IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 17 November 2023, accepted 13 January 2024, date of publication 18 January 2024, date of current version 25 January 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3355804

RESEARCH ARTICLE

Improved Search in Neuroevolution Using a Neural Architecture Classifier With the CNN Architecture Encoding as Feature Vector

JHON I. PILATAXI[1,2], (Graduate Student Member, IEEE), JORGE E. ZAMBRANO[1,2], (Graduate Student Member, IEEE), CLAUDIO A. PEREZ[1,2], (Senior Member, IEEE), AND KEVIN W. BOWYER[3], (Fellow, IEEE)

[1]Department of Electrical Engineering, and Advanced Mining Technology Center, Universidad de Chile, Santiago 8370451, Chile
[2]IMPACT, Center of Interventional Medicine for Precision and Advanced Cellular Therapy, Santiago 1025000, Chile
[3]Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN 46556, USA

Corresponding author: Claudio A. Perez (clperez@ing.uchile.cl)

This work was supported in part by Agencia Nacional de Investigacion y Desarrollo (ANID) under Grant FONDECYT 1231675; in part by the Basal funding for Scientific and Technological Center of Excellence, under Project AFB220002 and Project IMPACT #FB210024; and in part by the Department of Electrical Engineering, Universidad de Chile.

ABSTRACT Designing Convolutional Neural Networks (CNNs) for a specific task requires not only Deep Learning expertise but also knowledge of the problem. The goal of Neuroevolution is to find CNN architectures automatically through evolution. The search time, however, is a critical problem in Neuroevolution since multiple CNNs must be trained in the evolutionary process. In this work, we propose a Neural Architecture Classifier (NAC) to avoid training architectures that would not have good performance, based on knowledge of previously trained architectures. The NAC evaluates each CNN using the CNN architecture encoding as the input. A genetic algorithm (GA) is used for evolution, and a search space with few restrictions; hence, CNN architectures can have any width, depth or shape as well as the suitable number of skip connections. We applied this methodology to solving pattern recognition problems on six well-known datasets: five subsets of the MNIST-Variations (MNIST-V) dataset and the CIFAR-10 dataset. Experiments demonstrated that integrating the NAC in the GA reduces the search time by up to 44% compared to conventional GAs. Additionally, the architectures found by evolution with our classifier achieved a better performance (4.6% on average) than those found by traditional evolution. Our results improved the state-of-the-art by 5.2% (0.33% error reduction) on the most difficult dataset of the MNIST-V (MRDBI), and an average of 4% (0.24% error reduction) on the MNIST-V datasets. Additionally, our results on CIFAR-10 are close to those of the state-of-the-art but with significantly reduced search time. The application of a classifier can be relevant and useful not only for genetic algorithms, but also for other evolutionary algorithms.

INDEX TERMS Convolutional neural networks, genetic algorithms, neuroevolution, neural architecture classifier, search time.

I. INTRODUCTION

Convolutional Neural Networks (CNNs) consist of two main parts: the feature extractor or backbone and the classifier [1], [2], [3], [4]. The backbone extracts the most important features from an input image hierarchically, while the classifier corresponds to a fully connected neural network (NN) [4], [5], [6], [7]. This general design was inspired by the mammalian visual system since some of the main blocks can be identified with its early and intermediate processing stages [8], [9], [10], [11].

Designing an optimum architecture for a specific task is still an open problem, even for experienced researchers in Deep Learning (DL) [12], [13], [14]. In 1979, Fukushima

The associate editor coordinating the review of this manuscript and approving it for publication was Christian Pilato.

introduced the Neocognitron model [9]. This CNN was capable of solving pattern recognition problems including translation invariance [9], [15], [16], [17]. LeCun designed LeNet [10] to solve the handwritten digit recognition problem in 1998. In 2012, Krizhevsky et al. introduced AlexNet [18], which was the first CNN that won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), and significantly improved the state-of-the-art (SOTA) compared to traditional machine learning and computer vision methods [19]. In 2014, Simonyan and Zisserman introduced VGG [20]. This network used receptive fields smaller than AlexNet, and the small-size filters allowed VGG to increase the number of layers. These early approaches consisted of a stack of convolutional and pooling layers, followed by fully connected layers [2], [21]. However, modern architectures, such as GoogLeNet [22], ResNet [23], DenseNet [24], and PyramidNet [25], are not only deeper, but are also more efficient than the early models. More details about these architectures are provided in section II-A.

As stated previously, the mammalian visual cortex evolved over millions of years to perform pattern recognition tasks with remarkable precision [26], [27], [28], [29]. The evolutionary concept has been applied to CNNs to design better architectures that best fit a particular problem. Neuroevolution embodies the concept of evolving neural networks [30], and it is part of the Neural Architecture Search (NAS) [31], which consists of searching CNN and NN architectures automatically using various algorithms, including Reinforcement Learning [32], [33], [34], [35], [36], Gradient Descent [37], [38], and Evolutionary Computation (EC) [39], [40], [41], [42]. Furthermore, Neuroevolution can find better CNN architectures for a specific task efficiently, allowing researchers without DL experience to find optimum CNN architectures for their fields of study [14], [43]. Neuroevolution has been applied to image classification [14], [31], [43], [44], [45], [46], [47], [48], [49], [50], [51], medical imaging segmentation [52], [53], and human activity recognition [54], among other problems [55], [56], [57], [58], [59], [60], [61].

However, the main bottleneck in Neuroevolution is search time because it is a population-based method, and multiple CNNs must be trained to converge to a solution. In order to reduce the CNN training time, and consequently the search time, various studies have proposed fitness approximation techniques in the following ways: training for a few epochs [45], [46], [47], [48], using a subsampled dataset [62], working with low-resolution images [63], evolving cells or blocks instead of the complete architecture [64], evolving architectures with a small dataset and then transferring to a larger dataset [65], or by using a combination of them [66].

Recently, new approaches to fitness acceleration were proposed, such as 2-level Genetic Algorithm (2LGA) [14] and performance predictors [67], [68], [69], [70], [71], [72], [73], [74], [75], [76]. In the 2LGA [14], two levels were proposed: in the first one, the whole population is trained for a few epochs, whereas at the second level, the best individuals are trained for more epochs after a certain number of generations. Genetic operations were performed at both levels, and the offspring of the second level replaced a part of the first-level population. Performance predictors are designed to predict the fitness value of CNNs using a regression model based on the partial learning curve [67], [68] or architecture encoding information (end-to-end performance predictor) [69], [70], [71], [72]. Other performance predictor methods are shared-weights based, in which the weights are not directly trained but are updated using trained weights from other architectures [73], [74], [75], [76].

The search space is a crucial part of Neuroevolution because it defines all possible architectures that can be found. Some authors have used fixed-length encoding [47], [62], [77], in which the architecture depth is fixed and depends on the designer's criterion. However, it is impossible to know the suitable depth of the CNN needed to solve a new task. In contrast, other authors have proposed variable-length encoding, but nevertheless, the genetic operations must be redefined according to the encoding used. For instance, in [78] and [79] only mutation operations were performed; in [14], [43], and [80] crossover and mutation were defined for cells, blocks, and layer-based variable-length encoding, respectively. Layer-based encoding space considers a layer (convolutional, fully connected, activation, or batch normalization, among others) as a basic unit, and it only considers connections among adjacent layers (plain CNNs) [62], [80]. Block-based encoding spaces combine different layers in a block as a basic unit; traditional blocks are residual blocks [23], dense blocks [24], or inception blocks [22]. These encodings also require fewer parameters than layer-based encoding [43], [81]. Cell-based encoding is a special case of block-based encoding, in which all the blocks in the CNN are the same [14], [49], [82]. Although cell-based encoding reduces the number of parameters for encoding deep networks significantly, stacking the same block several times is not necessarily the best solution [82], [83]. Additionally, some studies have included training hyperparameters, such as batch size, optimizer, and learning rate, in their proposed encoding [14], [62].

In this work, we propose improving the search in Neuroevolution by using a Neural Architecture Classifier (NAC) to avoid training the entire population, training only those individuals that are classified with potential for good performance by the NAC. The GA progresses to the next generation reducing the number of CNNs trained, and thus reducing the search time. The NAC is a binary classifier trained using previously evaluated CNNs. We integrated the NAC into a genetic algorithm (GA) to classify each individual of the population as either 1 (indicating a potential for good classification performance, and thus, that individual is trained), or 0 (indicating a potential for poor

classification performance, and therefore, that individual is not trained).

The main contributions of this paper are the following: (1) Development of a NAC based on the CNN architecture encoding, and integration of the NAC in a GA to avoid training the entire population. Thus, only those individuals that are classified by the NAC with potential for good performance are trained. In this way, the NAC reduces the search time in Neuroevolution, (2) Introducing a new block-based encoding to represent a complete CNN, and (3) Applying the proposed method to six datasets of which the MNIST-Variations (MNIST-V) and CIFAR-10 datasets showed improvements relative to results of the SOTA. Also, data augmentation and anti-aliasing (AA) filters are considered in the results.

The rest of this paper is organized as follows: Section II contains a background on CNNs and Neuroevolution and summarizes related methods. The proposed method is provided in Section III. In Section IV the experiments are described. In Section V, the results and discussion are presented. Finally, Section VI contains the conclusions and future work.

II. BACKGROUND

In this section the fundamentals of CNNs, and the main related work is presented, which helps to place our proposed method in perspective.

A. CONVOLUTIONAL NEURAL NETWORKS

The results achieved with early CNN models had shown that increasing the depth of the CNNs improved performance. Nevertheless, it was not feasible to train deep networks because of the vanishing gradient (VG) problem [84], which refers to the gradient becoming insignificant during backpropagation.

In 2015, Srivastava et al.. developed highway networks, and proved experimentally that connections between non-adjacent layers (called skip connections) facilitated the training process of deep neural networks [85]. In the same year, He et al. used skip connections to develop residual blocks that connected the input and output of a block by addition [23]. Later, Hung et al. introduced dense blocks, in which the output of each layer is connected to every subsequent layer in the block by concatenation [24].

