

# A Survey on Evolutionary Neural Architecture Search

Yuqiao Liu<sup>ID</sup>, *Graduate Student Member, IEEE*, Yanan Sun<sup>ID</sup>, *Member, IEEE*, Bing Xue<sup>ID</sup>, *Senior Member, IEEE*, Mengjie Zhang<sup>ID</sup>, *Fellow, IEEE*, Gary G. Yen<sup>ID</sup>, *Fellow, IEEE*, and Kay Chen Tan<sup>ID</sup>, *Fellow, IEEE*

**Abstract**—Deep neural networks (DNNs) have achieved great success in many applications. The architectures of DNNs play a crucial role in their performance, which is usually manually designed with rich expertise. However, such a design process is labor-intensive because of the trial-and-error process and also not easy to realize due to the rare expertise in practice. Neural architecture search (NAS) is a type of technology that can design the architectures automatically. Among different methods to realize NAS, the evolutionary computation (EC) methods have recently gained much attention and success. Unfortunately, there has not yet been a comprehensive summary of the EC-based NAS algorithms. This article reviews over 200 articles of most recent EC-based NAS methods in light of the core components, to systematically discuss their design principles and justifications on the design. Furthermore, current challenges and issues are also discussed to identify future research in this emerging field.

**Index Terms**—Deep learning, evolutionary computation (EC), evolutionary neural architecture search (NAS), image classification.

## I. INTRODUCTION

DEEP neural networks (DNNs), as the cornerstone of deep learning [1], have demonstrated their great success in diverse real-world applications, including image classification [2], [3], natural language processing [4], speech recognition [5], to name a few. The promising performance of DNNs has been widely documented due to their deep architectures [1], which can learn meaningful features directly from the raw data almost without any explicit feature engineering.

Manuscript received January 5, 2021; revised May 29, 2021; accepted July 24, 2021. This work was supported in part by the National Natural Science Foundation of China under Grant 61803277, in part by the Science and Technology Innovation Talent Project of Sichuan Province under Grant 2021JDR0001, and in part by the Research Grants Council of the Hong Kong SAR under Grant PolyU11202418 and Grant PolyU11209219. (Corresponding author: Yanan Sun.)

Yuqiao Liu is with the College of Computer Science, Sichuan University, Chengdu 610065, China (e-mail: lyqguitar@gmail.com).

Yanan Sun is with the College of Computer Science, Sichuan University, Chengdu 610065, China, and also with the State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China (e-mail: ysun@scu.edu.cn).

Bing Xue and Mengjie Zhang are with the School of Engineering and Computer Science, Victoria University of Wellington, Wellington 6012, New Zealand (e-mail: bing.xue@ecs.vuw.ac.nz; mengjie.zhang@ecs.vuw.ac.nz).

Gary G. Yen is with the School of Electrical and Computer Engineering, Oklahoma State University, Stillwater, OK 74078 USA (e-mail: gyen@okstate.edu).

Kay Chen Tan is with the Department of Computing, Hong Kong Polytechnic University, Hong Kong (e-mail: kaychen.tan@polyu.edu.hk).

Color versions of one or more figures in this article are available at <https://doi.org/10.1109/TNNLS.2021.3100554>.

Digital Object Identifier 10.1109/TNNLS.2021.3100554

Generally, the performance of DNNs depends on two aspects: their architectures and the associated weights. Only when both achieve the optimal status simultaneously, the performance of the DNNs could be expected promising. The optimal weights are often obtained through a learning process: using a continuous loss function to measure the discrepancies between the real output and the desired one, and then the gradient-based algorithms are often used to minimize the loss. When the termination condition is satisfied, which is commonly a maximal iteration number, the algorithm can often find a good set of weights [1]. This kind of process has been very popular owing to its effectiveness in practice and has become the dominant practice for weight optimization [6], although they are principally local search [7] algorithms. On the other hand, obtaining the optimal architectures cannot be directly formulated by a continuous function, and there is even no explicit function to measure the process for finding optimal architectures.

To this end, there has been a long time that the promising architectures of DNNs are manually designed with rich expertise. This can be evidenced from the state of the arts, such as VGG [8], ResNet [2], and DenseNet [3]. These promising convolutional neural network (CNN) models are all manually designed by the researchers with rich knowledge in both neural networks and image processing. However, in practice, most end users are not with such kinds of knowledge. Moreover, DNN architectures are often problem-dependent. If the distribution of the data is changed, the architectures must be redesigned accordingly. Neural architecture search (NAS), which aims to automate the architecture designs of DNNs, is identified as a promising way to address the challenge aforementioned.

Mathematically, NAS can be modeled by an optimization problem formulated by

$$\begin{cases} \arg \min_A = \mathcal{L}(A, \mathcal{D}_{\text{train}}, \mathcal{D}_{\text{fitness}}) \\ \text{s.t. } A \in \mathcal{A} \end{cases} \quad (1)$$

where  $\mathcal{A}$  denotes the search space of the potential neural architectures, and  $\mathcal{L}(\cdot)$  measures the performance of the architecture  $A$  on the fitness evaluation data set  $\mathcal{D}_{\text{fitness}}$  after being trained on the training data set  $\mathcal{D}_{\text{train}}$ .  $\mathcal{L}(\cdot)$  is usually non-convex and non-differentiable [9]. In principle, NAS is a complex optimization problem experiencing several challenges, e.g., complex constraints, discrete representations, bi-level structures, computationally expensive characteristics,

and multiple conflicting criteria. The NAS algorithms refer to the optimization algorithms which are specifically designed to effectively and efficiently solve the problem represented by (1). The initial work of the NAS algorithms is commonly viewed as the work in [10], which was proposed by Google. The preprint version of this work was first released in the website of arXiv<sup>1</sup> in 2016, and then was formally accepted for publication by the International Conference on Learning Representations (ICLR) in 2017. Since then, a large number of researchers have been investing tremendous efforts in developing novel NAS algorithms.

Based on the optimizer used, the existing NAS algorithms can be broadly classified into three different categories: reinforcement learning (RL) [11]-based NAS algorithms, gradient-based NAS algorithms, and evolutionary computation (EC) [12]-based NAS (ENAS) algorithms. Specifically, the RL-based algorithms often require thousands of graphics processing cards (GPUs) performing several days even on median-scale data set, such as the CIFAR-10 image classification benchmark data set [13]. The gradient-based algorithms are more efficient than the RL-based algorithms. However, they often find the ill-conditioned architectures due to the improper relationship for adapting to gradient-based optimization. Unfortunately, the relationship has not been mathematically proven. In addition, the gradient-based algorithms require to construct a supernet in advance, which also highly requires expertise. The ENAS algorithms solve the NAS by exploiting EC techniques. Specifically, EC is a class of population-based computational paradigms, simulating the evolution of species or the behaviors of the population in nature, to solve challenging optimization problems. In particular, genetic algorithms (GAs) [14], genetic programming (GP) [15], and particle swarm optimization (PSO) [16] are widely used EC methods in practice. Owing to the promising characteristics of the EC methods in insensitiveness to the local minima and no requirement to gradient information, EC has been widely applied to solve complex non-convex optimization problems [17], even when the mathematical form of the objective function is not available [18].

In fact, the EC methods had been frequently used more than 20 years ago, searching for not only the optimal neural architectures but also the weights of neural networks simultaneously, which is also termed as neuroevolution [19]. The major differences between ENAS and neuroevolution lie in two aspects. First, neuroevolution often uses EC to search for both the neural architectures and the optimal weight values, while ENAS for now focuses mainly on searching for the architectures and the optimal weight values are obtained using the gradient-based algorithms<sup>2</sup> immediately after. Second, neuroevolution commonly applies to small-scale and

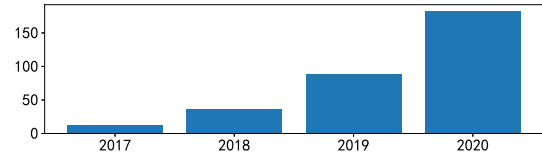


Fig. 1. Number of submissions refers to the works of evolutionary NAS. The data are collected from Google Scholar with the keywords of “evolutionary” OR “genetic algorithm” OR “particle swarm optimization” OR “PSO” OR “genetic programming” AND “architecture search” OR “architecture design” OR “CNN” OR “deep learning” OR “deep neural network” and the literature on NAS collected from the AutoML.org website by the end of 2020. With these initially collected data, we then have carefully checked each manuscript to make its scope accurate within the evolutionary NAS.

median-scale neuron networks, while ENAS generally works on DNNs, such as the deep CNNs [23], [24] and deep stacked autoencoders [25], which are stacked by the building blocks of deep learning techniques [1]. Generally, the first work of ENAS is often viewed as the LargeEvo algorithm [23] which was proposed by Google which released its early version in March 2017 in arXiv. Afterward, this article got accepted into the 34th International Conference on Machine Learning in June 2017. The LargeEvo algorithm used a GA to search for the best architecture of a CNN, and the experimental results on CIFAR-10 and CIFAR-100 [13] have demonstrated its effectiveness. Since then, a large number of ENAS algorithms have been proposed. Fig. 1 shows the number of similar works<sup>3</sup> published from 2017 to 2020 when this survey article was ready for submission. As can be seen from Fig. 1, from 2017 to 2020, the number of submissions grows with multiple scales.

A large number of related submissions have been made available publicly, but there is no comprehensive survey of the literature on the ENAS algorithms. Although recent reviews on NAS have been made in [18] and [26]–[28], they mainly focus on reviewing different methods to realize NAS, rather than concentrating on the ENAS algorithms. Specifically, Elsken *et al.* [26] divided NAS into three stages: search space, search strategy, and performance estimation strategy. Similarly, Wistuba *et al.* [27] followed these three stages with an additional review about the multiple objectives in NAS. Darwish *et al.* [18] made a summary of swarm intelligence (SI) and evolutionary algorithms (EAs) approaches for deep learning, with focuses on both NAS and other hyperparameter optimization. Stanley *et al.* [28] went through a review of neuroevolution, aiming at revealing the weight optimization rather than the architecture of neural networks. Besides, most of the references in these surveys are pre-2019 and do not include an update on the papers published during the past two years when most ENAS works were published. This article presents a survey involving a large number of ENAS papers, with the expectation to inspire some new ideas for enhancing the development of ENAS. To allow readers easily concentrating on the technical part of this survey, we also follow the three stages to introduce ENAS algorithms, which have been widely adopted by the existing NAS survey papers [18], [26], [27],

<sup>1</sup><https://arxiv.org/abs/1611.01578>

<sup>2</sup>Please note that we still categorize some existing algorithms as the ENAS algorithm, such as API [20], EvoCNN [21], and EvoDeep [22], although they also concern the weights. This is because the optimal weight values of the DNNs searched by them are still obtained by the gradient-based algorithms. They only searched for the best weight initialization values or the best weight initialization method of the DNNs.

<sup>3</sup>These “submissions” include the ones which have been accepted for publication after the peer-review process and also the ones which are only available on the arXiv website without the peer-review process.

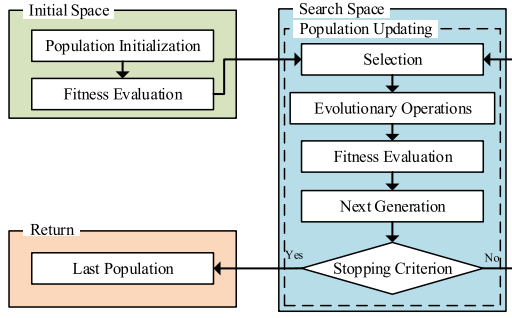


Fig. 2. Flowchart of a common ENAS algorithm.

but with essential modifications made to specifically suit ENAS algorithms.

The remainder of this article is organized as follows. The background of ENAS is discussed in Section II. Section III documents different encoding spaces, initial spaces, and search spaces of the ENAS algorithms. In Section IV, the encoding strategy and architecture representation are introduced. Section V summarizes the process of population updating, including evolutionary operators and selection strategies. Section VI shows multiple ways to speed up the evolution. Section VII presents the applications of ENAS algorithms. Section VIII discusses the challenges and prospects, and finally Section IX presents conclusions.

## II. BACKGROUND

As discussed above, ENAS is a process of searching for the optimal architectures of DNNs using EC methods. In this section, we will first introduce the unified flowchart of ENAS in Section II-A. Then, the categories of the EC methods based on their search strategies and the categories of the DNN architectures will be discussed in Sections II-B and II-C, respectively.

### A. Flowchart of ENAS

Fig. 2 shows an illustration of the flowchart of an ENAS algorithm. Specifically, the evolutionary process takes place in the initial space and the search space sequentially. First, a population is initialized within the initial space that is defined in advance. Each individual in the population represents a solution for the ENAS, i.e., a DNN architecture. Each architecture needs to be encoded as an individual before it joins the population. Second, the fitness of the generated individuals is evaluated. Note that there are two fitness evaluation steps as shown in Fig. 2, which commonly use the same evaluation criterion. Third, after the fitness evaluation of the initial population, the whole population starts the evolutionary process within the search space, which is shown by the dashed box in Fig. 2. In the evolutionary process, the population is updated by the selection and the evolutionary operators in each iteration, until the stopping criterion is met. Note that the selection stage is not necessary for some other EC paradigms like SI. Finally, a population that has finished the evolution is obtained. In Sections III–VI, these key steps are documented in detail.

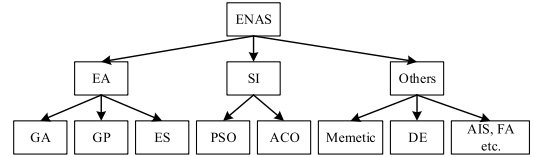


Fig. 3. Categories of ENAS from the EC methods regarding the search strategies.

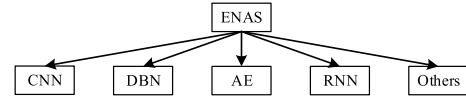


Fig. 4. Categories of ENAS from neural network perspective.

### B. Evolutionary Search Strategy

ENAS is distinguished from other NAS methods by its used optimization approach, i.e., the EC methods, where the optimization approaches can be further subdivided based on the search strategy that the optimization approach adopted. Fig. 3 provides such an illustration from three aspects: EAs, SI, and others. In practice, the EA-based methods account for the majority of the existing ENAS algorithms, where the GA takes a large part of the EA-based methods. The other categories of the EC methods are also important parts for realizing ENAS algorithms, such as GP, evolutionary strategy (ES), PSO, ant colony optimization (ACO) [29], differential evolution (DE) [30], and firefly algorithm (FA) [31]. In this article, the hill-climbing algorithm (HCA) is classified into EC paradigm because it can be regarded as an EA with a simple selection mechanism and without crossover operation [32]. HCA has been well-known as a widely used local search algorithm in memetic algorithms [33].

### C. Common Neural Networks in ENAS

The end product of ENAS is all about DNNs, which can be broadly divided into five different categories: CNN, deep belief network (DBN), stacked auto-encoder (SAE), recurrent neural network (RNN), and others. The brief fact can be seen from Fig. 4, and a detailed illustration is shown in Table III.

Most ENAS methods are proposed for searching the optimal CNN architectures. This is because many hand-crafted CNNs, such as VGG [8], ResNet [2], and DenseNet [3], have demonstrated their superiority in handling the image classification tasks which is the most successful applications in the deep learning field. Generally, the CNN architecture is composed of the convolutional layers, the pooling layers, and the fully connected layers, and a common example of CNNs is shown in Fig. 5(a). The optimization of the CNN architecture is mainly composed of three aspects: the hyperparameters of each layer [34], the depth of the architecture [21], and the connections between layers [35]. In practice, the majority of the ENAS methods consider the above three aspects collectively [23], [36].

DBN [37] is made up by stacking multiple restricted Boltzmann machines (RBMs), and an example can be seen in Fig. 5(b). Specifically, RBMs have connections only between layers, while without any interlayer connection.



