Received August 24, 2021, accepted September 3, 2021, date of publication September 9, 2021,
date of current version September 20, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3111175

Two-Level Genetic Algorithm for Evolving Convolutional Neural Networks for Pattern Recognition

DANIEL A. MONTECINO[1], CLAUDIO A. PEREZ[1], (Senior Member, IEEE), AND KEVIN W. BOWYER[2], (Life Fellow, IEEE)
[1]Department of Electrical Engineering and Advanced Mining Technology Center, Universidad de Chile, Santiago 8370451, Chile
[2]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 Investigación y Desarrollo (ANID) under Grant FONDECYT 1191610, and Project AFB180004, and in part by the Department of Electrical Engineering, and the Advanced Mining Technology Center, Universidad de Chile.

ABSTRACT The aim of Neuroevolution is to find neural networks and convolutional neural network (CNN) architectures automatically through evolutionary algorithms. A crucial problem in neuroevolution is search time, since multiple CNNs must be trained during evolution. This problem has led to fitness acceleration approaches, generating a trade-off between time and fitness fidelity. Also, since search spaces for this problem usually include only a few parameters, this increases the human bias in the search. In this work, we propose a novel two-level genetic algorithm (GA) for addressing the fidelity-time trade-off problem for the fitness computation in CNNs. The first level evaluates many individuals quickly, and the second evaluates only those with the best results more finely. We also propose a search space with few restrictions, and an encoding with unexpressed genes to facilitate the crossover operation. This search space allows CNN architectures to have any sizes, shapes, and skip-connections among nodes. The two-level GA was applied to the pattern recognition problem on seven datasets, five MNIST-Variants, Fashion-MNIST, and CIFAR-10, achieving significantly better results than all those previously published. Our results show an improvement of 39.89% (4.2% error reduction) on the most complex dataset of MNIST (MRDBI), and on average 30.52% (1.35% error reduction) on all the five datasets. Furthermore, we show that our algorithm performed as well as precise-training GA, but took only the time of a fast-training GA. These results can be relevant and useful not only for image classification problems but also for GA-related problems.

INDEX TERMS Convolutional neural network, automatic architecture design, deep learning, genetic algorithms, neuroevolution, image classification.

I. INTRODUCTION

Neural networks (NNs) and convolutional networks (CNNs) have been inspired, since their beginnings, by mammal brain structures including those of the visual system [1]–[5]. These networks were designed to learn from examples, with the learning process strongly depending on the architecture of the network [6]–[8].

Finding optimum architectures for a given task is still an open problem, even for researchers with experience in Deep Learning (DL) [9]–[11]. In 1979, Fukushima introduced the Neocognitron [12] which was one of the first CNNs that modeled the structures of the visual system and was successfully applied to pattern recognition using space invariance [13], [14]. In 1998, LeCun introduced CNNs for the problem of handwritten digit recognition [15]. Since then, many new architectures have been created by modifying the existing ones to achieve better results in various tasks. For instance, in 2012 Krizhevky *et al.* created AlexNet [16], which was the first CNN to win the *ILSVR* [17] competition. In 2014, Szegedy *et al.* designed GoogleNet [18]. Later, CNNs had success in improving face recognition, with DeepFace (97.35% on LFW) [19] and FaceNet (99.63% on LFW) [20]. Then, He *et al.* developed

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

ResNet [21]. Subsequently, Huang *et al.*, developed DenseNet [22], and more recently, Han *et al.* developed the PyramidNet [23]. More details are provided in section II-B.

One important fact is that the mammalian visual cortex evolved gradually over millions of years to perform complex pattern recognition tasks with outstanding precision [24], [25]. The simple and elegant idea of evolution is also applicable to CNNs, in developing better architectures that best fit each problem. Evolving neural networks is called Neuroevolution, and it has been a widely addressed issue [2], [8], [26]–[30].

Neuroevolution is a sub-topic of Neural Architecture Search (NAS), which consists of searching CNN and NN architectures automatically, using various methods such as Reinforcement Learning (RL), Gradient based Optimization (GD), Bayesian Optimization (BO), and evolutionary methods [27], [31]. Neuroevolution could strongly impact the development of future technologies based on DL in two ways. The first is allowing DL researchers to find better CNN architectures for specific tasks, thus having to spend less time and effort searching for them. The second is that Neuroevolution allows nonexperts to find near optimum CNN models. Neuroevolution could be used by researchers from other fields without requiring advanced expertise in DL. For instance, neuroevolution has been applied successfully to medical images [32]–[36], speech recognition [37]–[39], emotion recognition [40], [41], scene classification [42], among others [43]–[51].

One of the first studies in Neuroevolution was done by Miller *et al.* [52] who used a Genetic Algorithm (GA) to simultaneously evolve the architecture and weights of a neural network, which was called Topology and Weight Evolving Artificial Neural Networks (TWEANNs). Then, Stanley and Miikkulainen developed NEAT [28], another TWEANN algorithm, that allowed species to evolve separately to keep variability in the population. However, the encoding did not admit large architectures. To overcome this problem, Stanley *et al.* encoded the neural network indirectly in [53], so few parameters could code larger NNs through decoding rules.

Another pioneering work was performed using GAs to design biologically inspired receptive fields [24], [54], [55] into CNNs based on Fukushima's Neocognitron Model [56]. It was shown that biologically inspired receptive fields improved the recognition capacity of the CNN in the problems of face recognition and handwritten digit recognition [24]. Another contribution was showing that a genetic algorithm could find many improved solutions for the bank of receptive fields in the first layers of the CNN compared to standard fully connected NNs [24].

TWEANN algorithms can only evolve NNs, since CNNs have significantly more weights, and therefore, it was not possible to include them in the codification of the model. To overcome this limitation, new solutions used GAs to seek only the network architecture, and the weights were determined by training using backpropagation. Thus, networks were trained with a training set, and the accuracy achieved in a validation set was used to compute the fitness. Evolving CNNs became possible through this methodology. Real *et al.* [29], and Liu *et al.* [10] were among the first to develop this evolution scheme. Both achieved state-of-the-art results using simple evolutionary algorithms with only mutations.

The search time in GA-based NAS methods, and, therefore, in Neuroevolution methods, is mainly caused by a combination of two factors. First, they are population-based methods; thus, they must evaluate many individuals to converge, and second, there is training time required to train each CNN.

The training time of a single CNN could be long depending on the dataset size, and the size of the network. Also, since GAs are population based, many individuals are evaluated. All individuals in the population are trained from scratch in GA based NAS methods. Therefore, the training time is multiplied by the number of individuals in the evolution [8].

Therefore, some studies proposed fitness approximation techniques to reduce the CNN training time, followed by the evolution time. The approximation techniques [27] included training the models with fewer epochs [2], [10], [29], [57], with fewer data [58], with lower resolution images [59], with fewer cells and feature maps in the evolved models [30], or even combining them as in [60].

However, Zela *et al.* [61] stated that, "… short training exerts on both architectural choices and hyper-parameters, resulting in a poor correlation between the performance after short and long training periods." Therefore, even if these approximations reduce the training time, they also reduce the fitness fidelity, and can generate an error in the population ranking [27], [61]. Consequently, the GA searches for the best model that fits the restriction, but which is not necessarily the best model for the global problem. This generates a fidelity-time trade-off problem in fitness calculation, since the more reliable the fitness—and hence the ranking—, the more time is required to compute it, and the longer the evolution time.

Previous publications that used a fitness approximation approach did not quantify the impact on the test error performance [2], [30], [62]–[64], i.e., the trade-off problem between computational time and fitness fidelity was underestimated or neglected. This underestimation could be due to the great computational cost required for running GA experiments without approximations, but it could also be due to limited knowledge on the effect of using fitness approximations. Therefore, the decline in performance produced exclusively by the use of these approximations is neither known nor quantified.

In this work, we aim to address the fidelity-time trade-off problem by proposing a novel GA with two levels (2LGA). The 2LGA reduces the adverse effects of using fitness approximation techniques, for example, by training with fewer epochs. One level serves to evaluate many CNN networks rapidly, while the second makes more precise evaluations to improve fitness fidelity. At the first level (1L) of the 2LGA, the fitness of the CNN models is approximated by training each CNN with fewer epochs. In contrast, the second level (2L) has only the best individuals of the first level, and the fitness is computed more precisely by training the CNN models with more epochs. Also, both levels generate offspring (as it is shown in Fig. 1), but the offspring of the second level, which is more reliable, move to the first level of the next generation to provide feedback from individuals who are reliably the best. The idea of using a 2LGA is that at the first level, the ranking can have failures, changing the real order of the population. Therefore, selecting the best CNN models, and ordering them most reliably, reduces the error produced at the top of the ranking, and only re-evaluates a fraction of the population.

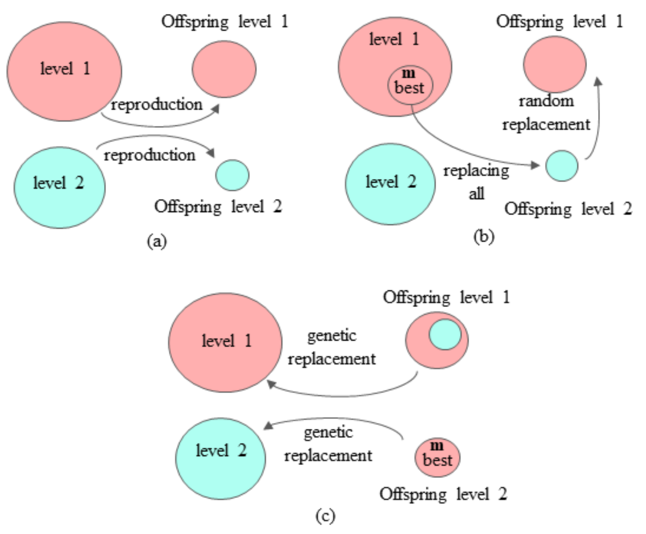


FIGURE 1. **Reproduction and combination process of GA levels. First, each level generates its offspring in (a). Then, in (b), the second-level offspring replace part of the first-level offspring, and then the best individuals of the first level take the place of the second-level offspring. Finally, at each level, the offspring replace the worst individuals in (c).**

The authors in [60] and [65] used a population that was sampled to train the individuals with higher fitness with more epochs. A single GA was used with three groups in which each group had different training epochs according to their fitness. However, genetic operators were applied only once to generate the complete population before sampling. Also, the crossover operator, which is a fundamental element for the exploitation in the GA, was not implemented, and structural parameters were not optimized by the GA.

