A Review on Convolutional Neural Network Encodings for Neuroevolution

Gustavo-Adolfo Vargas-Hákim, Efrén Mezura-Montes, *Senior Member, IEEE*, and Héctor-Gabriel Acosta-Mesa

***Abstract*—Convolutional neural networks (CNNs) have shown outstanding results in different application tasks. However, the best performance is obtained when customized CNNs architectures are designed, which is labor intensive and requires highly specialized knowledge. Over three decades, neuroevolution (NE) has studied the application of evolutionary computation to optimize artificial neural networks (ANNs) at different levels. It is well known that the encoding of ANNs highly impacts the complexity of the search space and the optimization algorithms' performance as well. As NE has rapidly advanced toward the optimization of CNNs topologies, researchers face the challenging duty of representing these complex networks. Furthermore, a compilation of the most widely used encoding methods is nonexistent. In response, we present a comprehensive review on the *state-of-the-art* of encodings for CNNs.**

***Index Terms*—Convolutional neural networks (CNNs), encodings, neural architecture search, neuroevolution (NE).**

I. Introduction

NEUROEVOLUTION (NE) has become the successful utilization of evolutionary algorithms (EAs) for the optimization of artificial neural networks (ANNs) [1]. Not only the weights in a network might be optimized but also its topology. For over three decades, the NE community has advanced the field toward different approaches and applications, ranging from reinforcement learning and control [2] to visual classification [3].

In recent years, there has been an increasing success of deep neural network architectures, such as convolutional neural networks (CNNs), which have shown important advancements in solving different tasks, specially those related with visual recognition [4]. However, the application of these techniques often requires several design decisions; CNNs are usually composed of many convolutional and pooling layers with hundreds of filters of different sizes [5], coupled with a number of other hyperparameters. It is then difficult to manually design a particular network for every given problem.

NE research has latterly offered an interesting variety of evolutionary and swarm intelligence methods with the aim of providing a solution for the highly complex problem of designing a CNN. Fig. 1 presents the methods used in the reviewed literature. The important growth of this field can be appreciated in Fig. 2, which displays the number of reviewed articles per year since 2013. A total of 135 papers have been acquired from databases, such as IEEE Xplore, ACM Digital Library, IOP Science, and Google Scholar. Keywords included *NE*, Neural Architecture Search, Evolutionary Deep Learning, Evolutionary Neural Architecture Pruning, and Evolutionary Neural Architecture Compression. Only those works concerning the design of CNN architectures using metaheuristics were considered. An intensive review of swarm intelligence and evolutionary methods for deep learning is presented in [6]. Comprehensive and general reviews on NE are presented in [1], [5], [7], and [8].

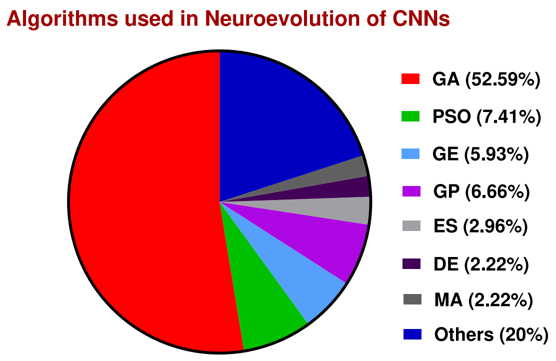

Fig. 1. NE methods for CNNs; GA, particle swarm optimization (PSO), GE, GP, evolution strategy (ES), differential evolution (DE), memetic algorithm (MA), and others that do not classify into any of these categories.

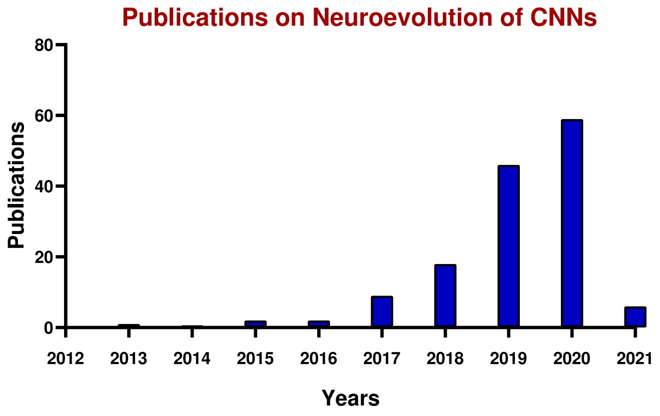

Fig. 2. Record of reviewed publications on NE for CNNs.

Manuscript received July 23, 2020; revised November 12, 2020 and March 30, 2021; accepted May 25, 2021. Date of publication June 11, 2021; date of current version January 31, 2022. The work of Gustavo-Adolfo Vargas-Hákim was supported by the Mexican National Council of Science and Technology (CONACyT). *(Corresponding author: Gustavo-Adolfo Vargas-Hákim).*

The authors are with the Artificial Intelligence Research Institute, University of Veracruz, Xalapa 91000, Mexico (e-mail: vargashakimg@gmail.com).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TEVC.2021.3088631.

Digital Object Identifier 10.1109/TEVC.2021.3088631

The computational representation of an ANN is called *neural encoding* and is the first and arguably the most important part of the design of an NE approach [9]. The subsequent layout of the EA and the complexity of the search space highly depends on the encoding [10]–[12]. Thus, the careful study of representations of CNNs is of crucial importance for the success of evolutionary architecture search. However, the choice of a suitable encoding for any given application is far from trivial, as the research toward the analysis of neural encodings has been scarce.

The purpose of this work is twofold. First, we aim to provide for an overview of the *state-of-the-art* regarding the encodings for CNNs. To the best of our knowledge, the last review on encodings for NE was published in 2011 [13]. By its time of publications, CNNs were not as widely utilized as nowadays and they were not considered in this review. The high volume of publications of evolutionary neural architecture search further motivates a review on these techniques. Second, this review aims to formalize the study of neural encodings of CNNs by proposing a novel taxonomy of these representations. Moreover, analysis and discussions about the observed benefits and losses of each subfamily of encodings are included.

The remainder of this article is organized as follows. A brief introduction to CNNs is presented in Section II. General aspects and assumptions about encodings are discussed in Section III. Section IV details indirect encodings, whilst Section V elaborates on direct encodings. Hybrid encodings are presented in Section VI. Section VII offers a general discussion on NE at the encoding level and proposes prospective research directions. The conclusion of this review is summarized in Section VIII.

II. Convolutional Neural Networks

CNNs, as known nowadays, were proposed by LeCun *et al.* [14]. This family of ANNs has the property of extracting features automatically from data besides being trained as a classifier. Usually, the input of the network is a tensor, e.g., a color image. However, there are successful approaches to the usage of other kinds of data, such as the text [15] and audio [16]. The aforementioned is achieved by a set of different layers with different properties described next.

Convolutional layers apply a form of discrete convolution between an input tensor, such as an image with different channels, and a function represented by a smaller 2-D array called *filter* or *kernel*. The mathematical statement of the discrete bidimensional convolution process between two functions f and g is presented in the following:

$$(f * g)(x, y) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} g(i, j) f(x - i, y - j) \qquad (1)$$

where x and y are the two variables of both f and g, e.g., a pixel's row and column, and i and j are the indexing values that shift f through g. A layer contains a set of filters, each of them in charge of a certain region of the input. The term *stride* refers to an integer number defining the separation between filters applied to the input, i.e., the steps taken by the

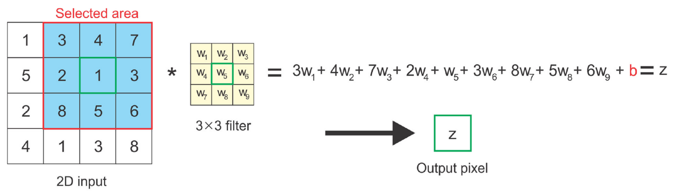

Fig. 3. Example of the convolution operation between a 4×4 input and a 3×3 filter. The result is mapped based on the position of the center value of the region of interest. When no padding is added, the result is smaller in dimensions. The *weight* values in the filter are learned parameters. The value b corresponds to the bias of the filter.

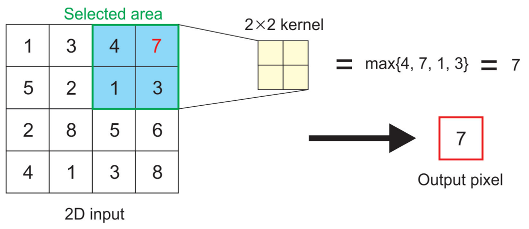

Fig. 4. Example of the max pooling operation between a 4×4 input and a 2×2 kernel. It is noticeable that pooling kernels do not have any *weight* values.

convolution process. A convolutional layer is often followed by *batch normalization* [17] and/or an activation function, such as ReLU [18], which is presented in (2)

$$\text{ReLU}(x) = \max(x, 0) \qquad (2)$$

where x is a numerical input and *max* is the function that returns the larger value among its arguments. The output of a convolutional layer is often called the *feature map* or *activation map* as it is expected to carry extracted features from data. Fig. 3 shows an example of the convolution operation. Instead of using individual inputs, a set of them might be grouped in what are called *batches* of size m.

The *pooling layer*, also known as the subsampling layer, reduces the spatial resolution of its input. A pooling layer consists of an empty kernel with a stride value. The most popular pooling methods are the *max pooling*, which extracts the highest value perceived by the kernel, and the *average pooling*, which obtains the arithmetic average of the values perceived by the kernel. It is important to note that both the stride size and the kernel size have an important role in the output's resolution. However, when the stride value is higher, the output's spatial resolution is linearly decreased in the same proportion. Fig. 4 shows an example of the convolution and pooling operations.

Batch normalization plays an important role during the training of a CNN. As explained in [17], this process is a powerful remedy against the internal covariate shift (ICS), which corresponds to the change of distribution of the network activations during the training process, as the weights keep changing. With a high ICS, the training process needs to compensate and continually adapt weights to new distributions, thus taking longer to optimize. Given a layer in the network that receives a d-dimensional input $x = (x^{(1)}, \ldots, x^{(d)})$, the mean μ_B and variance σ_B^2 for a batch B of size m must be computed. For each element x_i in the batch, the k-dimension normalization shown in (3) is applied

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \tag{3}$$

with ϵ being a small constant. The normalized element $\hat{x}_i$ has now a zero mean and a variance of 1. Nonetheless, this procedure might constrain the representation power of the layer. Hence, a transformation is applied over each normalized input $\hat{x}_i$ in the batch as shown in

$$y_i = \lambda \hat{x}_i + \beta \tag{4}$$

where λ and β are new learnable parameters associated to each input x_i in the batch. The batch normalization process can now be written as a transformation $\mathbf{BN}_{\lambda,\beta} : x_i \rightarrow y_i$. As discussed in [19], Batch Normalization can be applied in the form of a layer, augmenting the CNN topology.

Fully Connected Layer is a feedforward layer located at the end of the network. A collection of fully connected layers form a classifier. The data coming from the feature extraction section of the network must be *unfolded*, i.e., converted into a linear array, in order to fit the first fully connected layer. Each neuron has its own activation function, where ReLU is a common choice. The last layer is often equipped with the *softmax* activation function for multiclass classification tasks.

Convolutional layers, pooling layers, batch normalization, and fully connected layers are assembled together in such a way that the CNN abstracts useful features from data in order to learn patterns. Some architectures are built *in-line*, i.e., the output of a layer is the input of only the next layer. Examples of these networks are LeNet-5 [14], AlexNet [4], and VGGNet [20]. Some others utilize *skip connections*, when the output of a layer is connected to the input of a layer further than the next one. DenseNet [21] is a good example of this group of topologies. When a layer receives more than two inputs, elementwise summation or concatenation is carried out. If the inputs' spatial resolutions do not match, *padding*, i.e., adding "dummy" values at the borders, is applied.

The following section elaborates on the basic notions of neural encodings, so that a CNN can be represented and manipulated by an EA.