Recently, Zhang proposed combining a low-pass filter with common down-sampling methods (max-pooling, average-pooling, and stride-convolution) to create shift-invariant CNNs [86]. This proposed approach increased the accuracy of the ImageNet and CIFAR-10 classifications across several commonly used CNNs. Furthermore, he observed that using these filters provided better generalization [86].

Designing handcrafted-CNN architectures requires enormous effort, and one cannot expect to obtain optimal performance by applying the same architecture to different tasks. To design a CNN architecture for a specific task requires not only expertise in DL, but also knowledge of the problem to be solved.

B. NEUROEVOLUTION METHODS

As mentioned previously, the goal of Neuroevolution is to find CNN or NN architectures through EC methods, such as Particle Swarm Optimization (PSO), Grammatical Evolution (GE), Differential Evolution (DE), and Genetic Algorithms (GAs). Examples of the PSO approach are IPPSO [51] and psoCNN [48]; DECNN uses DE to evolve CNN hyperparameters [46]; AE-CNN [43], 2LGA [14], and evoCNN [45] are examples of GAs used to optimize CNN architectures; E-CNN uses GA and GE to evolve CNN Architectures [62]; HGAPSO is a hybrid method that uses PSO to optimize convolutional hyperparameters, and a GA to optimize the connections among convolutional layers in block-based encoding [87].

The first studies in Neuroevolution used GAs with architectures and weights encoded to evolve simultaneously [77], [88], [89]. However, this encoding did not support deep architectures due to the number of parameters that needed adjustment. To overcome this limitation, newer approaches optimize either the weights or the architecture. In the first case, the architecture remains fixed [90], [91]. In the second case, the encoding includes only architecture hyperparameters or both architecture and training hyperparameters, and the weights are adjusted through training using backpropagation [39], [40], [41], [42].

An early study with LS-Evolution achieved promising performance by evolving CNNs but required 2,730 GPU-days [79]. Other EC methods were proposed, such as GeNet [92], EIGEN [93], and CGP-CNN [94], which reduced the employed computational resources significantly by compromising the classification accuracy. AmoebaNet [64] was the first EC method that achieved the SOTA on the ImageNet classification problem, employing 450 GPUs (Graphics Processing Units) in 7 days. AE-CNN accelerated the process and needed only 27 GPU-days, but the classification accuracy was compromised [43].

Recently, Sun et al. modified the AE-CNN by combining it with an end-to-end performance predictor (E2EPP) [71]. AE-CNN+E2EPP took only 8 GPU-days, but the classification error increased compared to the original AE-CNN. The time consumed by AE-CNN+E2EPP is caused mainly by collecting the dataset to train the predictor model. In E2EPP methods, it is impossible to collect more data to train the predictor model because CNNs are not trained in the evolution. The main differences with our proposed method are: (1) we take advantage of the evolved-trained networks to adjust our NAC, and (2) we optimize even the connections inside blocks instead of using DenseNet or ResNet blocks as in AE-CNN.

To reduce the search time, Wang et al. proposed EffPNet [49] using a subset of the training set in the evolutionary process and a surrogate model to search the

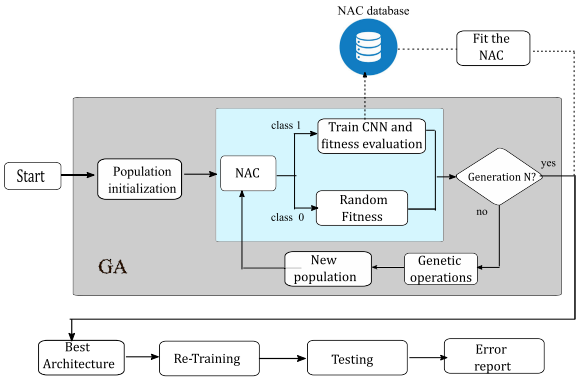

FIGURE 1. Algorithm overview. The light blue rectangle depicts our proposal for reducing the search time within the GA by means of the NAC. The trained networks and their best validation accuracy are stored to fit the NAC for further generations and other experiments.

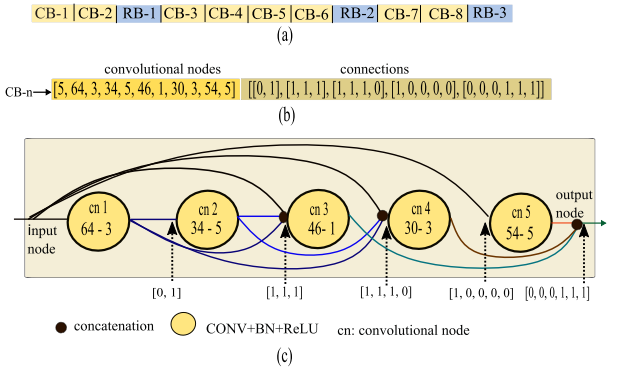

FIGURE 2. Example of architecture encoding and its interpretation. (a) an architecture encoding with 8 CBs and 3 RBs, (b) a CB with 5 convolutional nodes encoding the following: the first array provides the number of nodes in the block, and the number of filters and kernel size for each convolution (CONV); the binary vectors represent the node interconnection in the block, (c) A CB-n encoding interpretation.

optimal cell encoding. The surrogate model was a Support Vector Machine (SVM), and its objective was to identify whether or not a generated architecture should be trained for more epochs. The cell encoding, accuracy, and loss training history of ten epochs were used as input features to the SVM, which was adjusted at each generation [49]. The main differences with our proposed method are: (1) we encode the complete architecture instead of a cell to reduce the human bias, and (2) we apply a NAC to identify whether or not an architecture should be trained. Our NAC is trained with the previously trained CNNs, using only the architecture encoding. Li et al. proposed SHEDA-CNN [95] to optimize the hyperparameters of a 20-layer ResNet architecture. This method used a performance predictor. If the predicted fitness of a CNN was greater than the average fitness of the previously trained architectures, the CNN was trained.

III. METHODOLOGY

A. ALGORITHM OVERVIEW

Figure 1 shows the framework of our proposed method. We integrated the NAC into the GA. The NAC evaluates each individual of the GA population, thus avoiding training those individuals having been classified as low performing. Therefore, in each generation, only those individuals identified as promising by the NAC would be trained. Consequently, the GA progresses to the next generation training fewer individuals, and thus reducing search time.

B. SEARCH SPACE AND CNN ARCHITECTURE ENCODING

According to the methods previously used, there are various ways to define the encoding of the CNN architecture. Encoding can be applied using variable or fixed vector length, and it can encode either the complete network or just a cell that is stacked several times to build the final network. In our study, we use the variable-length encoding of the complete network in a search space with few restrictions, since architectures can have any size, shape, or depth. The encoding is based on blocks. There are two possible block types: a convolutional block (CB), and a reduction block (RB), but each block can be different because of the number of nodes in each one, the hyperparameters of each node, or the skip connections in each block.

CBs have two variable-length arrays. The first one encodes the number of convolutional nodes (k), the kernel size, and the feature map size of each node, while the second one encodes the connections inside the block. The connections inside the blocks are expressed as a list of k binary arrays, as illustrated by Figure 2(b). The first node input is the input of the block, and therefore, it is not necessary to represent that connection. The $l - th$ binary array corresponds to the input to the $l + 1 - th$ node. For example, the first binary array represents the input for the second node, and the last array corresponds to the output node. In a binary array, the value 1 represents a connection from a specific node. The first value represents a connection with the input of the block, and the following values represent a connection from the output of the following convolutional nodes, as is illustrated in Figure 2(c). The feature maps of different nodes are concatenated as in DenseNet [24]. Convolutional nodes are composed of convolutional and batch normalization layers [96], as well as the ReLU (Rectified Linear Unit) activation function. RBs are encoded by 1 and 0 for max and average pooling, respectively. RBs are used to sample the feature maps based on Zhang's proposal [86], which consists of a pooling layer followed by an AA filter. The number of convolutional blocks in the initial population is limited to six, but it can increase during the evolution. In addition, the number of reduction blocks is limited by the size of the input images. Table 1 presents information on the proposed search space.

Our proposed encoding represents the CNN backbone, and a fully connected layer is added to obtain the whole CNN architecture as in [14], [43], [44], [47], and [49].

TABLE 1. Parameter domain in the proposed search space.

Description	Value Domain
Convolutional kernel size	$\epsilon\{1, 3, 5\}$
Feature map size	ϵ [16-96]
Pooling type	$\epsilon\{max, average\}$
Number of convolutional blocks (CBs)	ϵ [3-12]
Number of convolutional nodes per block	ϵ [1-7] on MNIST-V ϵ [1-9] on CIFAR-10
Number of reduction blocks (RBs)	ϵ [1-3] on MNIST-V ϵ [1-4] on CIFAR-10

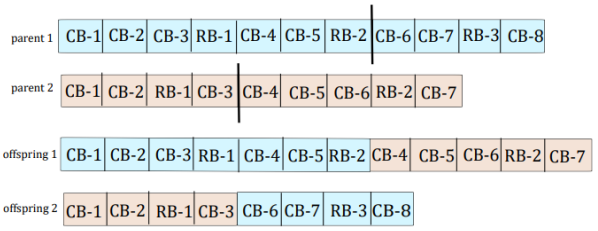

FIGURE 3. Crossover operation example in which the parents are divided into two parts, and their offspring have one part from each parent.

C. GENETIC OPERATIONS

To generate a new population during the evolution process, three operations are performed: crossover, mutation, and selection [97], [98], [99]. The crossover generates offspring from two selected parents. Then, mutation allows exploring the search space and modifying the offspring individuals. The selection finally defines the population for the next generation from the current population and its offspring.

1) CROSSOVER

The offspring generated are expected to have greater fitness than their parents [100]. However, selecting only the best individuals as parents could reduce the diversity in the population, and therefore, the best performance of the population would not be achieved because of a premature convergence of the GA [101]. For that reason, we used a binary tournament selection to choose parents [102]. The binary tournament selects two individuals randomly from the population, and the one with better fitness is selected as a parent [100].

