A Survey on Evolutionary Construction of Deep Neural Networks

Xun Zhou, A. K. Qin, *Senior Member, IEEE*, Maoguo Gong, *Senior Member, IEEE*, and Kay Chen Tan, *Fellow, IEEE*

***Abstract*—Automated construction of deep neural networks (DNNs) has become a research hot spot nowadays because DNN's performance is heavily influenced by its architecture and parameters, which are highly task-dependent, but it is notoriously difficult to find the most appropriate DNN in terms of architecture and parameters to best solve a given task. In this work, we provide an insight into the automated DNN construction process by formulating it into a multilevel multiobjective large-scale optimization problem with constraints, where the nonconvex, nondifferentiable, and black-box nature of this problem make evolutionary algorithms (EAs) to stand out as a promising solver. Then, we give a systematical review of existing evolutionary DNN construction techniques from different aspects of this optimization problem and analyze the pros and cons of using EA-based methods in each aspect. This work aims to help DNN researchers to better understand why, where, and how to utilize EAs for automated DNN construction and meanwhile, help EA researchers to better understand the task of automated DNN construction so that they may focus more on EA-favored optimization scenarios to devise more effective techniques.**

***Index Terms*—Automated design of DNNs, deep neural networks, evolutionary algorithms, optimization.**

I. Introduction

DEEP neural networks (DNNs) are one of the most powerful machine learning techniques nowadays, deriving the reviving and boom of artificial intelligence in recent years [1]–[3]. They are characterized by sophisticated task-oriented models (in the form of networks), which allow the most effective feature representation to be learned in a task-driven manner from the data of various types, such as images, texts, and time series [4]. The concept of DNNs appeared in the 1970s. After that, the development of DNNs continued but did not attract much attention. The starting point for the boom of DNNs occurred in 2012 when a specific DNN model named AlexNet, armed with the high computing horsepower of graphics processing units (GPUs), achieved the record-breaking classification performance on the ImageNet dataset [5]. Since then, DNNs have received ever-increasing attention, leading to the emergence of a new era in machine learning, namely, deep learning.

DNNs have different types of architectures suitable for dealing with different types of data. For example, convolutional neural networks (CNNs) are apt at learning features from the data with certain local structures such as images [4]. Recurrent neural networks (RNNs) are good at learning the temporal behavior from sequence data such as time series [4]. For the same type of DNNs, there also exist various kinds of models. For example, AlexNet [5], VGG [6], Inception [7], and ResNet [8] are popular CNN models while LSTM and GRU are commonly used RNN models [9]. It is well known that DNN's performance depends on both model architecture and model parameters. However, in practice, it poses great challenges to find the most suitable DNN model in terms of architecture and parameters to best solve a given task because it corresponds to solve a highly complex large-scale optimization problem of nonconvex and black-box nature [10].

The traditional way to address this issue assumes the model architecture is manually specified and leaves the task of learning model parameters to be solved by using a selected and configured model learner. This often leads to the suboptimal performance due to a lack of sufficient expert knowledge and human labor to make the best choice from a vast number of possible model architectures, model learners, and their associated parameters. Recent years have seen exponentially growing efforts in both academia and the industry on studying automated DNN construction techniques that aim to automatically determine the best-performing model architecture and parameters for a given task [10]–[15]. Such techniques commonly perform search-based optimization about model architecture, model parameters, and model learners, where optimization of model parameters is via a model parameters learner, such as stochastic gradient descent (SGD) and evolutionary algorithms (EAs) and nested into optimization of the model architecture via a model architecture learner, such as reinforcement learning (RL) [11], [12], EAs [13]–[15] and SGD [10].