Another important line of research in neuroevolution is the search space [8], [27]. A good search space facilitates the search for architectures, even for Random Search (RS) [10], [66]–[68]. Therefore, a good search space is essential for taking advantage of GAs. Some authors use a search space with fixed-length codification [69], [70]; thus, all the networks have the same number of layers or operations. These spaces restrict GAs, especially in problems where larger, or smaller, networks may be better solutions. By contrast, using spaces with variable-length networks, as in [2], [9]–[11], [29], [71], [72] makes coding difficult, especially for performing crossovers between individuals. For this reason, in some of these implementations, the authors only use mutation operations [10], [29]. However, avoiding crossover operations in GAs results in limited exploitation of the search-space [2], [11]. A variable-length scheme, and a crossover operation were proposed by Sun *et al.* [2] and by Johnson *et al.* [64]. However, these schemes do not consider skip-connections in the architectures, a fundamental element that prevents the vanishing-gradient problem [22], [73].

Previous work includes defining parameters for each operation, such as the kernel size in the convolution [62], [66]. Some of the studies considered operations with different parameter values, such as the convolution size (e.g. 3×3, 1×1). For example, Zoph *et al.* [66] adopted this methodology to create the widely used NASNet-space. However, one operation for each parameter may not be practical if there are many possible alternatives, such as in the number of convolution filters.

To reduce previous restrictions, we propose a flexible search space allowing variable-length architectures. Our search space includes skip-connections between operations, parameters of these operations, and hyper-parameters of the network (see Table 1). In this way, the GA has greater freedom to seek solutions reducing human bias. Furthermore, we propose a new coding, with mutation and crossover, through genes that are not expressed in the decoded CNN. Also, as in PiramydNet [23], we propose a decoding method that increases the number of filters as a function of the depth.

TABLE 1. **Overview of the encoded information in the proposed search space.**

Description		Value Domain
Structural Parameters	growth rate	$\in [1.5, 4.5]$
	number of cells	$\in \{1, 2\}$
	stem size	$\in \{32, 45\}$
Hyper-parameters	warmup	$\in [0, 0.5]$
	learning rate	$\in [0.002, 0.125]$
Node parameters	operation	$\in$ {Conv, Identity, Maxpool}
	joining method	$\in$ {CAT, SUM}
Conv. parameters	feature-map factor	$\in [0.1, 1.2]$
	activation	$\in$ {ReLU, ELU, PReLU}
	dropout	$\in [0, 0.6]$
	kernel size	$\in \{1, 3, 5\}$
Maxpool parameters	kernel size	$\in \{2, 3, 5\}$

The main contributions of our work are the following: (1) A novel 2LGA addressing the fidelity-time trade-off problem for the fitness computation in CNNs. The 2LGA reduces the adverse consequences of using fitness approximation techniques, such as training with fewer epochs. Any GA that uses approximations for fitness computation would benefit from the 2LGA, sacrificing a fraction of time, but increasing its performance significantly. (2) A novel search space that reduces the parameter restrictions, and a codification that allows the 2LGA to perform any crossover operation among CNN architectures to generate new individuals for the next generation. (3) Application of the 2LGA and the proposed search space to the MNIST-Variants, the Fashion-MNIST, and the CIFAR-10 datasets, reaching significantly better results than those previously published. (4) On the MNIST-Variants dataset, we show that the 2LGA effectively reduces the adverse aspects of fitness approximation techniques, by reaching the performance of an ordinary GA that fully trains CNNs, but taking a slightly longer time than one that approximates fitness.

II. BACKGROUND

A. GENETIC ALGORITHMS

Genetic algorithms are widely used [74], [75] in optimization and search problems, and they are inspired by evolution and natural selection processes [76]. GAs take advantage of bioinspired genetic operators, for selection, crossover, and mutation, to obtain high-quality solutions [76]. The main elements of the GA procedures are initialization, fitness evaluation, population operators (mutation and crossover), and selection [8]. In this paper, we use the concept of reproduction to encompass the processes of selection, mutation, and crossover. As some authors state, GAs are appropriate for searching CNN architecture problems since GAs are gradient-free and insensitive to local minima [2], [11], [77]. Furthermore, GAs have good global search capability [75], while gradient descent optimization methods could converge to a local minimum in cases of non-convex surfaces [78]. Moreover, GAs can even handle problems when an explicit or exact objective function is not available [77], which is the case with the CNNs architecture search problem.

B. HAND-DESIGNED CONVOLUTIONAL NETWORK ARCHITECTURES

In 1998, LeCun introduced convolutional networks for the problem of handwritten-digit recognition [15]. The architecture proposed by LeCun consisted of 3 convolutional layers with less than 20 feature maps. Subsequently, Krizhevky *et al.* used the concepts developed by LeCun to develop AlexNet [16] but increased the convolutional layers, the feature maps, and kernel sizes. The authors also introduced the ReLU activation function and Maxpool layers. Later, Szegedy *et al.* designed GoogleNet [18], a network with parallel convolutions with different kernel sizes. They also increased the depth of the network (number of layers), and its width (feature maps).

At that point, an increase in the depth of the networks proved to have better results, but larger networks were not feasible to train due to the vanishing gradient problem [79], [80]. To avoid this difficulty, He *et al.* developed a residual block that connects the input and the output of a block by adding them [21]. They could then train networks of 151 layers, avoiding the vanishing gradient problem. Subsequently, Huang *et al.*, (2017) extended the concept of skip connection, and developed a Dense block, which connects the outputs of each layer to every subsequent layer [22].

Recently, Han *et al.* developed PyramidNet [23], which gradually increases the feature map dimension with a linear (1) or exponential (2) function of the depth as follows:

$$D_k = \begin{cases} 16 & if\ k = 1 \\ \lfloor D_{k-1} + \frac{\alpha}{N} \rfloor & if\ 2 \leq k \leq N+1 \end{cases} \tag{1}$$

$$D_k = \begin{cases} 16 & if\ k = 1 \\ \lfloor D_{k-1} \cdot \alpha^{\frac{1}{N}} \rfloor & if\ 2 \leq k \leq N+1 \end{cases} \tag{2}$$

where D_k is the number of feature maps of the k-th residual unit, N is the total number of residual units, and α is a growth factor.

In addition, they proposed a residual unit with an identity-mapping shortcut, and since the branches of the residual unit have different feature map dimensions, they applied zero-padding at the channel axis to match the features in this dimension.

Although handcrafted CNN architectures have improved greatly, the process has taken years and enormous effort. Therefore, designing architectures automatically could speed up the development of DL research.

The proposed search space includes concepts such as the number of feature maps, activation function, network size, and gradual expansion, among others. Therefore, the results of previous research on CNN architectures serve as a base for our search space. The proposed search space allows the GA to find a combination and interconnections of elements that will yield improved results.

C. NAS METHODS

Various search strategies are used to explore the space of CNN architectures, such as Reinforcement Learning, Gradient Descent methods, Bayesian Optimization, and evolutionary methods [8], [27], [31].

In the RL approach, a controller samples CNNs from the search space seeking to optimize a reward. The controller's feedback is a reward to improve decision making, and choose an increasingly better model. The reward is computed by training the sampled model from scratch, and computing its validation accuracy [81], [82]. A pioneer in RL methods was Zoph and Le [81], who used a recurrent neural network as a controller, and the policy gradient algorithm to train this controller. Other approaches, such as MetaQNN [82], train the controller using Q-learning with an e-greedy exploration strategy, and experience replay. However, RL methods tend to require more computational resources than evolutionary algorithms [11], [83]. For example, Zoph *et al.* needed 28 days and 800 GPUs [81], while MetaQNN required ten days and 10 GPUs [82].

The GD-based methods model CNN architectures using continuous variables to optimize the search using gradient descent. A super graph or super network is built that has a predetermined number of nodes, but each node contains all the available operations [31], [84]. The objective is to find a subgraph with the optimal connections between node operations. One of the most well-known GD-based methods is DARTS (Differentiable Architecture Search) [84], in which a set of alpha variables weights every possible connection between operations. Network weights and alpha weights are optimized during training. Although this approach reduces the search time, building a super graph requires expertise [8], and a GPU with enough memory. Also, the size of the super graph grows linearly with the number of operations [31], and is, therefore, not so suitable for including multiple operations in the search, or for including parameters for each operation. GA methods are preferable when a smaller GPU is available; when the search space is complex and has many parameters to optimize.

Bayesian Optimization [85] is an optimization method that does not require explicit knowledge of the objective function, similar to GAs, and it is, therefore, widely used for hyperparameter optimization. The most common approach of BO-based methods applied to NAS is to model the validation accuracy of CNNs as Gaussian processes [86], [87]. However, BO methods cannot effectively handle variable-length encodings and time scales cubically with the number of observations [31].

Evolutionary Computation (EC) is population-based search and is frequently used as an optimization method. EC is based on the mechanisms of biological evolution [8]. Genetic Algorithms, Differential Evolution (DE), and Particle Swarm Optimization (PSO) are among the EC methods [8]. Some EC methods for NAS are PsoCNN [62], and IPPSO [9], which use a PSO approach to optimize the architecture of CNNs. DECNN [63] and evoCNN [2] are also NAS methods, that use DE and GAs, respectively. HGAPSO [57] is a hybrid two-level method that combines PSO and GA to search for the optimal CNNs. One level encodes and optimizes the parameters of the CNN, and the other level optimizes the connections between layers. The latter differs from our proposal since our method has the same coding at both levels, and the difference between levels is in the fitness computation.

1) NAS BENCHMARK DATASETS

We use seven datasets for testing our method (see Fig. 2). The first five datasets are MNIST-Basic (MB), 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). They compose the MNIST-Variants (MNIST-V) datasets. MNIST-V are datasets created by Larochelle *et al.* [88], that are modified versions of the original MNIST. MNIST-V datasets are considerably more challenging than the original MNIST, since alterations make the classification more difficult, and they have only 20% of the training images of the original MNIST, with five times more test images (see Table 2). The sixth dataset is Fashion-MNIST [89]. This dataset contains 70,000 grayscale images of fashion products, 60,000 for training, and 10,000 for testing. The images are of size 28 × 28, and are divided into 10 categories. The seventh dataset is the CIFAR-10 [90], which has size 32 × 32 color images (3 channels) of common objects. CIFAR-10 has 50,000 training images and 10,000 test images.

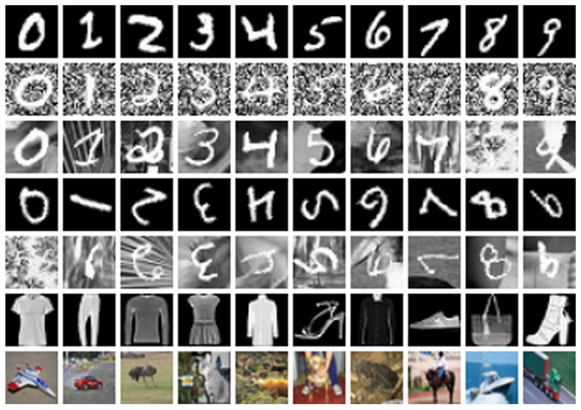


FIGURE 2. Examples of images of the six datasets. From top to bottom, the datasets are, Basic MNIST (MB), MNIST with random backgrounds (MRB), MNIST with background images (MBI), MNIST with rotated digits (MRD), MNIST with rotated digits and background images (MRDBI), fashion MNIST, and CIFAR-10.