In our proposed method, we use a one-point crossover operation as in [43]. Each parent is divided into two parts, and two offspring are generated by combining the first part of one parent with the second part of the other. The combination has a limitation: the number of RBs cannot be greater than the maximum (see Table 1). Therefore, if an offspring does not meet this constraint, another division point is selected. Figure 3 illustrates an example of a crossover operation.

2) MUTATION

We use four possible mutation types: adding, deleting, modifying, and mixing. The first three mutations are similar to those used in [43], and we added the last mutation to optimize the pooling layer position in CNNs. The added

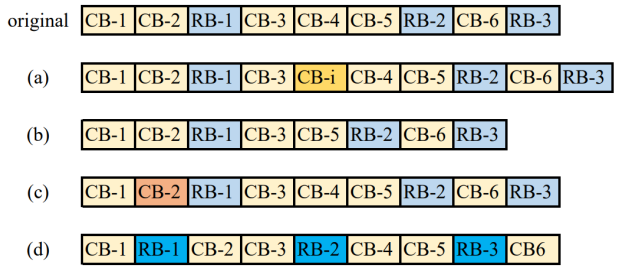

FIGURE 4. Examples of each mutation operation. (a) adding CB-i in 5^{th} position, (b) deleting CB-4, (c) modifying the CB-2, and (d) changing the position of RBs.

mutation increases the architecture length by adding a new block in a random position. The deleting mutation reduces the architecture length by removing a block in a random position. The modifying mutation changes the parameters of a random block. Finally, the mixing mutation changes the positions of all RB blocks randomly. Figure 4 shows an example of each mutation type.

It is worth mentioning that just one mutation type is selected randomly. It was also verified that the number of CBs was within the domain (see Table 1), and if it exceeded the limit, the excess number of blocks was eliminated.

3) SELECTION

Given the current population P_t and the generated offspring population Q_t, I individuals are selected by the binary-tournament selection, generating the population for the next generation P_{t+1}. After that, it is necessary to check if the two best individuals (elitism) are in P_{t+1}. Otherwise, the elitist individuals replace the two randomly selected individuals of P_{t+1}.

D. NEURAL ARCHITECTURE CLASSIFIER

An SVM was used as a neural architecture classifier to decide if a CNN should be trained or not. An SVM was used mainly for two reasons: first, the SVM does not require a large dataset to achieve good performance compared to other machine learning approaches, such as neural networks (NN); and second, an SVM finds the optimal separation hyperplane between the classes, even for non-linear problems [103], [104].

The architecture encoding, accuracy, and the loss training history of ten epochs were used as features to train SVM models in [49]. By contrast, we propose using only the CNN encoding as the input for the classifier. In this way, the classifier will learn to distinguish between configurations in the architectures generated by the GA that have the potential for good performance, and those that have the potential for poor performance. The NAC (SVM) classifies each generated architecture into two categories by the GA: 1 (indicating potential for good performance), or 0 (indicating potential for poor performance). Architectures classified as 1 by the SVM will be trained, while those classified as 0 are not trained.

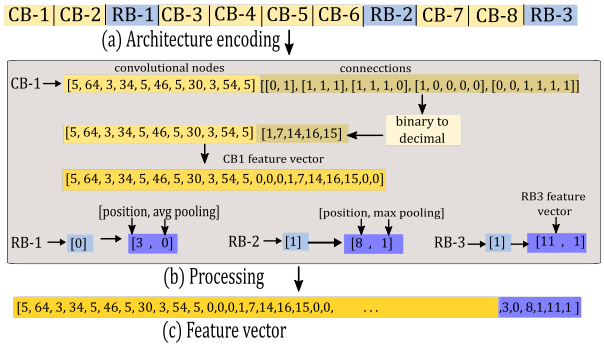

FIGURE 5. Transformation from architecture encoding to feature vector. (a) An example of architecture encoding with 8 CBs and 3 RBs. (b)Processing: in CBs, the binary arrays, which represent the interconnections in the block, were transformed to decimals. For RBs, the block position was added. (c) The final feature vector concatenates all the CBs before the RB feature vectors.

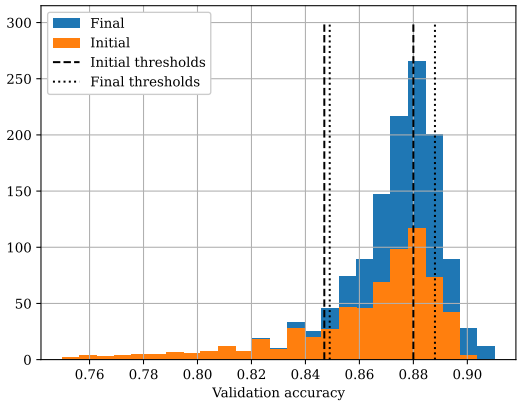

FIGURE 6. Validation accuracy distribution for initial (orange), and final (blue), data used to train the NAC in MRDBI. The dashed and dotted lines correspond to initial and final thresholds. For better visualization, the figure is shown in the range 0.75 and 0.92, but there are CNNs with lower performance.

1) FEATURE VECTOR

Due to the variable-length encoding, the feature vector is normalized by filling those architectures with a smaller number of convolutions per block with zeros, or a smaller number of blocks than the limits defined in Table 1. The original encoding is transformed into a feature vector as follows:

First, the list of binary arrays that represent inside-block connections is transformed to decimal values. Second, since the RBs have only the type of pooling, the position of the block in the complete network was considered as part of the feature vector. The final feature vector concatenates the feature vectors of all the CBs before the RB feature vectors, as shown in Figure 5. Finally, the vector is normalized by dividing it by the maximum value of each element according to the domain.

The length of the feature vector is

$$\text{length(Feat)} = \text{mCB}(3\text{mCN} + 1) + 2\text{mRB}, \qquad (1)$$

where mCB, mRB, and mCN, are the maximum number of CBs, RBs, and convolutional nodes, respectively. Factor 3 of mCN corresponds to the kernel size, feature map size, and node connections. Factor 2 of mRD is assumed because we consider the position of the RB, and the type of pooling.

2) LABELING

The GA was run without the NAC for two evolutions following a random initialization. After that, we ran our proposed method, and we trained the NAC after each run. We stored the architecture encoding and the best validation accuracy achieved. We defined high and low thresholds (T_H,T_L) for assigning the trained CNNs to either class 1 (potential for good performance), or class 0 (potential for poor performance). Class 1 was assigned to the architectures with accuracies higher than T_H, and class 0 to those with accuracies lower than T_L. Those architectures with accuracy between the two thresholds were not used to train the NAC. Initially we used 25% of the best and the worst trained CNNs to train the NAC. Then, the percentage of CNNs used was reduced by 5% after every two evolutions with the NAC. We adjusted the thresholds because the trained CNNs modified the validation accuracy distribution. This adjustment was necessary because the NAC eliminated CNNs with the potential for low performance. Figure 6 shows the accuracy distribution, and the thresholds for MRDBI, while Table 2 has the final thresholds for the six datasets.

TABLE 2. Final thresholds used in the NAC training.

Dataset	Low	High
MB	0.99	0.995
MBR	0.963	0.98
MBI	0.964	0.982
MRD	0.94	0.962
MRDBI	0.849	0.881
CIFAR-10	0.90	0.928

3) FITNESS EVALUATION

The fitness value of a CNN architecture depends on the classifier response that will determine whether or not the classifier will be trained. If the classifier identifies a CNN as class 1, it is trained from scratch, and its validation accuracy and training time are used to compute the fitness value. Otherwise, a random fitness value between 0 and 0.5 is assigned for those architectures that were not trained. The fitness is computed according to

$$\text{Fitness} = max(\text{acc}_{\text{val}}) - \frac{log(T_{\text{time}}/N_{\text{epochs}})}{1000}, \qquad (2)$$

where acc_{val} is the validation accuracy, and T_{time} and N_{epochs} are the training time and the number of epochs, respectively. The temporal component is used to prevent searching networks with large training times. The fitness function is similar to that proposed in [14].

IV. EXPERIMENTS

We compared the classification accuracy of our proposed method with GA, both with and without the NAC. The first experiments with the GA did not include the NAC for collecting data for the initial training of the NAC. In experiments with the NAC, we collected additional data (trained CNNs) to improve the NAC training.

A. BECHMARK DATASET

To test our proposed method, we used six well-known datasets for image classification tasks: five subsets of MNIST-V [105], and CIFAR-10 [106]. MNIST-V datasets are modified versions of the original MNIST, and include: MNIST Basic (MB) subset, MNIST with rotated digits (MRD), MNIST with random noise as background (MRB), MNIST with background images (MBI), and MNIST with rotated digits with background images (MRDBI) [105]. MNIST-V contains 62,000 images (28 × 28 pixels), 12,000 for training and 50,000 for testing [105]. CIFAR-10 contains 32×32-color images of 10 common objects, and it has 50,000 and 10,000 images for training and testing, respectively [106].

B. METHODS FOR COMPARISON

Using MNIST variants, we compared our results to the following SOTA evolutionary methods: DECNN [46], EvoCNN [45], IPPSO [51], psoCNN [48], psoECNN [47] HGAPSO [87], 2LGA [14], LF-MOGP [107], and IDECNN [108]. These methods do not use any data augmentation (DA). On CIFAR- 10, we compared our results to the following NAS methods: Johnson [80], LargeEvo [79], GeNet [92], CGP-CNN [94], CNN-GA [44], AE-CNN [43], AE-CNN+E2EPP [71], EvoApproxNAS [91], SHEDA-CNN [95], and NE-SGD [109]. All the methods selected use the following common DA techniques for CIFAR-10: four-pixel padding, random crop, and random flip [39], [40].

