2 shows some examples of the SVHN images; it can be seen that the digit of interest is centered, and multiple digits can be apparent in one image, which makes it harder for the recognition system.

Fig. 2. Examples of SVHN images [18]

D. Proposed framework

Figure 3 depicts the flowchart for the proposed hyperparameter tuning framework. In the beginning, the first population is initialized randomly so that every gene in the first population

takes a random value according to Table I. Next, the chromosomes undergo a series of genetic operations, such as mutation, crossover, and selection, to evolve from one generation to the next. The current population is modified, and a convergence test is applied. The operations of constructing a new CNN, evaluating its performance on the given dataset, and evolving the population's chromosomes are repeated until the closing generation. The The fittest chromosome and its corresponding CNN architecture are kept for further investigation and future usage.

IV. Results

A. Experimental setup

The presented approach for hyperparameters tuning in CNNs was developed in Python 3.7.14 and was run on Google Colab. Colab is an online cloud-based Jupyter notebook environment that allows users to train their machine learning and deep learning models on CPUs and GPUs with cloud storage capability. The corresponding CNN architectures of the evolved chromosomes were implemented in Keras, an open-source library of neural network components written in Python.

B. Parameters tuning

GAs involve abounding related parameters that require configuration. Several experiments were conducted using the SVHN dataset to find the best parameters combination, and the results are discussed in this subsection. Firstly, baseline settings were set to have an initial population for parameter tuning experiments of GAs and to prove the proposed approach's convergence. Table II provides the configuration used as the initial setting for the population at the inception. It is worth mentioning that values of the size of the population, generations count, and epochs were picked after trying many different values. However, for the remaining values in Table II, detailed comparison experiments for all the possible values are conducted, and results are reported in Table III.

Figure 4 depicts the improvement of average validation accuracy and the decline of validation loss over the various generations. Validation loss decreases towards the last generation; the minimum average value for validation loss is 0.668, which occurred in the nineteenth generation. Furthermore, the validation accuracy is improving when moving from one generation to the next; it reaches the maximum average value

TABLE II
Baseline design configurations

Parameter	Value/Method
Size of population	20
Generations count	20
Epochs	50 with early stopping
Rate of crossover	95%
Method of crossover	Single-point
Rate of Mutation	0.1%
Method of Selection	Random selection
Policy for replacement	Elitism