III. Encodings

The NE of augmenting topologies (NEATs) algorithm was a breakthrough in NE [22]. In their work, Stanley and Miikkulainen proposed a powerful method to evolve both the architecture of an ANN and its weights. The authors suggested the power of what is called a direct encoding. In his doctoral thesis, Stanley further explains the advantages of direct encodings over indirect encodings [23].

Traditional NE has dealt with conventional ANNs, i.e., feedforward neural networks and classical recurrent neural networks. These architectures are assembled in such a way that the most basic unit is the neuron. NEAT takes advantage of this particularity to propose an encoding based on a linear representation. Its encoding consists on a list of connections, which sets the connectivity of a list of nodes, i.e., neurons. This kind of representation is a direct encoding; all the information of the network is represented. Thus, the construction of the *phenotype* might need little to no processing from the *genotype*. An indirect encoding, on the other hand, is a genotype with a set of rules aiming to build the network; there is no explicit representation of the ANN.

In view of the success of CNNs in recent years, NE has evolved to deal with these highly complex architectures. CNNs possess the property of being modular, then such networks can be decomposed in terms of layers, e.g., convolutional, pooling, normalization, and fully connected. Recent networks, such as ResNet [24] and DenseNet [21] have exploited modularity even further, proposing blocks or modules containing arrangements of layers.

Although the idea of a neuron is still present in CNNs, it is not the most convenient unit to use as the building block for encodings. NE has adopted the concept of layer as the most primitive unit. The notion of direct encoding adapts to this new paradigm; a direct encoding is any representation that explicitly defines the characteristics of the layers or modules containing layers and their connectivity. A direct encoding is said to be manipulated at the phenotypic level, meaning that there is no rule involved to transform the encoding to the actual network. The availability of powerful deep learning frameworks, such as Tensorflow [25] or Pytorch [26], eases the construction of the CNN from a direct encoding, consisting most of the times on extracting the hyperparameters in certain order.

On the other hand, an indirect encoding is any representation that requires a set of rules to generate a CNN. Again, the network is not explicitly represented and cannot be directly built from the genotype. The genotypic space is completely different to the phenotypic space, and a mechanism to transform the genotype to the phenotype is always needed. Some researchers have recently explored what could be called Hybrid Encodings; representations that combine ideas from direct and also indirect encodings.

IV. Indirect Encodings

As mentioned before, an indirect encoding is a genotype that requires a set of rules to be transformed into a phenotype, i.e., a CNN. This family of encodings is represented by two different categories: 1) Binary Encodings and 2) Grammar Encodings. The specialized literature shows that this encoding paradigm is the least utilized. Nonetheless, it entails powerful benefits.

A. Binary Encodings

Binary encodings use binary arrays to represent the architecture of CNNs. One-dimensional binary arrays were some of the most primeval representations used in evolutionary computation, especially in genetic algorithms (GAs). They naturally encompass the advantage of being compatible with well-known variation operators during search. Moreover, the rules to generate the CNN based on a binary array tend to be straightforward. An important loss of these encodings is the constrained search space due to the nature of binary numbers and due to fixed-length (FL) encodings, which shorten flexibility. Additionally, bigger networks usually correspond to larger

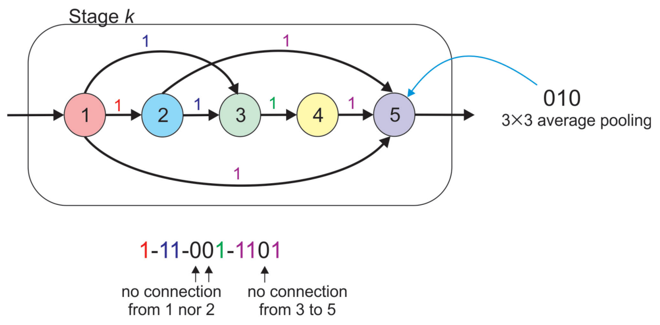

Fig. 5. Example of the binary encoding for a single stage used in [40]. Each node i, except the first one, has $i-1$ bits specifying connections from the previous nodes. Also, each node is represented by a *3-bit* string. In this example, *010* is related to a 3×3 average pooling.

strings, forcing the population to increase in size. The reviewed binary encodings can be classified into two broad categories: 1) binary strings and 2) binary matrices.

Binary strings are 1-D arrays of bits that have been utilized in creative ways to represent different CNN architectures. These encodings are oftentimes used to encode the hyperparameters of the different components of a CNN. Bivaeva [27] proposed a FL string for each layer, including convolutional, pooling, and fully connected layers. Each string is able to encode characteristics, such as convolutional filter size, pooling type and kernel size, and the activation functions. A similar approach was also used in [28]. Li *et al.* [29] also included a 3-bit ID substring at the beginning of each type of layer. This procedure makes the parsing process of the string easier.

Baldominos *et al.* [30], [31] introduced a 69-bit string that encodes the different hyperparameters of an FL CNN. This representation is divided between the convolutional part, which includes convolutional and pooling layers, and the fully connected part, that represents fully connected layers. Besides these layers, this encoding also determines the optimizer of the CNN, the batch size, the dropout rates, and the type of weight regularization in fully connected layers.

Wang *et al.* [32], [33] utilized a binary encoding resembling an IP address, allowing a variable-length (VL) representation of convolutional, pooling, and fully connected layers. It considers kernel sizes, stride values, number of filters, pooling type, and the number of neurons in fully connected layers.

The work by Chen *et al.* [34] is characterized by representing more complex CNN components using binary strings. In this case, the string can be divided into several 3-bit alleles that choose different popular blocks found in well-known architectures, such as Inception and ResNet. Eight possible blocks are available and the inner structure of each block is respected. Xie and Yuille [35] presented the Genetic CNN (see also [36]–[39]). They conceived the CNN as a sequence of stages, each containing a number of convolutional layers with equal characteristics. The layers inside a stage are connected using a fixed graph, where a binary string encodes the connectivity of the nodes. Neighbor stages are connected via spatial pooling.

Jiang *et al.* [40] expanded Genetic CNN by introducing two parts. The first part is the so-called Topology Encoding, which uses a binary string to represent the connectivity of convolutional nodes within a graph. The Operation Encoding is a binary string containing information about each node in the graph, concerning mainly with kernel sizes for both pooling and convolution. If L nodes are inside a single stage, the Operation Encoding has a length of $3L$ bits. Fig. 5 shows an example of this method.

In the work by Liu *et al.* [41], the CNN has three possible branches that are composed of a series of blocks composed of different operators. A *simple* branch consists on a convolutional layer. The other two *complex* branches are an inverted residual block and an inception module [42], respectively. If the three paths are activated at the same time, the data flow throw them and the result of each module is summed at the end. The first three bits of a binary string encode the activation (1) or deactivation (0) of each branch. The hyperparameters of the fixed elements inside each branch are encoded by subsequent bits.

Ye *et al.* [43] utilized a binary encoding based on the quantum computing theory. Convolutional layers are described by 12 bits, meanwhile pooling layers and fully connected layers are described by 11 bits each. In order to increase the storing capacity of binary numbers, a *quantum chromosome* q is built, which consists of a set of m pairs α_i and β_i such that $|\alpha_i|^2 + |\beta_i|^2 = 1$. These values are combined with the superposition nature of quantum bits to build the original binary string, which is later decoded into a CNN architecture.

Previously mentioned approaches built CNNs from scratch. In Evolutionary Neural Architecture Pruning/Compression, however, the objective is to reduce the existing architectures to their smallest form while maintaining performance. Binary strings excel at this task, by attaching the elements of a CNN to the Boolean values, in order to turn them on (1) or off (0). Zhou *et al.* [44] developed a pruning approach for biomedical image segmentation starting from the well-known U-Net architecture [45]. Each bit of a binary string corresponds to a filter in the original architecture. A similar approach was used in [46], but using AlexNet [4], VGG-16 [20], and ResNet [21] architectures. Zhou *et al.* [47] pruned the LeNet and VGG-19 architectures. Fernandes Jr. and Yen [48] pruned ResNet-like architectures [24] with two strings; one binary string represents the first layer of a Residual Block, whilst the second one represents the following layers.

Binary Matrices utilize a higher dimensional array of Boolean values to be used as an adjacency matrix. These encodings are solely for the connectivity of the predefined CNN components. In an adjacency matrix A of $N \times N$, a layer i connects to the layer j if and only if $A_{ij} = 1$. Note that $j > i$ as recurrent connections are forbidden in CNNs. Thus, the only important part is the upper right triangle in A. This approach was implemented in [49] and [50]. O'Neill *et al.* [51] utilized a $3 \times N \times N$ adjacency matrix, in which each of the three channels corresponds to one of the three N-layers DenseBlocks in the CNN.

Table I organizes the reviewed literature on Binary Encodings into their subfamilies, classifying them also into VL and FL encodings, and including the hyperparameters than can be encoded in these representations.

TABLE I
SUMMARY OF BINARY ENCODINGS

	Publications	Hyperparameters	Length
Binary String	[27] [28] [29] [30] [31] [34] [41] [43] [44] [46] [47] [48]	Filters' size, No. of Filters, Pooling Type, Kernel Size, Stride, No. of Neurons, Activation Function, Batch Size, Learning rate, Optimizer	Fixed
	[32] [33] [35] [36] [37] [38] [39] [40]	Filters' size, Kernel Size, Pooling Type, Connectivity	Variable
Binary Matrix	[49] [50] [51]	Connectivity	Fixed

TABLE II
SUMMARY OF GRAMMAR ENCODINGS

	Publications	Hyperparameters	Length
Grammar Encoding	[30] [53] [54] [55] [56] [57] [58] [59]	Filters' size, No. of Filters, Pooling Type, Kernel Size, Stride, No. of Neurons, Activation Function, Regularization, Dropout, Optimizer, Padding, Batch Norm., Connectivity	Fixed

B. Grammar Encodings

Grammar Encodings research has been scarce comparable to the Binary Encodings. However, it has shown promising results. These methods are based on the Grammatical Evolution theory [52], which requires the definition of a formal grammar in the Backus–Naur notation. However, once they have been characterized, they allow the creation of very flexible networks with a wide range of changeable parameters.

A grammar is defined as a 4-tuple $G = (N, T, S, P)$, where N is a nonempty set of nonterminal symbols, T is a nonempty set of terminal symbols, S is the starting symbol, and P is a set of production rules in Backus–Naur form. A grammar allows the generation of a language, i.e., all the possible sequences of terminal symbols that can be derived from the starting symbol. In this context, a language is a representation of CNN architectures.

Baldominos *et al.* [30], [53] presented this encoding method to reduce redundancies in the previously discussed 69-bit string. Their grammar allows to define several aspects of a CNN. For example, the rule $< conv > ::= < n\_kernels >< k\_size >< act\_fn >< pool >$ specifies that a convolutional layer consists of a number of kernels, the kernel size, the activation function, and the pooling operation. Each of these features are nonterminals that can also be defined using other rules. The authors encode the same characteristics they represent in their binary encoding. Lima *et al.* [54] adopted this encoding style, defining convolutional layers, which are composed of nodes corresponding to convolutions, max pooling, or average pooling.

Assunção *et al.* [55], [56] introduced DENSER, in which they proposed a grammar composed of two levels; the GA level, which depicts the macro-structure of the CNN as a sequence of layers. The second level is the dynamic structured GE (DSGE) level, encoding the microstructure of the genotype; it contains the parameters related to the GA level. Both levels can communicate in such a way that each element in the GA level works as an initial nonterminal symbol for the DSGE level. This mechanism not only grants a flexible definition of a CNN, but can be extended to other types of ANNs. Cetto *et al.* [57] implemented a simpler version of DENSER, by utilizing a Context Free Grammar instead of the DSGE.

Unlike DENSER, Fast DENSER [58], [59] introduces a list of connections per layer, which specifies the skip connections coming from previous layers. Furthermore, the maximum number of previous layers that can be used as input is now considered a new parameter, named the maximum number of levels back.

Table II summarizes the properties of Grammar Encodings.