C. PARAMETER SETTINGS

In our proposed method, we set some of the parameters according to the values commonly used in previous research [39], [40]. The population size and the maximum number of generations were both set to 20. The crossover and mutation probabilities were set to 0.9 and 0.2, respectively, and the elitism rate was set at 0.1. For population initialization, we set the maximum number of blocks to six and the maximum number of nodes per CB to seven for MNIST-V and nine for CIFAR-10. The maximum number of nodes in a CB remained constant throughout evolution, but individual sizes could increase by adding more blocks or nodes in a block through genetic operations. Another restriction was the maximum number of RBs, which depended on the image size. Since we used a stride of two in each RB, the maximum number of RBs was three for MNIST-V and four for CIFAR-10.

The training parameters for evolution were set as follows: the training set was divided into 80% for training and 20% for validation, the batch size at 128, and the number of epochs at 25 and 45 for MNIST-V and CIFAR-10, respectively. To prevent overfitting, we used early stopping [110] of 10 epochs, and a dropout layer (p=0.1) [111], [112], after each convolutional node. We used the cross-entropy loss function [113] and Adam optimizer [114] with 1e-3 as the learning rate. The validation accuracy was used to compute the fitness, and the best individual in each run was retrained for more epochs (long training), as in [14] and [49]. The table 3 presents a summary of the parameters used during evolution.

TABLE 3. Summary of the parameter settings for evolution.

Parameter	Value
population size	20
number of generations	20
crossover probability	0.9
mutation probability	0.2
elitism	0.1
batch size	128
learning rate	0.001
optimizer	Adam
loss function	cross-entropy
training epochs	25 on MNISTV 40 on CIFAR-10
early stopping	10

The parameters listed on Table 3 are static except for the learning rate. We use the Adam optimizer and therefore, the learning rate is adapted during the CNN training. It is important to mention that the parameters defined in Table 3 are commonly static in Neuroevolution, as in [14], [43], [45], and [80].

In the long training, the best individual was trained from scratch using the training set, and the error rate in the test subset was reported on Tables 4 to 6. For MNIST-V we retrained the models for 100 epochs with warm-up and step decay learning rates. The warm-up increased from 0 to 0.01 during 20 epochs, and on the step decay scales, the learning rate divided by 10 in epochs 40 and 80. Early stopping of 50 epochs was also used. For CIFAR-10, we retrained for 250 epochs, with warm-up of 30 epochs, adjusting the learning rate in epochs 90 and 180, and early stopping of 100 epochs. Additionally, we retrained the best individuals using random rotation and cutout as DA for 200 epochs on MNIST-V. Cutout randomly masks rectangular regions of the input images during training to improve generalization and robustness [115]. We applied a 13 × 13 mask. Random rotation with a range of 180 degrees in any direction was applied for MRDBI and MRD, while for MRB, MBI, and MB, the range was set at 20 degrees. For CIFAR-10, the CNN that achieved the best results was also retrained, adding cutout to the DA used in the evolutions. We applied the same mask size (16×16) as in the original cutout paper [115].

The codes are implemented in Python 3.9.7 and PyTorch 1.11.0. The experiments on MNIST-V datasets were performed on a Intel(R) Core(TM) i7-7700 CPU @ 3.60 GHz and two Nvidia GeForce GTX 1080 GPUs, while the experiments on CIFAR-10 dataset were executed on an

TABLE 4. Comparison of test errors among our results and those of the SOTA on MNIST-V (lower is better). The symbols (+) and (-) indicate that our results are better, or worse, than the corresponding peer competitor, and the symbol (=) means that there is no statistically significant difference.

Method	MB	MBR	MBI	MRD	MRDBI
DECNN (mean) [46]	1.46(+)	3.56(+)	8.69(+)	5.53(+)	37.55(+)
DECNN (best) [46]	1.03(+)	3.46(+)	5.67(+)	4.07(+)	32.85(+)
EvoCNN (mean) [45]	1.28(+)	3.59(+)	4.62(+)	5.46(+)	37.38(+)
EvoCNN (best) [45]	1.18(+)	2.80(+)	4.53(+)	5.22(+)	35.03(+)
IPPSO (mean) [51]	1.21(+)				34.5(+)
IPPSO (best) [51]	1.13(+)				33.0(+)
psoCNN (mean) [48]		2.53(+)	2.40(+)	6.42(+)	20.98(+)
psoCNN (best) [48]		1.79(+)	1.90(+)	3.58(+)	14.28(+)
psoECNN (mean) [47]	0.38(+)	1.8(+)	2.2(+)	3.23(+)	13.61(+)
psoECNN (best) [47]	0.35(+)	1.88(+)	2.46(+)	3.9(+)	11.61(+)
HGAPSO (mean) [87]	0.84(+)				12.23(+)
HGAPSO (best) [87]	0.74(+)				10.53(+)
2LGA (mean) [14]	0.65(+)	1.37(=)	1.34(=)	2.43(+)	6.73(=)
2LGA (best) [14]	0.61(+)	1.35(+)	1.26(+)	2.26(+)	6.33(+)
LF-MOGP (best) [107]	0.52(-)	2.41(+)	3.08(+)	5.45(+)	14.98(+)
IDECNN (mean) [108]	0.38(-)	2.07(+)	2.29(+)	4.16(+)	14.31(+)
IDECNN (best) [108]	0.29(-)	1.29(+)	1.1(-)	3.02(+)	10.04(+)
Our method (mean)	0.61	1.38	1.32	2.15	6.6
Our method (best)	0.55	1.28	1.24	1.95	6.0
Our method (std)	0.03	0.07	0.05	0.17	0.33

Intel(R) Xeon(R) W-2123 CPU @3.6 GHz and three Nvidia GeForce RTX2080Ti GPUs.

V. RESULTS AND DISCUSSION

A. OVERALL RESULTS

We compared our results to those of the SOTA using the same datasets and testing conditions. All the experiments were performed five times, and the mean and minimum test error were reported as in the SOTA. Table 4 shows the results on the five MNIST-V subsets, and One Sample T-Test [116] was used to compare with the SOTA as in [46] and [51]. Our method revealed the lowest error in all cases on the most challenging datasets, (MRDBI, MRD, and MBR). Our results demonstrated an improvement of 5.2% (0.33% error reduction) on MRDBI, 13.71% (0.31% error reduction) on MRD, and 0.77% (0.1% error reduction) on MBR, and our results were comparable to the SOTA on the other datasets. For CIFAR-10, studies from the SOTA reported just the minimum error achieved. For this reason, no statistically significant comparison is possible. However, on Table 5 we compared our results with SOTA methods and reported the relative improvement, with regard to [43], [44], [71], and [95].

Table 5 shows a comparison of our results on CIFAR-10 with those of the SOTA in terms of test error, number of parameters, and computational cost. The latter was measured in GPU-days as has been done in previous studies [43], [44], [71], [95], even though different GPUs were used. Our method achieved a 4.08% test error in 17.3 GPU-days, while AE-CNN obtained a 4.3% test error using 27 GPU-days. This result reduced the AE-CNN test error by 0.22% using only 64% of the GPU-days. The E2PP method reduced the search time to 8.5 GPU-days on AE-CNN, but the test error increased to 5.3%, and therefore, our GA with NAC achieved an improvement of 23% (1.22% error reduction). Some studies employed predefined CNN architectures, ResNet-20 and DPN-92 [117], that reach results with lower GPU-days, but are not directly comparable [95], [109]. Additionally, we trained the best architecture using cutout as additional data augmentation, reducing the test error to 3.14%. It is worth mentioning that the EffPNet, a method with an SVM as a surrogate model, achieved 3.49% test error, but the authors did not mention whether or not they applied any type of DA in training [49].

TABLE 5. Comparison of test error, computational cost in GPU-days, and number of parameters in our results and those of other Neuroevolution methods on CIFAR-10 (lower is better).

Method	Error (%)	GPU-days	Parameters (M)
Johnson [80]	15.15	0.5	
LargeEvo [79]	5.4	2730	40.1
GeNet [92]	7.1	17	
CGP-CNN [94]	5.98	27	1.68
CNN-GA [44]	4.78	35	2.9
AE-CNN [43]	4.3	27	2.0
AE-CNN+E2EPP [71]	5.3	8.5	
EvoApproxNAS [91]	6.8		1.11
SHEDA-CNN+cutout [95]	3.64	0.58	10.88
NE-SGD+cutout [109]	4.35	11	
Our method	4.08	17.3	4.18
Our method+cutout	3.14	17.3	4.18

TABLE 6. Comparison of test error (lower is better) among the best individuals found on MNIST-V, trained with (w) and without (w/o) DA, AA filter and cosine learning rate schedule (cos lr).

Training condition	MB	MBR	MBI	MRD	MRDBI
w/o AA, w/o DA (mean)	0.62	1.32	1.30	2.52	6.88
w/o AA, w/o DA (best)	0.6	1.29	1.29	2.41	6.79
w AA, w/o DA (mean)	0.58	1.32	1.27	2.00	6.08
w AA, w/o DA (best)	0.55	1.28	1.24	1.95	6.0
w/o AA, w DA (mean)	0.63	1.09	1.02	1.32	4.58
w/o AA, w DA (best)	0.51	1.15	0.9	1.27	4.28
w AA, w DA (mean)	0.51	1.08	1.1	1.07	4.01
w AA, w DA (best)	0.48	1.03	1.09	1.01	3.9
w AA, w DA, cos lr (mean)	0.52	1.06	1.02	1.34	3.47
w AA, w DA, cos lr (best)	0.48	0.95	1.0	1.22	3.36

B. ANTI-ALIASING FILTER AND DA FOR MNIST-V

For the ablation study, we trained the best CNNs with and without DA, with and without AA filters, and with the cosine annealing learning rate [118] instead of the step decay learning rate on the MNIST-V datasets. As can be seen on Table 6, DA improves the performance of our best CNN significantly. The error was reduced by 2.1% (an improvement of 33%) on MRDBI, and by an average of 0.7% (an improvement of 23%) when we trained with the step decay learning rate. Moreover, the error was reduced by 2.6% on MRDBI and 0.8% on average when the models were trained with a cosine annealing learning rate instead of a step decay schedule. Figure 7 shows the boxplots of the test error on the MRDBI dataset under the different training conditions.