TABLE 2. Overview of the datasets used to evaluate neuroevolution methods and to assess our proposed 2LGA method.

Dataset	train-val	test	description
MB	12,000	50,000	Redistribution of classic MNIST
MBI	12,000	50,000	MB with images as background
MRB	12,000	50,000	MB with noise as background
MRD	12,000	50,000	MB with rotated digits
MRDBI	12,000	50,000	MB with rotated digits and images as background
Fashion-MNIST	60,000	10,000	fashion products
CIFAR-10	50,000	10,000	Common objects

III. THE PROPOSED METHOD

We propose a 2LGA to balance fitness fidelity and computational time. In the proposed method, a GA is applied on both levels, and therefore each level has a population and genetic operators. On the first level, individuals have shorter training to reduce the search time, whereas, on the second level, the best individuals of the 1L are instructed more precisely by training them longer.

We used existing and widely used techniques (such as tournament selection, CNN training procedures, Gaussian Mutation, and integer mutation, among others) in some parts of our method since these are used in many GAs [8]. We, therefore, referenced them to complete the description of our method, and to make it reproducible. However, the contributions of our work, described in section I, are independent of them.

A. TWO-LEVEL GENETIC ALGORITHM

The algorithm starts by initializing the population of the 1L randomly, and then evaluating this population with a limited number of training epochs (short training). Subsequently, the GA selects the parents through tournament selection, which consists of randomly sampling K individuals from the population, and selecting the one with the best fitness [74]–[76]. Several tournaments are implemented to generate the next parent population. Then the GA generates offspring through crossover and mutations. Finally, the algorithm replaces part of the initial population with the offspring. This level evolves as a common GA until the 2L is generated.

To save computation time, the 2L of the GA is evaluated only every Q generations. When the 2L is generated for the first time, the best N individuals of the 1L are selected, and N becomes the population size of the 2L. Then, the fitness of the selected individuals is computed using a larger number of training epochs (long training). In subsequent generations, when the 2L is evaluated, both levels perform reproduction, as is shown in Fig. 1. Since both levels generate their offspring, the offspring of the 2L randomly replaces part of the 1L offspring. Every Q generation, the best $m(< N)$ individuals of the 1L become the 2L offspring. Genetic replacement is performed on both levels, where the worst individuals are replaced.

We place the 2L offspring in the 1L, instead of in the 2L, for two reasons: to provide feedback to the 1L about the 2L population, and to limit the 2L to the best of the 1L. In this way, individuals are accurately evaluated if they prove to be good candidates in the 1L, and not because they are offspring of good parents. In generations where the 2L is not evaluated, between Q generations, the 1L evolves as a common GA, with selection, reproduction, and replacement. Evolution ends when a maximum time or a maximum number of generations is reached as in [11], [57], [63], [83], [91], [92].

B. SEARCH SPACE AND ENCODING

It has been stated that a good search space allows search algorithms to find good solutions easily, even for random searches [10], [66]–[68]. Also, coding is essential for GAs to allow exploration of the search space.

Several works have proposed ways of coding the architecture of CNN networks [2], [11], [57], [62], [63], [69], [70], [83]. Coding can be variable-length or fixed-length and can code the entire network, or just one cell which is stacked to build the network. In this work, we adopted a coding method that belongs to the variable-length cell category [10], [11], [30], [57], [62], [63], [93]. We, however, propose a novel encoding, which will be explained in the section III-B4, that contains genes that may not be expressed, as can happen with real genes.

1) CNN ARCHITECTURE

In our cell coding scheme, shown in Fig. 3, the chromosome of an individual is a variable-length list of nodes. Each node

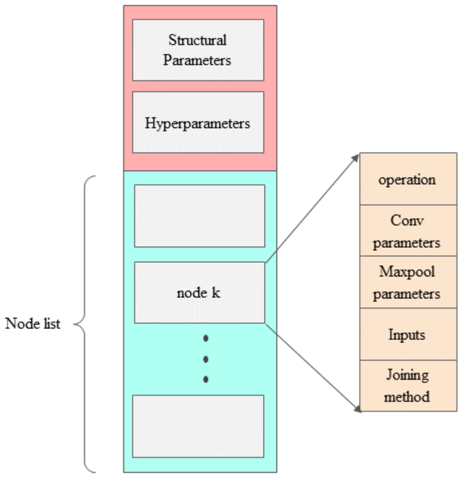


FIGURE 3. Representation of the chromosome of an individual. The chromosome has information about structural parameters, training hyper-parameters, and parameters concerning each node.

has information about its operation, its joining method for inputs, and its input connections.

Each node defines the operation that it is using, as is shown in Table 1, "Operation" in "Node parameters". And each node encodes the parameters for all operations, e.g., the activation function in convolutions, and defines its joining method for inputs as concatenation, or sum. The joining method for nodes with only one input, and information on unused operations is not expressed. By using unexpressed genes, performing crossover is direct, even if the nodes have different operations.

Input connections of a node can be with any of the previous nodes on the chromosome list, including the input node (shown as a green rectangle in Fig. 4). As a restriction, the k-th node has at least one input, and a maximum of k inputs when all the previous nodes are connected. In Fig. 4, node 2 (k=2) has 2 connections, one from node 0 (input node), and another from node 1. The connections are expressed as arrays of 0s and 1s, as shown in Fig. 4, in which a 1-value in the k-th position means a connection with the k-th node. In the example in Fig. 4, node 4 (k=4) has an array of (0, 1, 0, 1), in which 1-values appear in indexes 1 and 3. Therefore, the node has a connection with nodes 1 and 3.

The number of cells per block, and the number of blocks in the CNN network are defined as follows: Usually, these parameters are defined manually in Neuroevolution algorithms, but we include them within the coding, so that the numbers of cells and blocks are also optimized by the 2LGA. Also, it is common to apply the first convolution, called the stem, before building the cells.

In the same way, the number of stem filters is usually defined manually, but we also include this number as a parameter in the 2LGA. These parameters, the stem size, the number of blocks and number of cells, control the size of the network, and we therefore call them structural parameters (see Table 1). Besides these structural parameters, we design

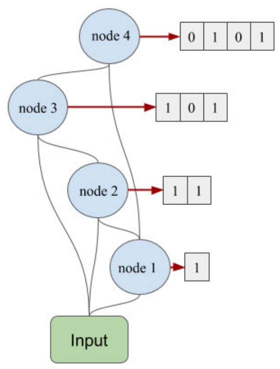


FIGURE 4. **Example of connection coding. The inputs are coded for each node in an array, as is shown with red arrows. For each array coding that the node inputs, the *k*-th value indicates the presence (value 1) or absence (value 0) of connection with the *k* node. Node *i* may have a minimum of one, and a maximum of *i* input connections.**

each CNN network with a gradual increase in the number of filters, as in PiramyNet [23]. We can then define the growth rate (parameter γ) that determines the increase in feature maps at the output of a block, with respect to its input feature maps. The number of output feature maps of a block is therefore given by (3):

$$\mathrm{F_{out}}(\mathrm{block_i}) = \gamma \cdot \mathrm{F_{in}}(\mathrm{block_i}) \tag{3}$$

where, $\mathrm{F_{out}}(\cdot)$ and $\mathrm{F_{in}}(\cdot)$ are functions that represent the number of output and input feature maps of their argument.

Given the growth rate, each node that performs a convolution has a defined number of output feature maps (4). We also include the growth rate parameter in the coding:

$$\mathrm{F_{out}}(\mathrm{node_k}) = F_0 \cdot (\gamma^{N_{\mathrm{blocks}}})^{k/N} \tag{4}$$

where $\mathrm{node_k}$ is the k-th node, F_0 is the stem size (initial number of feature maps), N_{blocks} is the number of blocks of the network, and N is the total number of operations of the net. Note that $\gamma^{N_{\mathrm{blocks}}}$ is the total growth of the network with respect to the stem size.

With the configuration described, all convolutions have a defined number of output feature maps, that depend on the growth rate factor γ, the number of cells, and the number of blocks. For the purpose of adding flexibility in defining the shape of the network, we added a second multiplicative factor f_k in the coding for each node, which can be in the range between 0.1 and 1.2 (see Table 1). Then, the number of output feature maps of a node is computed using (5):

$$\mathrm{F_{out}}(\mathrm{node_k}) = f_k \cdot F_0 \cdot (\gamma^{N_{\mathrm{blocks}}})^{k/N} \tag{5}$$

The shape of the network therefore depends on the number of cells, the number of blocks, the stem size, the growth rate, and each multiplier factor f_k.

It is worth noting that convolution layers are composed of a Batch-Normalization [94] and the Convolution. Also, to join operations of different depths (feature maps), a 1×1 convolution is used to project tensors with fewer feature maps to have the same feature maps as the biggest one.

2) HYPER-PARAMETERS

Since the optimal training parameters depend on each network architecture, we code them in the 2LGA. Therefore, the 2LGA will look for both the best architecture and its training parameters, thus avoiding optimizing an architecture for a set of training parameters. As a CNN network training method, we adopt a schedule learning rate with warmup [95] and linear decay, similar to [96]. The warmup implemented increases linearly from zero to a maximum learning rate during the first W epochs. Our GA therefore encodes the maximum learning rate and the number of warmup epochs as a proportion of the total epochs T. For instance, if ω is the proportion, the number of warmup epochs is given by (6):

$$W = \omega \cdot T \tag{6}$$

3) MUTATION OPERATION

We implemented two types of mutations: one that modifies the number of nodes (size mutation), and another that modifies the node parameters, (node mutation). Size mutation operates by either adding a new node, or by removing an existing one. For node mutation, all the information coded in the node can mutate (see Fig. 3). The node parameters are shown in Table 1, and the parameters are available for all operations. If the parameter to mutate is qualitative, a random value is chosen from a set of possible options. Mutation methods for real and integer values are Gaussian and integer values, respectively. These rules for mutation are also applied to structural and hyper-parameter mutations.

Adding a new node, as is shown in Fig. 5, is performed by choosing one node randomly from the existing ones in the node list shown in Fig. 3. Then, the selected node is duplicated, and the new node with mutations in its parameters is placed just before the original one. The inputs from the original node are transferred to those of the new node, and the output from the new node becomes the input to the original one. This process is shown and explained in Fig. 5. An example of removing a node is shown in Fig. 6, for a node with only one input and one output. The removed node is replaced with a connection from the previous node output to the next node input. If there are no nodes with these characteristics, a random connection of any node is removed, as long as there are still connections available.