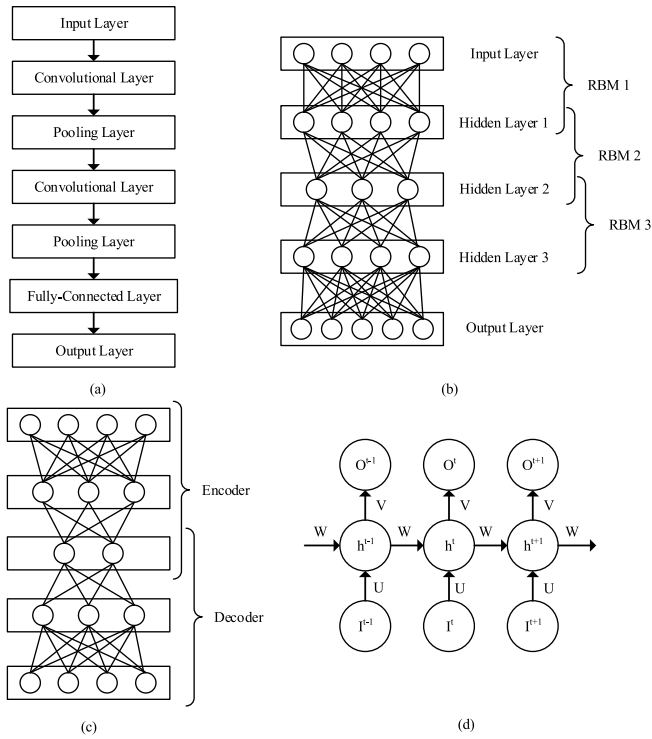


Fig. 5. Examples of different neural architectures in ENAS. (a) CNN. (b) DBN. (c) AE. (d) RNN.

Meanwhile, RBMs allow the DBN to be trained in an unsupervised manner to obtain a good weight initialization. There are two commonly used types of hyperparameters needed to be optimized in DBN: the number of neurons in each layer [38] and the number of layers [39], [40].

An SAE is constructed by stacking multiple of its building blocks which are AEs. An AE aims to learn meaningful representations from the raw input data by restoring its input data as its objective [41]. Generally, an AE is typically composed of two symmetrical components: the encoder and the decoder, and an example including both parts is shown in Fig. 5(c). Generally, some ENAS methods only take the encoder into evolution [25], [42] because the decoder part is symmetric with the encoder and can be derived accordingly. Yet, some other ENAS methods optimize the hyperparameters of encoders and decoders separately [43], [44].

The most significant difference between RNNs and the neural networks introduced above is its recurrent connection. Fig. 5(d) shows the time-expanded structure of an RNN, where the value of the current hidden layer  $h^t$  is influenced by the value at its previous time slot  $h^{t-1}$  and the output value of its previous layer. Because these layers are reused, all the weights (i.e.,  $U$ ,  $W$ , and  $V$  in the figure) are shared. Different from focusing on the number of neurons and the number of layers in the feedforward neural network, some ENAS methods concern about how many times the RNN should be unfold [45], [46]. In addition, there remain other neural networks like typical DNNs, which are made up of only fully connected layers, where the connections are formed by all the neurons in the two adjacent layers. Because such a kind of DNNs is not the major target investigated by NAS, we will not introduce it in detail.

TABLE I  
COMMON PARAMETERS OPTIMIZED IN DIFFERENT DNNs

	Parameters	
	global parameters	number of layers, connections between layers
CNN	convolution layer	filter size (width and height), stride size (width and height), feature map size, convolution type, standard deviation and mean value of the filter elements
	pooling layer	filter size (width and height), stride size (width and height), pooling type
	fully-connected layer	number of neurons, standard deviation and mean value of weights
DBN, AE	number of hidden layers, neurons per layer	
RNN	number of hidden layers, neurons per layer, number of time slot	

When the ENAS algorithms are applied to these DNNs, the goal is to find the best architecture-related parameters. Specifically, for CNNs, they are the number of convolutional layers, pooling layers, fully connected layers, and the parameters related to these layers (such as the kernel size of the convolutional layers, the pooling type, and the number of neurons for fully connected layers) as well as their connection situation (such as the dense connection and the skip connection). For DBN and SAE, they are the number of their building blocks, i.e., the RBM for DBN and the AE for SAE, and the number of neurons in each layer. For RNN, in addition to the architecture-related parameters mentioned above, the number of the time slot is also an important parameter to be optimized by ENAS algorithms. For the traditional DNNs, the ENAS algorithms often concern about the neuron number of each layer. In addition, some ENAS algorithms also concern the weights, such as the weight initialization method and weight initial values. Table I summarizes the detail of the common parameters optimized by the ENAS methods in different types of neural networks, including the CNNs as the most popular type. The details of these ENAS algorithms will be documented in the following sections.

### III. ENCODING SPACE

The encoding space contains all the valid individuals encoded in the population. In terms of the basic units, the encoding space can be divided into three categories according to the basic units they adopt. They are the layer-based encoding space, the block-based encoding space, and the cell-based encoding space. In addition, some ENAS methods do not take the configuration of the basic unit into consideration, but care the connections between units. Such a kind of encoding space is often called the topology-based encoding space.

In addition, the constraints on the encoding space are important. This is because they represent the human intervention which restricts the encoding space and lightens the burden of the evolutionary process. A method with a mass of constraints can obtain a promising architecture easily but prevent to design any novel architecture that does not follow the constraints. Furthermore, the different sizes of the search space would greatly affect the efficiency of the evolution. On the other hand, the effectiveness of the ENAS methods cannot be guaranteed if there is no constraint on the search space: one extreme case is that all the individuals in the search space are mediocre. In this

TABLE II  
DIFFERENT ENCODING SPACE AND THE CONSTRAINTS

	Fixed depth	Rich initialization	Partial fixed structure	Relatively few constraints
Layer-based encoding space	[34], [47]–[57]	[34], [58]–[62]	[20], [54], [63]–[73]	[21]–[23], [25], [39], [40], [42], [46], [74]–[106]
Block-based encoding space	[107]–[109]	[110]	[69], [110], [111]	[9], [24], [32], [76], [105], [112]–[122]
Cell-based encoding space			[123]–[125]	[36], [122], [126]–[132]
Topology-based encoding space		[133], [134]		[35], [135]–[140]

case, an excellent individual can be obtained even without selection. Table II shows different kinds of encoding spaces and the constraints of the existing ENAS algorithms, where the first row shows the constraints and the first column displays the encoding space. In the following, we will discuss them at length.

#### A. Encoding Space and Constraints

The layer-based encoding space denotes that the basic units in the encoding space are the primitive layers, such as convolution layers and fully connected layers. This would lead to a huge search space, since it tries to encode so much information in the search space. However, it may take more time to search for a promising individual because there are more possibilities to construct a well-performed DNN from the primitive layers. For example, a promising CNN is commonly composed of hundreds of primitive layers; accordingly, the search process will consume more time to search for such a deep architecture by iteratively stacking the primitive layers. In addition, the promising DNN may not be found if only with the primitive layers. For instance, the promising performance of ResNet is widely recognized due to its skip connections which cannot be represented by the primitive layers.

To alleviate the above problems, the block-based encoding space is developed, where various layers of different types are combined as the blocks to serve as the basic unit of the encoding space. The traditional blocks are ResBlock [2], DenseBlock [3], ConvBlock (Conv2d + BatchNormalization + Activation) [32], and InceptionBlock [141]. Specifically, layers in the blocks have a specific topological relationship, such as the residual connection in ResBlock and the dense connections in DenseBlock. These blocks have promising performance and often require fewer parameters to build the architecture. So it is principally easier to find a good architecture in the block-based encoding space compared with the layer-based encoding space. Some ENAS algorithms used these blocks directly, such as [9] and [76], while other methods proposed different blocks for different purposes. For example, Chen *et al.* [108] proposed eight blocks including ResBlock and InceptionBlock encoded in a 3-bit string and used Hamming distance to determine similar blocks. Song *et al.* [114] proposed three residual dense mixed blocks to reduce the amount of computation due to the convolution operation of image super-resolution tasks.

The cell-based encoding space is similar to the block-based one and can be regarded as a special case of the block-based space where all the blocks are the same. The ENAS algorithms using this space build the architectures by stacking repeated motifs. Chu *et al.* [129] divided the cell-based space into two independent parts: the micro part containing the parameters of cells and the macro part defining

the connections between different cells. To be more specific, the cell-based encoding space concentrates more on the micro part. Different from the block-based space, the layers in the cell can be combined more freely, and the macro part is always determined by human expertise [36], [142], [143]. Some widely used encoding spaces are classified in this category. For example, NAS-Bench-101 [142] and NAS-Bench-201 [143] search for different combinations of layers and connections in cells, and then stacked the cells sequentially. In addition, NASNet [144] and DARTS [145] search for two kinds of cells, namely, normal cell and reduction cell, and each stacked cell is connected to the two previous cells. The cell-based space greatly reduces the size of the encoding space. This is because all the basic units in the encoding space are the same, and the number of parameters in terms of constructing the promising DNN is much fewer. However, Frachon *et al.* [76] claimed that there is no theoretical basis that the cell-based space can help obtain a good architecture.

In contrast, the topology-based space does not consider the parameters or the structure of each unit (layer or block), yet they are only concerned about the connections between units. One classical example is the one-shot method which treats all the architectures as different subgraphs of a supergraph [26]. Yang *et al.* [137] proposed CARS to search for the architecture by choosing different connections in the supergraph, and then the architecture was built by the chosen subgraph. CARS can be classified into the topology-based category because it aims at deciding whether or not to keep the connections in the supergraph. But from the perspective of building the architecture, CARS can also be classified into the cell-based category. This is because the subgraph cell was stacked several times to build the architecture. Another typical case is pruning. Wu *et al.* [133] used a shallow VGGNet [8] on CIFAR-10, and the aim was to prune unimportant weight connections from the VGGNet.

As observed from Table II, the constraints on the encoding space mainly focus on three aspects: fixed depth, rich initialization, and partial fixed structure. The fixed depth means all the individuals in the population have the same depth. The fixed depth is a strong constraint and largely reduces the size of the encoding space. Note that the fixed depth is different from the *fixed-length encoding strategy* which will be introduced in Section IV. In genetic CNN [35], for example, the *fixed-length encoding strategy* only limits the maximum depth. The node which is isolated (no connection) is simply ignored. By this way, the individuals can obtain different depths. The second constraint is rich initialization (i.e., the *well-designed space* to be discussed in Section III-B), and it is also a limitation in practice that requires a lot of expertise. In this case, the initialized architectures are manually designed, which goes against the original intention

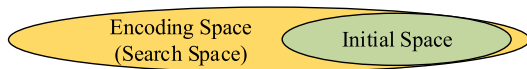


Fig. 6. Relationship between encoding space, search space, and initial space.

of NAS. The partial fixed structure means the architecture is partially settled. For example, in [64], a max-pooling layer is added to the network after every set of four convolution layers.

In Table II, the relatively few constraints category means that those methods have no restrictions like the three aspects discussed above. However, it does not imply there is no constraint. For example, in the classification task, the fully connected layer is often used as the tail of the whole DNNs in some methods [21], [82], [127]. Moreover, the maximum length is predefined in many methods including both *fixed-length encoding strategy* methods [35] and *variable-length encoding strategy* methods [21] resulting in preventing the method from discovering a deeper architecture. Wang *et al.* [75] tried to break the limit of maximum length using a Gaussian distribution initialization mechanism. Irwin-Harris *et al.* [146] broke the limit using the evolutionary operators, the crossover operator, and the mutation operator, to extend the depth to any size.

Generally, the encoding space can be served as the basis of the search space and the initial space. In practice, the encoding space is often the same as the search space, while the initial space is often a subspace of the encoding space. Fig. 6 shows the relationship between the three spaces. The search space is larger than the initial space when some manual constraints are added to the population initialization. When there are no such manual constraints, the search space and initial space are equivalent. Furthermore, the initial space determines what kind of individuals may appear in the initial population, and the search space determines what kind of individuals may appear in the evolutionary search process, as illustrated in Fig. 2. In Sections III-B and III-C, we will discuss the initial space and the search space in Sections III-B and III-C, respectively.

### B. Initial Space

In general, there are three types of architecture initialization approaches in the initial space: starting from trivial initial conditions [23], randomly initialization in the encoding space [21], and starting from a well-designed architecture (also termed as rich initialization) [34]. These three types of initialization correspond to three different initial spaces: *trivial space*, *random space*, and *well-designed space*.

The *trivial space* contains only a few primitive layers. For example, the LargeEvo algorithm [23] initialized the population in a *trivial space* where each individual constitutes just a single-layer model with no convolutions. Xie and Yuille [35] experimentally demonstrated that a *trivial space* can evolve to a competitive architecture. The reason for using as little experience as possible is to justify the advantage of the EC-based methods where some novel architectures can be discovered, and most of the discovery is different from the manually designed DNN architectures. On the contrary, the *well-designed space* contains the state-of-the-art architectures. In this way, a promising architecture

can be obtained at the beginning of the evolution, whereas it can hardly evolve to other novel architectures. Actually, many of the ENAS methods adopting this initial space focus on improving the performance on the well-designed architecture. For example, the architecture pruning aims at compressing DNNs by removing less important connections [134]. For a *random space*, all the individuals in the initial population are randomly generated in the limited space, and it has been adopted by many methods, such as [9], [21], and [67]. The aim of this type of initial spaces is also to reduce the intervention of human experience in the initial population.

### C. Search Space

After the initialization of the population, the ENAS methods start to search the architectures in the search space. Generally speaking, the search space is the same as the initial space when the random initial space is adopted. For other two types of initial spaces, however, due to the relatively small initial space, the search space will become much larger with the expectation that the promising architecture is included. It is worth noting that many methods do not directly define the search space, but restrict the search space using evolutionary operators. For example, Irwin-Harris *et al.* [146] did not specify the maximal depth; instead, they used the evolutionary operations to extend the architecture to any depth.

## IV. ENCODING STRATEGY

This section will discuss how to encode a network architecture into an individual of the EC methods. Each ENAS method needs to determine its encoding strategy before starting the first stage of the ENAS method, i.e., the population initialization. The most intuitive difference in different encoding strategies of the ENAS methods is the length of the encoded individuals.

Generally, the encoding strategies can be divided into two different categories according to whether the length of an individual changes or not during the evolutionary process. They are the *fixed-length encoding strategy* and the *variable-length encoding strategy*. Particularly, the individuals have the same length during the evolutionary process when the *fixed-length encoding strategy* is used. In contrast, the individuals are with different lengths during the evolutionary process if the *variable-length encoding strategy* is used. The advantage of the *fixed-length encoding strategy* is that it is easy to use standard evolutionary operations which are originally designed for individuals with equal lengths. In genetic CNN [35], for example, the fixed-length binary string helped the evolutionary operators (especially the crossover) with easy implementation. Another example is [115], where Loni *et al.* used a fixed-length string of genes to represent the architecture which led to easy implementation of the one-point crossover in the corresponding encoded information. However, a proper maximal length must be predefined for the fixed-length encoding strategy. Because the maximal length relates to the optimal depth of the DNN architecture, which is unknown in advance, the corresponding ENAS algorithms still rely on expertise and experience.



Compared with the *fixed-length encoding strategy*, the *variable-length encoding strategy* does not require the human expertise regarding the optimal depth in advance, which has the potential to be fully automated. In addition, the advantage of this encoding strategy is that it can define more details of the architecture with freedom. For example, when solving a new task where there is no expertise in knowing the optimal depth of the DNN, we just initialize the individuals with a random depth, and the optimal depth can be found through the variable-length encoding strategy, where the depths of the corresponding DNN can be changed during the evolutionary process. However, the variable-length encoding strategy also brings some drawbacks. Because the traditional evolutionary operators might not be suitable for this kind of encoding strategy, the corresponding evolutionary operators need to be redesigned, where an example can be seen in [21]. Another disadvantage is that due to the flexibility of variable-length encoding, it could generally result in over-depth architectures, which sometimes further leads to more time-consuming fitness evaluation. Note that some works claimed their use of the variable-length encoding strategy, where each individual is designed with a maximal length, and then the placeholder is used to indicate the gene's validation of the gene [97]. For example, the individual is designed to have 1000 genes, while some genes having the values of zeros do not participate in the evolutionary process. In this article, we also categorize these methods into the fixed-length encoding strategy.