Statistical tests were performed on the MRDBI results for the different training conditions, using ANOVA and Tukey HSD tests [119] to determine if the differences

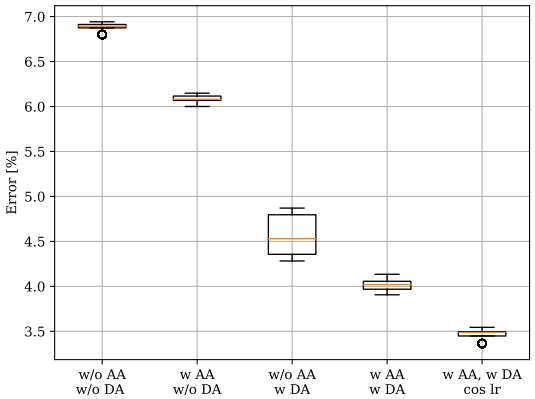

FIGURE 7. Test error on the MRDBI dataset of the best CNN with the following training conditions, with (w) and without (w/o): DA, AA filter, and cosine learning rate schedule (cos lr).

TABLE 7. Summary of the results of the Tukey HSD test for the best CNN test error, trained with (w) and without (w/o): DA, AA filter, and cosine learning rate schedule (cos lr) on MRDBI.

Group 1	Group 2	Mean-diff (% error)	reject
w/o AA, w/o DA	w AA, w/o DA	-0.80	True
w/o AA, w/o DA	w/o AA, w DA	-2.29	True
w AA, w/o DA	w/o AA, w DA	-2.06	True
w/o AA, w DA	w/o AA, w DA	-0.57	True
w/o AA, w DA	w/o AA, w DA, cos lr	-0.54	True

were statistically significant. The ANOVA test yielded a $p_{\text{value}} = 7e-20$ (< 0.05), meaning that the results are significantly different among test error scores. Table 7 shows a summary of Tukey HSD results, on which there are significant differences in the test error between each pair of experiments. Therefore, using the AA filters improves the CNN performance significantly compared to performing experiments without these filters. Also, the variance in test errors is reduced in experiments with DA. Finally, the cosine-schedule learning rate with suitable hyperparameters helps in reducing the number of test errors.

C. ADVANTAGES OF USING NAC

Figure 8 shows the validation accuracy of the CNNs trained by experiments with and without the NAC on six datasets. Notice that the NAC avoids training architectures that could result in low validation accuracy. For instance, the lowest validation accuracy scores in experiments without the NAC were 0.73, 0.81, 0.91, 0.76, 0.54, and 0.63 for the MB, MBR, MBI, RD, MRDBI, and the CIFAR-10 datasets, respectively. By contrast, when the GA integrated the NAC, the lowest accuracy in each dataset increased to 0.987, 0.953, 0.95, 0.92, 0.811, and 0.832, respectively on the same six datasets.

Figure 9 shows the validation accuracy distribution of the CNN trained in experiments developed on the MRDBI dataset. In cases without NAC, only 25% of the CNN achieved validation accuracy over 88%, and 36% obtained accuracy lower than 85%. Additionally, less than 13% of

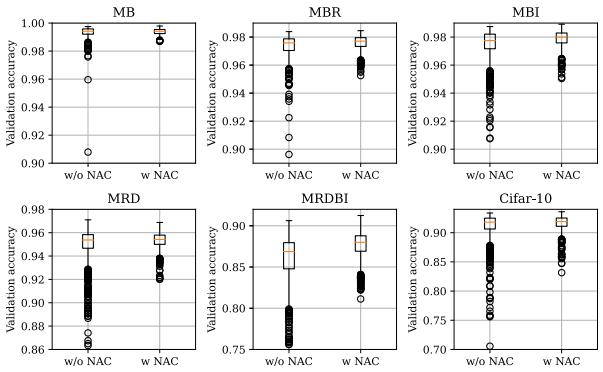

FIGURE 8. Validation accuracy achieved in experiments without (w/o) and with (w) NAC on each of the six datasets (MB, MBR, MBI, RD, MRDBI, and CIFAR-10).

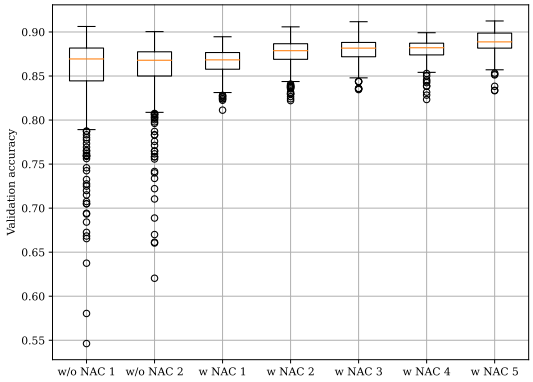

FIGURE 9. Validation accuracy distribution of each run on the MRDBI dataset. Two evolutions without (w/o) NAC, and five runs with (w) NAC. The NAC avoids training architectures that result in low validation accuracy.

the trained CNNs achieved performance under the initial T_L used to train the classifier, and more than 50% of the trained CNNs achieved more than the initial T_H in validation accuracy.

We performed several experiments with and without NAC to compare the search time on the same GPU. Table 8 shows the searching time in GPU-days, and the lowest test error scores achieved in experiments with and without NAC, respectively. The experiments demonstrated that including the NAC in the GA reduced the search time by 41% on average on six datasets. Moreover, as a result of our literature review, we found that previous work reported the GPU-days [43], [71], [92], [95] although they used different GPUs for computations, and compared the computational cost of various NAS methods. The measure in GPU-days was performed in [43], [71], [92], and [95] even though comparing GPU-days depends on the GPUs used.

D. EVOLUTIONARY TRAJECTORY

Figure 10 shows the evolutionary trajectory of an experiment on the MRDBI set. The green line represents the fitness of the best individual in each generation. The blue line

TABLE 8. Comparison of search time in GPU-days and the best test error in experiments with (w) and without (w/o) the NAC.

Dataset	Search Time			Test error		
	w NAC	w/o NAC	% reduction	w NAC	w/o NAC	% reduction
MB	0.63	1.12	43.7	0.55	0.6	8.3
MBR	0.8	1.32	39.3	1.28	1.32	3
MBI	0.87	1.56	44.2	1.24	1.29	3.8
MRD	1.1	1.72	36	1.95	1.89	-2.6
MRDBI	1.19	1.92	38	6	6.44	6.8
CIFAR-10	17.3	30.6	43.4	4.08	4.4	7.2

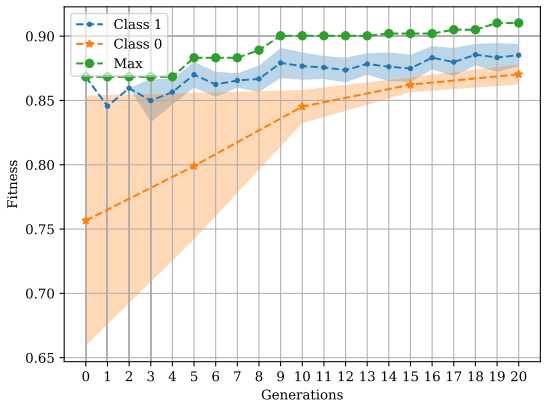

FIGURE 10. Evolutionary trajectory of a GA with a NAC on the MRDBI dataset. In green, the fitness of the best individual for each generation; in blue, the average fitness of trained CNNs during the evolution; and in orange, the average fitness of the rejected CNNs trained once evolution ends for analysis purposes.

represents the average fitness achieved by the trained population (class 1) in each generation. In order to know the behavior of the NAC during an evolution, individuals rejected by the classifier in the initial population, and in generations 5, 10, 15 and 20, were stored and trained with the same hyperparameter as evolution (orange line). It is clear that the GA with the NAC did not get stuck in a local minimum and improved the population during the evolution as shown in Figure 10. These results confirm that the NAC performed well in rejecting these CNN architectures for training.

Figure 11 shows the number of trained and non-trained CNNs for each generation in an experiment with NAC on the MRDBI dataset. In this evolution, 357 individuals were generated by the GA, but only 168 of those CNNs were trained. Additionally, it can be observed that less than 10% of the population was trained (not rejected by the NAC) in the first 25% of the generations. By contrast, 70% of the architectures were trained in the last 25% of the generations. These results show that the NAC helps improve the population performance over the generations. Starting with a random initial population, having individuals with high performance in early generations is not expected. Consequently, our NAC rejects most of those individuals, and the number of trained architectures increases with evolution because better individuals are generated.

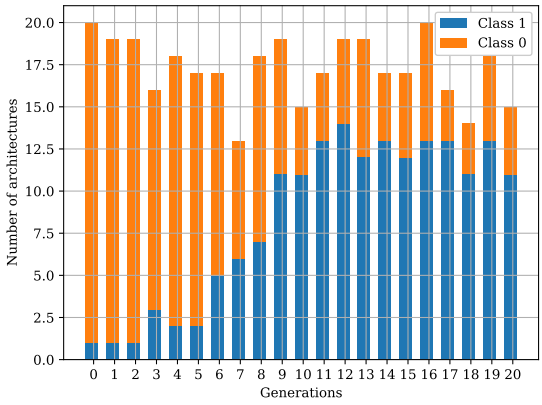

FIGURE 11. Number of trained (class 1) and non-trained (class 0) architectures in each generation in an experiment with NAC on the MRDBI dataset.

TABLE 9. Parameters of the best model in each dataset.