V. DIRECT ENCODINGS

Direct encodings have been more widely used in the reviewed literature. Among the advantages they provide is the ease to construct the network from the encoding, the availability of well-known data structures able to handle them, and an important number of variation operators capable of manipulating them. Most of the research that lean toward direct encodings employs a Block-chained Encodings, followed in popularity by Graph-based Encodings.

A. Block-Chained Encodings

The Block-chained Encodings exploit the modular properties of a CNN architecture, i.e., the abstraction of a topology as a sequence of modules of different types and characteristics. Suitable data structures for these family of representations are the common array, linked lists, stacks, and queues. In general, linear data structures make a good fit for these encodings. Particularly, these models are almost *free-to-go*, as the construction of the CNN depends only on an ordered information extraction. As will be seen, the capabilities of these encodings depend on the chosen level of abstraction.

To standardize the terminology, we use the term *block* to refer to each module in the chain. Based on the available publications, blocks can be classified into low-level, medium-level, and high-level blocks, depending on the level of abstraction. A block with a higher abstraction level contains several standard CNN components [60], arranged in a predefined way. Lower abstraction blocks represent single layers only [61]. It is possible that during the search, a low-level Block-chained Encoding finds architectures similar to those in medium-level and high-level blocks. Nonetheless, the justification of using blocks of higher abstraction lies in the efficiency of some well-known architectures, such as ResNet [24] and DenseNet [21].

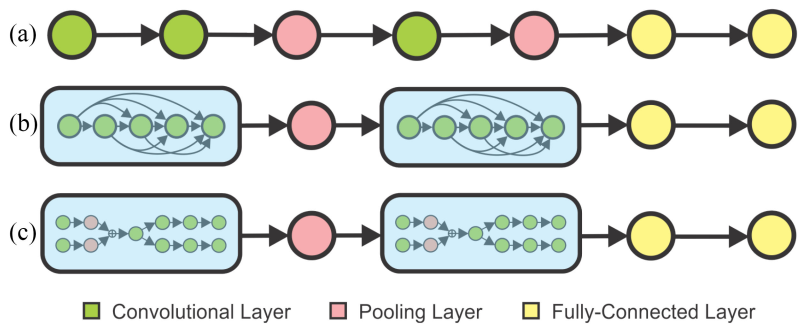

Fig. 6. Different abstraction level in block-chained encodings. (a) Low-level blocks representing simple layers [78]. (b) Medium-level blocks represent DenseBlocks [88]. (c) High-level blocks represent more complex arrangements of layers, similar to those of popular handmade CNNs [89].

Low-level blocks are used to represent the simplest of the CNN components, such as convolutional, pooling, and fully connected layers. With the lowest granularity, as in the work by Young *et al.* [62] (see also [63]–[68]), batch normalization, ReLU, and dropout are also considered as single blocks. In other works, such as [61], [69], and [70] batch normalization and/or ReLU activation function are included as part of convolutional blocks. The number of filters, their sizes, and stride values are usually contained inside each block. Pooling is fixed and employed only at the end of the chain.

Some authors utilized convolutional and pooling blocks only, while fixing a feedforward ANN at the end. In [71], each individual has an initial block that specifies its ID, its age, and its learning rate. Sun *et al.* [72] utilized convolutional and pooling blocks to evolve the encoder part of a convolutional autoencoder (CAE) as part of a classification system. The L2-norm regularization coefficients are included as the hyperparameters in convolutional blocks, besides the number of filters, filters' sizes, and so on. Similar works can be found in [73]–[75]. Hassanzadeh *et al.* [76], [77] evolved convolutional and pooling blocks for U-Net-like CNNs for image segmentation. In the former work, they evolved 2-D and 3-D CNNs. In the latter, they included the optimizer and the learning rate as part of each block. In both papers, the downsampling section (first half of the CNN) is repeated equally in the upsampling section (second half) by making the adequate adjustments. Skip connections and the so-called long connections of U-Net were also encoded.

Other encodings include fully connected blocks as well. The encoding proposed by Sun *et al.* [78] exemplifies the standardized version of this type of representation (see also [79]–[83]). As a novelty, the mean value and standard deviation of the weights of the convolutional filters inside convolutional blocks are also learned through the EA. Other similar approaches differ only in the number of encoded hyperparameters. Fernandes Jr. and Yen [84], for example, constrained the list of hyperparameters, consisting of the number of filters, filters' sizes, and number of neurons for the corresponding blocks. In contrast to [78] and [84], Martín *et al.* [85] evolved only the fully connected part of the CNNs. Martin *et al.* [86], [87] expanded the encoding used in [78] by including the optimizer, the number of training epochs, and the number of training samples at the beginning of the chain. Fig. 6(a) illustrates an example of a low-level Block-chained encoding, similar to the one used in [78].

A group of low-level Block-chained Encodings utilize multiple branches or chains. Hadjiivanov and Blair [90] favored the diversity by using a three-level structure: first, the genome level, related with speciation, is a skeleton of ordered blocks shared by all individuals belonging to the same species, second, the genotype levels, corresponding to the species, is the same genome with connections added between blocks, and finally, the phenotype level, that adds weights to the connections. Bochinski *et al.* [91] independently evolved a chain of convolutional blocks, and a chain of fully connected blocks, to be concatenated at the end (see also [92]–[94]). Kotyan and Vargas [95] co-evolved three populations: an architecture population that combines blocks in a block population, which mixes simple layers from a layer population.

Kim *et al.* [96] employed two equally sized chains, one containing different type of blocks and the other determining the number of outputs of each block. Dahou *et al.* [97] introduced a multibranch block chain, in which each path consists of a convolutional block and a max pooling block. The number of threads is equal to the number of convolutional filters in the blocks, and the resulting data from each subchain is concatenated at the end. Zhao *et al.* [98] employed a *supernetwork* based on the ShuffleNetV2 [99]. The EA evolves a path from this multibranch architecture, in which the option of each stage corresponds to a block (see also [100], [101]). A similar approach can be found in [102], which is a multibranch version of [78], and utilizes global pooling instead of fully connected layers. Kwasigroch *et al.* [103] designed a low-level Block-chained Encoding that includes skip connections, which aid in solving the vanishing-gradient problem [21]. A skip connection occurs when a layer sends its output to another nonconsecutive layer through a shortcut.

The types of available CNN components have diversified in the recent years, and so have the operators that can be used as blocks. Chu *et al.* [104] utilized a fixed convolutional backbone, and later evolved a series of convolutional blocks to perform three different types of operations; standard 2-D convolution, inverted bottleneck convolution, and grouped convolution. Awad *et al.* [105] also evolved convolutional blocks only, by using real numbers in the range

[0, 1], applying a different type of convolution depending on the range of the number corresponding to each block.

Medium-level blocks take abstraction one step further. Here, a block contains more than one operation or layer of the same type. By using this approach, a higher modularity is more easily achieved. Furthermore, the utilization of more abstract blocks comes with the advantage of applying well-established concepts in Deep Learning, such as skip connections. The hyperparameters of the individual layers are fixed in many cases, specially when the blocks correspond to those in architectures as ResNet and DenseNet.

Sapra and Pimentel [106] used a chain of medium-level blocks, called clusters, each containing a fixed number of convolutional, pooling, or fully connected layers. A similar approach is found in [107]–[110]. A constrained version of this chain was developed by Lu *et al.* [12], in which a maximum of five clusters can be used, with up to four layers inside each. Qu *et al.* [111] used blocks that can apply up to two different predefined operations in parallel, by concatenating the final result of each. Chen *et al.* [112] introduced a chain that starts with a 3×3 convolutional block. A small search space allows to choose the 20 consecutive blocks, and a large search space enforces the search of 40 blocks. The options are among ShuffleNetV2 [99] blocks of different sizes and the Xception block [113] of 3×3. Chen *et al.* [114] evolved chains of four blocks that contain convolutional, pooling, fully connected, and deconvolutional layers, for a convolutional variational autoencoder (VAE).

Liu *et al.* [115] used a stack to represent a CNN architecture in a hierarchical way. In this peculiar representation, the Lth level of the stack is a block recursively composed of all those previous levels. This configuration implies that the first levels correspond to low-level *motifs*, such as simple layers, meanwhile the medium and last levels are more abstract compositions.

As mentioned before, skip connections are a useful resource to improve the training phase of CNNs. Particularly, DenseNet and ResNet are two of the most important exponents of this architectural motif. A Residual Block or *ResBlock* is made of a series of convolutional layers, in which a skip connection connects the first's input to the last's output for channelwise summation [24]. In a DenseBlock, on the other hand, each of the layers connects to all the nonconsecutive layers through skip connections, and channelwise concatenation is applied between feature maps [21]. The *growth rate* g is a hyperparameter that is linked to the number of filters of each of the layers in a DenseBlock. The kth layer will receive $g(k-1)$ feature maps as input; thus, the growth rate defines a linear growth in the inputs of the layers. Wang *et al.* [116] utilized a single DenseBlock that is repeated several times (see also [117]). The assumption is that evolving and stacking multiple copies of the same component generates good performance at lower effort. Other encodings-based solely on DenseBlocks are found in [88] and [118], which encode the blocks as a two-integer vector of the form *(No. of Layers, Growth Rate)*. Sun *et al.* [119] employed only ResBlocks with two convolutional layers in each (see also [120]).

Sun *et al.* [121] proposed a block-chained encoding based on ResBlock and DenseBlocks of three and four layers, respectively, (see also [77], [122]). In [60], the DenseBlocks and ResBlocks are represented by three hyperparameters: 1) *type* for the type of block; 2) *output* for the size of output feature maps; and 3) *amount* for the number of layers. Pooling blocks are also considered and can appear only between the other types of blocks.

Other medium-level blocks contain layers connected in the form of fixed-size fixed-structure directed acyclic graphs [123]–[125]. Real *et al.* [126] utilized the so-called *normal cells* and *reduction cells*. The former contains a graph of layers that generates an output, whose size is equal to the input. The latter, on the other hand, reduces the output size, usually by half, with respect to the input. Furthermore, each cell can receive more than one input, and elementwise summation or concatenation is applied if needed.

Finally, *high-level blocks* take advantage of recently discovered structures that have advanced the performance of human-designed CNNs. Hassanzadeh *et al.* [127] evolved a chain of four possible attention blocks of a U-Net architecture for image segmentation. Each of them contains a configurable subblock in terms of the number of layers, number of filters, and their sizes, among other hyperparameters. Yao *et al.* [128] proposed to search for the components of the four modules that construct an object detection CNN. Each module has several possible options in the search space; backbone (versions of ResNet, MobineNetV2), feature fusion neck (enabled or not), region proposal network (one-stage detector or guided anchoring detector), and region-based CNN head (regular head, RetinaNet head, cascade head). Xu *et al.* [129] evolved an akin chain of high-level components for lane detection.

Gottapu and Dagli [130] encoded a series of *e-blocks* that contain n layers with skip connections. Each layer is made of a fixed template made of batch normalization, 1×1 convolution, ReLU, and an evolvable operator among a list of different types of convolutions. Song *et al.* [131] evolved the nonlinear mapping section of a CNN for single-image super resolution. This section is made of n blocks that are made of complex compositions of operations that combine Dense-like and Residual-like connectivity patterns, while also using different consecutive convolutional layers.

Zhang *et al.* [89] evolved the dense-like connectivity of a series of stages. The kth stage is made of n blocks, where each of them contains a concatenation of the two processing branches, followed by a series of evolvable subblocks that processed their inputs in parallel via a series of possible basic operators, to sum their outputs at the end (see Fig. 6(c)).

Table III organizes and summarizes the characteristics of Block-chained Encodings.