In addition, most of the DNN architectures can be represented as directed graphs which are made up of different basic units and the connections between the units. Therefore, the encoding for architecture can be divided into two aspects: configurations of basic units and connections, which will be discussed in the following subsections.

#### A. Encoding for Configurations of Basic Units

In practice, different basic units have different configurations, such as layers, blocks, and cells, which are demonstrated in Section III. For example, in CNNs, there are multiple parameters in the primitive layers which can be seen in Table I. As for the DenseBlock implemented by Sun *et al.* [9], only two parameters are needed to build the block. Because the configuration of cell is more flexible than that of blocks in CNNs, which can be regarded as a microcosm of a complete neural network. For example, the cells in [36] are made up by a combination of ten layers selected from eight different primitive layers. But the cells do not include some of the configurations of primitive layer, such as the feature map size which is an important parameter in the primitive layer [21].

#### B. Encoding for Connections

When the parameters (configurations) of the basic units in the architecture have been determined, the corresponding architecture cannot be built up immediately. Since edges are indispensable in directed graph, connections are also part of the architecture. The connections discussed in this section include not only the connections between basic units but also the connections within the basic units.

Generally, the architectures of neural networks can be divided into two categories: *linear architecture* and *non-linear architecture*. The former denotes the architectures containing sequential basic units. The latter indicates that there are skip connections or loop connections in the architecture. Note that the structure could be the macro structure consisting of basic units, or the micro structure within the basic units.

1) *Linear Architecture*: The linear architecture can be found in different kinds of architectures including the one generated from the layer-based encoding space and the block-based encoding space. Its widespread use in ENAS stems from its simplicity. No matter how complex the internal of the basic units is, many ENAS methods stack the basic units one by one to build up the skeleton of the architecture which is linear. For example, in AE-CNN [9], Sun *et al.* stacked different kinds of blocks to generate the architecture.

One special case is a linear architecture generated from the layer-based encoding space. In this case, there is no need to solely encode the connections, and only the parameters in each basic unit are enough to build an architecture. One classical example can be seen in [21] where Sun *et al.* explored a great number of parameters based on a linear CNN architecture. However, most architectures are not designed to be linear. The skip connections in ResNet [2] and the dense connections in DenseNet [3] show the ability to build a good architecture.

2) *Non-Linear Architecture*: First, we will introduce two approaches to encoding a non-linear architecture in this section. Specifically, the adjacent matrix is the most popular way to represent the connections for non-linear architectures. Genetic CNN [35] used a binary string to represent the connection, and the string can be transformed into a triangular matrix. In the binary string, "1" denotes that there is a connection between the two nodes while "0" denotes no connection in between. Lorenzo and Nalepa [135] used a matrix to represent the skip connections, and this work revolved around the adjacent matrix. Back in the 1990s, Kitano *et al.* [147] began to study the use of the adjacent matrix to represent network connection and explained the process from the connectivity matrix, to the bit-string genotype, to the network architecture phenotype. Another way to represent the connections is to use an ordered pair  $G = (V, E)$  with vertices  $V$  and edge  $E$  associated with a direction, to represent a directed acyclic graph. Irwin-Harris *et al.* [146] also used this strategy to encode the connections.

Second, the non-linear architecture is a more common case in both the macro architecture and the micro architecture. AmoebaNet-A [36], as an example of non-linear macro architecture, stacked these two kinds of cells for several times to build up an architecture. In addition, each cell receives two inputs from the previous two cells separately, which means that the direct connection and the skip connection are both used in this macro structure. Also in AmoebaNet, each cell is a non-linear architecture inside, which means the micro architecture is also non-linear. The non-linear architecture provides the architecture with more flexibility, which makes it more likely to build up a promising architecture than that of the linear ones.

TABLE III  
CATEGORIZATION OF EC AND DIFFERENT TYPES OF NEURAL NETWORK

			CNN	DBN	RNN	AE	Others
Single-objective	EA	GAs	[9], [20]–[24], [34]–[36], [48], [50], [53], [57], [60]–[62], [64], [66]–[68], [70]–[72], [75], [79], [82], [83], [86], [87], [89], [91], [93], [94], [96], [98]–[100], [104], [108], [111], [114], [118]–[120], [122], [123], [126]–[128], [132], [139], [148], [149]	[40], [150]	[45], [69], [94], [95], [151]	[43]	[84], [140], [152], [153]
		GP	[63], [112], [113], [116], [154], [155]		[156], [157]	[44]	[158]
		ES	[124]		[45], [46], [58], [159]–[161]	[42]	[140], [162]
	SI	ACO	[106]		[136], [138], [163], [164]		
		PSO	[74], [75], [97], [102], [103], [110], [131], [165]	[38], [39]		[25], [47]	[56], [85], [166]
	Other	Memetic	[135], [167]				
		DE	[49], [75]		[90]	[52]	[168]
		HCA	[32], [59], [80]				
		CVOA			[88]		
		Hyper-heuristic		[78]			
		FA	[77]				
Multi-objective	EA		[65], [73], [81], [101], [105], [107], [115], [115], [121], [129], [134], [137], [169]–[172]	[55], [173]	[174]		[175], [176]
	SI		[92], [109], [133]	[51]			

TABLE IV  
SELECTION STRATEGY

Elitism	[32], [42], [43], [46], [53], [59], [67], [76], [80], [87], [112], [116], [133], [135], [158]
Discard the worst or the oldest	[36], [60], [79], [89], [132], [151], [177]
Roulette	[35], [48], [64], [66], [108], [114], [115], [118], [134]
Tournament selection	[9], [20], [21], [23], [24], [63], [83], [95], [100], [119], [122], [127], [129], [130], [177]
Others	[61], [105]

## V. POPULATION UPDATING

This section discusses the population updating process as shown in Fig. 2. Generally, the population updating varies greatly among the existing ENAS algorithms because they may use different EC methods which are with different updating mechanisms. Table III shows the ENAS algorithms which are classified according to the EC methods that they use and different types of DNNs that they target at. Obviously, the EA-based ENAS algorithms dominate the ENAS. To be more specific, the GA-based ENAS is the most popular approach which largely owes to the convenience of architecture representation in GA. As a result, we give a detailed introduction to the EA-based ENAS including the selection strategy at first. Immediately after, we present a summary of the SI-based ENAS methods and others separately. The corresponding multiobjective ENAS algorithms are also introduced at the end of each section.

### A. EAs for ENAS

The dash box in Fig. 2 shows the general flow of population updating in EA-based ENAS. In this section, we will introduce the selection strategy and the evolutionary operations which collectively update the population. Specifically, the first stage of population updating is selection. The selection strategies can be classified into several types, where Table IV shows the main kinds of selection strategies. Note that the

selection strategy can be not only used in choosing individuals as parents to generate offspring with the evolutionary operators but also used in the environmental selection stage which chooses individuals to make up the next population. Zhang *et al.* [104] termed these two selections as mate selection and environmental selection separately.

The existing selection strategies can be divided into five categories: elitism, discarding the worst, roulette wheel selection, tournament selection, and others. The simplest strategy is elitism which retains the individuals with higher fitness. However, this can cause a loss of diversity in the population, which may lead the population falling into local optima. Discarding the worst is similar to elitism, which removes the individuals with poor fitness values from the population. Real *et al.* [36] used the aging evolution which discards the oldest individual in the population. Aging evolution can explore the search space more, instead of zooming in on good models too early, as non-aging evolution would. The same selection strategy was also used in [40]. Zhu *et al.* [60] combined these two approaches to discarding the worst individual and the oldest individual at the same time. Roulette wheel selection gives every individual a probability according to its fitness among the population to survive (or be discarded), regardless of whether it is the best or not. Tournament selection selects the best one from an equally likely sampling of individuals. Furthermore, Johnner and Wassner [61] used a ranking function to choose individuals by rank. A selection trick termed as niching was used in [71] and [128] to avoid stacking into local optima. This trick allows offspring worse than parent to survive for several generations until evolving to a better one.

Most of the methods focus on preserving the well-performed individuals; however, Liu *et al.* [126] emphasized the genes more than the survived individuals, where a gene can represent any components in the architecture. They believed that the individuals which consist of the fine-gene set are more likely to have promising performance.



Some selection methods aim at preserving the diversity of the population. Elsken *et al.* [105] selected individuals in inverse proportion to their density. Javaheripi *et al.* [178] chose the parents based on the distance (difference) during the mate selection. They chose two individuals with the highest distance to promote the exploration search.

In terms of evolutionary operations, mutation and crossover are two of the most commonly used operations in the EA-based ENAS algorithms. Particularly, mutation is only performed on a single individual, while the crossover takes two individuals to generate offspring.

The mutation operator aims to search the global optimum around the individual. A simple idea is to allow the encoded information to vary from a given range. Sun *et al.* [21] used polynomial mutation [179] on the parameters of layers which are expressed by real numbers. To make mutation not random, Lorenzo and Nalepa [135] proposed a novel Gaussian mutation based on a Gaussian regression to guide the mutation, i.e., the Gaussian regression can predict which architecture may be good, and the newly generated individuals are sampled in the regions of the search space, where the fitness values are likely to be high. This makes mutation to have a “direction.” Moreover, Maziarz *et al.* [180] used an RNN to guide the mutation operation. In this work, the mutation operations were not sampled at random among the possible architectural choices, but were sampled from distributions inferred by an RNN. Using an RNN to control the mutation operation can also be seen in other methods such as [129]. Some researchers investigated the diversity of the population after mutation. Qiang *et al.* [39] used a variable mutation probability. They used a higher probability in the early stage for better exploration and a lower probability in the later stage for better exploitation. It has been effectively applied to many other methods [70]. To maintain the diversity of the population after mutation operation, Tian *et al.* [67] used *force mutation* and distance calculation, which ensures the individual in the population is not particularly similar to other individuals (especially the best one). Kramer [128] used the  $(1+1)$ -ES that generates an offspring based on a single parent with bit-flip mutation and used a mutation rate to control and niching to overcome local optima.

The intensive computational cost of ENAS presents a bottleneck which will be discussed in Sections VI and VIII-C. To reduce the unaffordable computational cost and time, some kinds of mutation have also been designed. Zhang *et al.* [104] proposed an exchange mutation which exchanges the position of two genes of the individual, i.e., exchanging the order of layers. This will not bring new layers and the weights in neural networks can be completely preserved, which means that the offspring do not have to be trained from scratch. Chen *et al.* [181] introduced two function-preserving operators for DNNs, and these operators are termed as network morphisms [182]. The network morphisms aim to change the DNN architecture without the loss of the acquired experience. The network morphisms change the architecture from  $F(\cdot)$  to  $G(\cdot)$ , which satisfies the condition formulated as follows:

$$\forall x, \quad F(x) = G(x) \quad (2)$$

where  $x$  denotes the input of the DNN. The network morphisms can be regarded as a function-preserving mutation operation. With this operation, the mutated individuals will not have worse performance than their parents. To be more specific, Chen *et al.* [181] proposed *net2widernet* to obtain a wider net and *net2deepernet* to obtain a deeper net. Elsken *et al.* [32] extended the network morphisms with two popular network operations: skip connections and batch normalization. Zhu *et al.* [60] proposed five well-designed function-preserving mutation operations to guide the evolutionary process by the information which have already learned. To avoid local optimal, Chen *et al.* [123] added noises in some function-preserving mutation, and in the experiment, they found that by adding noises to pure network morphism, instead of compromising the efficiency, it in contrast improved the final classification accuracy. Note that all the network morphisms can only increase the capacity of a network because if one would decrease the network’s capacity, the function-preserving property could not be guaranteed [105]. As a result, the architecture generated by network morphisms is only going to get larger and deeper, which is not suitable for a device with limited computing resources, such as a mobile phone. In order for the network architecture to be reduced, Elsken *et al.* [105] proposed the approximate network morphism, which satisfies

$$\forall x, \quad F(x) \approx G(x). \quad (3)$$

For the crossover operator, the single-point crossover [183] is the most popular method in EA-based ENAS [48], [64], [66], [134] because of its implementation simplicity. However, single-point crossover can typically apply to two individuals with equal lengths only. Therefore, this cannot be applied to variable-length individuals. To this end, Sun *et al.* [24] proposed an efficient crossover operator for individuals with variable lengths. Sapra and Pimentel [79] proposed a disruptive crossover swapping the whole cluster (a sequence of layers) between both individuals at the corresponding positions rather than only focusing on the parameters of layers. Sun *et al.* [21] used the simulated binary crossover (SBX) [184] to do a combination of the encoded parameters from two matched layers. Note that the encoded parameters after SBX are not the same as that of both parents, which are quite different from other crossover operators.

EAs for multiobjective ENAS are gaining more and more attention from researchers. The single-objective ENAS algorithms are always concerned about only one objective, e.g., the classification accuracy, and these algorithms have only one goal: searching for the architecture with the highest accuracy. In general, most of the multiobjective ENAS algorithms aim at dealing with both the performance of the neural network and the number of parameters simultaneously [101], [105], [137], [170]. However, these objective functions are often in conflict with each other. For example, getting a higher accuracy often requires a more complicated architecture with the need of more computational resources. On the contrary, a device with limited computational resource, e.g., a mobile phone, cannot afford such sophisticated architectures.

The simplest way to tackle the multiobjective optimization problem is by converting it into a single-objective optimization

problem with weighting factors, i.e., the weighted summation method

$$F = \lambda f_1 + (1 - \lambda) f_2 \quad (4)$$

is the classical linear form to weight two objective functions  $f_1$  and  $f_2$  into a single objective function, where the  $\lambda \in (0, 1)$  denotes the weighting factor. In [65], [83], [113], and [185], the multiobjective optimization problem was solved using the available single-objective optimization methods by (4) of the weighted summation. Chen *et al.* [130] did not adopt the linear addition as the objective function, while using a nonlinear penalty term. However, the weights manually defined may incur bias [186].

Some algorithms have been designed and widely used in multiobjective optimization, such as NSGA-II [187] and MOEA/D [188], which have also been used in ENAS methods such as [107], [115], and [169]. These methods aim to find a Pareto front set (or nondominant set). Only these methods are in the multiobjective category of Table III. Some researchers have made improvements on these multiobjective optimization methods for better use in ENAS. Baldeon-Calisto and Lai-Yuen [107] chose the penalty-based boundary intersection approach in MOEA/D because training a neural network involves nonconvex optimization and the form of Pareto front is unknown. LEMONADE [105] divided the objective function into two categories:  $f_{\text{exp}}$  and  $f_{\text{cheap}}$ .  $f_{\text{exp}}$  denotes the expensive-to-evaluate objectives (e.g., the accuracy), while  $f_{\text{cheap}}$  denotes the cheap-to-evaluate objectives (e.g., the model size). In every iteration, they sampled parent networks with respect to sparsely distribution based on the cheap objectives  $f_{\text{cheap}}$  to generate offspring. Therefore, they evaluated  $f_{\text{cheap}}$  more times than  $f_{\text{exp}}$  to save time. Schoron *et al.* [81] also took the use of the LEMONADE proposed by Elsen *et al.* [105]. Because NSGA-III [189] may fall into the small model trap (this algorithm prefers small models), Yang *et al.* [137] have made some improvements to the conventional NSGA-III for favoring larger models.

### B. SI for ENAS

PSO is inspired by the bird flocking or fish schooling [16] and is easy to implement compared with other SI algorithms. Junior and Yen [74] used their implementation of PSO to update the particles based on the layer instead of the parameters of the layer. Gao *et al.* [165] developed a gradient-priority PSO algorithm to handle issues including the low convergence efficiency of PSO when there are a large number of hyperparameters to be optimized. They expected the particle to find the locally optimal solution at first, and then move to the global optimal solution.

For ACO, the individuals are generated in a quite different way. Several ants are in an ant colony.<sup>4</sup> Each ant moves from node to node following the pheromone instructions to build an architecture. The pheromone is updated every generation. The paths of well-performed architecture will maintain more pheromone to attract the next ant for exploitation, and at the

same time, the pheromone is also decaying (i.e., pheromone evaporation), which encourages other ants to explore other areas. Byla and Pang [106] let the ants choose the path from node to node in a graph whose depth increases gradually. Elsaid *et al.* [163] introduced different ant agent types to act according to specific roles to serve the needs of the colony, which is inspired by the real ant species.