Dataset	CB[a]	RB[b]	conv. nodes[c]
MB	3 (3)	3 (1 - 2)	12 (6 - 5 - 1)
MBR	7 (7)	2 (0 - 2)	25 (10 - 9 - 6)
MBI	6 (5)	3 (1 - 2)	23 (8 - 4 - 11)
MRD	7 (5)	3 (1 - 2)	26 (7 - 8 - 11)
MRDBI	7 (3)	3 (3 - 1)	22 (3 - 13 - 6)
CIFAR10	12 (6)	4 (2 - 2)	48 (11 - 28 - 9)

[a] number of convolutional blocks (unique blocks).
[b] number of reduction blocks (max - avg pooling).
[c] number of convolutional nodes (kernel size= 5-3-1).

E. SEARCH SPACE

We used a search space with few restrictions, encoding the complete architecture by two kinds of blocks, so that the CNNs could have any width, depth, or shape. Table 9 presents the value of the parameters of the best model in the search space domain. As can be seen on Table 9, the number of convolutional nodes, which represent the depth of the network, is 12 for the MB dataset (the easiest subset of MNIST-V), but the best model for CIFAR-10 has 48 convolutional nodes.

VI. CONCLUSION

In this work, we propose a novel GA to improve search in Neuroevolution. The new GA incorporates three main contributions. The most important one is that the proposed GA integrates the NAC into the conventional GA and thereby reducing the search time. Additionally, our proposed encoding and search space are different because we represent the complete CNN in blocks instead of stacking the same cell, as in [14] and [49]. Our proposed encoding allows CNNs to have any size, width, and skip connections through nodes. Furthermore, we included the skip connections in the block instead of using DenseNet or ResNet blocks, as in [43] and [71].

We applied the proposed method on pattern recognition problems using the MNIST-V and CIFAR-10 datasets. Our method achieved a significant reduction in the test error compared to previously published results: 0.77% on the MBR

set, 13.7% on the MRD set, and 5% on the MRDBI set without using any DA. In addition, our results on the CIFAR-10 are competitive with the SOTA, improving the error score and reducing the search time reported for other evolutionary methods using common DA.

We also trained the best model by applying rotation and cutout as DA, improving the test error achieved without DA by 12% on MB, 18% on MBR, 15% on MBI, 47% on MRD, and 35% on MRDBI; cutout was added to common DA on CIFAR-10 improving the test error by 23%.

We demonstrated experimentally that including a NAC in a GA reduces the search time substantially, improving the global results as well. By contrast, previous studies used an E2EPP, which compromised the performance of the GA. The idea of a classifier could also be useful in other evolutionary algorithms applied to Neuroevolution, or to other optimization problems.

For future work a search space in a multi-objective optimization problem could be used to optimize not only the accuracy of the CNNs, but also the number of parameters. In addition, the effectiveness of the proposed method on larger datasets, in other pattern recognition tasks, could be assessed. Similarly, a recent proposed approach [120], [121] to balancing exploration and exploitation could be included in future research to assess possible improvements in the results obtained by the GA.

Finally, some possible limitations of our proposed method are the time required for evolution, and the number of GPUs required for larger datasets. Integrating the NAC into the GA reduces the search time, but the computational time may increase and requires a large number of GPUs if Neuroevolution is applied to larger datasets. Despite the good results obtained in this study, the application of the proposed method to real-world problems should be examined and evaluated in the future.

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, May 2015.

[2] W. Rawat and Z. Wang, "Deep convolutional neural networks for image classification: A comprehensive review," *Neural Comput.*, vol. 29, no. 9, pp. 2352–2449, Sep. 2017.

[3] J. E. Zambrano, D. P. Benalcazar, C. A. Perez, and K. W. Bowyer, "Iris recognition using low-level CNN layers without training and single matching," *IEEE Access*, vol. 10, pp. 41276–41286, 2022.

[4] M. Z. Alom, T. M. Taha, C. Yakopcic, S. Westberg, P. Sidike, M. S. Nasrin, M. Hasan, B. C. Van Essen, A. A. S. Awwal, and V. K. Asari, "A state-of-the-art survey on deep learning theory and architectures," *Electronics*, vol. 8, no. 3, p. 292, Mar. 2019.

[5] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. Mohamed, and H. Arshad, "State-of-the-art in artificial neural network applications: A survey," *Heliyon*, vol. 4, no. 11, Nov. 2018, Art. no. e00938, doi: 10.1016/j.heliyon.2018.e00938.

[6] Y. LeCun, K. Kavukcuoglu, and C. Farabet, "Convolutional networks and applications in vision," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2010, pp. 253–256.

[7] G. W. Lindsay, "Convolutional neural networks as a model of the visual system: Past, present, and future," *J. Cognit. Neurosci.*, vol. 33, no. 10, pp. 2017–2031, Sep. 2021.

[8] D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *J. Physiol.*, vol. 160, no. 1, pp. 106–154, Jan. 1962.

[9] K. Fukushima, "Neural network model for a mechanism of pattern recognition unaffected by shift in position-neocognitron," *IEICE Tech. Rep., A*, vol. 62, no. 10, pp. 658–665, 1979.

[10] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[11] D. P. Benalcazar, J. E. Zambrano, D. Bastias, C. A. Perez, and K. W. Bowyer, "A 3D iris scanner from a single image using convolutional neural networks," *IEEE Access*, vol. 8, pp. 98584–98599, 2020.

[12] B. Wang, Y. Sun, B. Xue, and M. Zhang, "Evolving deep neural networks by multi-objective particle swarm optimization for image classification," in *Proc. Genetic Evol. Comput. Conf. (GECCO)*. New York, NY, USA: Association for Computing Machinery, Jul. 2019, pp. 490–498.

[13] B. Wang, B. Xue, and M. Zhang, "Particle swarm optimization for evolving deep convolutional neural networks for image classification: Single- and multi-objective approaches," in *Deep Neural Evolution: Deep Learning with Evolutionary Computation*. Singapore: Springer, 2020, pp. 155–184.

[14] D. A. Montecino, C. A. Perez, and K. W. Bowyer, "Two-level genetic algorithm for evolving convolutional neural networks for pattern recognition," *IEEE Access*, vol. 9, pp. 126856–126872, 2021.

[15] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biol. Cybern.*, vol. 36, no. 4, pp. 193–202, Apr. 1980.

[16] K. Fukushima, S. Miyake, and T. Ito, "Neocognitron: A neural network model for a mechanism of visual pattern recognition," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-13, no. 5, pp. 826–834, Sep. 1983.

[17] K. Fukushima, "Neocognitron: A hierarchical neural network capable of visual pattern recognition," *Neural Netw.*, vol. 1, no. 2, pp. 119–130, 1988.

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 25, 2012, pp. 1097–1105.

[19] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, Dec. 2015.

[20] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. 3rd Int. Conf. Learn. Represent. (ICLR)*, 2015, pp. 1–14.

[21] C. Bock, M. Moor, C. R. Jutzeler, and K. Borgwardt, "Machine learning for biomedical time series classification: From shapelets to deep learning," in *Artificial Neural Networks*. New York, NY, USA: Humana, 2021, pp. 33–71.

[22] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 1–9.

[23] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[24] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. 30th IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jan. 2017, pp. 2261–2269.

[25] D. Han, J. Kim, and J. Kim, "Deep pyramidal residual networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 6307–6315.

[26] C. A. Perez, P. A. Estévez, F. J. Galdames, D. A. Schulz, J. P. Perez, D. Bastías, and D. R. Vilar, "Trademark image retrieval using a combination of deep convolutional neural networks," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2018, pp. 1–7.

[27] C. A. Perez, C. A. Salinas, P. A. Estevez, and P. M. Valenzuela, "Genetic design of biologically inspired receptive fields for neural pattern recognition," *IEEE Trans. Syst., Man Cybern. B, Cybern.*, vol. 33, no. 2, pp. 258–270, Apr. 2003.

[28] B. A. Olshausen and D. J. Field, "Sparse coding with an overcomplete basis set: A strategy employed by V1?" *Vis. Res.*, vol. 37, no. 23, pp. 3311–3325, Dec. 1997. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0042698997001697

[29] K. O. Stanley, J. Clune, J. Lehman, and R. Miikkulainen, "Designing neural networks through neuroevolution," *Nature Mach. Intell.*, vol. 1, no. 1, pp. 24–35, Jan. 2019, doi: 10.1038/s42256-018-0006-z.