4) CROSSOVER OPERATION

Two types of crossover operations between two different individuals occur simultaneously. The first one is combining two nodes, and the second, combining two lists of nodes. All the nodes have the same type of information, such as operation type, convolution parameters, and Maxpool parameters, as is shown in Fig. 3. Therefore, performing crossover between two nodes of two individuals can be accomplished by combining parameters, one by one, from the two nodes. In this crossover, the parameters of the operations are

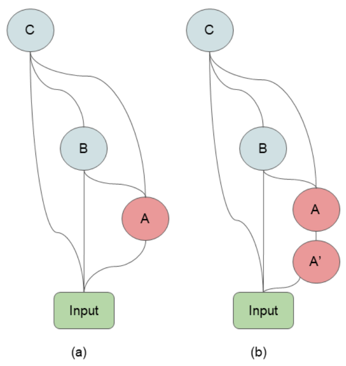


FIGURE 5. Node addition scheme in the mutation process. First (a), a node is chosen at random (red node A). Then in (b), it is duplicated, mutated (A'), and added right before the original. The original node keeps its outputs, while the new node acquires the inputs from the original node.

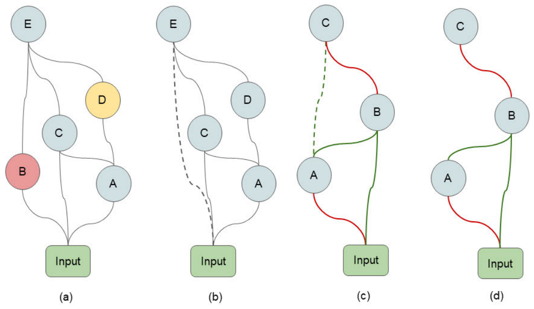


FIGURE 6. Node removing scheme in the mutation process. As in (a), when there are nodes available to remove (e.g. nodes B and D), one of them is selected randomly (node B marked in red). Then, in (b), it is removed and replaced with a connection (dotted line). When nodes are not available to remove as in (c), a detachable connection (green line) is selected randomly. The selected connection (dotted green line) is removed, resulting in (d).

recombined. In this way, the node keeps the information of unexpressed operations and the next generation inherits that information. Therefore, the information of all the operations evolves and is not lost by mutation or crossover.

Although all the nodes have the same information type, nodes in different positions, such as nodes 1 and 3 in Fig. 4, have input coding arrays of different sizes. As a consequence, we combine only nodes in the same relative position. For combining two lists of nodes, they are aligned, and the nodes in the same relative position are combined, as is shown in Fig. 7. Since the two lists can be of different sizes, as in the example shown in Fig. 7, the remaining nodes of the largest list are attached to the resulting chromosome, with a probability of 0.5.

Structural parameters and hyper-parameters from Table 1 are also combined. If the parameter has a qualitative value, one of each is selected randomly. If the parameter is numeric, crossover is performed according to (7) [97]:

$$\mathrm{val}_{\mathrm{cross}} = b \cdot \mathrm{val}_1 + (1-b) \cdot \mathrm{val}_2 \tag{7}$$

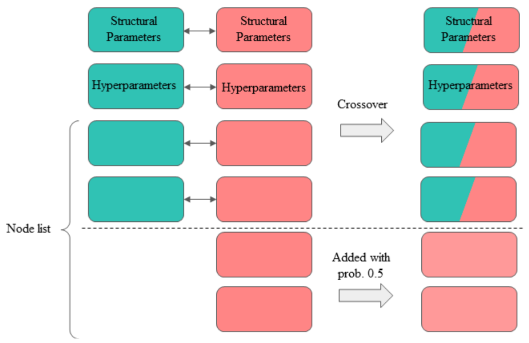


FIGURE 7. An example of crossover between chromosomes of different lengths. The chromosomes at the left are combined node by node, including their structural parameters and hyper-parameters. Then, the remaining nodes of the largest chromosome are attached to the new one, with a probability of 0.5.

where val_1 and val_2 are the numerical values to perform crossover, $\mathrm{val}_{\mathrm{cross}}$ is the new value, and b is a random value between 0 and 1. If the values are integers, $\mathrm{val}_{\mathrm{cross}}$ is rounded to the nearest integer number.

5) FITNESS CALCULATION

For the fitness computation, each CNN model is trained from scratch with the training set. At each training epoch, the error rate is computed in the validation set (val-error-rate), and it is saved in a history vector. Once the training ends, the minimum error rate of the last 30% of the epochs is used in the fitness. In addition, training time in seconds is measured, and a temporal component is included in determining the fitness value. The fitness value is obtained by (8).

$$\mathrm{Fitness} = \mathrm{val}_{\mathrm{error}} + \frac{\log(T_{\mathrm{time}}/N_{\mathrm{epochs}})}{1000} \tag{8}$$

where $\mathrm{val}_{\mathrm{error}}$ is the error calculated in the validation set, and T_{time} is the total training time. The temporal component in the fitness function was introduced to prevent the 2LGA from searching for networks with large training times. The average epoch time is used by dividing the total training time by the number of epochs N_{epochs}. We use the logarithm to avoid over-penalizing larger networks. The factor $1/1000$ was used to assign the temporal component an order of magnitude less than the error rate.

To train CNN models, we use SGD and Adam optimizer, with a schedule learning rate with warmup and linear decay, as in [57], [62], [93]. Therefore, in the first W warmup epochs, the learning rate increases linearly from zero to a maximum value. Then, the LR decays linearly from the maximum value to zero. Both the value of the maximum learning rate, and W are encoded, so that they change for each individual in the 2LGA population.

We use label smoothing of 0.1 as regularization [98] and we do not use data augmentation (DA) in the evolution. As in previous work, once the GA finishes, the individual with the best fitness is trained from scratch with the training and validation set, and with more epochs than those used for the fitness computation [11], [57], [83], [93]. The accuracy achieved in the test set is reported for each experiment. In the Fashion-MNIST experiments, we also train the winner individual of the evolution with simple DA. This DA is commonly used for this dataset and consists of random flip (RF) and random crop (RC) [16].

IV. EXPERIMENT DESIGN

Although the proposed method can be applied to a variety of problems, such as regression or detection, we chose image classification because it has been widely tackled by neuroevolutionary algorithms [2], [10], [11], [30], [62], [83].

To evaluate our 2LGA, we performed three groups of experiments. In the first group, we compared the performance of our method with other evolutionary algorithms in seven image classification problems (MNIST-V, Fashion-MNIST, and CIFAR-10). We compared our results to those of state-of-the-art (SOTA) studies on the same datasets. We compared them with PsoCNN [62], IPPSO [9], HGAPSO [57], DECNN [63], and EvoCNN [2] for MNIST-V; with REMNET [99], DeepSwarm [100], EvoCNN [2], PsoCNN [62], with some hand designed CNNs for Fashion-MNIST; and with Johnson [64], SOBA [71], CNN-GA [83], HGAPSO [57], and AE-CNN [101] for CIFAR-10.

For the MNIST-V, Fashion-MNIST, and CIFAR-10 datasets, we compared our method with other Evolutionary NAS algorithms, and for Fashion-MNIST, we also included the best handcrafted methods. Most handcrafted methods use a simple DA technique composed of RF and RC. We, therefore, make two different comparisons: using RF and RC, and without using any DA. For CIFAR-10, we chose the SOTA results extracted from [8] that use the common DA techniques for this dataset (padding 4 pixels, RC, and RF). Other methods include more elaborated training techniques and data augmentation methods [5], [30], [72], [102].

In the second group of experiments we compared our 2LGA with two ordinary GAs (O-GAs). In this way, we could highlight the benefits of using 2LGAs in terms of time and accuracy. The third group of experiments consists of evaluating our search space by comparing the search using Random Search algorithms, and GAs.

In the next section, we describe the experiments performed, the parameters used for the 2LGA, and the training parameters for each model.

A. ORDINARY GENETIC ALGORITHMS AND RANDOM SEARCH

We compared the proposed 2LGA with two ordinary GAs. The first O-GA gives short training to all the models, serving as a lower bound for the 2LGA in terms of evolution time. The second O-GA has a long training for all the models, and therefore is a reference for the accuracy the 2LGA could achieve. In both O-GA experiments the same search space is used as in the 2LGA. Also, except for training epochs and population size, all the parameters in the 2LGA and O-GAs were the same. Additionally, we set the population size of the O-GAs to evaluate the same number of networks as the 2LGA during the complete evolution.

We also performed two Random Search experiments. As our 2LGA conducts training using two different epochs, we performed one RS for each of them; that is, one RS that gives a short training to all the models, and another that trains them longer. In addition, to have a fair comparison with our GA algorithm, we ran RS experiments until they reached the maximum evolution time of the 2LGA experiments applied to the same dataset. RS optimizes the same fitness (8) as the GAs.

In the RS we performed the following procedure to generate new individuals: First, we selected the number of nodes of the cell randomly. The possible number of nodes was between 1 and 7. Here, the upper bound 7 was selected as the mean number of nodes from all the winners of evolutions on the same dataset, plus one. Second, for each node, we randomly selected node parameters, convolution parameters, and maxpool parameters. We sampled the parameters uniformly within the bounds in Table 1. Third, for each node in the i-th position, we generated random connections, making sure that all nodes had at least one input and one output. Then, we randomly selected structural parameters and hyper-parameters from Table 1.

The experiments for O-GAs and RS were performed on the MRDBI dataset because it is the most difficult among the MNIST-V sets. We ran each experiment five times, and we reported the mean and lowest error and performed statistical tests to compare the results as in [57], [63], [103]–[105].

B. PARAMETER SETTING

For the first level of our 2LGA, we chose genetic parameters commonly used in previous publications [11], [83]. We set the population size, and the number of generations to 20, the crossover probability to 0.9, the mutation probability to 0.1, and the percentage of elitism to 25% [30].

For the 2L, we used a population with eight individuals, which is smaller than that for the 1L, and a percentage of elitism of 50%. The K value for tournament selection of the first and second levels is 25% of each population (5 and 2, respectively). Additionally, the 2L was evaluated every three generations (Q=3), and it was also evaluated at the end of the evolution, before obtaining the best individual of the population.

The search space parameters —or its boundaries— are shown in Table 1. To initialize the population, we set the maximum number of operations (or nodes) to five. Nevertheless, individuals can mutate and increase their size during evolution.

Training parameters were set as follows: we used 80% of the training set for training, and 20% for validation; the batch size was 128; the numbers of epochs of the first and second level were 18 and 54, respectively. The optimizer was Adam [106], and the schedule learning rate is described in Section III-B5. The individuals were trained with the training set, and the error in the validation set was used to compute the fitness. The best individual of the evolution was retrained with the train and validation sets for 90 epochs, and the error rate in the test set is reported as in [2], [57], [62], [63]. For CIFAR-10, the number of epochs were 36 and 108 for 1L and 2L, respectively, and 600 for testing. For better comparison, we did not use label smoothing, and we removed the temporal component of (8). All other parameters were unchanged.