SI for multiobjective ENAS started only in the last two years and the research of this field is scarce which can be seen from Table III. Li *et al.* [51] used the bias-variance framework on their proposed multiobjective PSO to get a more accurate and stable architecture. Wu *et al.* [133] used the MOPSO [190] for neural networks pruning. The  $G_{\text{best}}$  is selected according to the crowding distance in the nondominant solutions set. Wang *et al.* [109] used the OMOPSO [191], which selects the leaders using a crowding factor and the  $G_{\text{best}}$  is selected from the leaders. To better control the balance between convergence and diversity, Jiang *et al.* [92] proposed an MOPSO/D algorithm based on an adaptive penalty-based boundary intersection.

### C. Other EC Techniques for ENAS

Different from GAs, the mutation of DE exploits the information from three individuals. Some ENAS methods like [52], [90] chose DE to guide the offspring generation. However, there is little difference between different DE-based ENAS algorithms.

Wang *et al.* [192] proposed a hybrid PSO-GA method. They used PSO to guide the evolution of the parameters in each block encoded in decimal notation. Meanwhile, using GA to guide the evolution of the shortcut connections is encoded in binary notation. Because PSO performs well on continuous optimization and GA is suitable for optimization with binary values, this hybrid method can search architectures effectively.

HCA can be interpreted as a very simple EA. For example, in [32] the evolutionary operators only contain mutation and no crossover, and the selection strategy is relatively simple. The memetic algorithm is the hybrids of EAs and local search. Evans *et al.* [167] integrated the local search (as gradient descent) into GP as a fine-tuning operation. The CVOA [88] was inspired by the new respiratory virus, COVID-19. The architecture was found by simulating the virus spreads and infecting healthy individuals. Hyper-heuristic contains two levels: high-level strategy and low-level heuristics, and a domain barrier is between these two levels. Hence, the high-level strategy is still useful when the application domain is changed. AIS was inspired by theories related to the mammal immune system and do not require the crossover operator compared with the GA [76].

## VI. EFFICIENT EVALUATION

In this section, we will discuss the strategies to improve the efficiency of evaluations, with the consideration that the evaluation is often the most time-consuming stage of ENAS algorithms [193].

Real *et al.* [23] used 250 computers to finish the LargeEvo algorithm over 11 days. Such computational resources are

<sup>4</sup>The population in ACO also termed as colony.

TABLE V  
DIFFERENT METHODS TO SHORTEN THE EVALUATION TIME

Weight inheritance	[23], [32], [59], [60], [66], [71], [76], [80], [81], [87], [89], [91], [105], [106], [110], [122], [123], [127], [133], [135]
Early stopping policy	[25], [34], [47], [66], [67], [69], [75], [76], [82], [83], [86], [89], [89], [97], [99], [103], [109], [113], [116], [127], [165]
Reduced training set	[50], [87], [126], [131], [198]
Reduced population	[99], [124], [126]
Population memory	[24], [34], [61], [119]

not available for everyone interested in NAS. Almost all the methods evaluate individuals by training them first and evaluating them on the validation/test dataset. Since the architecture is becoming more and more complex, it will take a lot of time for training each architecture to convergence. So it is needed to investigate new methods to shorten the evaluation time and reduce the dependency on large amounts of computational resources. Table V lists five of the most common methods to reduce the time: weight inheritance, early stopping policy, reduced training set, reduced population, and population memory. We would like to introduce the five kinds of methods first, then other kinds of promising methods next, and finally the surrogate-assisted methods at the end of this section.

Because the evolutionary operators do not completely disrupt the architecture of an individual, some parts of the newly generated individuals are the same as their parents. The weights of the same parts can be easily inherited. With the weight inheritance, the neural networks no longer need to be trained completely from scratch. This method has been used in [166] 20 years ago. Moreover, as mentioned in Section V, the network morphisms change the network architecture without loss of the acquired experience. This could be regarded as the ultimate weight inheritance because it solved the weight inheritance problem in the changed architecture part. The ultimate weight inheritance lets the new individual completely inherit the knowledge from their parents, which will save a lot of time.

The early stopping policy is another method which has been used widely in NAS. The simplest way is to set a fixed and relatively small number of training epochs. This method is used in [25], where training the individuals after a small number of epochs is sufficient. Similarly, Assunção *et al.* [82] let the individuals undergo the training for the same and short time for each epoch (although this time is not fixed and will increase with the epoch), to allow the promising architectures have more training time to get a more precise evaluation. So *et al.* [194] set hurdles after a fixed number of epochs. The weak individuals stop training early to save time. However, the early stopping policy can lead to inaccurate estimation about individuals' performance (especially the large and complicated architecture), which can be seen in Fig. 7. In Fig. 7, *individual2* performs better than *individual1* before epoch  $t_1$ , whereas *individual1* performs better in the end. Yang *et al.* [137] also discussed this phenomenon. So, it is crucial to determine at which point to stop. Note that neural networks can converge or hardly improve its performance after several epochs, as seen in  $t_1$  for *individual2* and  $t_2$  for

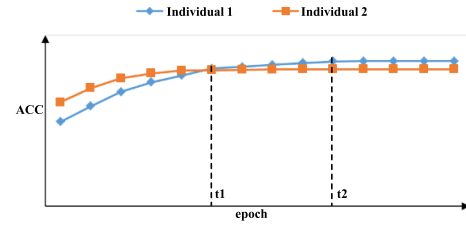


Fig. 7. Two learning curves of two different individuals.

*individual1* in Fig. 7. Using the performance estimated at this point can evaluate an individual's relatively accurately with less training time. Therefore, some methods such as [34] and [89] stopped training when observing there is no significant performance improvement. Sukanuma *et al.* [116] used the early stopping policy based on a reference curve. If the accuracy curve of an individual is under the reference curve for successive epochs, then the training will be terminated and this individual is regarded as a poor one. After every epoch, the reference curve is updated by the accuracy curve of the best offspring.

The reduced training set, i.e., using a subset of data assuming similar properties to a large dataset, can also shorten the time effectively. Liu *et al.* [126] explored promising architectures by training on a subset and used transfer learning on the large original dataset. Because there are so many benchmark datasets in the image classification field, the architecture can be evaluated on a smaller dataset (e.g., CIFAR-10) first and then applied on a large dataset (such as CIFAR-100 and ImageNet [195]). The smaller dataset can be regarded as the proxy for the large one.

The reduced population is a unique method of ENAS since other NAS approaches do not have a population. Assunção *et al.* [99] reduced the population based on their previous algorithm [196], [197] to speed up the evolution. However, simply reducing the population may not explore the search space well in each epoch and may lose the global search ability. Another way is reducing the population dynamically. For instance, Fan *et al.* [124] used the  $(\mu + \lambda)$  evolution strategy and divided the evolution into three stages with population reduction, which aims to find balance between the limited computing resources and the efficiency of the evolution. The large population in the first stage is to ensure the global search ability, while the small population in the last stage is to shorten the evolution time. Instead of reducing the population, Liu *et al.* [198] evaluated the downsizing architecture with smaller size at an early stage of evolution. Similarly, Wang *et al.* [131] did not evaluate the whole architecture but starting with a single block, and then the blocks were stacked to build architecture as evolution proceeds.

The population memory is another category of the unique methods of ENAS. It works by reusing the corresponding architectural information that has previously appeared in the population. In the population-based methods, especially the GA-based methods (e.g., in [21]), it is natural to maintain well-performed individuals in the population in successive generations. Sometimes, the individuals in the next population directly inherit all the architecture information of their parents without any modification, so it is not necessary to evaluate



the individuals again. Fujino *et al.* [34] used memory to record the fitness of individuals, and if the same architecture encoded in an individual appears, the fitness value is retrieved from memory instead of being reevaluated. Similarly, Miah *et al.* [119] and Sun *et al.* [24] used a hashing method for saving pairs of architecture and fitness of each individual and reusing them when the same architecture appears again. Johner and Wassner [61] prohibited the appearance of architectures that have appeared before offspring generation. This does reduce the time; however, the best individuals are forbidden to remain in the population which may lead the population evolve toward a bad direction.

There are many other well-performed methods to reduce the time in ENAS. Rather than training thousands of different architectures, one-shot model [199] trained only one SuperNet to save time. Different architectures, i.e., the SubNets, are sampled from the SuperNet with the shared parameters. Yang *et al.* [137] believed the traditional ENAS methods without using SuperNet which were less efficient for models to be optimized separately. In contrast, the one-shot model optimizes the architecture and the weights alternatively. But the weight sharing mechanism brings difficulty in accurately evaluating the architecture. Chu *et al.* [200] scrutinized the weight-sharing NAS with a fairness perspective and demonstrated the effectiveness. However, there remain some doubts that cannot explain clearly the one-shot model. The weights in the SuperNet are coupled. It is unclear why inherited weights for a specific architecture are still effective [139].

Making the use of hardware can reduce the time, too. Jiang *et al.* [92] used a distributed asynchronous system which contained a major computing node with 20 individual workers. Each worker is responsible for training a single block and uploading its result to the major node in every generation. Wang *et al.* [109] designed an infrastructure which has the ability to leverage all the available GPU cards across multiple machines to concurrently perform the objective evaluations for a batch of individuals. Colangelo *et al.* [201], [202] designed a reconfigurable hardware framework that fits the ENAS. As they claimed, this was the first work of conducting NAS and hardware co-optimization.

Furthermore, Lu *et al.* [101] adopted the concept of proxy models, which are small-scale versions of the intended architectures. For example, in a CNN architecture, the number of layers and the number of channels in each layer are reduced. However, the drawback of this method is obvious: the loss of prediction accuracy. Therefore, they performed an experiment to determine the smallest proxy model that can provide a reliable estimate of performance at a larger scale.

All the above methods obtain the fitness of individuals by directly evaluating the performance on the validation dataset. An alternative way is to use indirect methods, namely, the performance predictors. As summarized in [203], the performance predictors can be divided into two categories: performance predictors based on the learning curve and end-to-end performance predictors, both of which are based on the training-predicting learning paradigm. This does not mean the performance predictor does not undergo the training phase at all, while it means learning from the information obtained

in the training phase, and uses the knowledge learned to make a reasonable prediction for other architectures. Rawal and Miikkulainen [157] took the use of the learning-curve-based predictor, where the fitness is not calculated in the last epoch but is predicted by the sequence of fitness from first epochs. Specifically, they used a long short-term memory (LSTM) [204] as a sequence to sequence model, predicting the final performance using the learning results of the first several epochs on the validation data set. Sun *et al.* [203] adopted a surrogate-assisted method which is called end-to-end performance predictor. The predictor does not need any extra information about the performance of individuals to be evaluated. The performance predictor is essentially a regression model mapping from the architecture to its performance. The regression model needs to be trained with sufficient training data pairs first, and each pair consists of architecture and its corresponding performance. Specifically, they chose random forest [205] as the regression model to accelerate the fitness evaluations in ENAS. When the random forest receives a newly generated architecture as input, the adaptive combination of a huge number of regression trees which have been trained in advance in the forest gives the prediction.

## VII. APPLICATIONS

This section discussed different application fields of ENAS have involved. Generally, the ENAS algorithms can be applied to wherever DNNs can be applied. Table VI shows the wide range of applications and Table VII displays the performance of extraordinary ENAS methods on two popular and challenging datasets for image classification tasks, namely, CIFAR-10 and CIFAR-100. Both these tables can show what ENAS has achieved so far.

### A. Overview

Table VI shows the applications of the existing ENAS algorithms, which contains a wide range of real-world applications.

Generally, these applications can be grouped into the following five different categories.

- 1) Image and signal processing, including image classification which is the most popular and competitive field, image-to-image processing (including image restoration, image denoising, super-resolution, and image inpainting), emotion recognition, speech recognition, language modeling, and face de-identification.
- 2) Biological and biomedical tasks, including medical image segmentation, malignant melanoma detection, sleep heart study, and assessment of human sperm.
- 3) Predictions and forecasting about all sorts of data, including the prediction of wind speed, car park occupancy, time-series data, financial and usable life, the forecasting of electricity demand time series, traffic flow, electricity price, and municipal waste.
- 4) Engineering, including engine vibration prediction, unmanned aerial vehicle (UAV), bearing fault diagnosis, and predicting general aviation flight data.
- 5) Others, including crack detection of concrete, gamma-ray detection, multitask learning, identify galaxies, video understanding, and comics understanding.

TABLE VI  
APPLICATIONS OF EXISTING ENAS ALGORITHMS

Category	Applications	References
(1)	Image classification	[9], [21]–[25], [34]–[36], [43], [44], [47], [48], [53], [54], [60]–[62], [65]–[67], [71], [72], [74], [75], [75]–[77], [81], [83], [86], [87], [91]–[93], [97]–[102], [104]–[106], [108]–[113], [115]–[117], [120], [122], [123], [125], [127], [128], [130]–[135], [137], [139], [146], [154], [162], [167], [169], [170], [172], [178], [180], [202], [206]–[209]
(1)	Image to image	[42], [50], [96], [114], [121], [129], [149]
(1)	Emotion recognition	[94], [165]
(1)	Speech recognition	[58], [84], [103], [140], [160]
(1)	Language modeling	[117], [157]
(1)	Face De-identification	[210]
(2)	Medical image segmentation	[39], [40], [73], [107], [118], [124], [135]
(2)	Malignant melanoma detection	[59], [80]
(2)	Sleep heart study	[168]
(2)	Assessment of human sperm	[119]
(3)	Wind speed prediction	[159]
(3)	Electricity demand time series forecasting	[88]
(3)	Traffic flow forecasting	[51]
(3)	Electricity price forecasting	[90]
(3)	Car park occupancy prediction	[45]
(3)	Energy consumption prediction	[95]
(3)	Time series data prediction	[151]
(3)	Financial prediction	[211]
(3)	Usable life prediction	[55]
(3)	Municipal waste forecasting	[46]
(4)	Engine vibration prediction	[136], [164]
(4)	UAV	[153]
(4)	Bearing fault diagnosis	[52]
(4)	Predicting general aviation flight data	[138]
(5)	Crack detection of concrete	[64]
(5)	Gamma-ray detection	[82]
(5)	Multitask learning	[212]
(5)	Identify Galaxies	[148]
(5)	Video understanding	[70]
(5)	Comics understanding	[57], [213]

### B. Comparisons on CIFAR-10, CIFAR-100 and ImageNet

In Table VI, it is obvious to see that many ENAS methods are applied to image classification tasks. The benchmark data set, CIFAR-10, which contains a total of ten classes, and the CIFAR-100 are the advanced data sets including a hundred classes. These two data sets have been widely used in image classification tasks, and the accuracy on these two challenging data sets can represent the ability of the architecture searched. In addition, ImageNet [195] is a more challenging benchmark data set, which contains 1000 classes and more than one million images. Because CIFAR-10 and CIFAR-100 are relatively small and easy to be over-fitting nowadays [122], the results of the ENAS methods on ImageNet are also included. We have collected the well-performed ENAS methods tested on these three datasets and showed the results in Table VII, where the methods are ranked in an ascending order of their best accuracy on CIFAR-10, i.e., the methods are ranked in a descending order of their error rate based on the conventions. The data shown under column “CIFAR-10” and “CIFAR-100” denotes the error rate of each method on the corresponding

datasets. Especially, as for ImageNet, we report both top1 and top5 error rates. Furthermore, the “GPU days,” which was initially proposed in [21], denotes the total search time of each method, and it can be calculated as follows:

$$\text{GPU Days} = \text{The number of GPUs} \times t \quad (5)$$

where the  $t$  denotes the number of days that each method searched for. In Table VII “Parameters” denotes the total number of parameters which can represent the capability of architecture and the complexity. In addition, the symbol “—” in Table VII implies there is no result publically reported by the corresponding paper. The year reported in this table is its earliest time made public. Furthermore, there are additional notes provided for Table VII. First, if there are several results reported from literature, such as CGP-DCNN [113] and CARS [137], we choose to report one or two of the most representative architectures. Second, we name two algorithms without proper names in Table VII (i.e., Firefly-CNN [77] and EEDNAS-NNMM [123]) based on the EC methods they used and the first letter of the title. Third, we report the classification errors of CGP-CNN [116] in the format of “best (mean  $\pm$  std).” Finally, the symbols of “+” and “−” in the column of “GPU days” denote the meaning of “more than” and “less than,” respectively.