B. Graph-Based Encodings

Graph-based Encodings are related to Block-Chained Encodings, as both exploit the modularity of CNNs. A graph is a nonlinear data structure defined as a 2-tuple $G = (N, E)$, where N is the nonempty set of nodes and E is the nonempty set of edges connecting the nodes. Researchers have taken advantage of these structures to represent highly flexible

TABLE III
SUMMARY OF BLOCK-CHAINED ENCODINGS

	Publications	**Hyperparameters**	**Length**
Low-level	[65] [70] [74] [76] [77] [82] [100] [101] [104] [105]	Filters' size, No. of Filters, Pooling Type, Kernel Size, Stride, No. of Neurons, Activation Function, Batch Size, Learning rate, Optimizer, Connectivity	Fixed
	[61] [62] [63] [64] [66] [67] [68] [69] [71] [72] [73] [75] [78] [79] [80] [81] [83] [84] [85] [86] [87] [90] [91] [92] [93] [94] [95] [96] [97] [98] [102] [103]		Variable
Medium-level	[106] [109] [110] [12] [111] [112] [118] [120] [122] [123] [125] [126]	No. of Layers, No. of Filters, Pooling Type, Activation Function, Optimizer, Connectivity	Fixed
	[107] [108] [114] [115] [116] [119] [88] [117] [121] [127] [124] [117]		Variable
High-level	[128] [129] [130] [131] [132] [89]	Filters' size, Connectivity	Fixed

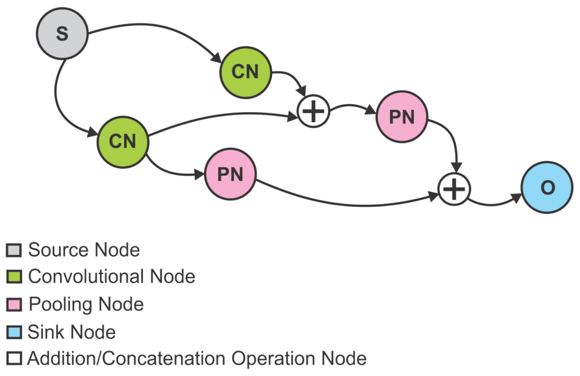

Fig. 7. Example of a graph encoding. Nodes represent operations, such as convolution and pooling. When more than one input is received inside a node, elementwise addition or concatenation might be applied.

CNNs. Nevertheless, flexibility comes at a cost of complexity. As a reminder, the construction of CNNs based on direct encodings usually consists in an ordered extraction of information. In graph-based encodings, the CNN information retrieval must be carefully conducted, as branches might have been created by the EA. Another inconvenience is that cycles are not permitted in this representation, as they could generate feedback loops, which are not considered in the original CNN. Thus, the variation operators must avoid cycles. There are two main types of Graph-based Encoding: 1) graph encodings and 2) tree encodings.

Graph encodings are the more general version of this family of representations and correspond to directed acyclic graphs. Similarly, as in low-level Block-chained Encodings, each node of the graph usually corresponds to a simple operator, such as a convolutional layer, and edges transfer the information between nodes.

Desell [132] utilized a graph, very similar to that used in NEAT. Instead of encapsulating neurons as nodes, the author classified them as input node, output node, and convolutional nodes. The idea is to let evolution optimize a CNN solely by determining the size of the filters and how they are connected (see also [133], [134]). Real *et al.* [135], in contrast, defined the edges as operators, e.g., convolution and pooling, whilst nodes are data holders represented by rank-3 tensors. If one node receives two edges as inputs of different dimensions, padding or interpolation is implemented as needed to adjust the size of a randomly selected input according to the other (see also [136], [137]).

Chen *et al.* [11] evolved a CNN composed of a sequence of graphs. Each graph can take up to two inputs that can be outputs from previous graphs. Here, each graph could be understood as being part of a medium-level block; however, the authors emphasized the predominant nature of the graphs in this encoding (see also [138]–[140]). Byla and Pang [141] utilized the Ant Colony Optimization algorithm, which allowed to build the graphs from scratch, instead of initializing a population of them. Yu *et al.* [142] utilized a *super network* represented as a grid, where each point corresponds to an operator. A graph is built by connecting different points of the grid (macrolevel structure), while the types of operators are defined by using the micro-level structure.

Miikkulainen *et al.* [143], and Liang *et al.* [144] maintained two populations of individuals using CoDeepNEAT. A population of *blueprint* chromosomes, consisting of graphs, whose nodes point to a second population of *module* chromosomes, which are also graphs containing several common operations of CNN. Both populations are evolved independently with the purpose of giving the EA a mechanism through which it could find its own modules. Fernando *et al.* [145] introduced the differentiable pattern-producing networks (DPPNs), based on the compositional pattern-producing networks (CPPNs) described in HyperNEAT [146]. DPPNs are represented by graphs, and map values from a coordinate vector to an output vector. This method relies on the geometric properties of data. The original CPPNs for CNNs were used in [147].

Suganuma *et al.* [148] utilized Cartesian genetic programming (CGP) to optimize the architectures of a CNN. In this framework, the CNNs are graphs of so called Highly Functional Modules, which are convolutional layers, ResBlocks, max pooling layers, and operations of summation and concatenation. A second attempt to use CGP was proposed in [149], where a string of integers describes the different nodes in the graph and their connectivity (see also [150]). An example of a graph encoding is shown in Fig. 7.

Tree encodings utilize the tree data structures, and are particularly well suited for genetic programming (GP). Irwin-Harris *et al.* utilized the *leaf nodes* in the tree as input nodes, while the *root node* corresponds to the output. All the intermediate nodes can take the form of convolutional, pooling or concatenation/summing operations. As in [151]–[153], only the convolutional section of a CNN is evolved. Bi *et al.* [154] also included operators, such as square root, absolute value,

TABLE IV
SUMMARY OF GRAPH-BASED ENCODINGS

	Publications	Hyperparameters	Length
Graph	[133] [134] [135] [136] [137] [138] [11] [142] [139] [140] [141] [143] [144] [145] [146] [148] [149] [150] [151]	No. of Layers, No. of Filters, Filters' size, Pooling Type, Kernel Size, Batch Norm., Connectivity	Variable
Tree	[152] [153] [154] [155] [156]		

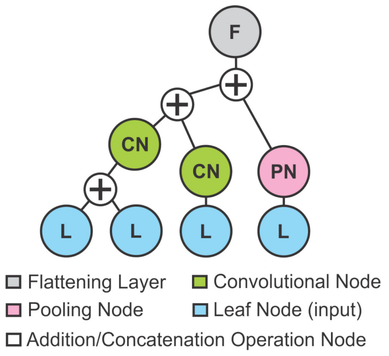

Fig. 8. Tree Encoding such as the one described by Irwin-Harris *et al.* [152].

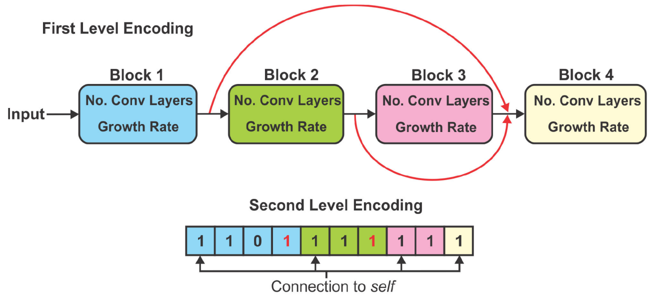

Fig. 9. Example of the hybrid encoding proposed by Wang *et al.* [163].

and weighted subtraction. Evans *et al.* [155] incorporated statistical operators, such as the mean and standard deviation, and a set of simple operators near the end of the tree, such as addition and product, in order to let GP to evolve its own classifier after the feature extraction. Fig. 8 illustrates an example of a tree encoding.

Table IV summarizes the properties of Graph-based Encodings.

VI. Hybrid Encodings

Hybrid encodings are a family of neural encodings that combine ideas from both direct and indirect encodings schemes at some extent. The research toward novel, often *out of the ordinary* encodings is highly desired in order to tackle some of the limitations one or the other kind of strategy has. Additionally, other promising encodings based on hyperparameters and on real numbers are also considered in this category.

Block-chained Encodings and Graph-based Encodings have been mixed together to achieve higher flexibility. Lu *et al.* [156] utilized a series of medium-level blocks represented by evolvable graphs, which is called *operation encoding*. The *path encoding* is an integer vector whose entries determine the connectivity between blocks. The encoding by Zhang *et al.* [157] utilized the previously discussed normal cells and reduction cells, however, the graphs inside blocks are not entirely fixed. Each node contains information about its inputs and its outputs, which details the connectivity of the graphs (see also [158]).

Graph-based Encodings and Binary Encodings have also been combined to take advantage of the capabilities of the later to represent the connectivity in the former. Wyk and Bosman [159] utilized a binary adjacency matrix for the connectivity of an directed acyclic graph of CNN operators. Kang and Ahn [160] used a chromosome composed of n pairs of vectors. The first vector in each pair contains a series of k convolutional and pooling layers, while the second vector encapsulates $k + 1$ binary strings that turns on and off the different connections of each operator. In the end, each pair of vectors describes a graph inside a block. Yotchon and Jewajinda [161] employed a binary string to interconnect a series of operations in graphs that constitute cells inside medium-level blocks (see also [162]).

Wang *et al.* [163] combined a Block-chained Encoding and a Binary Encoding. The *first-level encoding* is a chain of DenseBlocks with a number of layers and a growth rate. The *second-level encoding* is a binary string attached to each DenseBlock in order to evolve the connectivity patterns of the inner layers (1 corresponds to an existing connection, and 0 means otherwise). Fig. 9 illustrates this encoding method. The connections between reduction cells and normal cells have also been represented with binary strings in [164].

Some methods encoded CNNs hyperparameters into vectors. These approaches are important to Hybrid Encodings, as these components could be evolved separately from the architecture's skeleton. Liu *et al.* [165] evolved a FL integer vector of hyperparameters that can generate topologies of different sizes (similarly as in [166]). A similar approach was utilized in [167], but the size of the final CNN is always fixed. Mostafa *et al.* [168] utilized a real-based vector to represent hyperparameters. Baldeon-Calisto and Lai-Yuen [169] combined real-based hyperparameters, such as the learning rate and the dropout rate, and integer-based hyperparameters, such as the number of filters per layer and the filters' sizes of three layers in a U-Net.

Finally, real-based encodings are starting to be explored. Most of the previously mentioned representations are used for discrete optimization. However, recent works on the gradient-based Neural Architecture Search have demonstrated that it is possible to use continuous optimization methods for the design of CNN architectures [170], [171]. Kobayashi and Nagao [172] encoded an acyclic graph in which nodes are data holders, and edges are operations. Between nodes i and j, there is a possible operation o_{ij} with a weight w_{ij}^{n}. A real vector determines which one, from a list operations, is utilized between each pair of nodes. Javaheripi *et al.* [173] used a real-based vector for pruning a CNN architecture. Each value corresponds to a filter in a predefined architecture, and values closer to 0 deactivate the filters. An special mention on [174] and [175] is to be made, as their method can be easily adapted to several types of encodings.

Table V organizes and summarizes the Hybrid Encodings.

VII. Discussion

This work has presented the trends on how to successfully encode a CNN for an EA to manipulate. Based on the literature review above, there is an ample range of options. The NE research community has the important challenge of choosing an encoding, as there does not exist any strict empirical evidence on the impact of each subfamily. Moreover, the benefits and losses of different methods have not been neither holistically analyzed nor discussed until now. In this section, we examine advantages and disadvantages of the encodings schemes based on different categories of interest. Furthermore, we point out important niches of opportunity for future research involving neural encodings for NE.

A. Strengths and Weaknesses

First, we focus the discussion on the general, application-agnostic strengths, and weaknesses of the reviewed methods.

Direct encodings come with the advantage of not requiring a decoding step. Additionally, the vast amount of tested methods provide the interested users for different variation operators that can be introduced into several metaheuristics. Block-chained Encodings are easy to handle and are compatible with the modular nature of CNNs. Graph-based Encodings go one step further in flexibility, allowing to find more complex architectures if needed. As a drawback, Graph-based Encodings are harder to manipulate. Also, excessive modularity might result in human bias during search, as well-known handmade substructures come into play.