V. RESULTS

In this section, we present the results of the experiments described of our proposed 2LGA. We compare the 2LGA results to the best of those previously published using the same datasets. In addition, we show the results of the comparison of our 2LGA with O-GAs and with RS algorithms. As in previously published studies, [2], [57], [62], [63], all the results presented in this section were achieved in the test set, which is only used to evaluate the evolution-winning individual. Also, all the experiments were performed five times; we report the best and the mean values here.

A. RESULTS ON BENCHMARK DATASET EXPERIMENTS

We assessed our 2LGA on the MNIST-V and Fashion-MNIST datasets. Table 3 shows the results (% error) on MNIST-V subsets: MB, MRB, MBI, MRD, and MRDBI. As can be seen on Table 3, our 2LGA achieved significantly better results than those previously published. Specifically, our 2LGA achieved an improvement of 17.57% (0.13% error reduction) on the MB subset, 24.58% (0.44% error reduction) on the MRB subset, 33.68% (0.64% error reduction) on the MBI subset, 36.87% (1.32% error reduction) on the MRD subset, and 39.89% (4.2% error reduction) on the MRDBI subset. It is worth noting that the greatest improvement, 39.89% (4.2% error reduction), was achieved on the most difficult dataset, MRDBI.

TABLE 3. Comparison of test errors among the 2LGA and the previously published best results in MNIST-variants (lower is better).

Method	MB	MRB	MBI	MRD	MRDBI
PsoCNN (mean) [62]	–	2.53	2.40	6.42	20.98
PsoCNN (best) [62]	–	1.79	1.90	3.58	14.28
IPPSO (mean) [9]	1.21	–	–	–	33.00
IPPSO (best) [9]	1.13	–	–	–	34.50
HGAPSO (mean) [57]	0.84	–	–	–	12.23
HGAPSO (best) [57]	0.74	–	–	–	10.53
DECNN (mean) [63]	1.46	3.56	8.69	5.53	37.55
DECNN (best) [63]	1.03	3.46	5.67	4.07	32.85
EvoCNN (mean) [2]	1.28	3.59	4.62	5.46	37.38
EvoCNN (best) [2]	1.18	2.80	4.53	5.22	35.03
Ours 2LGA (mean)	0.65	1.37	1.34	2.43	6.73
Ours 2LGA (best)	0.61	1.35	1.26	2.26	6.33

Table 4 shows the results on the Fashion-MNIST dataset for our 2LGA and for those published previously. Among the previously published results, the first five methods are

TABLE 4. Comparison of test error of the 2LGA with previously published results in Fashion-mnist.

Method	% Error without DA	% Error with DA (RF + RC)
GoogleNet [18]	6.30	–
ResNet18 [21]	–	5.10
DenseNet [22]	–	4.60
MobileNet[a] [107]	–	5.00
WideResNet-28-10[b] [108]	–	4.63
REMNet [99]	5.54	–
EvoCNN (mean) [2]	7.28	–
EvoCNN (best) [2]	5.47	–
DeepSwarm (mean) [100]	6.75	–
DeepSwarm (best) [100]	6.44	–
PsoCNN (mean) [62]	5.90	–
PsoCNN (best) [62]	5.53	–
Ours 2LGA (mean)	4.82	4.62
Ours 2LGA (best)	4.76	4.56

The results in the second column are without any Data Augmentation (DA) during training. In the third column, DA with Random Flip and Random Crop is shown. for 2LGA, only the winner individual is trained with RF+RC
[a]On this implementation, only RF was used.
[b]In the early version of the dataset, some images were duplicated in the test and training sets. Therefore, the results reported by the original paper [109] are better. The results reported here are from [110], in which the experiments were repeated with the last version of the dataset.

hand-crafted CNNs, and the next four are from evolutive algorithms. The first two, REMNet and evoCNN, use genetic algorithms, whereas DeepSwarm and psoCNN use Particle Swarm Optimization algorithms. The evolutive methods do not use any kind of DA, but the nonevolutionary approaches do. Thus, we present results of our 2LGA without any DA in the second column, and with RF and RC DA in the third column. Therefore, Table 4 shows methods with the same training conditions so that the results could be comparable. Other DA techniques were not used for this comparison. As can be seen on Table 4, our 2LGA also achieved better results than those published previously. Specifically, our 2LGA achieved an improvement of 12.98% (0.71% error reduction) with no DA, and 0.87% (0.04% error reduction) with RF+RC. Also, Table 4 shows that previously published evolutive methods reached lower error rates than handcrafted methods if no DA was used. Conversely if DA was used (RF+RC), previously published hand-crafted methods reached lower error rates than evolutive methods of competitors. However, our 2LGA achieved the lowest test error.

In Table 5 we show the results from the CIFAR-10 experiments. Competitors in Table 5 used common DA techniques. As can be seen, our method achieves the SOTA results on this dataset.

We also performed a comparison with the results from DARTS [84], as it is one of the most relevant GD-based NAS methods, and the implementation is available online. The implementation of DARTS provides the training procedure, with several additional methods for data augmentation and training techniques including cutout [111], auxiliary-tower, schedule-drop-path, and long training for 600 epochs [84]. We, therefore, performed two experiments to be able to compare our results to those of DARTS: We (a) trained our 2LGA

TABLE 5. Comparison of test errors (%) among the 2LGA and the previously published best results in CIFAR-10 (lower is better).

Method	Test Error (%)
Johnson [64]	15.15
SOBA [71]	4.78
CNN-GA [83]	4.78
HGAPSO [57]	4.37
AE-CNN [101]	4.30
Ours 2LGA	3.95

and DARTS models with long training, i.e., 600 epochs, with no data augmentation and no additional training techniques, and (b) trained the 2LGA model with data augmentation, and additional training techniques, including long training. We finally compared the results of the two experiments (a-b) with those computed/published regarding DARTS [84].

In experiment (a), with long training, no data augmentation, and no additional training techniques, the DARTS model achieved a 4.83% error in the test set of CIFAR-10, while our 2LGA model reached an error of 3.95% on the same dataset. Therefore, our 2LGA model achieves 0.88 less test error (an 18% improvement) compared to DARTS. In experiment (b), with data augmentation and additional training techniques, our 2LGA model achieved a 2.68% test error, while the result for DARTS was 2.76% test error [84] (an 2.9% improvement). Furthermore, the best results using DARTS were obtained after the search when the researchers increased the model size (number of cells and feature maps) [84]. One of the advantages of the 2LGA is that the results are obtained automatically without intervention by the researchers.

B. RESULTS ON ORDINARY GENETIC ALGORITHMS AND RANDOM SEARCH EXPERIMENTS

Table 6 and Fig. 8 show the main results of O-GA experiments, using two different numbers of training epochs. As can be seen, the O-GA that gives the population short training with 18 epochs (O-GA+18eps) achieves the worst mean error, but the results are reached in the shortest time. In contrast, the O-GA that trains the population longer with 54 epochs (O-GA+54eps), achieves the same results as our 2LGA, surpassing the state-of-the-art shown on Table 3. But the O-GA+54eps requires 3 times more time than our 2LGA method.

TABLE 6. Comparison of 2LGA with two ordinary genetic algorithms (O-GAs) and with two random search (RS) algorithms.

Experiment	% Error (best)	% Error (mean)	Fitness (mean)	Evolution time (hrs.)
O-GA + 18 eps	7.12	7.34	7.93	5.8
O-GA + 54 eps	6.45	6.78	7.05	19.9
2LGA (18, 54 eps)	6.33	6.73	7.13	6.6
RS +18 eps	8.13	8.42	8.96	9.1
RS + 54 eps	7.04	7.25	7.49	9.2

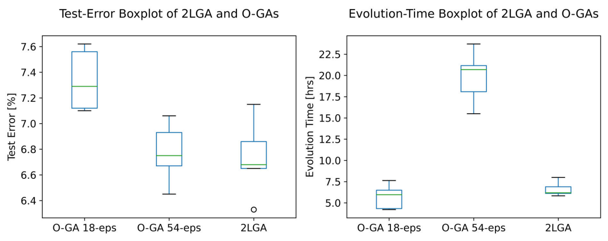


FIGURE 8. Box-plots of test error (a) and evolution time (b) for the O-GA experiments. Each experiment was repeated 5 times on the MRDBI dataset.

Further, statistical tests were performed with these experiments using the ANOVA test to determine if differences among test-error results and evolution times were statistically significant. We used the Tukey HSD test [112] to verify which results were statistically different. The ANOVA test applied to the test-error results yielded an F-value = 8.37 and p-value = 0.005 (<0.05), and for evolution time, F-value = 73.6 and p-value = 1.8e-7. Thus, it was determined that there are significant differences among group results in both test error and evolution time.

For the Tukey HSD test, we compared the test-error and evolution times of the experiments. The Tukey HSD results are shown on Table 7 and Table 8 for test-error and evolution time, respectively. The Tukey HSD results in test errors (Table 7) show that the 2LGA error is significantly different from the error of the O-GA+18eps, but it is the same as the error of the O-GA+54eps. Conversely, the Tukey HSD results in evolution time (Table 8) indicate that the result reached by the 2LGA is significantly different from that of the O-GA+54eps, but it is not different from that of the O-GA+18eps.

TABLE 7. Tukey hsd test results of test error with two ordinary genetic algorithms (O-GA) and a 2LGA.

Group1	Group2	Mean-diff (% error)	Reject
O-GA + 18eps	O-GA + 54eps	-0.566	True
O-GA + 18eps	2LGA	-0.604	True
O-GA + 54eps	2LGA	-0.038	False

TABLE 8. Tukey hsd test results from evolution time among two O-GA and a two-level genetic algorithm.

Group1	Group2	Mean-diff (hrs.)	Reject
O-GA + 18eps	O-GA + 54eps	14.1	True
O-GA + 18eps	2LGA	-0.9	False
O-GA + 54eps	2LGA	-13.2	True

Therefore, *the 2LGA achieves the same performance as a GA with 54 epochs but taking only a similar time to that of the O-GA with 18 epochs.*

We also performed two RS experiments, one by using 18 epochs, and the second using 54. The experiments were stopped when they reached the maximum evolution time of our 2LGA.