In principle, there is no totally fair comparison, due to the following two reasons.

- 1) The encoding space including the initial space and the search space is different from each other. There are two extreme cases in the initial space: trivial initialization which starts at the simplest architecture and rich initialization which starts from a well-designed architecture (e.g., ResNet-50 [2]). Meanwhile the size of the search space is largely different, e.g., [48] only takes the kernel size into the search space.
- 2) Different tricks exploited in the methods, e.g., the “cutout,” can make the comparisons unfair, too. The “cutout” refers to a regularization method [216] used in the training of CNNs, which could improve the final performance appreciably.

Table VII shows the progress of ENAS for image classification according to the accuracy on CIFAR-10: LargeEvo algorithm [23] (5.4%, 2017), LEMONADE [105] (2.58%, 2018), and NSGANet [101] (2.02%, 2019). Many ENAS methods have a lower error rate than ResNet-110 [2] with 6.43% error rate on CIFAR-10, which is a manually well-designed architecture. Therefore, the architecture found by ENAS can reach the same level or exceed the architecture designed by experts. It shows that the ENAS is reliable and can be used in other application fields.

## VIII. CHALLENGES AND ISSUES

Despite the positive results of the existing ENAS methods, there are still some challenges and issues which need to be addressed. In the following, we will briefly discuss them.

### A. Effectiveness

The effectiveness of ENAS is questioned by many researchers. Wistuba *et al.* [27] noted that the random search

TABLE VII  
COMPARISON OF THE CLASSIFICATION ERROR RATE ON CIFAR-10, CIFAR-100, AND IMAGENET

ENAS Methods	GPU Days	Parameters(M)	CIFAR-10(%)	CIFAR-100(%)	ImageNet(Top1/Top5 %)	Year
CGP-DCNN [113]	—	1.1	8.1	—	—	2018
EPT [87]	2	—	7.5	—	—	2020
GeNet [35]	17	—	7.1	—	—	2017
		156	—	29.03	27.87 / 9.74	
EANN-Net [108]	—	—	7.05 $\pm$ 0.02	—	—	2019
DeepMaker [115]	3.125	1	6.9	—	—	2020
		1.89	—	24.87	—	
GeneCai (ResNet-50) [178]	0.024	—	6.4	—	25.7 / 7.9	2020
CGP-CNN (ConvSet) [112]	—	1.52	6.75	—	—	2017
CGP-CNN (ResSet) [112]	—	1.68	5.98	—	—	2017
MOPSO/D-Net [92]	0.33	8.1	5.88	—	—	2019
ReseNet-50 (20% pruned) [134]	—	6.44	5.85	—	—	2018
ImmuNeCS [76]	14	—	5.58	—	—	2019
EIGEN [93]	2	2.6	5.4	—	—	2018
	5	11.8	—	21.9	—	
LargeEvo [23]	2750	5.4	5.4	—	—	2017
		40.4	—	23	—	
CGP-CNN (ConvSet) [116]	31	1.5	5.92 (6.48 $\pm$ 0.48)	—	—	2019
	—	2.01	—	26.7 (28.1 $\pm$ 0.83)	—	
CGP-CNN (ResSet) [116]	30	2.01	5.01 (6.10 $\pm$ 0.89)	—	—	2019
	—	4.6	—	25.1 (26.8 $\pm$ 1.21)	—	
MOCNN [109]	24	—	4.49	—	—	2019
NASH [32]	4	88	4.4	—	—	2017
	5	111.5	—	19.6	—	
HGAPSO [75]	7+	—	4.37	—	—	2018
DPP-Net [214], [215]	2	11.39	4.36	—	—	2018
		0.45	5.84	—	—	
		4.8	—	—	26.0 / 8.2	
AE-CNN [9]	27	2	4.3	—	—	2019
	36	5.4	—	20.85	—	
SI-ENAS [104]	1.8	—	4.07	18.64	—	2020
EPSOCNN [131]	4-	6.77	3.69	—	—	2019
Hierarchical Evolution [100]	300	—	3.63 $\pm$ 0.10	—	20.3 / 5.2	2017
		—	—	—	—	
EA-FPN [127]	0.5	5.8	3.57	—	—	2018
	1	7.2	—	21.74	—	
AmoebaNet-A [36]	3150	3.2	3.34 $\pm$ 0.06	—	—	2018
		86.7	—	—	17.2 / 3.9	
Firefly-CNN [77]	—	3.21	3.3	22.3	—	2019
		2.9	3.22	—	—	
CNN-GA [24]	35	2.9	—	—	—	2018
	40	4.1	—	20.53	—	
	35	—	—	—	25.2 / 7.7	
JASQNet [130]	3	3.3	2.9	—	—	2018
		1.8	2.97	—	—	
		4.9	—	—	27.2 / —	
RENASNet [122]	6	3.5	2.88 $\pm$ 0.02	—	—	2018
		5.36	—	—	24.3 / 7.4	
NSGA-Net [170]	4	3.3	2.75	—	—	2018
		2.4	3	—	—	
CARS [137]	0.4	3.6	2.62	—	—	2019
		3.7	—	—	27.2 / 9.2	
		5.1	—	—	24.8 / 7.5	
		—	—	—	—	
LEMONADE [105]	80	13.1	2.58	—	—	2018
		0.5	4.57	—	—	
		—	—	—	28.3 / 9.6	
EENA [60]	0.65	8.47	2.56	—	—	2019
EEDNAS-NNMM [123]	0.5	8.49	—	17.71	—	2019
		4.7	2.55	—	—	
NSGANet [101]	27	0.2	4.67	—	—	2019
		4	2.02	—	—	
		0.2	—	25.17	—	
		4.1	—	14.38	—	
		5.0	—	—	23.8 / 7.0	

can get a well-performed architecture and has proven to be an extremely strong baseline. Yu *et al.* [217] showed the state-of-the-art NAS algorithms performed similar to the random policy on average. Liashchynskyi and Liashchynskyi [218] compared with grid search, random search, and GA for NAS, which showed that the architecture obtained by GA and random search has similar performance. There is no need to use complicated algorithms to guide the search process if random search can outperform NAS based on EC approaches.

However, the evolutionary operator in [218] only contains a recombination operator, which limited the performance

of ENAS algorithms. Although random search can find a well-performed architecture in the experiments, it cannot guarantee that it will find a good architecture every time. Moreover, recent researches [100], [139] also showed that the evolutionary search was more effective than random search. Furthermore, the experiments in Amoebanet [36] and NAS-Bench-201 [143] also showed that ENAS can search for and find better architectures. Thus, there is an urgent need to design an elaborated experiment to reveal what components in ENAS are responsible for the effectiveness of the algorithm, especially in a large encoding space.



In Section V, two types of operators have been introduced. We note that some methods such as the LargeEvo algorithm [23] only use single individual-based operator (mutation) to generate offspring. The main reason that they did not involve the crossover operator in their method come from two reasons: the first is for simplicity [174], and the second is that simply combining a section of one individual with a section of another individual seems “ill-suited” to the neural network paradigm [76]. Second, Wistuba *et al.* [27] believed that there was no indication that a recombination operation applied to two individuals with high fitness would result in an offspring with similar or better fitness.

However, the supplemental materials in [24] demonstrated the effectiveness of the crossover operator in this method. This method can find a good architecture with the help of crossover operation. On the contrary, when crossover is not used, the architecture found is not promising, unless it runs for a long time. In fact, the mutation operator let an individual explore the neighboring region, and it is a gradually incremental search process like searching step by step. Crossover (recombination) can generate offspring dramatically different from the parents, which is more likely a stride. So, this operator has the ability to efficiently find a promising architecture. Chu *et al.* [121] preferred that while a crossover mainly contributes to exploitation, a mutation is usually aimed to introduce exploration. These two operators play different roles in the evolutionary process. But there is not a sufficient explanation of how the crossover operator works. Maybe some additional experiments need to be done on the methods without the crossover operator.

The EC approach generally performs well when dealing with practical problems (e.g., the vehicle routing problem). This is mainly because EC is designed collectively by considering the domain knowledge in most cases [219], [220]. Similarly, another key reason to ensure the effectiveness of the ENAS algorithm is that the EC approach can effectively consider (and embed) some domain knowledge of the neural architecture. Since the original intention of ENAS is to design neural architectures automatically without manual experience, the future development of ENAS will need to continue to reduce any artificial experience by including prior knowledge of the encoding space. In this case, how to effectively embed domain knowledge into ENAS is a big challenge.

### B. Scalability

The scale of the data sets used in most ENAS methods is relatively large. Taking the image classification task as an example, the MNIST data set [221] is one of the earliest data sets. In total, it contains 70 000  $28 \times 28$  grayscale images. Later in 2009, CIFAR-10 and CIFAR-100 [13] including 60 000  $32 \times 32$  color images are medium-scale datasets. One of the most well-known large-scale datasets is ImageNet [195], which provides more than 14 million manually annotated high-resolution images. Unlike CIFAR-10 and CIFAR-100, which are commonly used in ENAS, fewer methods choose to verify their performance on ImageNet [122], [134], [180]. This can be explained by the data shown in Table VII, where the GPU days are usually tens or even thousands. It is unaffordable for

most researchers when the medium-scale data set changes to the larger one.

However, Chen *et al.* [122] believed that the results on the larger datasets like ImageNet are more convincing because CIFAR-10 is easy to be over-fitting. A popular way to deal with this is using a proxy on CIFAR-10 and transfer to ImageNet [36], [105]. Another alternative approach is to use the down-scaled dataset of a large-scale dataset such as ImageNet64  $\times$  64 [222].

### C. Efficient Evaluation Method and Reduce Computational Cost

Section VI has introduced the most popular and effective ways to reduce the fitness evaluation time and the computational cost. In a nutshell, it can be described as a question that strikes the balance between the time spent and the accuracy of the evaluation. Because of the unbearable time for fully training the architecture, we must compromise as little as we can on the evaluation accuracy in exchange for significant reduction in the evaluation time without sufficient computing resources.

Although a lot of ENAS methods have adopted various means to shorten the evaluation time. Even though Sun *et al.* [203] specifically proposed a method for acceleration, the research direction of search acceleration is just getting started. The current approaches have many limitations that need to be addressed. For example, although the LargeEvo algorithm [23] used weight inheritance to shorten the evaluation time and reduced the computational cost, it still ran for several days with the use of lots of computational resources which cannot be easily accessed by many researchers. Furthermore, there is no baseline and common assessment criteria for search acceleration methods. It is a major challenge to propose a novel method to evaluate the architecture accurately and quickly.

### D. Interpretability

CNNs are known as black-box-like solutions, which are hard to interpret [63]. Although some works have been done to visualize the process of feature extraction [223], they are uninterpretable due to a large number of learned features [154]. The low interpretability of the manual designed architecture becomes a big obstacle to the development of neural networks. To overcome this obstacle, some studies [63], [154], [167] used GP to automatically design the neural network. Being well-known for its potential interpretability, GP aims at solving problems by automatically evolving computer programs [224].

All the above studies [63], [154], [167] gave a further analysis to demonstrate the interpretability. Specifically, Evans *et al.* [154] have made a visualization on the JAFFE dataset [225] to expound how the evolved convolution filter served as a form of edge detection, and the large presence of white color in the output of the convolution can help the classification. In their subsequent work [167], they made a visualization of the automatically evolved model on the Hands data set [226], where the aggregation function extracts the minimum value of a specific area in the hand image to determine whether the hand is open or closed. Furthermore, Bi *et al.* [63] displayed the features described by the

evolved functions like convolution, max-pooling and addition, and the generated salient features are discriminative for face classification.

Despite the interpretability the existing work made, all these GP-based ENAS methods only aim at shallow NNs and the number of the generated features is relatively small. However, all the most successful NNs have a deep architecture. This is due partly to the fact that very few GP-based methods can be run on GPUs. It is necessary to use deep GP to evolve a deeper GP tree and make a further analysis on the deeper architecture in the future.

#### E. Future Applications

Table VI shows various applications which have been explored by ENAS. But these are just a small part of all areas of neural network applications. ENAS can be applied wherever neural networks can be applied and automate the process of architecture designed which should have been done by experts. Moreover, plenty of image classification successes of ENAS have proven that ENAS has the ability to replace experts in many areas. The automated architecture design is a trend.

However, this process is not completely automated. The encoding space (search space) still needs to be designed by experts for different applications. For example, for the image processing tasks, CNNs are more suitable, so the encoding space contains the layers including convolution layers, pooling layers and fully connected layers. For the time-series data processing, RNNs are more suitable, so the encoding space may contain cells including  $\Delta$ -RNN cell, LSTM [204], gated recurrent unit (GRU) [227], minimally gated unit (MGU) [228], and update-gated RNN (UGRNN) [229]. The two manually determined encoding spaces already contain a great deal of artificial experience and the components without guaranteed performance are excluded. The problem is: can a method search the corresponding type of neural network for multiple tasks in a large encoding space, including all the popular widely used components? Instead of searching one multitask network [212] which learns several tasks at once with the same neural network, the aim is to find appropriate networks for different tasks in one large encoding space.

#### F. Fair Comparisons

Section VII-B gives a brief introduction of the unfair comparisons. The unfairness mainly comes from two aspects: 1) the tricks including cutout [216], ScheduledDropPath [144], and so on and 2) the different encoding spaces. For aspect 1), some ENAS methods [24] have reported the results with and without the tricks. For aspect 2), the well-designed search space is widely used in different ENAS methods. For instance, the NASNet search space [144] is also used in [36] and [132] because it is well-constructed so that even random search can perform well. The comparison under the same condition can tell the effectiveness of different search methods.

Fortunately, the first public benchmark data set for NAS, the NAS-Bench-101 [142] has been proposed. The data set contains 432K unique convolutional architectures based on the cell-based encoding space. Each architecture can query the corresponding metrics, including test accuracy, training time,

and so on, directly in the data set without the large-scale computation. NAS-Bench-201 [143] was proposed recently and is based on another cell-based encoding space, which does not have limits on edges. Compared with NAS-Bench-101, which was only tested on CIFAR-10, this data set collects the test accuracy on three different image classification datasets (CIFAR-10, CIFAR-100, and ImageNet-16-120 [222]). But the encoding space is relatively small, and only contains 15.6K architectures. Experiments with different ENAS methods on these benchmark data sets can get a fair comparison and it will not take too much time. However, these data sets are only based on the cell-based encoding spaces and cannot contain all the search space of the existing methods, because the other basic units (layers and blocks) are built using more hyperparameters, which may lead to a larger encoding space.

In the future, a common platform making fair comparisons needs to be built. This platform must have several benchmarks encoding space, such as the NASNet search space, NAS-Bench-101, and NAS-Bench-201. All the ENAS methods can be directly tested on the platform. Furthermore, this platform also needs to solve the problem that different kinds of GPUs have different computing power, which may cause an inaccurate GPU days based on different standards. The GPU days cannot be compared directly until they have a common baseline of computing power.

### IX. CONCLUSION

This article provides a comprehensive survey of ENAS. We introduced ENAS from four aspects: population representation, encoding space, population updating, and fitness evaluation following the unified flow, which can be seen in Fig. 2. The various applications and the performance of the state-of-the-art methods on image classification are also summarized in tables to demonstrate the wide applicability and the promising ability. Challenges and issues are also discussed to identify future research direction in this field.

To be specific, first, the encoding space is introduced by categories. We divide the encoding space into two parts: initial space and search space, where the former one defines the initial conditions whereas the latter one defines the architectures that can be found in the evolutionary process. Also, different encoding strategies and architecture representations are discussed. Second, the process of population updating including the evolutionary operators, the multiobjective search strategy, and the selection strategy are presented. A variety of EC paradigms use respective metaphors to generate new individuals. Based on standard algorithms, many improved methods have been proposed to obtain a stable and reliable search capability. Furthermore, we introduce the existing methods to reduce the need for large amounts of time and computing resources, which is a huge obstacle to efficiency.