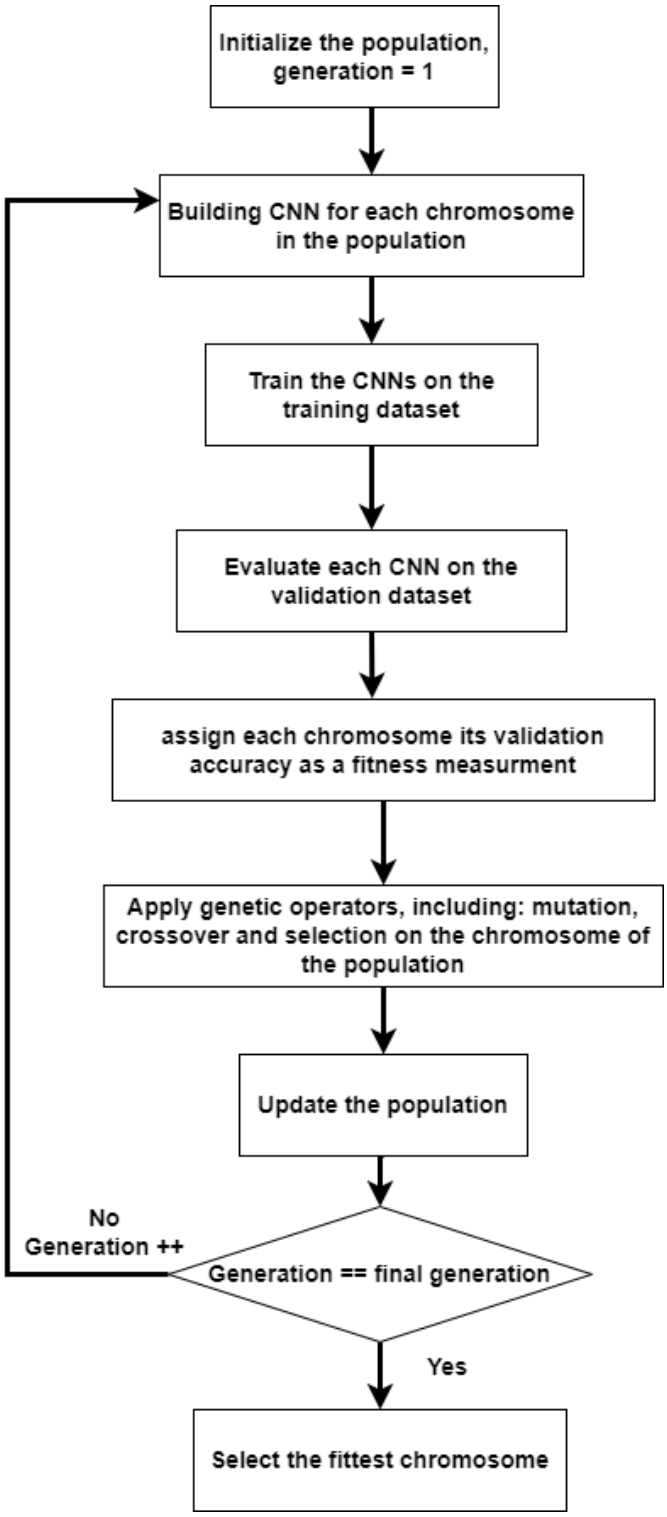

Fig. 3. Proposed method's Flowchart

of 0.812 in the nineteenth generation.

Fine-tuning for each parameter of the evolutionary process is tested, including crossover rate, mutation rate, crossover method, selection method, and replacement policy. The evolutionary-related parameters are initialized according to the values in Table II except the parameter under testing. Table

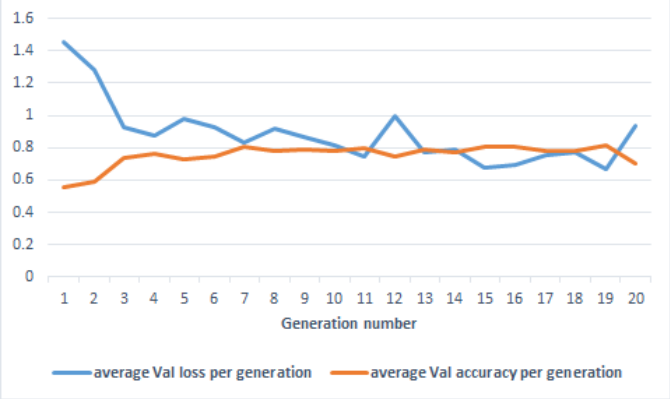

Fig. 4. Average validation accuracy and loss across various generations

III reports the results of various fine-tuning parameters related to the genetic evolution process and the average validation accuracy achieved for each case. Crossover is a crucial factor in GAs because it combines the genes of the parent's chromosomes to generate offspring. The rate of crossover specifies the probability of randomly mated parents. In the first experiment, the crossover rates tested are 80%, 90%, and 95%, and the highest achieved average validation accuracy is 91.4% with a crossover rate equals to 95%.

Regarding mutation, some gene values are randomly changed according to a predefined rate for mutation. This rate affects the diversity of the population because the proposed mutation operator mainly targets the "Active gene". Hence, it is responsible for adding or discarding a whole layer from the phenotype of the mutated chromosome. The rate of mutation values that were explored is 0.1%, 5%, and 10%. From Table III, it is apparent that the value of 0.1% for mutation rate is the best since it achieves the highest validation accuracy of 0.923.

TABLE III
FINE TUNING RESULTS

Parameter	Method/Value	Average validation accuracy
Crossover rate	80%	0.887
	90%	0.893
	95%	0.914
Mutation rate	0.1%	0.923
	5%	0.851
	10%	0.792
Crossover method	Single-point	0.902
	Two-point	0.846
Method of selection	Roulette wheel selection	0.878
	Random selection	0.923
	Tournament selection	0.897
Replacement policy	Elitism	0.919
	Generational	0.836