Indirect Encodings are generally more compact and easier to manipulate. Binary Encodings are appropriate to a diverse range of variation operators and are useful to represent connections and also for Neural Architecture Pruning. Grammar Encodings are well managed by grammatical evolution (GE) and can adapt to build different types of networks. Indirect Encodings are also related to the *genetic bottleneck* problem. The biological genome is by several orders of magnitude reduced in capacity compared with the number of neural connections it encodes [176]. For this reason, it is reasonable to expect that an indirect representation should suffice in developing very complex and powerful CNN architectures. Unadvantageously, Indirect Encodings are less flexible, and usually encapsulate a reduced, less diverse search space.

Hybrid Encodings have the potential of addressing the disadvantages of each family by combining their assets. In general terms, we recommend Block-chained Encodings as the first option for users, due to the high volume of research relying on this method, and the accumulated success evidence of several research works that utilize the same encodings. However, more investigation on novel encodings is highly encouraged.

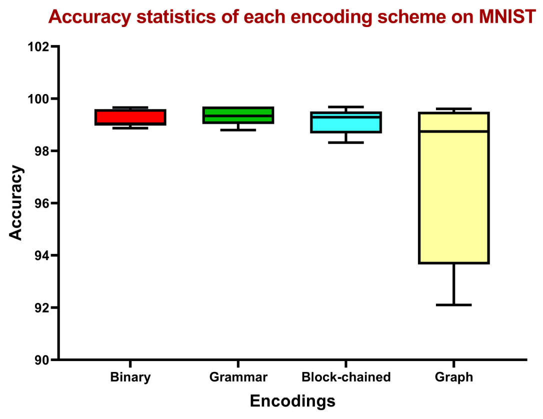

Fig. 10. Accuracy distributions for each encoding on the MNIST dataset. None of the reviewed hybrid encodings utilized this dataset, thus such a scheme was not included.

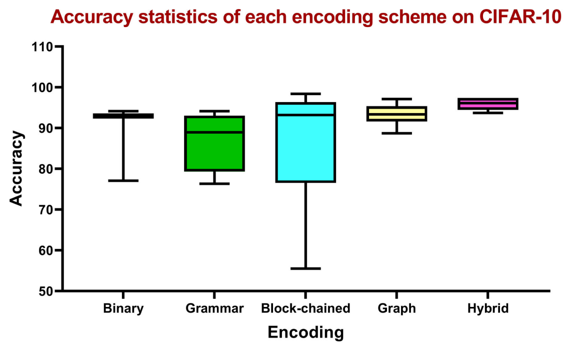

Fig. 11. Accuracy distributions for each encoding on the CIFAR-10 dataset.

B. Performance

The reviewed encoding schemes have shown highly competitive results, with the majority of them being applied for image classification. Accordingly, we use this scenario to analyze the performance of the different encoding families on the most extensively chosen datasets: MNIST and CIFAR-10. Figs. 10 and 11 display the statistical summary of accuracy results of each encoding family from the reviewed literature on the MNIST and CIFAR-10 datasets, respectively. None of the Hybrid Encodings was tested on the former.

On MNIST, Binary, Grammar, and Block-chained Encodings reached more robust results in comparison to Graph-based Encodings. Due to the more challenging nature of CIFAR-10 classification, more diverse outcomes are observed. Block-chained Encodings, specifically, have the largest range in performance (specially downward). This might be due to the diversity in the types of blocks that have been explored. Those approaches relying on Hybrid Encodings have acquired the highest and most robust results. Although these data do not intend to be fully conclusive, these results are good indicators that: 1) some encodings might work better than others depending on the application and 2) that the selection of an encoding plays an important role [84]. Hence, the study of neural encodings is very relevant.

Encouragingly, several works have explored the application of NE for CNNs to other applications in recent years, including

TABLE V
SUMMARY OF HYBRID ENCODINGS

	Publications	Hyperparameters	Length
Hybrid	[157] [158] [159] [164] [165]	No. of Filters, Filters' Size, Connectivity	Variable
	[160] [161] [162] [163]		Fixed
Hyperparameters	[166] [167] [168] [169] [170] [173] [174]	No. of Filters, Filters' size, Pooling Type, Kernel Size, Learning rate, Optimizer, Activation Function, Connectivity	Fixed

image segmentation [49], [76], [77], [122], [127], [128], [142], [169], object detection [75], [98], [159], image enhancement and denoising [104], [131], [165], video-game playing and reinforcement learning [74], [120], speech and language recognition [97], [111], and intrusion detection [137]. It has been observed that as the application increases its complexity, so does the encoding. For example, many of the medium-level and high-level encodings were applied to object detection, which is a complex task that has been approached using more complex CNNs than those for image classification. Following this evidence, we suggest to always consider how much search power would be required for an NE algorithm to find structures similar to those that have been successful for a given task, e.g., attention blocks, and to level up the modularity of the chosen encoding accordingly.

C. Computational Power

The availability of computational power is another aspect worthwhile taking into account. NE comes with the disadvantage of being computationally expensive. Although the benefits outweigh this obstacle, it is important to take informed decisions on the encoding to be used. If computational resources are limited, Indirect Encoding or low-level Block-chained Encodings are preferred, as they usually map to a reduced or fixed search space. Other Black-chained Encodings and Graph Encodings might rapidly increase the size of the CNNs, requiring more training and inference time. Furthermore, Neural Architecture Pruning methods based on Binary Encoding can be useful to fit an existing CNN into the available computational resource.

If computational resources are not constrained, choosing an encoding might depend on the complexity of the application. It has been seen that tasks, such as object detection and image segmentation, for example, require a more complex arrangement of CNN components. Graph-based Encodings and medium-level and high-level Block-chained Encodings are suggested, as more abstraction can and flexibility can be achieved in order to solve those tasks. When the size of the data is considerable, these encodings are also useful to avoid underfitting.

Hybrid Encodings can separate the representation into two different search spaces. This can be helpful to simplify the search process at some extent, by reducing the computation power consumption whilst maintaining a remarkable flexibility.

D. Expressiveness

The neural encodings display and interesting, yet concerning tradeoff between modularity and expressiveness. An

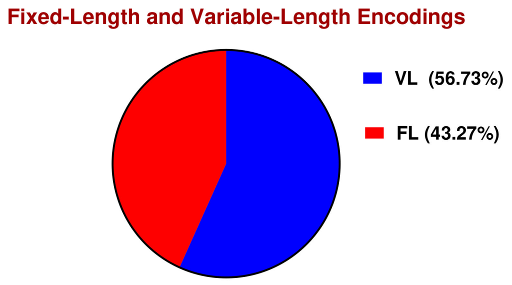

Fig. 12. Distribution of VL encodings and FL encodings among the reviewed papers.

encoding with higher modularity corresponds to less flexible, i.e., expressive CNNs. Medium-level and High-level Block-chained Encodings, for instance, utilize well-known structures or human-made architectures, which limits the EA to freely discover novel topologies. On the other hand, those encodings that demonstrate more flexibility could not be suitable to discover novel modules that can be reutilized (such as DenseBlocks and ResBlocks). Low-level Block-chained Encodings, Graph-based Encodings, and Grammar Encodings have a better expressiveness, yet less modularity.

Another aspect to consider is the length of the encodings. As shown in Fig. 12, VL encodings predominate over FL encodings. Encodings with variable size are challenging as they require very specialized variation operators, and the search space is in constant change. However, these types of representation favor *complexification*, which means that the architectures become increasingly more complex as the task demands it. If there is not a clear conception of how complex a CNN should be, a VL, usually less modular Indirect Encoding is suggested.

E. Usability

Another point to take into account, is the end user. Many works emphasize that designing a CNN architecture is a trial-and-error tedious work. The designer's expertise, not only in Deep Learning but also in the application domain, is central for the success of CNNs. If an NE framework focuses on the accessibility for more general users, e.g., the industry staff or researchers in other fields, Direct Encodings are a first choice. These encodings usually have a self-explanatory nature. Block-chained encodings, for example, clearly state the size, the order, and the internal characteristics of the network. Graph-based encodings follow in this line. However, the level of abstraction in the nodes plays an important role in the self-explainability of the encoding. If an Indirect Encoding is to be used, it is advisable to provide for rules not only to construct the CNN, but to explain its architecture to the user. Grammar Encodings are explainable only to people experienced in formal languages, yet being a difficult task. Binary Encodings, on the other hand, tend to be completely *obscure*.

F. Future Challenges

Researchwise, the literature shows that there are still open questions and challenges in what we call *neural enconding engineering*. The impact of the different encoding strategies is subject to analysis; as many research works have demonstrated outstanding results, it is still not clear what makes a particular encoding better than the others, with respect to the different kinds of data. Further investigation is required to evaluate the impact of several encodings on the same task. On the other hand, Hybrid Encodings are promising for a number of reasons. As the previous section revealed, Hybrid Encodings are the mixture of two or possibly more schemes. One advantage of using two types of encodings with different classes is to separate *representation duties*, i.e., the aspects of the network to be present in the encoding. Hybrid representations would allow to separate, for example, connectivity from hyperparameters. Binary Encodings could be a good choice for connectivity and medium-level blocks can be a good tradeoff between abstraction and flexibility. Using formal grammars to express connectivity patterns is also subject for experimentation.

From another point of view, the solution to the problem of encoding CNNs might be a *matter of root*. Novel data structures could improve the way these networks are interpreted. We are constrained to the current available data structures to a degree and, as in general Evolutionary Computation, they represent an important niche of research.

Similarly as in CNNs, other ANNs architectures have attracted the attention of the NE community. Hence, it is also encouraged to develop a similar encoding taxonomy for other neural architectures such as Recurrent Neural Networks and Autoencoders.

As long as Deep Learning continues to exhibit favorable outcomes, more powerful automatic algorithm generation techniques will flourish. NE is going to play an important role toward that goal and more than ever requires the attention of researchers to unravel the evolution of ANNs.

VIII. Conclusion

In NE of CNNs, the encoding is essential for the design of the EA and is a pivotal part in the optimization performance. Through a comprehensive review on the literature from the last decade, a novel taxonomy of encodings for CNNs has been proposed: Binary Encodings, Grammar Encodings, Block-chained Encodings, Graph-based Encodings, and Hybrid Encodings. Some notions about how to choose an encoding have been described based on elements such as the complexity of the problem to solve, and the available computational resources.

Although this field is becoming more active through the years, more research is required in order to better understand the effects on each type of encoding for several applications and different metaheuristics. This novel taxonomy on encodings intends to direct future research toward the design of better and more specialized encodings for CNNs.

References

[1] K. O. Stanley, J. Clune, J. Lehman, and R. Miikkulainen, “Designing neural networks through neuroevolution,” *Nat. Mach. Intell.*, vol. 1, no. 1, pp. 24–35, Jan. 2019.

[2] F. J. Gomez, J. Schmidhuber, and R. Miikkulainen, “Efficient non-linear control through neuroevolution,” in *Proc. Eur. Conf. Mach. Learn.*, 2006, pp. 654–662.

[3] P. Verbancsics and J. Harguess, “Image classification using generative neuro evolution for deep learning,” in *Proc. IEEE Winter Conf. Appl. Comput. Vis.*, 2015, pp. 488–493.

[4] A. Kizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” in *Proc. 25th Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[5] A. Baldominos, Y. Saez, and P. Isasi, “On the automated, evolutionary design of neural networks: Past, present, and future,” *Neural Comput. Appl.*, vol. 32, no. 2, pp. 519–545, Mar. 2019.

[6] A. Darwish, A. E. Hassanien, and S. Das, “A survey of swarm and evolutionary computing approaches for deep learning,” *Artif. Intell. Rev.*, vol. 53, no. 3, pp. 1767–1812, May 2020.

[7] E. Galvin and P. Mooney. *Neuroevolution in Deep Neural Networks: Current Trends and Future Challenges*. Acccessed: Jun. 2020. [Online]. Available: https://arxiv.org/pdf/2006.05415.pdf

[8] Y. Liu, Y. Sun, B. Xue, M. Zhang, G. G. Yen, and K. C. Tan. (2020). *A Survey on Evolutionary Neural Architecture Search*. [Online]. Available: https://arxiv.org/abs/2008.10937

[9] M. Suchorzewski, “Evolving scalable and modular adaptive networks with developmental symbolic encoding,” *Evol. Intell.*, vol. 4, no. 3, pp. 145–163, May 2011.