The results are shown on the last two rows of Table 6. RS achieved a SOTA on the MRDBI dataset, by achieving an improvement of 33.1% (3.5 error reduction) with respect to the best previously published result on the HGAPSO (see Table 3) when models were trained with 54 epochs. A 22.8% improvement (2.4 error reduction) was reached when the model was trained with 18 epochs. Since RS is one of the simplest search algorithms, the improvement achieved is primarily due to the proposed search space. Also, RS+54 gets better results than the O-GA+18 epochs. This suggests that precise training is a key to achieving good results, and also supports the hypothesis that RS is a difficult baseline to overcome, despite being one of the simplest search algorithms [10], [66]–[68]. On Table 6, we show the mean fitness of the winners for each method. The fitness is greater than that estimated with the mean test error because, when testing, models are customarily trained using both the training and validation partition of the data sets, and with more epochs (90 epochs) [11], [57], [83], [93]. Also, methods with short training (O-GA+18 and RS+18) tend to have greater validation error. As expected, O-GA+18 outperforms RS+18, and O-GA+54 (and 2LGA) outperforms RS+54 since GA methods are designed to optimize the fitness.

C. ANALYSIS OF THE SELECTION OF THE BEST INDIVIDUALS

In this section, we compare the rankings made by the 2LGA and that achieved by an O-GA with long training of all individuals, on the same population. For this purpose, we gave a long training to the individuals for each generation of the 1L that had been given short training during the evolution of the 2LGA. Then, we computed Kendall's rank correlation coefficient [113], as in [67], [114], between the ranking of the short-trained 1L and the ranking of the long-trained 1L. Fig. 9 shows how the ranking changes when using our proposed 2LGA instead of using an O-GA with long training.

In Fig. 9 lines that connect points indicate the change in the position of the individual after long training. For example, the individual that is in the first position when it is given short training becomes the eighth individual when it is trained longer, while the best long-trained individual is the seventh short-trained individual. We repeated this computation considering only the N best individuals of the 1L, i.e., those that form the 2L. An example is shown in Fig. 10, where the best 8 of the 1L are rearranged in the 2L. In addition, we computed the percentage of individuals that were correctly selected to form the 2L (green connections in Fig. 9). We computed the average of these indices throughout all the generations in which the 2L is evaluated. This experiment was repeated five times, using five independent runs of 2LGA.

The results are presented on Table 9 and show that both the correlation among the rankings with short and long training, and the percentage of well-classified top-N individuals are positives. This explains why the 18 epoch O-GA can find good solutions. However, the ranking between the top-N of the short-trained 1L and the long-trained one is close to zero, which, as in [114], indicates that a short training is not appropriate for ranking networks of similar performance.

These results explain the effectiveness of the 2LGA which takes advantage of the positive correlation between long and short training to select a part of the best networks. Then, the low correlation at the top of the 1L is fixed by re-training and rearranging the top-N individuals.

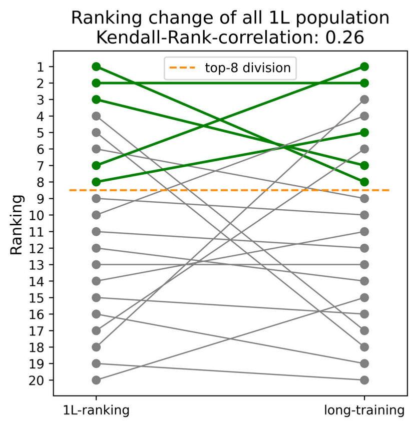


FIGURE 9. Change in the ranking of the entire 1L population when it receives short training (left column) and when given long training (right column). In this example, five of the 8 top individuals (62.5%) are selected correctly (green lines) to compose the 2L.

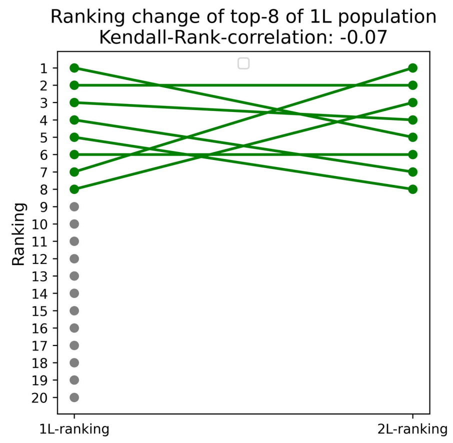


FIGURE 10. Change in the ranking of the top-8 of 1L population when it receives short training (left column) and when given long training (right column).

TABLE 9. Results of the selection of best-individuals. each value is the average of all evaluated generations, and of 5 independent evolutions.

Correlation of entire 1L	Correlation of top-8 of 1L	Percentage of correctly selected individuals
0.36	0.17	55.6 %

Thanks to this, the 2LGA can significantly outperform a short-training O-GA.

VI. ANALYSIS

The proposed search space is large and flexible, allowing CNNs to have any kind of architecture (skip connections, shape, and size), and to encode the internal parameters of each operation. These features make the search space sparser, and thus the sampled architectures tend to be different. This sparsity is beneficial for the search when evaluating the models with few epochs. According to the work reported in [114], the more different the architectures are, the fewer epochs are needed to rank them. Therefore, when evaluating the search space with only a few epochs, as in O-GA+18, and RS+18, shown on Table 6, the results are included in the state-of-the-art. However, as evolution progresses, the models tend to be more similar. Therefore, evaluating the models for a longer time benefits the search, achieving the best results as in 2LGA and O-GA+54 on Table 6.

On the other hand, the proposed 2LGA takes advantage of both a precise and an approximated training. First, the 2LGA exploits the speed of short training taking an evolution time similar to that of a common GA with short training. Second, the 2LGA exploits long training to improve ranking fidelity. As a result, the 2LGA achieves the same performance as if all models were trained precisely but in a significantly shorter time. The 2LGA achieves better results than an O-GA with short training because the difference in the ranking between short-trained networks and fully trained ones is considerably large. The 2LGA achieves the same results as the O-GA with long training because the O-GA trains unpromising models unnecessarily, and 2LGA with long training trains only the most promising ones.

Using the 2LGA reduces the adverse effects of using fitness approximation techniques, such as short training. The method, therefore, allows the GA to find architectures that achieve a lower test error with a small increase in search time. Furthermore, using a second level is complementary and compatible with other GAs that use fitness approximation techniques, so any GA that incorporates this method would improve its performance, sacrificing only a fraction of time.

We selected one CNN architecture that achieved the best results in test error to show in this section. The selected architecture is the best of the experiments performed on the MRDBI dataset which is the same dataset used for comparisons with ordinary GA, and with Random Search. The best architecture found by the 2LGA for this dataset, achieved a test error of 6.33%, with an evolution time of 6.9 hrs. By contrast, the best architecture found by the common GA that only uses short training achieved 7.12% test error for the same dataset, and its evolution took 6.3 hours. Thus, the 2LGA achieved 1.21% (an improvement of 17%) less test error in just 0.6 of an additional hour. On the other hand, the best architecture found by the common GA that only uses long training, achieved 6.45% test error, and its evolution took 21.2 hours, i.e., the 2LGA took 14.3 fewer hours, and achieved 0.12% less test error (an improvement of 1.9%). The best architecture found by the 2LGA is shown in Fig. 11. This figure contains the cell structure and the global structure. This architecture is composed of 2 blocks, 2 cells per block, with each block having 4 convolutions.

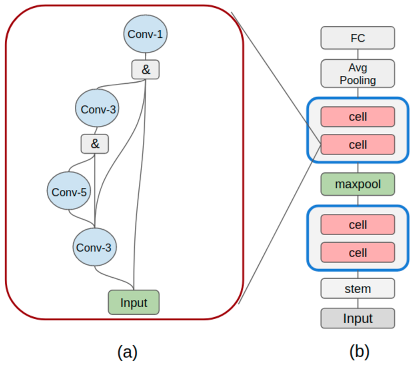


FIGURE 11. Best architecture found by the 2LGA for the MRDBI dataset.

Regarding the time and fidelity balance, it is important to explain that the 2LGA combines two types of training: short training and long training. Each training type has its fidelity, and its training time, both defined by the number of epochs. The greater the number of training epochs, the greater the fidelity, but the training time is also longer. If a GA uses only short training, as is common in the previous research [2], [30], [62]–[64], the low fidelity of the fitness will affect the overall performance of the GA. Therefore, the GA will have a higher test error than if it had used long training. Experiencing greater error when using short training occurs because the GA cannot rank individuals reliably according to their fitness. This is the trade-off between computational time and fitness fidelity. The 2LGA introduces long training to reduce the adverse consequences caused by the low fidelity of short training. Therefore, the balance is at a global level, at the GA level. Although the evolution time increases by just a fraction of the total time, the GA finds architectures that achieve lower test error.

VII. CONCLUSION

In this work we proposed a novel 2LGA for addressing the fidelity-time trade-off problem for the fitness computation in CNNs. In the 2LGA, the fitness is computed for all individuals with lower fidelity to save time. Then, the fitness for the best individuals that compose the second level is computed with higher fidelity. Since only a fraction of the population composes the second level, there is just a small increase in evolution time. However, as most of the best individuals are in the second level, the fitness is computed with high fidelity of only promising individuals. Therefore, there is a considerable gain in accuracy.

Our proposal includes a novel search space that reduces the parameter restrictions, and a codification that allows the 2LGA to perform any crossover operation among CNN architectures to generate new individuals for the next generation. This search space allows CNN architectures to have any size, shape, and skip-connections among nodes. Besides the structural components, the search space is composed of nodes, which can have any operation, such as convolution or maxpooling, and each operation has its own parameters (kernel size, activation function, etc.). In addition, we designed a variable-length encoding scheme with mutation and crossover on this complex search space. To improve the crossover operation among nodes, we encoded the parameters of all possible operations on each node, but only one is expressed, as in real genes. In this way, performing crossover between individuals is direct, as all the individuals have the same coded parameters.

The application of the proposed 2LGA to the pattern recognition problems of the MNIST-V dataset, and the Fashion-MNIST dataset, achieved significantly better results than all those that had been published previously. Our 2LGA achieved an improvement on the error reduction relative to previously published results of: 17.57% on the MB subset, 24.58% on the MRB subset, 33.68% on the MBI subset, 36.87% on the MRD subset, and 39.89% on the MRDBI subset. It is worth noting that the highest improvement in error reduction, of 39.89%, was achieved on the most difficult dataset, the MRDBI. On the Fashion-MNIST dataset our 2LGA resulted in an improvement in error reduction of 12.98% with no DA, and 0.87% with RF+RC DA.

We also show that 2LGA outperforms O-GAs on the MNIST-Variants dataset with short training and achieves the same results as those of O-GAs with long training, but while reducing the evolution time by nearly 66%.

To evaluate the search space, we carried out experiments with one of the simplest search algorithms, RS. The results showed that our search space helped the RS method to outperform even the best results published previously; those that used more sophisticated search algorithms than RS. This methodology may be extended to other areas besides image classification.

Finally, the designed 2LGA method receives a dataset as input and returns a fully trained CNN. To use this system, the user does not need to have profound DL knowledge, or to expend effort in tuning parameters. Therefore, the system could be useful for DL researchers in finding competitive CNNs with little effort, and it could help researchers without DL knowledge from other fields to incorporate CNN models in their research.