The crossover method was investigated for the third fine-tuning experiment; the crossover is responsible for producing a new generation of offspring by swapping the genes between the mated parents. There are two categories for the method of crossover: single-point and two-point. In a single-point crossover, a random gene on one parent is picked, and the genes beyond that specified gene are swapped between the mating parents. While in the two-point crossover method, two genes are randomly chosen, and the genes located in between them are exchanged from one mating parent to the other. Here, the Single-point crossover method tremendously improved the validation accuracy compared to the two-point crossover method, as apparent in Table III. In the next experiment, the selection method was investigated; the main objective of the selection operator was to select a portion of the population as candidate parents to produce offspring for the next generation. Usually, the fittest individuals are selected according to their corresponding fitness score calculated using the fitness function. The selection methods investigated here are tournament selection, roulette wheel selection, and random selection. Selecting an individuals in tournament selection includes running several tournaments between a predefined subset of chromosomes from the population pool; the chromosome with the highest fitness after each tournament is chosen. For roulette wheel selection, the offspring with the highest fitness value is most likely to be selected. Finally, in random selection, the parents are selected randomly from the existing population. Table III shows that the random selection method was capable of achieving the highest validation accuracy.

Lastly, a replacement policy is adopted to replace a subset of the old population to maintain the population size when evolving from one generation to the next. The "Elitism" policy replaces a subset of the current population with the best-evolved offspring. On the other hand, all the newly evolved offspring are replacing their parents in the generational replacement. Elitism policy attained a higher validation accuracy than a generational policy with an average validation accuracy of 91.9%.

TABLE IV
BEST EVOLVED ARCHITECTURE

Layer (type)	Output Shape	Param #
conv2d_712 (Conv2D)	(None, 32, 32, 16)	448
batch_normalization_608 (BatchNormalization)	(None, 32, 32, 16)	64
activation_852 (Activation)	(None, 32, 32, 16)	0
dropout_852 (Dropout)	(None, 32, 32, 16)	0
max_pooling2d_240 (MaxPooling2D)	(None, 16, 16, 16)	0
conv2d_713 (Conv2D)	(None, 16, 16, 32)	4640
activation_853 (Activation)	(None, 16, 16, 32)	0
dropout_853 (Dropout)	(None, 16, 16, 32)	0
conv2d_714 (Conv2D)	(None, 16, 16, 256)	73984
activation_854 (Activation)	(None, 16, 16, 256)	0
dropout_854 (Dropout)	(None, 16, 16, 256)	0
max_pooling2d_241 (MaxPooling2D)	(None, 8, 8, 256)	0
conv2d_715 (Conv2D)	(None, 8, 8, 128)	295040
activation_855 (Activation)	(None, 8, 8, 128)	0
dropout_855 (Dropout)	(None, 8, 8, 128)	0
max_pooling2d_242 (MaxPooling2D)	(None, 4, 4, 128)	0
flatten_224 (Flatten)	(None, 2048)	0
dense_364 (Dense)	(None, 10)	20490
Total params: 394,666		
Trainable params: 394,634		
Non-trainable params: 32		

Table IV lists the best-obtained architecture that was evolved using the proposed approach, this architecture achieves a validation accuracy of (0.9231), and it evolved in the 12^{th} generation. The evolved architecture consists of four convolutional layers and one fully-connected layer, the output layer. Only the first convolutional layer has batch normalization, while all the convolutional layers contain max pooling functionality except the second layer.

V. Conclusions

In this paper, we proposed a hyperparameters tuning framework for CNNs using GAs, which are promising and achieve optimal validation accuracy on the SVHN dataset. The best-achieved validation accuracy is 0.9231, while the average validation accuracy across 20 generations and a population size of 20 was (0.754) with a standard deviation of (0.1897). Many suggestions can further boost this proposed framework, including but not limited to: (a) adjusting the proposed chromosome structure to increase the gene number so that further hyperparameters of the CNNs can be automatically tuned, such as: stride, pooling type, filter size, etc., (b) estimating the diversity of the population across various generations, (c) considering adopting a variable chromosome length, (d) benchmarking the best evolved CNN architecture with the state-of-the-art architectures, (e) examining the proposed framework against more sophisticated, such as CIFAR-10 and CIFAR-100, and (f) compare the proposed evolutionary approach with other nature-inspired optimization approaches, including particle swarm and ant colony optimization.