[10] A. Eiben and J. Smith, *Introduction to Evolutionary Computing*, 2nd ed. Heidelberg, Germany: Springer, 2015.

[11] Y. Chen *et al.*, “RENAS: Reinforced evolutionary neural architecture search,” in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 4782–4791.

[12] Z. Lu, G. Sreekumar, E. Goodman, W. Banzhaf, K. Deb, and V. N. Boddeti, “Neural architecture transfer,” *IEEE Trans. Pattern Anal. Mach. Intell.*, early access, Jan. 19, 2021, doi: 10.1109/TPAMI.2021.3052758.

[13] J. Fekiač, I. Zelinka, and J. C. Burguillo, “A review of methods for encoding neural networks topologies in evolutionary computation,” in *Proc. 25th Eur. Conf. Model. Simulat. ECMS*, 2011, pp. 410–416.

[14] 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, May 1998.

[15] Y. Kim, “Convolutional neural networks for sentence classification,” in *Proc. Conf. Empirical Methods Nat. Lang. Process. (EMNLP)*, 2014, pp. 1746–1751.

[16] Y. Han, J. Kim, and K. Lee, “Deep convolutional neural networks for predominant instrument recognition in polyphonic music,” *IEEE/ACM Trans. Audio, Speech, Language Process.*, vol. 25, no. 1, pp. 208–221, Jan. 2017.

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

[18] R. H. Hahnloser, R. Sarpeshkar, M. A. Mahowald, R. J. Douglas, and H. S. Seung, “Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit,” *Nature*, vol. 405, pp. 947–951, May 1998.

[19] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, “How does batch normalization help optimization?” in *Proc. 31st Adv. Neural Inf. Process. Syst.*, 2018, pp. 2483–2493. [Online]. Available: http://papers.nips.cc/paper/7515-how-does-batch-normalization-help-optimization.pdf

[20] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2015, pp. 1–6.

[21] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2017, pp. 4700–4708.

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

[23] K. O. Stanley, "Efficient evolution of neural networks through complexification," Ph.D. dissertation, Dept. Comput. Sci., Univ. Texas Austin, Austin, TX, USA, 2004.

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

[25] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proc. 12th USENIX Symp. Oper. Syst. Design Implement.*, 2016, pp. 265–283.

[26] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 8026–8037.

[27] V. Bibaeva, "Using metaheuristics for hyper-parameter optimization of convolutional neural networks," in *Proc. IEEE 28th Int. Workshop Mach. Learn. Signal Process. (MLSP)*, 2018, pp. 1–6.

[28] H. Chung and K. S. Shin, "Genetic algorithm-optimized multi-channel convolutional neural network for stock market prediction," *Neural Comput. Appl.*, vol. 32, no. 12, pp. 7897–7914, 2020.

[29] Y. Li, J. Xiao, Y. Chen, and L. Jiao, "Evolving deep convolutional neural networks by quantum behaved particle swarm optimization with binary encoding for image classification," *Neurocomputing*, vol. 362, no. 14, pp. 156–165, Oct. 2019.

[30] A. Baldominos, Y. Saez, and P. Isasi, "Evolutionary convolutional neural networks: An application to handwriting recognition," *Neurocomputing*, vol. 283, no. 29, pp. 38–52, Mar. 2018.

[31] A. Baldominos, Y. Saez, and P. Isasi, "Hybridizing evolutionary computation and deep neural networks: An approach to handwriting recognition using committees and transfer learning," *Complexity*, vol. 2019, Mar. 2019, Art. no. 2952304.

[32] 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)*, 2018, pp. 1–8.

[33] 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. Aust. Joint Conf. Artif. Intell.*, 2018, pp. 237–250.

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

[35] L. Xie and A. Yuille, "Genetic CNN," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, 2017, pp. 1379–1388.

[36] R. Akut and S. Kulkami, "NeuroEvolution: Using genetic algorithm for optimal design of deep learning models," in *Proc. IEEE Int. Conf. Elect. Comput. Commun. Technol. (ICECCT)*, 2019, pp. 1–6.

[37] M. Ahmad, M. Abdullah, H. Moon, S. J. Yo, and D. Han, "Image classification based on automatic neural architecture search using binary crow search algorithm," *IEEE Access*, vol. 8, pp. 189891–189912, 2020.

[38] A. Ma, Y. Wan, Y. Zhong, J. Wang, and L. Zhang, "SceneNet: Remote sensing scene classification deep learning network using multi-objective neural evolution architecture search," *ISPRS J. Photogrammetry Remote Sens.*, vol. 172, pp. 171–188, Feb. 2021.

[39] S. Lee, J. Kim, H. Kang, D.-Y. Kang, and J. Park, "Genetic algorithm based deep learning neural network structure and hyperparameter optimization," *Appl. Sci.*, vol. 11, no. 2, p. 744, 2021.

[40] J. Jiang, F. Han, Q. Ling, J. Wang, T. Li, and H. Han, "Efficient network architecture search via multiobjective particle swarm optimization based on decomposition," *Neural Netw.*, vol. 123, pp. 305–316, Mar. 2020.

[41] J. Liu, S. Zhou, Y. Wu, K. Chen, W. Ouyang, and D. Xu, "Block proposal neural architecture search," *IEEE Trans. Image Process.*, vol. 30, pp. 15–25, 2020.

[42] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2015, pp. 1–9.

[43] W. Ye, R. Liu, Y. Li, and L. Jiao, "Quantum-inspired evolutionary algorithm for convolutional neural networks architecture search," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, 2020, pp. 1–8.

[44] Y. Zhou, G. G. Yen, and Z. Yi, "Evolutionary compression of deep neural networks for biomedical image segmentation," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 8, pp. 2916–2929, Aug. 2020.

[45] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention (MICCAI)* (Lecture Notes in Computer Science), vol. 9351. Munich, Germany: Springer, 2015, pp. 6642–6652.

[46] Y. Wang, C. Xu, J. Qiu, C. Xu, and D. Tao, "Towards evolutionary compression," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Disc. Data Min.*, 2018, pp. 2476–2485.

[47] Y. Zhou, G. G. Gen, and Z. Yi, "A knee-guided evolutionary algorithm for compressing deep neural networks," *IEEE Trans. Cybern.*, vol. 51, no. 3, pp. 1626–1638, Mar. 2021.

[48] F. E. Fernandes Jr., and G. G. Yen, "Pruning deep convolutional neural networks architectures with evolution strategy," *Inf. Sci.*, vol. 552, pp. 29–47, Apr. 2021.

[49] P. R. Lorenzo and J. Nalepa, "Memetic evolution of deep neural networks," in *Proc. Genet. Evol. Comput. Conf.*, 2018, pp. 505–512.

[50] D. O'Neill, B. Xue, and M. Zhang, "The evolution of adjacency matrices for sparsity of connection in DenseNets," in *Proc. Int. Conf. Image Vis. Comput. New Zealand (IVCNZ)*, 2019, pp. 1–6.

[51] D. O'Neill, B. Xue, and M. Zhang, "Neural architecture search for sparse densenets with dynamic compression," in *Proc. Genet. Evol. Comput. Conf.*, 2020, pp. 386–394.

[52] M. O'Neill and C. Ryan, "Grammatical evolution," *IEEE Trans. Evol. Comput.*, vol. 5, no. 4, pp. 349–358, Aug. 2001.

[53] A. Baldominos, Y. Saez, and P. Isasi, "Evolutionary design of convolutional neural networks for human activity recognition in sensor-rich environments," *Sensors*, vol. 18, no. 4, p. 1288, Apr. 2018.

[54] R. H. Lima, A. T. Pozo, and R. Santana, "Automatic design of convolutional neural networks using grammatical evolution," in *Proc. IEEE 8th Braz. Conf. Intell. Syst. (BRACIS)*, 2019, pp. 329–334.

[55] F. Assunção, N. Lourenco, P. Machado, and B. Ribeiro, "DENSER: Deep evolutionary network structured representation," *Genet. Program. Evol. Mach.*, vol. 20, no. 1, pp. 5–35, 2018.

[56] F. Assunção *et al.*, "Automatic design of artificial neural networks for gamma-ray detection," *IEEE Access*, vol. 7, pp. 110531–110540, 2019.

[57] T. Cetto, J. Byrne, X. Xu, and D. Moloney, "Size/accuracy trade-off in convolutional neural networks: An evolutionary approach," in *Proc. INNS Big Data Deep Learn. Conf.*, 2019, pp. 17–26.

[58] F. Assuncao, N. Lourenco, P. Machado, and B. Ribeiro, "Fast denser: Efficient deep neuroevolution," in *Proc. Eur. Conf. Genet. Program.*, 2019, pp. 197–212.

[59] F. Assuncao, N. Lourenco, B. Ribeiro, and P. Machado, "Incremental evolution and development of deep artificial neural networks," in *Proc. Eur. Conf. Genet. Program.*, 2020, pp. 35–51.

[60] Y. Sun, H. Wang, B. Xue, Y. Jin, G. G. Yen, and M. Zhang, "Surrogate-assisted evolutionary deep learning using an end-to-end random forest-based performance predictor," *IEEE Trans. Evol. Comput.*, vol. 24, no. 2, pp. 350–364, Apr. 2020.

[61] J. Prellberg and O. Kramer, "Lamarckian evolution of convolutional neural networks," in *Proc. Int. Conf. Parallel Probl. Solving Nat.*, 2018, pp. 424–435.

[62] S. R. Young *et al.*, "Evolving deep networks using HPC," in *Proc. Mach. Learn. HPC Environ.*, 2017, pp. 1–7.

[63] R. Keshari. (2017). *DEVOL: Automated Deep Neural Network Design Via Genetic Programming*. [Online]. Available: https://github.com/RohitKeshari/devol

[64] J.-D. Dong, A.-C. Juan, W. Wei, and M. Sun, "PPP-NET: Platform-aware progressive search for Pareto-optimal neural architectures," in *Proc. Int. Conf. Learn. Represent. (ICLR) Workshop*, 2018, pp. 1–4.

[65] A. A. Ahmed, S. M. S. Darwish, and M. M. El-Sherbiny, "A novel automatic CNN architecture design approach based on genetic algorithm," in *Proc. Int. Conf. Adv. Intell. Syst. Informat.*, Cairo, Egypt, 2019, pp. 473–482.

[66] F. M. Johner and J. Wassner, "Efficient evolutionary architecture search for CNN optimization on GTSRB," in *Proc. 18th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA)*, 2019, pp. 56–61.

[67] S. Fujino, T. Hatanaka, N. Mori, and K. Matsumoto, "Evolutionary deep learning based on deep convolutional neural network for anime storyboard recognition," *Neurocomputing*, vol. 338, pp. 393–398, Apr. 2019.

[68] M. Loni *et al.*, "DenseDisp: Resource-aware disparity map estimation by compressing Siamese neural architecture," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, 2020, pp. 1–8.

[69] A. Rikhtegar, M. Pooyan, and M. T. Manzuri-Shalmani, "Genetic algorithm-optimised structure of convolutional neural network for face recognition applications," *IET Comput. Vis.*, vol. 10, no. 6, pp. 559–566, 2016.

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

[71] F. Badan and L. Sekanina, "Optimizing convolutional neural networks for embedded systems by means of neuroevolution," in *Proc. Int. Conf. Theory Practice Nat. Comput.*, 2019, pp. 109–121.

[72] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "A particle swarm optimization-based flexible convolutional autoencoder for image classification," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 8, pp. 2295–2309, Aug. 2019.

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

[74] S. M. J. Jalali, P. M. Kebria, A. Khosravi, K. Saleh, D. Nahavandi, and S. Nahavandi, "Optimal autonomous driving through deep imitation learning and neuroevolution," in *Proc. IEEE Int. Conf. Syst. Man Cybern. (SMC)*, 2019, pp. 1215–1220.