REFERENCES

[1] Y. Bengio, A. Courville, and P. Vincent, ''Representation learning: A review and new perspectives,'' *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 8, pp. 1798–1828, Aug. 2013.

[2] 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.

[3] G. Zhong, W. Jiao, W. Gao, and K. Huang, ''Automatic design of deep networks with neural blocks,'' *Cognit. Comput.*, vol. 12, no. 1, pp. 1–12, Jan. 2020.

[4] 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, 1962.

[5] H. Zhu, Z. An, C. Yang, K. Xu, E. Zhao, and Y. Xu, ''EENA: Efficient evolution of neural architecture,'' in *Proc. IEEE/CVF Int. Conf. Comput. Vis. Workshop (ICCVW)*, Oct. 2019, pp. 1891–1899.

[6] A. Baldominos, Y. Saez, and P. Isasi, ''On the automated, evolutionary design of neural networks: Past, present, and future,'' *Neural Comput. Appl.*, vol. 32, pp. 519–545, Mar. 2019.

[7] S. Litzinger, A. Klos, and W. Schiffmann, ''Compute-efficient neural network architecture optimization by a genetic algorithm,'' in *Proc. 28th Int. Conf. Artif. Neural Netw.* Munich, Germany: Springer, Sep. 2019, pp. 387–392, doi: 10.1007/978-3-030-30484-3_32.

[8] Y. Liu, Y. Sun, B. Xue, M. Zhang, G. G. Yen, and K. Chen Tan, ''A survey on evolutionary neural architecture search,'' 2020, *arXiv:2008.10937*. [Online]. Available: http://arxiv.org/abs/2008.10937

[9] 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.

[10] 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: OpenReview.net, Apr./May 2018. [Online]. Available: https://openreview.net/forum?id=BJQRKzbA- and https://dblp.org/rec/conf/iclr/LiuSVFK18.bib

[11] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, ''Automatically evolving CNN architectures based on blocks,'' 2018, *arXiv:1810.11875*. [Online]. Available: http://arxiv.org/abs/1810.11875

[12] 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.

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

[14] 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.

[15] 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, Nov. 1998.

[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton, ''ImageNet classification with deep convolutional neural networks,'' *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017.

[17] 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.

[18] 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.

[19] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, ''DeepFace: Closing the gap to human-level performance in face verification,'' in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2014, pp. 1701–1708.

[20] F. Schroff, D. Kalenichenko, and J. Philbin, ''FaceNet: A unified embedding for face recognition and clustering,'' in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 815–823.

[21] 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.

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

[23] 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.

[24] 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.

[25] 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, 1997.

[26] K. O. Stanley, J. Clune, J. Lehman, and R. Miikkulainen, "Designing neural networks through neuroevolution," *Nature Mach. Intell.*, vol. 1, pp. 24–35, Jan. 2019.

[27] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *J. Mach. Learn. Res.*, vol. 20, p. 55, Jan. 2019. [Online]. Available: http://jmlr.org/papers/v20/18-598.html

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

[29] 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. 34th Int. Conf. Mach. Learn.*, vol. 70, 2017, pp. 2902–2911.

[30] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proc. AAAI Conf. Artif. Intell.*, Jul. 2019, vol. 33, no. 1, pp. 4780–4789.

[31] X. He, K. Zhao, and X. Chu, "AutoML: A survey of the state-of-the-art," *Knowl.-Based Syst.*, vol. 212, Jan. 2021, Art. no. 106622.

[32] R. Tsukada, L. Zou, and H. Iba, *Evolving Deep Neural Networks for X-Ray Based Detection of Dangerous Objects*. Singapore: Springer, 2020, pp. 325–355.

[33] M. B. Calisto and S. K. Lai-Yuen, "AdaEn-Net: An ensemble of adaptive 2D–3D fully convolutional networks for medical image segmentation," *Neural Netw.*, vol. 126, pp. 76–94, Jun. 2020.

[34] G. Li, W. Zhou, W. Chen, F. Sun, Y. Fu, F. Gong, and H. Zhang, "Study on the detection of pulmonary nodules in CT images based on deep learning," *IEEE Access*, vol. 8, pp. 67300–67309, 2020.

[35] 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.

[36] R. G. Babukarthik, V. A. K. Adiga, G. Sambasivam, D. Chandramohan, and J. Amudhavel, "Prediction of COVID-19 using genetic deep learning convolutional neural network (GDCNN)," *IEEE Access*, vol. 8, pp. 177647–177666, 2020.

[37] T. Tanaka, T. Moriya, T. Shinozaki, S. Watanabe, T. Hori, and K. Duh, "Automated structure discovery and parameter tuning of neural network language model based on evolution strategy," in *Proc. IEEE Spoken Lang. Technol. Workshop (SLT)*, Dec. 2016, pp. 665–671.

[38] V. Passricha and R. K. Aggarwal, "PSO-based optimized CNN for Hindi ASR," *Int. J. Speech Technol.*, vol. 22, pp. 1123–1133, Oct. 2019.

[39] T. Shinozaki and S. Watanabe, "Structure discovery of deep neural network based on evolutionary algorithms," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Apr. 2015, pp. 4979–4983.

[40] Z. Gao, Y. Li, Y. Yang, X. Wang, N. Dong, and H.-D. Chiang, "A GPSO-optimized convolutional neural networks for EEG-based emotion recognition," *Neurocomputing*, vol. 380, pp. 225–235, Mar. 2020.

[41] C.-C. Chung, W.-T. Lin, R. Zhang, K.-W. Liang, and P.-C. Chang, "Emotion estimation by joint facial expression and speech tonality using evolutionary deep learning structures," in *Proc. IEEE 8th Global Conf. Consum. Electron. (GCCE)*, Oct. 2019, pp. 221–224.

[42] A. Rajagopal, G. P. Joshi, A. Ramachandran, R. T. Subhalakshmi, M. Khari, S. Jha, K. Shankar, and J. You, "A deep learning model based on multi-objective particle swarm optimization for scene classification in unmanned aerial vehicles," *IEEE Access*, vol. 8, pp. 135383–135393, 2020.

[43] L. M. Zhang, "Genetic deep neural networks using different activation functions for financial data mining," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Oct. 2015, pp. 2849–2851.

[44] 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 *Applications of Evolutionary Computation*, P. Kaufmann and P. A. Castillo, Eds. Cham, Switzerland: Springer, 2019, pp. 488–503.

[45] Y.-Y. Hong, J. V. Taylar, and A. C. Fajardo, "Locational marginal price forecasting using deep learning network optimized by mapping-based genetic algorithm," *IEEE Access*, vol. 8, pp. 91975–91988, 2020.

[46] 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.

[47] S. S. Mostafa, F. Mendonca, A. G. Ravelo-Garcia, G. Julia-Serda, 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.

[48] Y. Zhang, P. Li, and X. Wang, "Intrusion detection for IoT based on improved genetic algorithm and deep belief network," *IEEE Access*, vol. 7, pp. 31711–31722, 2019.

[49] S. A. Ali, B. Raza, A. K. Malik, A. R. Shahid, M. Faheem, H. Alquhayz, and Y. J. Kumar, "An optimally configured and improved deep belief network (OCI-DBN) approach for heart disease prediction based on Ruzzo–Tompa and stacked genetic algorithm," *IEEE Access*, vol. 8, pp. 65947–65958, 2020.

[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] W. Jian, Y. Zhou, and H. Liu, "Densely connected convolutional network optimized by genetic algorithm for fingerprint liveness detection," *IEEE Access*, vol. 9, pp. 2229–2243, 2021.

[52] G. F. Miller, P. M. Todd, and S. U. Hegde, "Designing neural networks using genetic algorithms," in *Proc. 3rd Int. Conf. Genetic Algorithms*, vol. 89, 1989, pp. 379–384.

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

[54] C. A. Perez, C. A. Salinas, and P. Estevez, "Designing biologically inspired receptive fields for neural pattern recognition technology," in *Proc. IEEE Int. Conf. Syst., Man Cybern. e-Syst. e-Man Cybern. Cyberspace*, vol. 1, Oct. 2001, pp. 58–63.

[55] C. A. Perez and C. Salinas, "Genetic selection of biologically inspired receptive fields for computational vision," in *Proc. 1st Joint BMES/EMBS Conf. IEEE Eng. Med. Biol. 21st Annu. Conf. Annu. Fall Meeting Biomed. Eng. Soc.*, vol. 2, Oct. 1999, p. 924.

[56] K. Fukushima and S. Miyake, "Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition," in *Competition and Cooperation in Neural Nets*, S.-I. Amari and M. A. Arbib, Eds. Berlin, Germany: Springer, 1982, pp. 267–285.

[57] 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 *Proc. 16th Pacific Rim Int. Conf. Artif. Intell.* Cuvu, Fiji: Springer, Aug. 2019, pp. 650–663, doi: 10.1007/978-3-030-29894-4_52.

[58] A. Klein, S. Falkner, S. Bartels, P. Hennig, and F. Hutter, "Fast Bayesian optimization of machine learning hyperparameters on large datasets," in *Proc. 20th Int. Conf. Artif. Intell. Statist.*, vol. 54, A. Singh and J. Zhu, Eds. Fort Lauderdale, FL, USA: PMLR, Apr. 2017, pp. 528–536.

[59] P. Chrabaszcz, I. Loshchilov, and F. Hutter, "A downsampled variant of ImageNet as an alternative to the CIFAR datasets," 2017, *arXiv:1707.08819*. [Online]. Available: http://arxiv.org/abs/1707.08819

[60] 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.

[61] A. Zela, A. Klein, S. Falkner, and F. Hutter, "Towards automated deep learning: Efficient joint neural architecture and hyperparameter search," 2018, *arXiv:1807.06906*. [Online]. Available: http://arxiv.org/abs/1807.06906

[62] F. E. Fernandes Junior 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.

[63] B. Wang, Y. Sun, B. Xue, and M. Zhang, "A hybrid differential evolution approach to designing deep convolutional neural networks for image classification," in *Proc. Australas. Joint Conf. Artif. Intell.*, vol. 11320, 2018, pp. 237–250.

[64] F. Johnson, A. Valderrama, C. Valle, B. Crawford, R. Soto, and R. Nanculef, "Automating configuration of convolutional neural network hyperparameters using genetic algorithm," *IEEE Access*, vol. 8, pp. 156139–156152, 2020.

[65] D. R. So, Q. V. Le, and C. Liang, "The evolved transformer," in *Proc. 36th Int. Conf. Mach. Learn. (ICML)* (Proceedings of Machine Learning Research), vol. 97, K. Chaudhuri and R. Salakhutdinov, Eds. Long Beach, CA, USA: PMLR, Jun. 2019, pp. 5877–5886.

[66] 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.