References

[1] A. S. Tarawneh, A. B. Hassanat, E. Alkafaween, B. Sarayrah, S. Mnasri, G. A. Altarawneh, M. Alrashidi, M. Alghamdi, and A. Almuhaimeed, "Deepknuckle: Deep learning for finger knuckle print recognition," *Electronics*, vol. 11, no. 4, p. 513, 2022.

[2] K. O. Stanley, D. B. D'Ambrosio, and J. Gauci, "A hypercube-based encoding for evolving large-scale neural networks," *Artificial life*, vol. 15, no. 2, pp. 185–212, 2009.

[3] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy *et al.*, "Evolving deep neural networks," in *Artificial intelligence in the age of neural networks and brain computing*. Elsevier, 2019, pp. 293–312.

[4] S. R. Young, D. C. Rose, T. P. Karnowski, S.-H. Lim, and R. M. Patton, "Optimizing deep learning hyper-parameters through an evolutionary algorithm," in *Proceedings of the workshop on machine learning in high-performance computing environments*, 2015, pp. 1–5.

[5] S. S. Tirumala, S. Ali, and C. P. Ramesh, "Evolving deep neural networks: A new prospect," in *2016 12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*. IEEE, 2016, pp. 69–74.

[6] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin, "Large-scale evolution of image classifiers," in *International Conference on Machine Learning*. PMLR, 2017, pp. 2902–2911.

[7] L. Xie and A. Yuille, "Genetic cnn," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 1379–1388.

[8] M. Suganuma, S. Shirakawa, and T. Nagao, "A genetic programming approach to designing convolutional neural network architectures," in *Proceedings of the genetic and evolutionary computation conference*, 2017, pp. 497–504.

[9] F. Johnson, A. Valderrama, C. Valle, B. Crawford, R. Soto, and R. Ñanculef, "Automating Configuration of Convolutional Neural Network Hyperparameters Using Genetic Algorithm," *IEEE Access*, vol. 8, pp. 156 139–156 152, 2020, conference Name: IEEE Access.

[10] A. Reiling, W. Mitchell, S. Westberg, E. Balster, and T. Taha, "CNN Optimization with a Genetic Algorithm," in *2019 IEEE National Aerospace and Electronics Conference (NAECON)*, Jul. 2019, pp. 340–344, iSSN: 2379-2027.

[11] P. Liu, M. D. El Basha, Y. Li, Y. Xiao, P. C. Sanelli, and R. Fang, "Deep Evolutionary Networks with Expedited Genetic Algorithms for Medical Image Denoising," *Medical Image Analysis*, vol. 54, pp. 306–315, May 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1361841518307734

[12] X. Xiao, M. Yan, S. Basodi, C. Ji, and Y. Pan, "Efficient Hyperparameter Optimization in Deep Learning Using a Variable Length Genetic Algorithm," Jun. 2020, arXiv:2006.12703 [cs]. [Online]. Available: http://arxiv.org/abs/2006.12703

[13] N. M. Aszemi and P. Dominic, "Hyperparameter Optimization in Convolutional Neural Network using Genetic Algorithms," *International Journal of Advanced Computer Science and Applications*, vol. 10, no. 6, 2019.

[14] S. M. J. Jalali, M. Ahmadian, S. Ahmadian, R. Hedjam, A. Khosravi, and S. Nahavandi, "X-ray image based COVID-19 detection using evolutionary deep learning approach," *Expert Systems with Applications*, vol. 201, p. 116942, Sep. 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417422003724

[15] A. E. Ibor, F. A. Oladeji, O. B. Okunoye, and C. O. Uwadia, "Novel adaptive cyberattack prediction model using an enhanced genetic algorithm and deep learning (AdacDeep)," *Information Security Journal: A Global Perspective*, vol. 31, no. 1, pp. 105–124, Jan. 2022. [Online]. Available: https://www.tandfonline.com/doi/full/10.1080/19393555.2021.1883777

[16] A. Al-Hyari and S. Areibi, "Design space exploration of convolutional neural networks based on evolutionary algorithms," *Journal of Computational Vision and Imaging Systems*, vol. 3, no. 1, 2017.

[17] S. Vani and T. M. Rao, "An experimental approach towards the performance assessment of various optimizers on convolutional neural network," in *2019 3rd international conference on trends in electronics and informatics (ICOEI)*. IEEE, 2019, pp. 331–336.

[18] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.