[75] K. R. G. Operiano, H. Iba, and W. Pora, "Neuroevolution architecture backbone for $x-$ray object detection," in *Proc. IEEE Symp. Comput. Intell. (SSCI)*, 2020, pp. 2296–2303.

[76] T. Hassanzadeh, D. Essam, and R. Sarker, "EVOU-NET: An evolutionary deep fully convolutional neural network for medical image segmentation," in *Proc. 35th Annu. ACM Symp. Appl. Comput.*, 2020, pp. 181–189.

[77] T. Hassanzadeh, D. Essam, and R. Sarker, "2D to 3D evolutionary deep convolutional neural networks for medical image segmentation," *IEEE Trans. Med. Imag.*, vol. 40, no. 2, pp. 712–721, Feb. 2021.

[78] 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.

[79] S. R. Young, D. C. Rose, T. P. Karnowski, S.-H. Lim, and R. M. Patton, "Optimizing deep learning hyper-parameters through an evolutionary algorithm," in *Proc. Workshop Mach. Learn. High Perform. Comput. Environ.*, 2015, pp. 1–5.

[80] Q. Zhang, B. Li, and Y. Wu, "Evolutionary structure optimization of convolutional neural networks for deployment on resource limited systems," in *Proc. Int. Conf. Intell. Comput.*, 2018, pp. 742–753.

[81] N. Mitschke, M. Heizmann, K.-H. Noffz, and R. Wittmann, "Gradient based evolution to optimize the structure of convolutional neural networks," in *Proc. 25th IEEE Int. Conf. Image Process. (ICIP)*, 2018, pp. 3438–3442.

[82] S. Litzinger, A. Klos, and W. Schiffmann, "Compute-efficient neural network architecture optimization by a genetic algorithm," in *Proc. Int. Conf. Artif. Neural Netw.*, 2018, pp. 886–893.

[83] B. Dahal and J. Zhan, "Effective mutation and recombination for evolving convolutional networks," in *Proc. 3rd Int. Conf. Appl. Intell. Syst.*, 2020, pp. 1–6.

[84] F. E. Fernandes Jr., and G. G. Yen, "Particle swarm optimization of deep neural networks architectures for image classification," *Swarm Evol. Comput.*, vol. 49, pp. 62–74, Sep. 2019.

[85] A. Martín, V. M. Vargas, P. A. Gutiérrez, D. Camacho, and C. Hervás-Martínez, "Optimising convolutional neural networks using a hybrid statistically-driven coral reef optimisation algorithm," *Appl. Soft Comput.*, vol. 90, May 2020, Art. no. 106144.

[86] A. Martin, F. Fuentes-Hurtado, V. Naranjo, and D. Camacho, "Evolving deep neural networks architectures for Android malware classification," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, 2017, pp. 1659–1666.

[87] A. Martín, R. Lara-Cabrera, F. Fuentes-Hurtado, V. Naranjo, and D. Camacho, "EvoDeep: A new evolutionary approach for automatic deep neural networks parametrisation," *J. Parallel Distrib. Comput.*, vol. 117, pp. 180–191, Jul. 2018.

[88] B. Wang, Y. Sun, B. Xue, and M. Zhang, "Evolving deep neural networks by multi-objective particle swarm optimization for image classification," in *Proc. Genet. Evol. Comput. Conf.*, 2019, pp. 490–498.

[89] N. Zhang, J. Wang, J. Yang, X. Qu, and J. Xiao, "Multi-objective cuckoo algorithm for mobile devices network architecture search," in *Proc. Int. Conf. Artif. Neural Netw. (ICANN)*, 2020, pp. 312–324.

[90] A. Hadjiivanov and A. Blair, "Epigenetic evolution of deep convolutional models," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, 2019, pp. 1478–1486.

[91] E. Bochinski, T. Senst, and T. Sikora, "Hyper-parameter optimization for convolutional neural network committees based on evolutionary algorithms," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, 2017, pp. 3924–3928.

[92] I. Strumberger, E. Tuba, N. Bacanin, M. Zivkovic, M. Beko, and M. Tuba, "Designing convolutional neural network architecture by the firefly algorithm," in *Proc. IEEE Int. Young Eng. Forum (YEF-ECE)*, 2019, pp. 59–65.

[93] P. Vidnerová and R. Neruda, "Multi-objective evolution for deep neural network architecture search," in *Proc. Int. Conf. Neural Inf. Process.*, 2020, pp. 270–281.

[94] P. Vidnerová, Š. Procházka, and R. Neruda, "Multi-objective evolution for convolutional neural network architecture search," in *Proc. Int. Conf. Artif. Intell. Soft Comput.*, 2020, pp. 261–270.

[95] S. Kotyan and D. V. Vargas, "Towards evolving robust neural architectures to defend from adversarial attacks," in *Proc. Genet. Evol. Comput. Conf.*, 2020, pp. 135–136.

[96] Y.-H. Kim, B. Reddy, S. Yun, and C. Seo, "NEMO: Neuro-evolution with multiobjective optimization of deep neural network for speed and accuracy," in *Proc. JMLR Workshop Conf.*, 2017, pp. 1–8.

[97] A. Dahou, M. A. Elaziz, J. Zhou, and S. Xiong, "Arabic sentiment classification using convolutional neural network and differential evolution algorithm," *Comput. Intell. Neurosci.*, vol. 2019, Feb. 2019, Art. no. 2537689.

[98] Z. Zhao, M. Jiang, S. Guo, Z. Wang, F. Chao, and K. C. Tan, "Improving deep learning based optical character recognition via neural architecture search," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, 2020, pp. 1–7.

[99] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "ShuffleNet V2: Practical guidelines for efficient CNN architecture design," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, vol. 11218, 2018, pp. 122–138.

[100] W. Wang and L. Zhu, "Reliable network search based on evolutionary algorithm," in *Proc. IEEE Int. Conf. Comput. Control Robot. (ICCCR)*, 2021, pp. 279–282.

[101] H. Tang *et al.*, "Searching efficient 3D architectures with sparse point-voxel convolution," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2020, pp. 685–702.

[102] J. Dong, L. Zhang, B. Hou, and L. Feng, "A memetic algorithm for evolving deep convolutional neural network in image classification," in *Proc. IEEE Symp. Comput. Intell. (SSCI)*, 2020, pp. 2663–2669.

[103] A. Kwasigroch, M. Grochowski, and A. Mikolajczyk, "Neural architecture search for skin lesion classification," *IEEE Access*, vol. 8, pp. 9061–9071, 2020.

[104] X. Chu, B. Zhang, and R. Xu, "Multi-objective reinforced evolution in mobile neural architecture search," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2020, pp. 99–113.

[105] N. Awad, N. Mallik, and F. Hutter, "Differential evolution for neural architecture search," in *Proc. 1st Workshop Neural Architect. Search Int. Conf. Learn. Represent. (ICLR)*, 2020.

[106] D. Sapra and A. D. Pimentel, "An evolutionary optimization algorithm for gradually saturating objective functions," in *Proc. Genet. Evol. Comput. Conf.*, 2020, pp. 886–893.

[107] M. Loni, A. Zoljodi, S. Sinaei, M. Daneshtalab, and M. Sjodin, "NeuroPower: Designing energy efficient convolutional neural network architecture for embedded systems," in *Proc. Int. Conf. Artif. Neural Netw. (ICANN)*, 2019, pp. 208–222.

[108] M. Loni, S. Sinaei, A. Zoljodi, M. Daneshtalab, and M. Sjodin, "DeepMaker: A multi-objective optimization framework for deep neural networks in embedded systems," *Microprocess. Microsyst.*, vol. 73, Mar. 2020, Art. no. 102989.

[109] Z. Lu, K. Deb, E. Goodman, W. Banzhaf, and V. N. Boddeti, "NSGANetV2: Evolutionary multi-objective surrogate-assisted neural architecture search," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2019, pp. 35–51.

[110] J. Hajewski, S. Oliveira, and X. Xing, "Evolving deep autoencoders," in *Proc. Genet. Evol. Comput. Conf.*, 2020, pp. 123–124.

[111] X. Qu, J. Wang, and J. Xiao, "Evolutionary algorithm enhanced neural architecture search for text-independent speaker verification," in *Proc. INTERSPEECH*, 2020, pp. 961–965.

[112] Y. Chen, T. Yang, X. Zhang, G. Meng, X. Xiao, and J. Sun, "DetNAS: Backbone search for object detection," in *Proc. 32nd Adv. Neural Inf. Process. Syst.*, 2019, pp. 6642–6652. [Online]. Available: http://papers.nips.cc/paper/8890-detnas-backbone-search-for-object-detection.pdf

[113] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2017, pp. 1800–1807.

[114] X. Chen, Y. Sun, M. Zhang, and D. Peng, "Evolving deep convolutional variational autoencoders for image classification," *IEEE Trans. Evol. Comput.*, early access, Dec. 24, 2020, doi: 10.1109/TEVC.2020.3047220.

[115] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical representations for efficient architectures search," in *Proc. Int. Conf. Learn. Represent.*, 2018, pp. 1–6.

[116] B. Wang, B. Xue, and M. Zhang, "Particle swarm optimisation for evolving deep neural networks for image classification by evolving and stacking transferable blocks," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, 2020, pp. 1–8.

[117] H. Zhu, Z. An, C. Yang, K. Xu, E. Zhao, and Y. Xu, "EENA: Efficient evolution of neural architecture," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV) Workshops*, 2019, pp. 1891–1899. [Online]. Available: http://openaccess.thecvf.com/content_ICCVW_2019/html/NeurArch/Zhu_EENA_Efficient_Evolution_of_Neural_Architecture_ICCVW_2019_paper.html

[118] A. Rajagopal *et al.*, "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.

[119] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Automatically designing CNN architectures using genetic algorithm for image classification," *IEEE Trans. Cybern.*, vol. 50, no. 9, pp. 3840–3854, Sep. 2020.

[120] M. P. P. Faria, R. M. S. Julia, and L. B. P. Tomaz, "Improving fifa player agents decision-making architectures based on convolutional neural networks through evolutionary techniques," in *Proc. Braz. Conf. Intell. Syst. (BRACIS)*, 2019, pp. 371–386.

[121] 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.

[122] 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.

[123] D. Zhou *et al.*, "EcoNAS: Finding proxies for economical neural architecture search," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2020, pp. 11393–11401.

[124] Z. Lu *et al.*, "Multi-objective evolutionary design of deep convolutional neural networks for image classification," *IEEE Trans. Evol. Comput.*, vol. 25, no. 2, pp. 277–291, Apr. 2021.

[125] H. Tan, C. He, D. Tang, and R. Cheng, "Efficient evolutionary neural architecture search (NAS) by modular inheritable crossover," in *Proc. Int. Conf. Bio Inspired Comput. Theories Appl.*, 2020, pp. 761–769.

[126] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, 2019, pp. 4780–4789.

[127] T. Hassanzadeh, D. Essam, and R. Sarker, "Evolutionary attention network for medical image segmentation," in *Proc. Int. Conf. Digit. Image Comput. Techn. Appl. (DICTA)*, 2020, pp. 1–8.

[128] L. Yao, H. Xu, W. Zhang, X. Liang, and Z. Li, "SM-NAS: Structural-to-modular neural architecture search for object detection," in *Proc. AAAI Conf. Artif. Intell.*, 2020, pp. 12661–12668.

[129] H. Xu, S. Wang, X. Cai, W. Zhang, X. Liang, and X. Li, "CurveLane-NAS: Unifying lane-sensitive architecture search and adaptive point blending," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2020, pp. 689–704.

[130] R. D. Gottapu and C. H. Dagli, "Efficient architecture search for deep neural networks," *Procedia Comput. Sci.*, vol. 168, pp. 19–25, 2020.

[131] D. Song, C. Xu, X. Jia, Y. C. C. Xu, and Y. Wang, "Efficient residual dense block search for image super-resolution," in *Proc. AAAI Conf. Artif. Intell.*, vol. 34, 2020, pp. 12007–12014.