[30] R. Miikkulainen, "Neuroevolution BT," in *Encyclopedia of Machine Learning and Data Mining*, C. Sammut and G. I. Webb, Eds. New York, NY, USA: Springer, 2017, doi: 10.1007/978-1-4899-7687-1_594.
[31] Z. Lu, I. Whalen, Y. Dhebar, K. Deb, E. D. Goodman, W. Banzhaf, and V. N. Boddeti, "Multiobjective evolutionary design of deep convolutional neural networks for image classification," *IEEE Trans. Evol. Comput.*, vol. 25, no. 2, pp. 277–291, Apr. 2021.
[32] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," 2016, *arXiv:1611.01578*.
[33] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," 2016, *arXiv:1611.02167*.
[34] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 4095–4104.
[35] Z. Zhong, J. Yan, W. Wu, J. Shao, and C.-L. Liu, "Practical block-wise neural network architecture generation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 2423–2432.
[36] Z. Zhong, Z. Yang, B. Deng, J. Yan, W. Wu, J. Shao, and C.-L. Liu, "BlockQNN: Efficient block-wise neural network architecture generation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 7, pp. 2314–2328, Jul. 2021.
[37] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," in *Proc. 7th Int. Conf. Learn. Represent. (ICLR)*, 2019, pp. 1–13.
[38] A. Zela, T. Elsken, T. Saikia, Y. Marrakchi, T. Brox, and F. Hutter, "Understanding and robustifying differentiable architecture search," 2019, *arXiv:1909.09656*.
[39] Y. Bi, B. Xue, P. Mesejo, S. Cagnoni, and M. Zhang, "A survey on evolutionary computation for computer vision and image analysis: Past, present, and future trends," *IEEE Trans. Evol. Comput.*, vol. 27, no. 1, pp. 5–25, Feb. 2023.
[40] Y. Liu, Y. Sun, B. Xue, M. Zhang, G. G. Yen, and K. C. Tan, "A survey on evolutionary neural architecture search," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 34, no. 2, pp. 550–570, Feb. 2023.
[41] X. Zhou, A. K. Qin, M. Gong, and K. C. Tan, "A survey on evolutionary construction of deep neural networks," *IEEE Trans. Evol. Comput.*, vol. 25, no. 5, pp. 894–912, Oct. 2021.
[42] K. T. Chitty-Venkata, M. Emani, V. Vishwanath, and A. K. Somani, "Neural architecture search benchmarks: Insights and survey," *IEEE Access*, vol. 11, pp. 25217–25236, 2023.
[43] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Completely automated CNN architecture design based on blocks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 4, pp. 1242–1254, Apr. 2020.
[44] Y. Sun, B. Xue, M. Zhang, G. G. Yen, and J. Lv, "Automatically designing CNN architectures using the genetic algorithm for image classification," *IEEE Trans. Cybern.*, vol. 50, no. 9, pp. 3840–3854, Sep. 2020.
[45] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Evolving deep convolutional neural networks for image classification," *IEEE Trans. Evol. Comput.*, vol. 24, no. 2, pp. 394–407, Apr. 2020.
[46] B. Wang, Y. Sun, B. Xue, and M. Zhang, "A hybrid differential evolution approach to designing deep convolutional neural networks for image classification," in *Advances in Artificial Intelligence* (Lecture Notes in Computer Science, Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 11320. Cham, Switzerland: Springer, 2018, pp. 237–250.
[47] T. Lawrence, L. Zhang, C. P. Lim, and E.-J. Phillips, "Particle swarm optimization for automatically evolving convolutional neural networks for image classification," *IEEE Access*, vol. 9, pp. 14369–14386, 2021.
[48] F. E. Fernandes Jr. and G. G. Yen, "Particle swarm optimization of deep neural networks architectures for image classification," *Swarm Evol. Comput.*, vol. 49, pp. 62–74, Sep. 2019, doi: 10.1016/j.swevo.2019.05.010.
[49] B. Wang, B. Xue, and M. Zhang, "Surrogate-assisted particle swarm optimization for evolving variable-length transferable blocks for image classification," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 8, pp. 3727–3740, Aug. 2022.
[50] H. Chen, F. Miao, and X. Shen, "Hyperspectral remote sensing image classification with CNN based on quantum genetic-optimized sparse representation," *IEEE Access*, vol. 8, pp. 99900–99909, 2020.
[51] B. Wang, Y. Sun, B. Xue, and M. Zhang, "Evolving deep convolutional neural networks by variable-length particle swarm optimization for image classification," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jul. 2018, pp. 1–8.
[52] T. Hassanzadeh, D. Essam, and R. Sarker, "An evolutionary DenseRes deep convolutional neural network for medical image segmentation," *IEEE Access*, vol. 8, pp. 212298–212314, 2020.
[53] K. R. G. Operiano, H. Iba, and W. Pora, "Neuroevolution architecture backbone for X-ray object detection," in *Proc. IEEE Symp. Ser. Comput. Intell. (SSCI)*, Dec. 2020, pp. 2296–2303.
[54] A. Baldominos, Y. Saez, and P. Isasi, "Evolutionary design of convolutional neural networks for human activity recognition in sensor-rich environments," *Sensors*, vol. 18, no. 4, p. 1288, Apr. 2018.
[55] A. ElSaid, S. Benson, S. Patwardhan, D. Stadem, and T. Desell, "Evolving recurrent neural networks for time series data prediction of coal plant parameters," in *Proc. 22nd Int. Conf. Appl. Evol. Comput. Evo Appl. EvoStar*, Leipzig, Germany. Cham, Switzerland: Springer, 2019, pp. 488–503.
[56] S. S. Mostafa, F. Mendonça, A. G. Ravelo-Garcia, G. Gabriel Juliá-Serdá, and F. Morgado-Dias, "Multi-objective hyperparameter optimization of convolutional neural network for obstructive sleep apnea detection," *IEEE Access*, vol. 8, pp. 129586–129599, 2020.
[57] H. Xie, L. Zhang, and C. P. Lim, "Evolving CNN-LSTM models for time series prediction using enhanced grey wolf optimizer," *IEEE Access*, vol. 8, pp. 161519–161541, 2020.
[58] T. N. Fatyanosa and M. Aritsugi, "An automatic convolutional neural network optimization using a diversity-guided genetic algorithm," *IEEE Access*, vol. 9, pp. 91410–91426, 2021.
[59] N. Klyuchnikov, I. Trofimov, E. Artemova, M. Salnikov, M. Fedorov, A. Filippov, and E. Burnaev, "NAS-Bench-NLP: Neural architecture search benchmark for natural language processing," *IEEE Access*, vol. 10, pp. 45736–45747, 2022.
[60] J. P. Perez and C. A. Perez, "Face patches designed through neuroevolution for face recognition with large pose variation," *IEEE Access*, vol. 11, pp. 72861–72873, 2023.
[61] F. Boutros, P. Siebke, M. Klemt, N. Damer, F. Kirchbuchner, and A. Kuijper, "PocketNet: Extreme lightweight face recognition network using neural architecture search and multistep knowledge distillation," *IEEE Access*, vol. 10, pp. 46823–46833, 2022.
[62] A. Baldominos, Y. Saez, and P. Isasi, "Evolutionary convolutional neural networks: An application to handwriting recognition," *Neurocomputing*, vol. 283, pp. 38–52, Mar. 2018, doi: 10.1016/j.neucom.2017.12.049.
[63] P. Chrabaszcz, I. Loshchilov, and F. Hutter, "A downsampled variant of ImageNet as an alternative to the CIFAR datasets," 2017, *arXiv:1707.08819*.
[64] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proc. 33rd AAAI Conf. Artif. Intell. (AAAI), 31st Innov. Appl. Artif. Intell. Conf., 9th AAAI Symp. Educ. Adv. Artif. Intell. (EAAI)*, vol. 33, 2019, pp. 4780–4789.
[65] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 8697–8710.
[66] D. Zhou, X. Zhou, W. Zhang, C. C. Loy, S. Yi, X. Zhang, and W. Ouyang, "EcoNAS: Finding proxies for economical neural architecture search," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 11393–11401.
[67] T. Domhan, J. T. Springenberg, and F. Hutter, "Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves," in *Proc. 24th Int. Joint Conf. Artif. Intell.*, 2015, pp. 3460–3468.
[68] A. Klein, S. Falkner, J. T. Springenberg, and F. Hutter, "Learning curve prediction with Bayesian neural networks," in *Proc. 5th Int. Conf. Learn. Represent. (ICLR)*, Toulon, France, Apr. 2017. [Online]. Available: https://openreview.net/pdf?id=S11KBYclx
[69] B. Deng, J. Yan, and D. Lin, "Peephole: Predicting network performance before training," 2017, *arXiv:1712.03351*.
[70] R. Istrate, F. Scheidegger, G. Mariani, D. Nikolopoulos, C. Bekas, and A. C. I. Malossi, "TAPAS: Train-less accuracy predictor for architecture search," in *Proc. AAAI Conf. Artif. Intell.*, 2019, vol. 33, no. 1, pp. 3927–3934.
[71] Y. Sun, H. Wang, B. Xue, Y. Jin, G. G. Yen, and M. Zhang, "Surrogate-assisted evolutionary deep learning using an end-to-end random forest-based performance predictor," *IEEE Trans. Evol. Comput.*, vol. 24, no. 2, pp. 350–364, Apr. 2020.
[72] Y. Sun, X. Sun, Y. Fang, G. G. Yen, and Y. Liu, "A novel training protocol for performance predictors of evolutionary neural architecture search algorithms," *IEEE Trans. Evol. Comput.*, vol. 25, no. 3, pp. 524–536, Jun. 2021.

[73] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "SMASH: One-shot model architecture search through hypernetworks," in *Proc. 6th Int. Conf. Learn. Represent. (ICLR)*, Vancouver, BC, USA, May 2018. [Online]. Available: https://openreview.net/pdf?id=rydeCEhs-

[74] B. Wang, B. Xue, and M. Zhang, "A transfer learning based evolutionary deep learning framework to evolve convolutional neural networks," in *Proc. Genetic Evol. Comput. Conf. Companion (GECCO)*, Jul. 2021, pp. 287–288.

[75] S. Hu, R. Cheng, C. He, Z. Lu, J. Wang, and M. Zhang, "Accelerating multi-objective neural architecture search by random-weight evaluation," *Complex Intell. Syst.*, vol. 9, no. 2, pp. 1183–1192, Apr. 2023, doi: 10.1007/s40747-021-00594-5.

[76] H. Tan, R. Cheng, S. Huang, C. He, C. Qiu, F. Yang, and P. Luo, "RelativeNAS: Relative neural architecture search via slow-fast learning," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 34, no. 1, pp. 475–489, Jan. 2023.

[77] G.-A. Vargas-Hákim, E. Mezura-Montes, and H.-G. Acosta-Mesa, "A review on convolutional neural network encodings for neuroevolution," *IEEE Trans. Evol. Comput.*, vol. 26, no. 1, pp. 12–27, Feb. 2022.

[78] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical representations for efficient architecture search," in *Proc. 6th Int. Conf. Learn. Represent. (ICLR)*, Vancouver, BC, Canada, Apr./May 2018. [Online]. Available: https://openreview.net/pdf?id=BJQRKzbA-

[79] 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 *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 2902–2911.

[80] 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. 156139–156152, 2020.

[81] Z. Chen, Y. Zhou, and Z. Huang, "Auto-creation of effective neural network architecture by evolutionary algorithm and ResNet for image classification," in *Proc. IEEE Int. Conf. Syst., Man Cybern. (SMC)*, Oct. 2019, pp. 3895–3900.

[82] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "MnasNet: Platform-aware neural architecture search for mobile," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 2815–2823.