Although the state-of-the-art methods have achieved some success, ENAS still faces challenges and issues. The first important issue is whether the EC-based search strategy has advantages. If the result is at the same level as the baseline (e.g., random search), it is unnecessary to design the complex evolutionary operators. A sophisticated experiment is in urgent need to tell the effectiveness, especially in a large

encoding space. Second, the crossover operator is a multi-individual based operator and there is no sufficient explanation to how the crossover operator works well on ENAS. Besides, ENAS is just beginning a new era, so there is a lot of uncharted territory to be explored. Moreover, a unified standard or platform is demanded to make a fair comparison.

## REFERENCES

- [1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015.
- [2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.
- [3] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 4700–4708.
- [4] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018, *arXiv:1810.04805*. [Online]. Available: <http://arxiv.org/abs/1810.04805>
- [5] Y. Zhang, W. Chan, and N. Jaitly, "Very deep convolutional networks for end-to-end speech recognition," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Mar. 2017, pp. 4845–4849.
- [6] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," in *Neural Networks: Tricks of the Trade*. New York, NY, USA: Springer, 2012, pp. 437–478.
- [7] J. K. Kearney, W. B. Thompson, and D. L. Boley, "Optical flow estimation: An error analysis of gradient-based methods with local optimization," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-9, no. 2, pp. 229–244, Mar. 1987.
- [8] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*. [Online]. Available: <http://arxiv.org/abs/1409.1556>
- [9] 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. 2019.
- [10] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," 2016, *arXiv:1611.01578*. [Online]. Available: <http://arxiv.org/abs/1611.01578>
- [11] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *J. Artif. Intell. Res.*, vol. 4, no. 1, pp. 237–285, Jan. 1996.
- [12] T. Bäck, D. B. Fogel, and Z. Michalewicz, "Handbook of evolutionary computation," *Release*, vol. 97, no. 1, p. B1, 1997.
- [13] A. Krizhevsky *et al.*, "Learning multiple layers of features from tiny images," Univ. Toronto, Toronto, ON, Canada, Tech. Rep., 2009, p. 12.
- [14] D. E. Goldberg, *Genetic Algorithms*. London, U.K.: Pearson, 2006.
- [15] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone, *Genetic Programming*. New York, NY, USA: Springer, 1998.
- [16] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. IEEE ICNN*, vol. 4, Nov./Dec. 1995, pp. 1942–1948.
- [17] Y. Sun, G. G. Yen, and Z. Yi, "IGD indicator-based evolutionary algorithm for many-objective optimization problems," *IEEE Trans. Evol. Comput.*, vol. 23, no. 2, pp. 173–187, Apr. 2018.
- [18] 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, Mar. 2020.
- [19] D. Floreano, P. Dürri, and C. Mattiussi, "Neuroevolution: From architectures to learning," *Evol. Intell.*, vol. 1, no. 1, pp. 47–62, Mar. 2008.
- [20] E. Dufourq and B. A. Bassett, "Automated problem identification: Regression vs classification via evolutionary deep networks," in *Proc. South Afr. Inst. Comput. Scientists Inf. Technol. (SAICSIT)*, 2017, pp. 1–9.
- [21] 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.
- [22] 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.
- [23] E. Real, "Large-scale evolution of image classifiers," in *Proc. 34th Int. Conf. Mach. Learn. (ICML)*, vol. 70, Aug. 2017, pp. 2902–2911.
- [24] Y. Sun, B. Xue, M. Zhang, G. G. Yen, and J. Lv, "Automatically designing CNN architectures using the genetic algorithm for image classification," *IEEE Trans. Cybern.*, vol. 50, no. 9, pp. 3840–3854, Sep. 2020.
- [25] Y. Sun, B. Xue, G. G. Yen, and M. Zhang, "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.
- [26] T. Elsken, J. Hendrik Metzen, and F. Hutter, "Neural architecture search: A survey," 2018, *arXiv:1808.05377*. [Online]. Available: <http://arxiv.org/abs/1808.05377>
- [27] M. Wistuba, A. Rawat, and T. Pedapati, "A survey on neural architecture search," 2019, *arXiv:1905.01392*. [Online]. Available: <http://arxiv.org/abs/1905.01392>
- [28] K. O. Stanley, J. Clune, J. Lehman, and R. Miikkilainen, "Designing neural networks through neuroevolution," *Nature Mach. Intell.*, vol. 1, no. 1, pp. 24–35, Jan. 2019.
- [29] M. Dorigo, V. Maniezzo, and A. Colnari, "Ant system: Optimization by a colony of cooperating agents," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 26, no. 1, pp. 29–41, Feb. 1996.
- [30] K. Price, R. M. Storn, and J. A. Lampinen, *Differential Evolution: A Practical Approach to Global Optimization*. New York, NY, USA: Springer, 2006.
- [31] X.-S. Yang, "Firefly algorithms for multimodal optimization," in *Proc. Int. Symp. Stochastic Algorithms*. New York, NY, USA: Springer, 2009, pp. 169–178.
- [32] T. Elsken, J.-H. Metzen, and F. Hutter, "Simple and efficient architecture search for convolutional neural networks," 2017, *arXiv:1711.04528*. [Online]. Available: <http://arxiv.org/abs/1711.04528>
- [33] H. Wang, D. Wang, and S. Yang, "A memetic algorithm with adaptive Hill climbing strategy for dynamic optimization problems," *Soft Comput.*, vol. 13, nos. 8–9, pp. 763–780, Jul. 2009.
- [34] S. Fujino, N. Mori, and K. Matsumoto, "Deep convolutional networks for human sketches by means of the evolutionary deep learning," in *Proc. Joint 17th World Congr. Int. Fuzzy Syst. Assoc., 9th Int. Conf. Soft Comput. Intell. Syst. (IFSAS-SCIS)*, Jun. 2017, pp. 1–5.
- [35] L. Xie and A. Yuille, "Genetic CNN," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 1379–1388.
- [36] 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.
- [37] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Comput.*, vol. 18, no. 7, pp. 1527–1554, 2006.
- [38] J. K. Kim, Y. S. Han, and J. S. Lee, "Particle swarm optimization–deep belief network–based rare class prediction model for highly class imbalance problem," *Concurrency Comput., Pract. Exper.*, vol. 29, no. 11, p. e4128, Jun. 2017.
- [39] N. Qiang, B. Ge, Q. Dong, F. Ge, and T. Liu, "Neural architecture search for optimizing deep belief network models of fMRI data," in *Proc. Int. Workshop Multiscale Multimodal Med. Imag.* New York, NY, USA: Springer, 2019, pp. 26–34.
- [40] W. Zhang *et al.*, "Identify hierarchical structures from task-based fMRI data via hybrid spatiotemporal neural architecture search net," in *Proc. Int. Conf. Med. Image Comput. Comput.-Assist. Intervent.* New York, NY, USA: Springer, 2019, pp. 745–753.
- [41] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep Learning*, vol. 1. Cambridge, MA, USA: MIT Press, 2016.
- [42] M. Suganuma, M. Ozay, and T. Okatani, "Exploiting the potential of standard convolutional autoencoders for image restoration by evolutionary search," 2018, *arXiv:1803.00370*. [Online]. Available: <http://arxiv.org/abs/1803.00370>
- [43] J. Hajewski and S. Oliveira, "An evolutionary approach to variational autoencoders," in *Proc. 10th Annu. Comput. Commun. Workshop Conf. (CCWC)*, Jan. 2020, pp. 0071–0077.
- [44] L. Rodriguez-Coayahuil, A. Morales-Reyes, and H. J. Escalante, "Evolving autoencoding structures through genetic programming," *Genetic Program. Evolvable Mach.*, vol. 20, no. 3, pp. 413–440, Sep. 2019.
- [45] A. Camero, J. Toutouh, D. H. Stolfi, and E. Alba, "Evolutionary deep learning for car park occupancy prediction in smart cities," in *Proc. Int. Conf. Learn. Intell. Optim.* New York, NY, USA: Springer, 2018, pp. 386–401.
- [46] A. Camero, J. Toutouh, and E. Alba, "Random error sampling-based recurrent neural network architecture optimization," 2019, *arXiv:1909.02425*. [Online]. Available: <http://arxiv.org/abs/1909.02425>
- [47] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "An experimental study on hyper-parameter optimization for stacked auto-encoders," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jul. 2018, pp. 1–8.



- [48] A. Singh, S. Saha, R. Sarkhel, M. Kundu, M. Nasipuri, and N. Das, "A genetic algorithm based kernel-size selection approach for a multi-column convolutional neural network," 2019, *arXiv:1912.12405*. [Online]. Available: <http://arxiv.org/abs/1912.12405>
- [49] 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, pp. 1–16, Feb. 2019.
- [50] H. Shu and Y. Wang, "Automatically searching for U-Net image translator architecture," 2020, *arXiv:2002.11581*. [Online]. Available: <http://arxiv.org/abs/2002.11581>
- [51] L. Li, L. Qin, X. Qu, J. Zhang, Y. Wang, and B. Ran, "Day-ahead traffic flow forecasting based on a deep belief network optimized by the multi-objective particle swarm algorithm," *Knowl.-Based Syst.*, vol. 172, pp. 1–14, May 2019.
- [52] S. R. Saufi, Z. A. B. Ahmad, M. S. Leong, and M. H. Lim, "Differential evolution optimization for resilient stacked sparse autoencoder and its applications on bearing fault diagnosis," *Meas. Sci. Technol.*, vol. 29, no. 12, Dec. 2018, Art. no. 125002.
- [53] D. Kang and C. W. Ahn, "Efficient neural network space with genetic search," in *Proc. Int. Conf. Bio-Inspired Comput., Theories Appl.* New York, NY, USA: Springer, 2019, pp. 638–646.
- [54] B. Cheung and C. Sable, "Hybrid evolution of convolutional networks," in *Proc. 10th Int. Conf. Mach. Learn. Appl. Workshops*, vol. 1, Dec. 2011, pp. 293–297.
- [55] C. Zhang, P. Lim, A. K. Qin, and K. C. Tan, "Multiobjective deep belief networks ensemble for remaining useful life estimation in prognostics," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 10, pp. 2306–2318, Oct. 2017.
- [56] F. Ye, "Particle swarm optimization-based automatic parameter selection for deep neural networks and its applications in large-scale and high-dimensional data," *PLoS ONE*, vol. 12, no. 12, Dec. 2017, Art. no. e0188746.
- [57] S. Fujino, N. Mori, and K. Matsumoto, "Recognizing the order of four-scene comics by evolutionary deep learning," in *Proc. Int. Symp. Distrib. Comput. Artif. Intell.* New York, NY, USA: Springer, 2018, pp. 136–144.
- [58] T. Tanaka, T. Moriya, T. Shinozaki, S. Watanabe, T. Hori, and K. Duh, "Automated structure discovery and parameter tuning of neural network language model based on evolution strategy," in *Proc. IEEE Spoken Lang. Technol. Workshop (SLT)*, Dec. 2016, pp. 665–671.
- [59] A. Kwasigroch, M. Grochowski, and M. Mikołajczyk, "Deep neural network architecture search using network morphism," in *Proc. 24th Int. Conf. Methods Models Autom. Robot. (MMAR)*, Aug. 2019, pp. 30–35.
- [60] H. Zhu, Z. An, C. Yang, K. Xu, E. Zhao, and Y. Xu, "EENA: Efficient evolution of neural architecture," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. Workshop (ICCVW)*, Oct. 2019, pp. 1891–1899.
- [61] F. M. Johnner and J. Wassner, "Efficient evolutionary architecture search for CNN optimization on GTSRB," in *Proc. 18th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA)*, Dec. 2019, pp. 56–61.
- [62] M. Shen, K. Han, C. Xu, and Y. Wang, "Searching for accurate binary neural architectures," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. Workshop (ICCVW)*, Oct. 2019, pp. 2041–2044.
- [63] Y. Bi, B. Xue, and M. Zhang, "An evolutionary deep learning approach using genetic programming with convolution operators for image classification," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jun. 2019, pp. 3197–3204.
- [64] S. Gibb, H. M. La, and S. Louis, "A genetic algorithm for convolutional network structure optimization for concrete crack detection," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jul. 2018, pp. 1–8.
- [65] L. M. Zhang, "A new compensatory genetic algorithm-based method for effective compressed multi-function convolutional neural network model selection with multi-objective optimization," 2019, *arXiv:1906.11912*. [Online]. Available: <http://arxiv.org/abs/1906.11912>
- [66] 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. Inform.* New York, NY, USA: Springer, 2019, pp. 473–482.
- [67] H. Tian, S.-C. Chen, M.-L. Shyu, and S. Rubin, "Automated neural network construction with similarity sensitive evolutionary algorithms," in *Proc. IEEE 20th Int. Conf. Inf. Reuse Integr. Data Sci. (IRI)*, Jul. 2019, pp. 283–290.
- [68] S. Wei, S. Zou, F. Liao, W. Lang, and W. Wu, "Automatic modulation recognition using neural architecture search," in *Proc. Int. Conf. High Perform. Big Data Intell. Syst. (HPBD IS)*, May 2019, pp. 151–156.
- [69] P. Ortego, A. Diez-Oliván, J. Del Ser, F. Veiga, M. Penalva, and B. Sierra, "Evolutionary LSTM-FCN networks for pattern classification in industrial processes," *Swarm Evol. Comput.*, vol. 54, May 2020, Art. no. 100650.
- [70] A. Piergiovanni, A. Angelova, A. Toshev, and M. Ryoo, "Evolving space-time neural architectures for videos," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 1793–1802.
- [71] J. Prellberg and O. Kramer, "Lamarckian evolution of convolutional neural networks," in *Proc. Int. Conf. Parallel Problem Solving Nature*. New York, NY, USA: Springer, 2018, pp. 424–435.
- [72] 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.*, Nov. 2015, pp. 1–5.
- [73] M. G. B. Calisto and S. K. Lai-Yuen, "Self-adaptive 2D-3D ensemble of fully convolutional networks for medical image segmentation," in *Proc. Med. Imag., Image Process., Int. Soc. Opt. Photon.*, vol. 11313, 2020, Art. no. 113131W.
- [74] G. G. Yen and F. E. Fernandes, Jr., "Particle swarm optimization of deep neural networks architectures for image classification," *Swarm Evol. Comput.*, vol. 49, pp. 62–74, Sep. 2019.
- [75] B. Wang, Y. Sun, B. Xue, and M. Zhang, "A hybrid differential evolution approach to designing deep convolutional neural networks for image classification," in *Proc. Australas. Joint Conf. Artif. Intell.* New York, NY, USA: Springer, 2018, pp. 237–250.
- [76] L. Frachon, W. Pang, and G. M. Coghill, "ImmuNeCS: Neural committee search by an artificial immune system," 2019, *arXiv:1911.07729*. [Online]. Available: <http://arxiv.org/abs/1911.07729>
- [77] A. I. Sharaf and E.-S. F. Radwan, "An automated approach for developing a convolutional neural network using a modified firefly algorithm for image classification," in *Applications of Firefly Algorithm and its Variants*. New York, NY, USA: Springer, 2020, pp. 99–118.
- [78] N. R. Sabar, A. Turky, A. Song, and A. Sattar, "An evolutionary hyper-heuristic to optimise deep belief networks for image reconstruction," *Appl. Soft Comput.*, vol. 97, Dec. 2020, Art. no. 105510.
- [79] D. Sapra and A. D. Pimentel, "An evolutionary optimization algorithm for gradually saturating objective functions," in *Proc. Genetic Evol. Comput. Conf.*, Jun. 2020, pp. 886–893.
- [80] A. Kwasigroch, M. Grochowski, and A. Mikołajczyk, "Neural architecture search for skin lesion classification," *IEEE Access*, vol. 8, pp. 9061–9071, 2020.
- [81] C. Schorn, T. Elsken, S. Vogel, A. Runge, A. Guntoro, and G. Ascheid, "Automated design of error-resilient and hardware-efficient deep neural networks," 2019, *arXiv:1909.13844*. [Online]. Available: <http://arxiv.org/abs/1909.13844>
- [82] F. Assunção *et al.*, "Automatic design of artificial neural networks for gamma-ray detection," *IEEE Access*, vol. 7, pp. 110531–110540, 2019.
- [83] D. Laredo, Y. Qin, O. Schütze, and J.-Q. Sun, "Automatic model selection for neural networks," 2019, *arXiv:1905.06010*. [Online]. Available: <http://arxiv.org/abs/1905.06010>
- [84] M. U. Anwar and M. L. Ali, "Boosting neuro evolutionary techniques for speech recognition," in *Proc. Int. Conf. Electr., Comput. Commun. Eng. (ECCE)*, Feb. 2019, pp. 1–5.
- [85] S. Y. Teng, A. C. M. Loy, W. D. Leong, B. S. How, B. L. F. Chin, and V. Maša, "Catalytic thermal degradation of *Chlorella vulgaris*: Evolving deep neural networks for optimization," *Bioresour. Technol.*, vol. 292, Nov. 2019, Art. no. 121971.
- [86] S. Litzinger, A. Klos, and W. Schiffmann, "Compute-efficient neural network architecture optimization by a genetic algorithm," in *Proc. Int. Conf. Artif. Neural Netw.* New York, NY, USA: Springer, 2019, pp. 387–392.
- [87] D. Sapra and A. D. Pimentel, "Constrained evolutionary piecemeal training to design convolutional neural networks," in *Proc. Int. Conf. Ind., Eng. Appl. Appl. Intell. Syst.* New York, NY, USA: Springer, 2020, pp. 709–721.
- [88] F. Martínez-Álvarez *et al.*, "Coronavirus optimization algorithm: A bioinspired Metaheuristic based on the COVID-19 propagation model," 2020, *arXiv:2003.13633*. [Online]. Available: <http://arxiv.org/abs/2003.13633>
- [89] E. Rapaport, O. Shriki, and R. Puzis, "EEGNAS: Neural architecture search for electroencephalography data analysis and decoding," in *Proc. Int. Workshop Hum. Brain Artif. Intell.* New York, NY, USA: Springer, 2019, pp. 3–20.
- [90] L. Peng, S. Liu, R. Liu, and L. Wang, "Effective long short-term memory with differential evolution algorithm for electricity price prediction," *Energy*, vol. 162, pp. 1301–1314, Nov. 2018.
- [91] B. Dahal and J. Zhan, "Effective mutation and recombination for evolving convolutional networks," in *Proc. 3rd Int. Conf. Appl. Intell. Syst.*, Jan. 2020, pp. 1–6.