[132] T. Desell, "Large scale evolution of convolutional neural networks using volunteer computing," in *Proc. Genet. Evol. Comput. Conf.*, 2017, pp. 127–128.

[133] H. Zhang, S. Kiranyaz, and M. Gabbouj. (2018). *Finding Better Topologies for Deep Convolutional Neural Networks by Evolution*. [Online]. Available: https://arxiv.org/abs/1809.03242

[134] D. K. Barnes, S. R. Davis, E. M. Hand, and S. J. Louis, "A first step toward incremental evolution of convolutional neural networks," in *Proc. Genet. Evol. Comput. Conf.*, 2020, pp. 115–116.

[135] E. Real *et al.*, "Large-scale evolution of image classifiers," in *Proc. 34th Int. Conf. Mach. Learn.*, vol. 70, 2017, pp. 2902–2911.

[136] X. Zheng *et al.*, "Rethinking performance estimation in neural architecture search," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2020, pp. 11353–11362.

[137] B. Su, R. Li, and H. Zhang, "Evolving deep convolutional neural network for intrusion detection based on neat," in *Proc. IEEE 23rd Int. Symp. Wireless Pers. Multimedia Commun. (WPMC)*, 2020, pp. 1–6.

[138] M. Witsuba, "Deep learning architecture search by neuro-cell-based evolution with function-preserving mutations," in *Proc. Joint Eur. Conf. Mach. Learn. Knowl. Disc. Databases*, 2018, pp. 243–258.

[139] K. Maziarz, A. Khorlin, Q. de Laroussilhe, and A. Gesmundo, "Evolutionary-neural hybrid agents for architecture search," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2019, pp. 1–9.

[140] C. Saltori, S. Roy, N. Sebe, and G. Iacca, "Regularized evolutionary algorithm for dynamic neural topology search," in *Proc. Int. Conf. Image Anal. Process.*, 2019, pp. 219–230.

[141] E. Byla and W. Pang, "DeepSwarm: Optimising convolutional neural networks using swarm intelligence," in *Proc. U.K. Workshop Comput. Intell.*, 2019, pp. 119–130.

[142] Q. Yu *et al.*, "C2FNAS: Coarse-to-fine neural architecture search for 3D medical image segmentation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2020, pp. 4125–4134.

[143] R. Miikkulainen *et al.*, "Evolving deep neural networks," in *Artificial Intelligence in the Age of Neural Networks and Brain Computing*, J. Fagerberg, D. C. Mowery, and R. R. Nelson, Eds. London, U.K.: Academic, 2019, ch. 15, pp. 293–312.

[144] J. Liang, E. Meyerson, B. Hodjat, D. Fink, K. Mutch, and R. Miikkulainen, "Evolutionary neural automl for deep learning," in *Proc. Genet. Evol. Comput. Conf.*, 2019, pp. 401–409.

[145] C. Fernando *et al.*, "Convolution by evolution: Differentiable pattern producing networks," in *Proc. Genet. Evol. Comput. Conf.*, 2016, pp. 109–116.

[146] 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, 2009.

[147] P. Verbancsics and J. Harguess. (2013). *Generative Neuroevolution for Deep Learning*. [Online]. Available: https://arxiv.org/abs/1312.5355

[148] M. Suganuma, S. Shinichi, and T. Nagao, "A genetic programming approach to designing convolutional neural network architectures," in *Proc. Genet. Evol. Comput. Conf.*, 2017, pp. 497–504.

[149] M. Suganuma, M. Kobayashi, S. Shirakawa, and T. Nagao, "Evolution of deep convolutional neural networks using cartesian genetic programming," *Evol. Comput.*, vol. 28, no. 1, pp. 141–163, 2020.

[150] M. Kobayashi, S. Arai, and T. Nagao, "Evolutionary generative contribution mappings," in *Proc. IEEE Int. Conf. Syst. Man Cybern. (SMC)*, 2020, pp. 1657–1664.

[151] J. D. Homburg, M. Adams, M. Thies, T. Korthals, M. Hesse, and U. Rückert, "Constraint exploration of convolutional network architectures with neuroevolution," in *Proc. Int. Work Conf. Artif. Neural Netw.*, 2019, pp. 735–746.

[152] W. Irwin-Harris, Y. Sun, B. Xue, and M. Zhang, "A graph-based encoding for evolutionary convolutional neural network architecture design," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, 2019, pp. 546–553.

[153] A. McGhie, B. Xue, and M. Zhang, "GPCNN: Evolving convolutional neural networks using genetic programming," in *Proc. IEEE Symp. Comput. Intell. (SSCI)*, 2020, pp. 2684–2691.

[154] Y. Bi, B. Xue, and M. Zhang, "An evolutionary deep learning approach using genetic programming with convolutional operators for image classification," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, 2019, pp. 3197–3204.

[155] B. P. Evans, H. Al-Sahaf, B. Xue, and M. Zhang, "Evolutionary deep learning: A genetic programming approach to image classification," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, 2018, pp. 1–6.

[156] Z. Lu *et al.*, "NSGA-NET: Neural architecture search using multi-objective genetic algorithm," in *Proc. Genet. Evol. Comput. Conf.*, 2019, pp. 419–427.

[157] H. Zhang, Y. Jin, R. Cheng, and K. Hao, "Efficient evolutionary search of attention convolutional networks via sampled training and node inheritance," *IEEE Trans. Evol. Comput.*, vol. 25, no. 2, pp. 371–385, Apr. 2021.

[158] C. Broni-Bediako, Y. Murata, L. H. B. Mormille, and M. Atsumi, "Evolutionary NAS with gene expression programming of cellular encoding," in *Proc. IEEE Symp. Comput. Intell. (SSCI)*, 2020, pp. 2670–2676.

[159] G. J. V. Wyk and A. S. Bosman, "Evolutionary neural architecture search for image restoration," in *Proc. IEEE Int. Joint Conf. Neural Netw. (IJCNN)*, 2019, pp. 1–8.

[160] D. Kang and C. W. Ahn, "Efficient neural network space with genetic search," in *Proc. Int. Conf. Bio Inspired Comput. Theories Appl.*, 2020, pp. 638–646.

[161] P. Yotchon and Y. Jewajinda, "Hybrid multi-population evolution based on genetic algorithm and regularized evolution for neural architecture search," in *Proc. 17th Int. Joint Conf. Comput. Sci. Softw. Eng. (JCSSE)*, 2020, pp. 1451–1452.

[162] M. Hu, W. Wang, L. Liu, and Y. Liu, "Apenas: An asynchronous parallel evolution based multi-objective neural architecture search," in *Proc. IEEE Int. Conf. Parallel Distrib. Process. Appl. Big Data Cloud Comput. Sustain. Comput. Commun. Soc. Comput. Netw.*, 2020, pp. 153–159.
[163] 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. Pac. Rim Int. Conf. Artif. Intell.*, 2019, pp. 650–663.
[164] Z. Yang *et al.*, "CARs: Continuous evolution for efficient neural architecture search," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2020, pp. 1826–1835.
[165] P. Liu, M. D. E. Basha, Y. Li, Y. Xiao, P. C. Sanelli, and R. Fang, "Deep evolutionary networks with expedited genetic algorithms for medical image denoising," *Med. Image Anal.*, vol. 54, pp. 306–315, May 2019.
[166] W. Zhu, W. Yeh, J. Chen, D. Chen, A. Li, and Y. Lin, "Evolutionary convolutional neural networks using ABC," in *Proc. 11th Int. Conf. Mach. Learn. Comput.*, 2019, pp. 156–162.
[167] B. K. Oh and K. Kim, "Multi-objective optimization method to search for the optimal convolutional neural network architecture for long-term structural health monitoring," *IEEE Access*, vol. 9, pp. 44738–44750, 2021.
[168] S. S. Mostafa, F. Mendonça, A. G. Ravelo-Garcia, G. Juliá-Serdá, and F. Morgado-Dias, "Multi-objective hyperparameter optimization of convolutional neural network for obstructive sleep APNEA detection," *IEEE Access*, vol. 8, pp. 129586–129599, 2020.
[169] M. Baldeon-Calisto and S. K. Lai-Yuen, "AdaResU-Net: Multiobjective adaptive convolutional neural network for medical image segmentation," *Neuroecomputing*, vol. 392, pp. 325–340, Jun. 2020.
[170] H. Liu, K. Simonyan, and Y. Yang, "DARTs: Differentiable architecture search," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2019, pp. 1–9.
[171] X. Chen, L. Xie, J. Wu, and Q. Tian, "Progressive differentiable architecture search: Bridging the depth gap between search and evaluation," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, 2019, pp. 1294–1303.
[172] M. Kobayashi and T. Nagao, "An evolution-based approach for efficient differentiable architecture search," in *Proc. Genet. Evol. Comput. Conf. Companion*, 2020, pp. 131–132.
[173] M. Javaheripi, M. Samragh, T. Javidi, and F. Koushanfar, "GeneCAI: Genetic evolution for acquiring compact AI," in *Proc. Genet. Evol. Comput. Conf.*, 2020, pp. 350–358.
[174] T. Elsken, J. H. Metzen, and F. Hutter, "Efficient multi-objective neural architecture search via lamarckian evolution," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2019, pp. 1–9.
[175] C. Schorn, T. Elsken, S. Vogel, A. Runge, A. Guntoro, and G. Ascheid, "Automated design of error-resilient and hardware-efficient deep neural networks," *Neural Comput. Appl.*, vol. 32, pp. 18327–18345, Dec. 2020.
[176] A. M. Zador, "A critique of pure learning and what artificial neural networks can learn from animal brains," *Nat. Commun.*, vol. 10, p. 3770, Nov. 2019.

Gustavo-Adolfo Vargas-Hákim received the B.Sc. degree in mechatronics engineering from the Universidad Popular Autónoma del Estado de Puebla, Puebla, Mexico, in 2019. He is currently pursuing the master's degree in artificial intelligence with the University of Veracruz, Xalapa, Mexico.

During his engineering studies, he completed in two international robotics contests, obtaining the third and second places. He had the opportunity to finish his undergraduate studies as an exchange student with Oklahoma State University, OK, USA. His research interests span artificial neural networks and evolutionary computation, specifically in the domain of computer vision.

Efrén Mezura-Montes (Senior Member, IEEE) was born in Xalapa, Mexico, in 1973. He received the Diploma degree in computer systems engineering from the University of the Americas Puebla, Cholula, Mexico, in 1997, the M.Sc. degree in artificial intelligence from the University of Veracruz, Xalapa, in 2001, and the Ph.D. degree in computer science from the Center for Research and Advanced Studies, National Polytechnic Institute, Mexico City, Mexico, in 2004.

He has published over 160 papers in peer-reviewed journals and conferences. His current research interests include design, study, and application of nature-inspired metaheuristic algorithms to solve complex optimization problems.

Dr. Mezura-Montes is a member of the IEEE Computational Intelligence Society Evolutionary Computation Technical Committee and the IEEE Systems, Man and Cybernetics Society Soft Computing Technical Committee. He is also a member of the Mexican National Researchers System Level 3, a Regular Member of the Mexican Academy of Sciences, and the Board of Directors Member of the Mexican Computing Academy and Technical Committee Member of the Mexican Network on Applied Computational Intelligence.

Héctor-Gabriel Acosta-Mesa received the B.Sc. degree in computer systems engineering from the Veracruz Institute of Technology, Veracruz, Mexico, in 1991, the M.Sc. degree in artificial intelligence from the University of Veracruz, Xalapa, Mexico, in 1997, and the Ph.D. degree in artificial intelligence from the Artificial Intelligence Vision Research Unit, Department of Psychology, University of Sheffield, Sheffield, U.K., in 2003.

He is a Researcher with the Artificial Intelligence Research Institute, University of Veracruz. He is a Researcher of the National Council of Science and Technology of Mexico (level 1). His interests are the applications of machine learning techniques and computer vision, particularly in the area of medical imaging.