[83] L. Frachon, W. Pang, and G. M. Coghill, "ImmuNeCS: Neural committee search by an artificial immune system," 2019, *arXiv:1911.07729*.

[84] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Trans. Neural Netw.*, vol. 5, no. 2, pp. 157–166, Mar. 1994.

[85] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Training very deep networks," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, vol. 28, 2015, pp. 1–9.

[86] R. Zhang, "Making convolutional networks shift-invariant again," in *Proc. 36th Int. Conf. Mach. Learn.*, vol. 97, 2019, pp. 12712–12722.

[87] B. Wang, Y. Sun, B. Xue, and M. Zhang, "A hybrid GA-PSO method for evolving architecture and short connections of deep convolutional neural networks," in *Trends in Artificial Intelligence* (Lecture Notes in Computer Science, Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 11672. Cham, Switzerland: Springer, 2019, pp. 650–663.

[88] X. Yao and Y. Liu, "A new evolutionary system for evolving artificial neural networks," *IEEE Trans. Neural Netw.*, vol. 8, no. 3, pp. 694–713, May 1997.

[89] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evol. Comput.*, vol. 10, no. 2, pp. 99–127, Jun. 2002.

[90] A. M. Anter, A. W. Mohamed, M. Zhang, and Z. Zhang, "A robust intelligence regression model for monitoring Parkinson's disease based on speech signals," *Future Gener. Comput. Syst.*, vol. 147, pp. 316–327, Oct. 2023.

[91] H. Zhang, K. Hao, L. Gao, B. Wei, and X. Tang, "Optimizing deep neural networks through neuroevolution with stochastic gradient descent," *IEEE Trans. Cognit. Develop. Syst.*, vol. 15, no. 1, pp. 111–121, Mar. 2023.

[92] L. Xie and A. Yuille, "Genetic CNN," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 1388–1397.

[93] J. Ren, Z. Li, J. Yang, N. Xu, T. Yang, and D. J. Foran, "EIGEN: Ecologically-inspired GENetic approach for neural network structure searching from scratch," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 9051–9060.

[94] M. Suganuma, S. Shirakawa, and T. Nagao, "A genetic programming approach to designing convolutional neural network architectures," in *Proc. Genetic Evol. Comput. Conf. (GECCO)*, Jul. 2017, pp. 497–504.

[95] J.-Y. Li, Z.-H. Zhan, J. Xu, S. Kwong, and J. Zhang, "Surrogate-assisted hybrid-model estimation of distribution algorithm for mixed-variable hyperparameters optimization in convolutional neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 34, no. 5, pp. 2338–2352, May 2023.

[96] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. 32nd Int. Conf. Mach. Learn.*, vol. 37. Lille, France: PMLR, 2015, pp. 448–456.

[97] L. M. Schmitt, "Theory of genetic algorithms," *Theor. Comput. Sci.*, vol. 259, nos. 1–2, pp. 1–61, 2001.

[98] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002.

[99] Y. Sun, G. G. Yen, and Z. Yi, "IGD indicator-based evolutionary algorithm for many-objective optimization problems," *IEEE Trans. Evol. Comput.*, vol. 23, no. 2, pp. 173–187, Apr. 2019.

[100] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press, 1998.

[101] Y. Leung, Y. Gao, and Z.-B. Xu, "Degree of population diversity—A perspective on premature convergence in genetic algorithms and its Markov chain analysis," *IEEE Trans. Neural Netw.*, vol. 8, no. 5, pp. 1165–1176, Sep. 1997.

[102] B. L. Miller and D. E. Goldberg, "Genetic algorithms, tournament selection, and the effects of noise," *Complex Syst.*, vol. 9, no. 3, pp. 193–212, 1995.

[103] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, 1995.

[104] C. M. Bishop and N. M. Nasrabadi, *Pattern Recognition and Machine Learning*. New York, NY, USA: Springer, 2006, vol. 4.

[105] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio, "An empirical evaluation of deep architectures on problems with many factors of variation," in *Proc. ACM Int. Conf.*, vol. 227, 2007, pp. 473–480.

[106] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Univ. Toronto, Toronto, ON, Canada, Tech. Rep. TR-2009, 2009. [Online]. Available: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

[107] Q. Liu, X. Wang, Y. Wang, and X. Song, "Evolutionary convolutional neural network for image classification based on multi-objective genetic programming with leader–follower mechanism," *Complex Intell. Syst.*, vol. 9, no. 3, pp. 3211–3228, Jun. 2023.

[108] A. Ghosh, N. D. Jana, S. Mallik, and Z. Zhao, "Designing optimal convolutional neural network architecture using differential evolution algorithm," *Patterns*, vol. 3, no. 9, Sep. 2022, Art. no. 100567.

[109] M. Pinos, V. Mrazek, and L. Sekanina, "Evolutionary approximation and neural architecture search," *Genetic Program. Evolvable Mach.*, vol. 23, no. 3, pp. 351–374, Sep. 2022.

[110] Y. Yao, L. Rosasco, and A. Caponnetto, "On early stopping in gradient descent learning," *Constructive Approximation*, vol. 26, no. 2, pp. 289–315, Aug. 2007.

[111] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, Jun. 2014.

[112] S. Park and N. Kwak, "Analysis on the dropout effect in convolutional neural networks," in *Proc. 13th Asian Conf. Comput. Vis. (ACCV)*, Taipei, Taiwan. Cham, Switzerland: Springer, Nov. 2017, pp. 189–204.

[113] A. Mao, M. Mohri, and Y. Zhong, "Cross-entropy loss functions: Theoretical analysis and applications," 2023, *arXiv:2304.07288*.

[114] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. 3rd Int. Conf. Learn. Represent. (ICLR)*, San Diego, CA, USA, May 2015. [Online]. Available: https://arxiv.org/abs/1412.6980

[115] T. DeVries and G. W. Taylor, "Improved regularization of convolutional neural networks with cutout," 2017, *arXiv:1708.04552*.

[116] Student, "The probable error of a mean," *Biometrika*, vol. 6, no. 1, pp. 1–25, Mar. 1908.

[117] Y. Chen, J. Li, H. Xiao, X. Jin, S. Yan, and J. Feng, "Dual path networks," in *Advances in Neural Information Processing Systems*, vol. 30. Red Hook, NY, USA: Curran Associates, 2017.

[118] I. Loshchilov and F. Hutter, "SGDR: Stochastic gradient descent with warm restarts," in *Proc. 5th Int. Conf. Learn. Represent. (ICLR)*, Toulon, France, Apr. 2017, pp. 1–16.

[119] H. Abdi and L. J. Williams, "Tukey's honestly significant difference (HSD) test," *Encyclopedia Res. Design*, vol. 3, no. 1, pp. 1–5, 2010.
[120] A. M. Anter, M. A. Elaziz, and Z. Zhang, "Real-time epileptic seizure recognition using Bayesian genetic whale optimizer and adaptive machine learning," *Future Gener. Comput. Syst.*, vol. 127, pp. 426–434, Feb. 2022.
[121] A. M. Anter and Z. Zhang, "RLWOA-SOFL: A new learning model-based reinforcement swarm intelligence and self-organizing deep fuzzy rules for fMRI pain decoding," *IEEE Trans. Affect. Comput.*, doi: 10.1109/TAFFC.2023.3285997.

JHON I. PILATAXI (Graduate Student Member, IEEE) was born in Quito, Ecuador, in 1990. He received the B.S. degree in electronics and control engineering from Escuela Politecnica Nacional (EPN), in 2014. He is currently pursuing the Ph.D. degree in electrical engineering with Universidad de Chile, Santiago. From 2015 to 2021, he was a Lecturer with EPN. His research interests include neuroevolution, human activity recognition, machine learning, and deep learning.

JORGE E. ZAMBRANO (Graduate Student Member, IEEE) was born in Latacunga, Ecuador, in 1991. He received the B.S. degree in electronics and instrumentation engineering from Escuela Politécnica del Ejército (ESPE), in 2015. He is currently pursuing the Ph.D. degree in electrical engineering with Universidad de Chile, Santiago. His research interests include biometrics as well as medical image analysis by means of image processing, machine learning, and deep learning.

CLAUDIO A. PEREZ (Senior Member, IEEE) received the B.S. degree in electrical engineering, the P.E. degree in electrical engineering, and the M.S. degree in biomedical engineering from Universidad de Chile, in 1980 and 1985, respectively, and the Ph.D. degree from The Ohio State University, in 1991. He was a Fulbright Student with The Ohio State University, where he received a Presidential Fellowship, in 1990. He was a Visiting Scholar with UC Berkeley, in 2002, through the Alumni Initiatives Award Program from the Fulbright Foundation. He was the Department Chairperson, from 2003 to 2006, and the Director of the Office of Academic and Research Affairs, School of Engineering, Universidad de Chile, from 2014 to 2018. He is currently a Professor with the Department of Electrical Engineering, Universidad de Chile. His research interests include biometrics, image processing applications, convolutional neural networks, and pattern recognition. He is a Senior Member of the IEEE, Systems, Man and Cybernetics Society and the IEEE Computational Intelligence Society.

KEVIN W. BOWYER (Fellow, IEEE) is currently a Schubmehl-Prein Family Professor with the Department of Computer Science and Engineering, University of Notre Dame, and the Director of the College of Engineering Summer International Programs. His main research interests include computer vision and pattern recognition, including biometrics, data mining, object recognition, and medical image analysis. He is a fellow of IEEE for contributions to algorithms for recognizing objects in images and a fellow of IAPR for contributions to computer vision, pattern recognition, and biometrics. He was a recipient of the IEEE Computer Society Technical Achievement Award "for pioneering contributions to the science and engineering of biometrics," and the inaugural IEEE Biometrics Council Meritorious Service Award. He served as the inaugural Editor-in-Chief for the new IEEE TRANSACTIONS ON BIOMETRICS, BEHAVIOR, AND IDENTITY SCIENCE (TBIOM).