- [92] 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.
- [93] J. Ren, Z. Li, J. Yang, N. Xu, T. Yang, and D. J. Foran, "EIGEN: Ecologically-inspired GENetic approach for neural network structure searching from scratch," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 9059–9068.
- [94] C.-C. Chung, W.-T. Lin, R. Zhang, K.-W. Liang, and P.-C. Chang, "Emotion estimation by joint facial expression and speech tonality using evolutionary deep learning structures," in *Proc. IEEE 8th Global Conf. Consum. Electron. (GCCE)*, Oct. 2019, pp. 221–224.
- [95] A. Almalaq and J. J. Zhang, "Evolutionary deep learning-based energy consumption prediction for buildings," *IEEE Access*, vol. 7, pp. 1520–1531, 2018.
- [96] G. J. van Wyk and A. S. Bosman, "Evolutionary neural architecture search for image restoration," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2019, pp. 1–8.
- [97] B. Wang, Y. Sun, B. Xue, and M. Zhang, "Evolving deep convolutional neural networks by variable-length particle swarm optimization for image classification," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jul. 2018, pp. 1–8.
- [98] M. P. Wang, "Evolving knowledge and structure through evolution-based neural architecture search," M.S. thesis, Dept. Comput. Sci., NTNU, Trondheim, Norway, 2019.
- [99] F. Assunção, N. Lourenço, P. Machado, and B. Ribeiro, "Fast denser: Efficient deep neuroevolution," in *Proc. Eur. Conf. Genetic Program.* New York, NY, USA: Springer, 2019, pp. 197–212.
- [100] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical representations for efficient architecture search," 2017, *arXiv:1711.00436*. [Online]. Available: <http://arxiv.org/abs/1711.00436>
- [101] Z. Lu *et al.*, "Multi-objective evolutionary design of deep convolutional neural networks for image classification," 2019, *arXiv:1912.01369*. [Online]. Available: <http://arxiv.org/abs/1912.01369>
- [102] P. R. Lorenzo, J. Nalepa, M. Kawulok, L. S. Ramos, and J. R. Pastor, "Particle swarm optimization for hyper-parameter selection in deep neural networks," in *Proc. Genetic Evol. Comput. Conf.*, Jul. 2017, pp. 481–488.
- [103] V. Passricha and R. K. Aggarwal, "PSO-based optimized CNN for Hindi ASR," *Int. J. Speech Technol.*, vol. 22, no. 4, pp. 1123–1133, Dec. 2019.
- [104] H. Zhang, Y. Jin, R. Cheng, and K. Hao, "Sampled training and node inheritance for fast evolutionary neural architecture search," 2020, *arXiv:2003.11613*. [Online]. Available: <http://arxiv.org/abs/2003.11613>
- [105] T. Elsken, J. H. Metzen, and F. Hutter, "Efficient multi-objective neural architecture search via Lamarckian evolution," 2018, *arXiv:1804.09081*. [Online]. Available: <http://arxiv.org/abs/1804.09081>
- [106] E. Byla and W. Pang, "Deepswarm: Optimising convolutional neural networks using swarm intelligence," in *Proc. UK Workshop Comput. Intell.* New York, NY, USA: Springer, 2019, pp. 119–130.
- [107] M. Baldeon-Calisto and S. K. Lai-Yuen, "AdaResU-Net: Multiobjective adaptive convolutional neural network for medical image segmentation," *Neurocomputing*, vol. 392, pp. 325–340, Jun. 2020.
- [108] Z. Chen, Y. Zhou, and Z. Huang, "Auto-creation of effective neural network architecture by evolutionary algorithm and ResNet for image classification," in *Proc. IEEE Int. Conf. Syst., Man Cybern. (SMC)*, Oct. 2019, pp. 3895–3900.
- [109] B. Wang, Y. Sun, B. Xue, and M. Zhang, "Evolving deep neural networks by multi-objective particle swarm optimization for image classification," in *Proc. Genetic Evol. Comput. Conf.*, Jul. 2019, pp. 490–498.
- [110] B. Fielding and L. Zhang, "Evolving image classification architectures with enhanced particle swarm optimisation," *IEEE Access*, vol. 6, pp. 68560–68575, 2018.
- [111] 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.* New York, NY, USA: Springer, 2019, pp. 17–26.
- [112] M. Suganuma, S. Shirakawa, and T. Nagao, "A genetic programming approach to designing convolutional neural network architectures," in *Proc. Genetic Evol. Comput. Conf.*, Jul. 2017, pp. 497–504.
- [113] M. Loni, A. Majd, A. Loni, M. Daneshmand, M. Sjödin, and E. Troubitsyna, "Designing compact convolutional neural network for embedded stereo vision systems," in *Proc. IEEE 12th Int. Symp. Embedded Multicore/Many-Core Syst. Chip (MCSoc)*, Sep. 2018, pp. 244–251.
- [114] D. Song, C. Xu, X. Jia, Y. Chen, C. Xu, and Y. Wang, "Efficient residual dense block search for image super-resolution," 2019, *arXiv:1909.11409*. [Online]. Available: <http://arxiv.org/abs/1909.11409>
- [115] M. Loni, S. Sinaei, A. Zoljodi, M. Daneshmand, and M. Sjödin, "Deep-Maker: A multi-objective optimization framework for deep neural networks in embedded systems," *Microprocessors Microsyst.*, vol. 73, Mar. 2020, Art. no. 102989.
- [116] 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, Mar. 2020.
- [117] R. Miikkulainen *et al.*, "Evolving deep neural networks," in *Artificial Intelligence in the Age of Neural Networks and Brain Computing*. Amsterdam, The Netherlands: Elsevier, 2019, pp. 293–312.
- [118] 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.*, Mar. 2020, pp. 181–189.
- [119] E. Miah, S. Abolghasem Mirroshandel, and A. Nasr, "Genetic neural architecture search for automatic assessment of human sperm images," 2019, *arXiv:1909.09432*. [Online]. Available: <http://arxiv.org/abs/1909.09432>
- [120] 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)*, Oct. 2018, pp. 3438–3442.
- [121] X. Chu, B. Zhang, R. Xu, and H. Ma, "Multi-objective reinforced evolution in mobile neural architecture search," 2019, *arXiv:1901.01074*. [Online]. Available: <http://arxiv.org/abs/1901.01074>
- [122] Y. Chen *et al.*, "Reinforced evolutionary neural architecture search," 2018, *arXiv:1808.00193*. [Online]. Available: <http://arxiv.org/abs/1808.00193>
- [123] Y. Chen, T. Pan, C. He, and R. Cheng, "Efficient evolutionary deep neural architecture search (NAS) by noisy network morphism mutation," in *Proc. Int. Conf. Bio-Inspired Comput., Theories Appl.* New York, NY, USA: Springer, 2019, pp. 497–508.
- [124] Z. Fan, J. Wei, G. Zhu, J. Mo, and W. Li, "Evolutionary neural architecture search for retinal vessel segmentation," 2020, *arXiv:2001.06678*. [Online]. Available: <http://arxiv.org/abs/2001.06678>
- [125] K. Chen and W. Pang, "ImmuneNetNAS: An immune-network approach for searching convolutional neural network architectures," 2020, *arXiv:2002.12704*. [Online]. Available: <http://arxiv.org/abs/2002.12704>
- [126] P. Liu, M. D. El Basha, Y. Li, Y. Xiao, P. C. Sanelli, and R. Fang, "Deep evolutionary networks with expedited genetic algorithms for medical image denoising," *Med. Image Anal.*, vol. 54, pp. 306–315, May 2019.
- [127] M. Wistuba, "Deep learning architecture search by neuro-cell-based evolution with function-preserving mutations," in *Proc. Joint Eur. Conf. Mach. Learn. Knowl. Discovery Databases*. New York, NY, USA: Springer, 2018, pp. 243–258.
- [128] O. Kramer, "Evolution of convolutional highway networks," in *Proc. Int. Conf. Appl. Evol. Comput.* New York, NY, USA: Springer, 2018, pp. 395–404.
- [129] X. Chu, B. Zhang, H. Ma, R. Xu, and Q. Li, "Fast, accurate and lightweight super-resolution with neural architecture search," 2019, *arXiv:1901.07261*. [Online]. Available: <http://arxiv.org/abs/1901.07261>
- [130] Y. Chen *et al.*, "Joint neural architecture search and quantization," 2018, *arXiv:1811.09426*. [Online]. Available: <http://arxiv.org/abs/1811.09426>
- [131] B. Wang, B. Xue, and M. Zhang, "Particle swarm optimisation for evolving deep neural networks for image classification by evolving and stacking transferable blocks," 2019, *arXiv:1907.12659*. [Online]. Available: <http://arxiv.org/abs/1907.12659>
- [132] C. Saltori, S. Roy, N. Sebe, and G. Iacca, "Regularized evolutionary algorithm for dynamic neural topology search," in *Proc. Int. Conf. Image Anal. Process.* New York, NY, USA: Springer, 2019, pp. 219–230.
- [133] T. Wu, J. Shi, D. Zhou, Y. Lei, and M. Gong, "A multi-objective particle swarm optimization for neural networks pruning," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jun. 2019, pp. 570–577.
- [134] Y. Hu, S. Sun, J. Li, X. Wang, and Q. Gu, "A novel channel pruning method for deep neural network compression," 2018, *arXiv:1805.11394*. [Online]. Available: <http://arxiv.org/abs/1805.11394>
- [135] P. R. Lorenzo and J. Nalepa, "Memetic evolution of deep neural networks," in *Proc. Genetic Evol. Comput. Conf.*, Jul. 2018, pp. 505–512.
- [136] A. ElSaid, F. El Jamiy, J. Higgins, B. Wild, and T. Desell, "Optimizing long short-term memory recurrent neural networks using ant colony optimization to predict turbine engine vibration," *Appl. Soft Comput.*, vol. 73, pp. 969–991, Dec. 2018.