[67] K. Yu, C. Sciuto, M. Jaggi, C. Musat, and M. Salzmann, "Evaluating the search phase of neural architecture search," 2019, *arXiv:1902.08142*. [Online]. Available: http://arxiv.org/abs/1902.08142

[68] L. Li and A. Talwalkar, "Random search and reproducibility for neural architecture search," in *Proc. 35th Uncertainty Artif. Intell. Conf.*, vol. 115, Jul. 2020, pp. 367–377.

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

[70] D. Kang and C. W. Ahn, "Efficient neural network space with genetic search," in *Bio-Inspired Computing: Theories and Applications*, L. Pan, J. Liang, and B. Qu, Eds. Singapore: Springer, 2020, pp. 638–646.

[71] B. Fielding and L. Zhang, "Evolving image classification architectures with enhanced particle swarm optimisation," *IEEE Access*, vol. 6, pp. 68560–68575, 2018.

[72] A. A. Ahmed and S. M. Darwish, "A meta-heuristic automatic CNN architecture design approach based on ensemble learning," *IEEE Access*, vol. 9, pp. 16975–16987, 2021.

[73] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger, "Deep networks with stochastic depth," in *Computer Vision–(ECCV)*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds. Cham, Switzerland: Springer, 2016, pp. 646–661.

[74] M. A. A. Albadr, S. Tiun, M. Ayob, F. T. AL-Dhief, K. Omar, and F. A. Hamzah, "Optimised genetic algorithm-extreme learning machine approach for automatic COVID-19 detection," *PLoS ONE*, vol. 15, no. 12, Dec. 2020, Art. no. e0242899, doi: 10.1371/journal.pone.0242899.

[75] M. A. Albadr, S. Tiun, M. Ayob, and F. AL-Dhief, "Genetic algorithm based on natural selection theory for optimization problems," *Symmetry*, vol. 12, no. 11, p. 1758, Oct. 2020. [Online]. Available: https://www.mdpi.com/2073-8994/12/11/1758

[76] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press, 2020.

[77] X. Yao, "Evolving artificial neural networks," *Proc. IEEE*, vol. 87, no. 9, pp. 1423–1447, Sep. 1999.

[78] S. Ruder, "An overview of gradient descent optimization algorithms," 2016, *arXiv:1609.04747*. [Online]. Available: http://arxiv.org/abs/1609.04747

[79] S. Hochreiter, "Untersuchungen zu dynamischen neuronalen netzen," Ph.D. dissertation, Institut für Informatik, Technische Universität, Munich, Germany, 1991.

[80] 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.

[81] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *Proc. 5th Int. Conf. Learn. Represent. (ICLR)*. Toulon, France: OpenReview.net, Apr. 2017. [Online]. Available: https://openreview.net/forum?id=r1Ue8Hcxg and https://dblp.org/rec/conf/iclr/ZophL17.bib

[82] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," in *Proc. 5th Int. Conf. Learn. Represent. (ICLR)*. Toulon, France: OpenReview.net, Apr. 2017. [Online]. Available: https://openreview.net/forum?id=S1c2cvqee and https://openreview.net/forum?id=S1c2cvqee

[83] 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.

[84] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," in *Proc. 7th Int. Conf. Learn. Represent. (ICLR)*. New Orleans, LA, USA: OpenReview.net, May 2019. [Online]. Available: https://openreview.net/forum?id=S1eYHoC5FX and https://dblp.org/rec/conf/iclr/LiuSY19.bib

[85] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas, "Taking the human out of the loop: A review of Bayesian optimization," *Proc. IEEE*, vol. 104, no. 1, pp. 148–175, Jan. 2016.

[86] H. Mendoza, A. Klein, M. Feurer, J. T. Springenberg, and F. Hutter, "Towards automatically-tuned neural networks," in *Proc. Workshop Autom. Mach. Learn.*, 2016, pp. 58–65.

[87] M. Wistuba, "Bayesian optimization combined with successive halving for neural network architecture optimization," in *Proc. AutoML@ PKDD/ECML*, 2017, pp. 2–11.

[88] H. Larochelle, Y. Bengio, J. Louradour, and P. Lamblin, "Exploring strategies for training deep neural networks," *J. Mach. Learn. Res.*, vol. 10, no. 1, pp. 1–40, Jan. 2009.

[89] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms," 2017, *arXiv:1708.07747*. [Online]. Available: http://arxiv.org/abs/1708.07747

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

[91] P. A. Estévez, C. A. Perez, and E. Goles, "Genetic input selection to a neural classifier for defect classification of radiata pine boards," *Forest Products J.*, vol. 53, nos. 7–8, pp. 87–94, 2003.

[92] C. Perez, L. Castillo, L. Cament, P. Estevez, and C. Held, "Genetic optimisation of illumination compensation methods in cascade for face recognition," *Electron. Lett.*, vol. 46, no. 7, pp. 498–500, 2010.

[93] M. Suganuma, S. Shirakawa, and T. Nagao, "A genetic programming approach to designing convolutional neural network architectures," in *Proc. 27th Int. Joint Conf. Artif. Intell. (IJCAI)*, Jul. 2018, pp. 5369–5373.

[94] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. 32nd Int. Conf. Mach. Learn.* (Proceedings of Machine Learning Research), vol. 37, F. Bach and D. Blei, Eds. Lille, France: PMLR, Jul. 2015, pp. 448–456.

[95] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch SGD: Training ImageNet in 1 hour," 2017, *arXiv:1706.02677*. [Online]. Available: http://arxiv.org/abs/1706.02677

[96] L. N. Smith and N. Topin, "Super-convergence: Very fast training of neural networks using large learning rates," *Proc. SPIE*, vol. 11006, pp. 369–386, May 2019.

[97] S. Picek, D. Jakobovic, and M. Golub, "On the recombination operator in the real-coded genetic algorithms," in *Proc. IEEE Congr. Evol. Comput.*, Jun. 2013, pp. 3103–3110.

[98] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Los Alamitos, CA, USA, Jun. 2016, pp. 2818–2826.

[99] G. Kyriakides and K. Margaritis, "Regularized evolution for macro neural architecture search," in *Proc. 16th IFIP WG 12.5 Int. Conf. AIAI*, vol. 584. Neos Marmaras, Greece: Springer, 2020, pp. 111–122, doi: 10.1007/978-3-030-49186-4_10.

[100] E. Byla and W. Pang, "Deepswarm: Optimising convolutional neural networks using swarm intelligence," in *Advances in Intelligent Systems and Computing*, vol. 1043. Cham, Switzerland: Springer, 2020, pp. 119–130, doi: 10.1007/978-3-030-29933-0_10.

[101] 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.

[102] Z. Lu, I. Whalen, V. Boddeti, Y. Dhebar, K. Deb, E. Goodman, and W. Banzhaf, "NSGA-Net: Neural architecture search using multi-objective genetic algorithm," in *Proc. Genetic Evol. Comput. Conf.*, 2019, pp. 419–427.

[103] J. Tapia, C. Perez, and K. Bowyer, "Gender classification from the same iris code used for recognition," *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 8, pp. 1760–1770, Aug. 2016.

[104] K. Chang, K. Bowyer, and P. Flynn, "An evaluation of multimodal 2D+3D face biometrics," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, no. 4, pp. 619–624, Apr. 2005.

[105] 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.

[106] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. 3rd Int. Conf. Learn. Represent. (ICLR)*, Y. Bengio and Y. LeCun, Eds. San Diego, CA, USA, May 2015. [Online]. Available: http://arxiv.org/abs/1412.6980 and https://dblp.org/rec/journals/corr/KingmaB14.bib

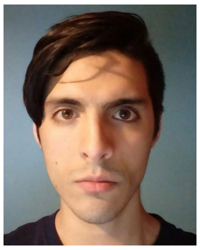
[107] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*. [Online]. Available: http://arxiv.org/abs/1704.04861

[108] S. Zagoruyko and N. Komodakis, "Wide residual networks," in *Proc. Brit. Mach. Vis. Conf. (BMVC)*, Sep. 2016, p. 87.

[109] Z. Zhong, L. Zheng, G. Kang, S. Li, and Y. Yang, "Random erasing data augmentation," in *Proc. AAAI Conf. Artif. Intell.*, vol. 34, 2020, pp. 13001–13008.

[110] A. Nøkland and L. H. Eidnes, "Training neural networks with local error signals," in *Proc. 36th Int. Conf. Mach. Learn.* (Proceedings of Machine Learning Research), vol. 97, K. Chaudhuri and R. Salakhutdinov, Eds. Jun. 2019, pp. 4839–4850.

[111] T. Devries and G. W. Taylor, "Improved regularization of convolutional neural networks with cutout," *CoRR*, vol. abs/1708.04552, 2017.
[112] H. Abdi and L. J. Williams, "Tukey's honestly significant difference (HSD) test," *Encyclopedia Res. Des.*, vol. 3, no. 1, pp. 583–585, 2010.
[113] M. G. Kendall, "A new measure of rank correlation," *Biometrika*, vol. 30, pp. 81–93, Jun. 1938.
[114] G. Kyriakides and K. Margaritis, "The effect of reduced training in neural architecture search," *Neural Comput. Appl.*, vol. 32, no. 23, pp. 17321–17332, Dec. 2020.



DANIEL A. MONTECINO was born in Santiago, Chile, in 1994. He received the B.S. degree (Hons.) in electrical engineering from the Universidad de Chile, Santiago, in 2016, where he is currently pursuing the master's degree in electrical engineering. He is also working as a Research Assistant with the Department of Electrical Engineering, Universidad de Chile. His current research interests include image processing, genetic algorithm, machine learning, and deep learning.

CLAUDIO A. PEREZ (Senior Member, IEEE) received the B.S. degree in electrical engineering, in 1980, the P.E. Title in electrical engineering and M.S. degree in biomedical engineering from Universidad de Chile, both in 1985, and the Ph.D. degree from 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 at Berkeley, in 2002, through the Alumni Initiatives Award Program from Fulbright Foundation. From 2003 to 2006, he was the Department Chairman and the Director of the Office of Academic and Research Affairs at the 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, neural network structures, genetic algorithms, and pattern recognition. He is a Senior Member of the IEEE Systems, Man and Cybernetics and the IEEE-CIS societies.

KEVIN W. BOWYER (Life Fellow, IEEE) is currently the Schubmehl-Prein Family Professor with the Department of Computer Science and Engineering, University of Notre Dame, and also works as 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 the IEEE, "for contributions to algorithms for recognizing objects in images" and a fellow of the IAPR, "for contributions to computer vision, pattern recognition and biometrics." He received an IEEE Computer Society Technical Achievement Award "for pioneering contributions to the science and engineering of biometrics," and received the inaugural IEEE Biometrics Council Meritorious Service Award. He is serving as the Inaugural Editor-in-Chief for the IEEE TRANSACTIONS ON BIOMETRICS, BEHAVIOR, AND IDENTITY SCIENCE (T-BIOM).