- [137] Z. Yang *et al.*, "CARS: Continuous evolution for efficient neural architecture search," 2019, *arXiv:1909.04977*. [Online]. Available: <http://arxiv.org/abs/1909.04977>
- [138] T. Desell, S. Clachar, J. Higgins, and B. Wild, "Evolving deep recurrent neural networks using ant colony optimization," in *Proc. Eur. Conf. Evol. Comput. Combinat. Optim.* New York, NY, USA: Springer, 2015, pp. 86–98.
- [139] Z. Guo *et al.*, "Single path one-shot neural architecture search with uniform sampling," 2019, *arXiv:1904.00420*. [Online]. Available: <http://arxiv.org/abs/1904.00420>
- [140] T. Shinozaki and S. Watanabe, "Structure discovery of deep neural network based on evolutionary algorithms," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Apr. 2015, pp. 4979–4983.
- [141] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 1–9.
- [142] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter, "NAS-bench-101: Towards reproducible neural architecture search," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 7105–7114.
- [143] X. Dong and Y. Yang, "Nas-bench-201: Extending the scope of reproducible neural architecture search," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2020, pp. 1–16. [Online]. Available: <https://openreview.net/forum?id=HJxyZkBKDr>
- [144] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 8697–8710.
- [145] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," 2018, *arXiv:1806.09055*. [Online]. Available: <http://arxiv.org/abs/1806.09055>
- [146] 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)*, Jun. 2019, pp. 546–553.
- [147] H. Kitano, "Designing neural networks using genetic algorithms with graph generation system," *Complex Syst.*, vol. 4, no. 4, pp. 461–476, 1990.
- [148] D. Jones, A. Schroeder, and G. Nitschke, "Evolutionary deep learning to identify galaxies in the zone of avoidance," 2019, *arXiv:1903.07461*. [Online]. Available: <http://arxiv.org/abs/1903.07461>
- [149] K. Ho, A. Gilbert, H. Jin, and J. Collomosse, "Neural architecture search for deep image prior," 2020, *arXiv:2001.04776*. [Online]. Available: <http://arxiv.org/abs/2001.04776>
- [150] J. D. Lamos-Sweeney, "Deep learning using genetic algorithms," Ph.D. dissertation, Dept. Comput. Sci., Rochester Inst. Technol., Rochester, NY, USA, 2012.
- [151] A. ElSaid, S. Benson, S. Patwardhan, D. Stadem, and T. Desell, "Evolving recurrent neural networks for time series data prediction of coal plant parameters," in *Proc. Int. Conf. Appl. Evol. Comput. (EvoStar)*. New York, NY, USA: Springer, 2019, pp. 488–503.
- [152] Y. Sun, G. G. Yen, and Z. Yi, "Evolving unsupervised deep neural networks for learning meaningful representations," *IEEE Trans. Evol. Comput.*, vol. 23, no. 1, pp. 89–103, Feb. 2018.
- [153] A. Behjat, S. Chidambaram, and S. Chowdhury, "Adaptive genomic evolution of neural network topologies (AGENT) for state-to-action mapping in autonomous agents," in *Proc. Int. Conf. Robot. Autom. (ICRA)*, May 2019, pp. 9638–9644.
- [154] B. 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)*, Jul. 2018, pp. 1–6.
- [155] S. Bianco, M. Buzzelli, G. Ciocca, and R. Schettini, "Neural architecture search for image saliency fusion," *Inf. Fusion*, vol. 57, pp. 89–101, May 2020.
- [156] P. J. Angeline, G. M. Saunders, and J. B. Pollack, "An evolutionary algorithm that constructs recurrent neural networks," *IEEE Trans. Neural Netw.*, vol. 5, no. 1, pp. 54–65, Jan. 1994.
- [157] A. Rawal and R. Miikkulainen, "From nodes to networks: Evolving recurrent neural networks," 2018, *arXiv:1803.04439*. [Online]. Available: <http://arxiv.org/abs/1803.04439>
- [158] M. Märtens and D. Izzo, "Neural network architecture search with differentiable Cartesian genetic programming for regression," in *Proc. Genetic Evol. Comput. Conf. Companion*, Jul. 2019, pp. 181–182.
- [159] M. Neshat *et al.*, "An evolutionary deep learning method for short-term wind speed prediction: A case study of the lillgrund offshore wind farm," 2020, *arXiv:2002.09106*. [Online]. Available: <http://arxiv.org/abs/2002.09106>
- [160] T. Tanaka, T. Shinozaki, S. Watanabe, and T. Hori, "Evolution strategy based neural network optimization and LSTM language model for robust speech recognition," *Cit. on*, vol. 130, pp. 1–4, Sep. 2016.
- [161] A. Camero, J. Toutouh, J. Ferrer, and E. Alba, "Waste generation prediction under uncertainty in smart cities through deep neuroevolution," *Revista Facultad de Ingeniería Universidad de Antioquia*, vol. 93, no. 93, pp. 128–138, Aug. 2019.
- [162] I. Loshchilov and F. Hutter, "CMA-ES for hyperparameter optimization of deep neural networks," 2016, *arXiv:1604.07269*. [Online]. Available: <http://arxiv.org/abs/1604.07269>
- [163] A. A. ElSaid, A. G. Ororbia, and T. J. Desell, "The ant swarm neuro-evolution procedure for optimizing recurrent networks," 2019, *arXiv:1909.11849*. [Online]. Available: <http://arxiv.org/abs/1909.11849>
- [164] A. ElSaid, F. E. Jamiy, J. Higgins, B. Wild, and T. Desell, "Using ant colony optimization to optimize long short-term memory recurrent neural networks," in *Proc. Genetic Evol. Comput. Conf.*, Jul. 2018, pp. 13–20.
- [165] Z. Gao, Y. Li, Y. Yang, X. Wang, N. Dong, and H.-D. Chiang, "A GPSO-optimized convolutional neural networks for EEG-based emotion recognition," *Neurocomputing*, vol. 380, pp. 225–235, Mar. 2020.
- [166] C. Zhang, H. Shao, and Y. Li, "Particle swarm optimisation for evolving artificial neural network," in *Proc. Conf. IEEE Int. Conf. Syst., Man Cybern. Cybern. Evolving Syst., Hum., Org., Complex Interact. (SMC)*, vol. 4, Oct. 2000, pp. 2487–2490.
- [167] B. P. Evans, H. Al-Sahaf, B. Xue, and M. Zhang, "Genetic programming and gradient descent: A memetic approach to binary image classification," 2019, *arXiv:1909.13030*. [Online]. Available: <http://arxiv.org/abs/1909.13030>
- [168] I. De Falco, G. De Pietro, A. Della Cioppa, G. Sannino, U. Scafuri, and E. Tarantino, "Evolution-based configuration optimization of a deep neural network for the classification of obstructive sleep apnea episodes," *Future Gener. Comput. Syst.*, vol. 98, pp. 377–391, Sep. 2019.
- [169] J. Huang, W. Sun, and L. Huang, "Deep neural networks compression learning based on multiobjective evolutionary algorithms," *Neurocomputing*, vol. 378, pp. 260–269, Feb. 2020.
- [170] Z. Lu *et al.*, "NSGA-Net: Neural architecture search using multi-objective genetic algorithm," in *Proc. Genetic Evol. Comput. Conf.*, Jul. 2019, pp. 419–427.
- [171] H. Zhu and Y. Jin, "Real-time federated evolutionary neural architecture search," 2020, *arXiv:2003.02793*. [Online]. Available: <http://arxiv.org/abs/2003.02793>
- [172] J. Dong, A. Cheng, D. Juan, W. Wei, and M. Sun, "PPP-Net: Platform-aware progressive search for pareto-optimal neural architectures," in *Proc. 6th Int. Conf. Learn. Represent. (ICLR)*, Vancouver, BC, Canada, Apr./May 2018, pp. 1–4. [Online]. Available: <https://openreview.net/forum?id=BINT3TAIM>
- [173] D. Hossain and G. Capi, "Multiobjective evolution of deep learning parameters for robot manipulator object recognition and grasping," *Adv. Robot.*, vol. 32, no. 20, pp. 1090–1101, Oct. 2018.
- [174] J. Bayer, D. Wierstra, J. Togelius, and J. Schmidhuber, "Evolving memory cell structures for sequence learning," in *Proc. Int. Conf. Artif. Neural Netw.* New York, NY, USA: Springer, 2009, pp. 755–764.
- [175] S. Roy and N. Chakraborti, "Development of an evolutionary deep neural net for materials research," in *Proc. 149th Annu. Meeting, Exhib. Supplemental (TMS)*. New York, NY, USA: Springer, 2020, pp. 817–828.
- [176] J. Liang, E. Meyerson, B. Hodjat, D. Fink, K. Mutch, and R. Miikkulainen, "Evolutionary neural AutoML for deep learning," in *Proc. Genetic Evol. Comput. Conf.*, Jul. 2019, pp. 401–409.
- [177] J. An, H. Xiong, J. Ma, J. Luo, and J. Huan, "StyleNAS: An empirical study of neural architecture search to uncover surprisingly fast end-to-end universal style transfer networks," 2019, *arXiv:1906.02470*. [Online]. Available: <http://arxiv.org/abs/1906.02470>
- [178] M. Javaheripi, M. Samragh, T. Javidi, and F. Koushanfar, "GeneCAI: Genetic evolution for acquiring compact AI," 2020, *arXiv:2004.04249*. [Online]. Available: <http://arxiv.org/abs/2004.04249>
- [179] K. Deb, *Multi-Objective Optimization Using Evolutionary Algorithms*, vol. 16. Hoboken, NJ, USA: Wiley, 2001.
- [180] K. Maziarz, M. Tan, A. Khorlin, M. Georgiev, and A. Gesmundo, "Evolutionary-neural hybrid agents for architecture search," *CoRR*, vol. abs/1811.09828, pp. 1–11, Nov. 2018. [Online]. Available: <http://arxiv.org/abs/1811.09828>
- [181] T. Chen, I. Goodfellow, and J. Shlens, "Net2Net: Accelerating learning via knowledge transfer," 2015, *arXiv:1511.05641*. [Online]. Available: <http://arxiv.org/abs/1511.05641>
- [182] T. Wei, C. Wang, Y. Rui, and C. W. Chen, "Network morphism," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 564–572.



- [183] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press, 1998.
- [184] K. Deb and R. B. Agrawal, "Simulated binary crossover for continuous search space," *Complex Syst.*, vol. 9, no. 2, pp. 115–148, 1995.
- [185] S. Kotyan and D. Vasconcellos Vargas, "Evolving robust neural architectures to defend from adversarial attacks," 2019, *arXiv:1906.11667*. [Online]. Available: <http://arxiv.org/abs/1906.11667>
- [186] K. Deb, "Multi-objective optimization," in *Search Methodologies*. New York, NY, USA: Springer, 2014, pp. 403–449.
- [187] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002.
- [188] Q. Zhang and H. Li, "MOEA/D: A multiobjective evolutionary algorithm based on decomposition," *IEEE Trans. Evol. Comput.*, vol. 11, no. 6, pp. 712–731, Dec. 2007.
- [189] K. Deb and H. Jain, "An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: Solving problems with box constraints," *IEEE Trans. Evol. Comput.*, vol. 18, no. 4, pp. 577–601, Apr. 2013.
- [190] C. A. C. Coello, G. T. Pulido, and M. S. Lechuga, "Handling multiple objectives with particle swarm optimization," *IEEE Trans. Evol. Comput.*, vol. 8, no. 3, pp. 256–279, Jun. 2004.
- [191] M. R. Sierra and C. A. C. Coello, "Improving pso-based multi-objective optimization using crowding, mutation and  $\epsilon$ -dominance," in *Proc. Int. Conf. Evol. Multi-Criterion Optim.* New York, NY, USA: Springer, 2005, pp. 505–519.
- [192] 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. Pacific Rim Int. Conf. Artif. Intell.* New York, NY, USA: Springer, 2019, pp. 650–663.
- [193] X. Zhou, A. K. Qin, M. Gong, and K. C. Tan, "A survey on evolutionary construction of deep neural networks," *IEEE Trans. Evol. Comput.*, early access, May 13, 2021, doi: [10.1109/TEVC.2021.3079985](https://doi.org/10.1109/TEVC.2021.3079985).
- [194] D. R. So, C. Liang, and Q. V. Le, "The evolved transformer," 2019, *arXiv:1901.11117*. [Online]. Available: <http://arxiv.org/abs/1901.11117>
- [195] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2009, pp. 248–255.
- [196] F. Assunção, N. Lourenço, P. Machado, and B. Ribeiro, "Evolving the topology of large scale deep neural networks," in *Proc. Eur. Conf. Genetic Program.* New York, NY, USA: Springer, 2018, pp. 19–34.
- [197] F. Assunção, N. Lourenço, P. Machado, and B. Ribeiro, "DENSER: Deep evolutionary network structured representation," *Genetic Program. Evolvable Mach.*, vol. 20, no. 1, pp. 5–35, Mar. 2019.
- [198] J. Liu, M. Gong, Q. Miao, X. Wang, and H. Li, "Structure learning for deep neural networks based on multiobjective optimization," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 6, pp. 2450–2463, Jun. 2017.
- [199] G. Bender, P. Kindermans, B. Zoph, V. Vasudevan, and Q. V. Le, "Understanding and simplifying one-shot architecture search," in *Proc. 35th Int. Conf. Mach. Learn. (ICML)*, J. G. Dy and A. Krause, Eds., Stockholm, Sweden, Jul. 2018, pp. 549–558. [Online]. Available: <http://proceedings.mlr.press/v80/bender18a.html>
- [200] X. Chu, B. Zhang, R. Xu, and J. Li, "FairNAS: Rethinking evaluation fairness of weight sharing neural architecture search," 2019, *arXiv:1907.01845*. [Online]. Available: <http://arxiv.org/abs/1907.01845>
- [201] P. Colangelo, O. Segal, A. Speicher, and M. Margala, "Artificial neural network and accelerator co-design using evolutionary algorithms," in *Proc. IEEE High Perform. Extreme Comput. Conf. (HPEC)*, Sep. 2019, pp. 1–8.
- [202] P. Colangelo, O. Segal, A. Speicher, and M. Margala, "Evolutionary cell aided design for neural network architectures," 2019, *arXiv:1903.02130*. [Online]. Available: <http://arxiv.org/abs/1903.02130>
- [203] 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.
- [204] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [205] T. K. Ho, "Random decision forests," in *Proc. 3rd Int. Conf. Document Anal. Recognit.*, vol. 1, Aug. 1995, pp. 278–282.
- [206] S. S. Tirumala, "Evolving deep neural networks using coevolutionary algorithms with multi-population strategy," *Neural Comput. Appl.*, vol. 32, no. 16, pp. 1–14, 2020.
- [207] F. Assunção, N. Lourenço, P. Machado, and B. Ribeiro, "Fast-DENSER++: Evolving fully-trained deep artificial neural networks," 2019, *arXiv:1905.02969*. [Online]. Available: <http://arxiv.org/abs/1905.02969>
- [208] F. Assunção, N. Lourenço, B. Ribeiro, and P. Machado, "Incremental evolution and development of deep artificial neural networks," in *Proc. 23rd Eur. Conf. Genetic Program. (EuroGP)*, Seville, Spain, New York, NY, USA: Springer, Jul. 2020, pp. 35–51.
- [209] C. Wei, C. Niu, Y. Tang, Y. Wang, H. Hu, and J. Liang, "NPENAS: Neural predictor guided evolution for neural architecture search," 2020, *arXiv:2003.12857*. [Online]. Available: <http://arxiv.org/abs/2003.12857>
- [210] J. Song, Y. Jin, Y. Li, and C. Lang, "Learning structural similarity with evolutionary-GAN: A new face de-identification method," in *Proc. 6th Int. Conf. Behav., Econ. Socio-Cultural Comput. (BESC)*, Oct. 2019, pp. 1–6.
- [211] L. M. Zhang, "Genetic deep neural networks using different activation functions for financial data mining," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Oct. 2015, pp. 2849–2851.
- [212] J. Liang, E. Meyerson, and R. Miikkulainen, "Evolutionary architecture search for deep multitask networks," in *Proc. Genetic Evol. Comput. Conf.*, Jul. 2018, pp. 466–473.
- [213] S. Fujino, T. Hatanaka, N. Mori, and K. Matsumoto, "The evolutionary deep learning based on deep convolutional neural network for the anime storyboard recognition," in *Proc. Int. Symp. Distrib. Comput. Artif. Intell.* New York, NY, USA: Springer, 2017, pp. 278–285.
- [214] J.-D. Dong, A.-C. Cheng, D.-C. Juan, W. Wei, and M. Sun, "DPP-Net: Device-aware progressive search for Pareto-optimal neural architectures," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 517–531.
- [215] A.-C. Cheng *et al.*, "Searching toward Pareto-optimal device-aware neural architectures," in *Proc. Int. Conf. Comput.-Aided Design*, Nov. 2018, pp. 1–7.
- [216] T. DeVries and G. W. Taylor, "Improved regularization of convolutional neural networks with cutout," 2017, *arXiv:1708.04552*. [Online]. Available: <http://arxiv.org/abs/1708.04552>
- [217] K. Yu, C. Sciuto, M. Jaggi, C. Musat, and M. Salzmann, "Evaluating the search phase of neural architecture search," 2019, *arXiv:1902.08142*. [Online]. Available: <http://arxiv.org/abs/1902.08142>
- [218] P. Liaschynskiy and P. Liaschynskiy, "Grid search, random search, genetic algorithm: A big comparison for NAS," 2019, *arXiv:1912.06059*. [Online]. Available: <http://arxiv.org/abs/1912.06059>
- [219] J. Wang, Y. Zhou, Y. Wang, J. Zhang, C. L. P. Chen, and Z. Zheng, "Multiobjective vehicle routing problems with simultaneous delivery and pickup and time windows: Formulation, instances, and algorithms," *IEEE Trans. Cybern.*, vol. 46, no. 3, pp. 582–594, Mar. 2015.
- [220] Y. Zhou and J. Wang, "A local search-based multiobjective optimization algorithm for multiobjective vehicle routing problem with time windows," *IEEE Syst. J.*, vol. 9, no. 3, pp. 1100–1113, Sep. 2015.
- [221] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.
- [222] P. Chrabaszcz, I. Loshchilov, and F. Hutter, "A downsampled variant of ImageNet as an alternative to the CIFAR datasets," 2017, *arXiv:1707.08819*. [Online]. Available: <http://arxiv.org/abs/1707.08819>
- [223] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Proc. Eur. Conf. Comput. Vis.* New York, NY, USA: Springer, 2014, pp. 818–833.
- [224] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, vol. 1. Cambridge, MA, USA: MIT Press, 1992.
- [225] F. Cheng, J. Yu, and H. Xiong, "Facial expression recognition in JAFFE dataset based on Gaussian process classification," *IEEE Trans. Neural Netw.*, vol. 21, no. 10, pp. 1685–1690, Oct. 2010.
- [226] J. Triesch and C. von der Malsburg, "Robust classification of hand postures against complex backgrounds," in *Proc. 2nd Int. Conf. Autom. Face Gesture Recognit.*, 1996, pp. 170–175.
- [227] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," 2014, *arXiv:1412.3555*. [Online]. Available: <http://arxiv.org/abs/1412.3555>
- [228] G.-B. Zhou, J. Wu, C.-L. Zhang, and Z.-H. Zhou, "Minimal gated unit for recurrent neural networks," *Int. J. Automat. Comput.*, vol. 13, no. 3, pp. 226–234, 2016.
- [229] J. Collins, J. Sohl-Dickstein, and D. Sussillo, "Capacity and trainability in recurrent neural networks," 2016, *arXiv:1611.09913*. [Online]. Available: <http://arxiv.org/abs/1611.09913>