Manuscript received November 1, 2020; revised February 13, 2021 and April 30, 2021; accepted April 30, 2021. Date of publication May 13, 2021; date of current version October 1, 2021. This work was supported in part by the National Key Research and Development Project, Ministry of Science and Technology, China, under Grant 2018AAA0101301; in part by the National Natural Science Foundation of China under Grant 61876162; in part by the Research Grants Council of the Hong Kong SAR under Grant PolyU11202418 and Grant PolyU11209219; and in part by the Australian Research Council (ARC) under Grant LP180100114 and Grant DP200102611. *(Corresponding authors: Maoguo Gong; Kay Chen Tan.)*

Xun Zhou is with the Department of Computer Science, City University of Hong Kong, Hong Kong (e-mail: xunzhou6-c@my.cityu.edu.hk).

A. K. Qin is with the Department of Computer Science and Software Engineering, Swinburne University of Technology, Hawthorn, VIC 3122, Australia (e-mail: kqin@swin.edu.au).

Maoguo Gong is with the International Research Center for Intelligent Perception and Computation, Xidian University, Xi'an 710071, China (e-mail: gong@ieee.org).

Kay Chen Tan is with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong, and also with Shenzhen Research Institute, The Hong Kong Polytechnic University, Shenzhen, China (e-mail: kaychen.tan@polyu.edu.hk).

This article has supplementary material provided by the authors and color versions of one or more figures available at https://doi.org/10.1109/TEVC.2021.3079985.

Digital Object Identifier 10.1109/TEVC.2021.3079985

EAs [16] are a family of search-based optimization techniques that evolve a population of candidate solutions via nature-inspired operators to search the optima, which have shown great competence in tackling challenging optimization problems. Due to EA's strong capability in exploring complicated search spaces [13], high flexibility in integrating problem-solving knowledge [17], and easy-to-parallelize nature [18]–[20], there is a fast-growing interest in applying EAs in automated DNN construction, namely, evolutionary construction of DNNs. Specifically, EAs have been applied for learning model parameters [21]–[47], designing model architectures [13], [15], [17], [18], [48]–[68], [68]–[126], and optimizing model learners [19], [20], [127]–[134].

There exist quite a few survey papers on relevant topics, focusing on a specific type of optimization problems such as model architecture optimization (commonly known as "neural architecture search") [135]–[137], a specific type of DNN models such as CNNs [138], or a specific type of optimization techniques, such as RL [139] and EAs [140], [141]. Different from the existing survey papers, this work provides a comprehensive and systematic review of using EAs to address various optimization problems involved in the automated DNN construction process based on an insight into this process from the perspective of optimization. It also includes discussions on the pros and cons of EA-based techniques compared to non-EA-based ones in different optimization scenarios. We aim to help DNN researchers to better understand why, where, and how to use EAs for automated DNN construction and also help EA researchers to better understand the task of automated DNN construction so that they may focus more on EA-favored optimization scenarios to devise more effective techniques.

Our major contributions include the following.

1) Define and analyze various optimization problems involved in the automated DNN construction process, revealing the motivations of using EAs as the solver.
2) Provide the taxonomy and survey of existing EA-based techniques in different optimization scenarios and discuss the pros and cons of EA-based techniques compared to non-EA-based ones in every scenario.
3) Summarize applications, challenges, and trends regarding the evolutionary DNN construction, where the publicly available datasets and codes used in studies are reported in a supplementary document, which, to the best of our knowledge, was never provided in existing survey works.

The remainder of this article is organized as follows. Section II provides an insight into automated DNN construction from the perspective of optimization. Section III introduces several fundamentals of evolutionary DNN construction. Sections IV–VII review evolutionary DNN construction techniques in different optimization scenarios, including model parameter optimization, model architecture optimization, model learner optimization, and miscellaneous. Section VIII summarizes applications, challenges, and trends about evolutionary DNN construction. Finally, Section IX concludes the article.

II. Insights Into Automated DNN Construction

Automated DNN construction aims at finding the most appropriate DNN model architecture and parameters to best solve a given task, which typically requires dealing with the following optimization problems.

Model architecture optimization, widely known as the neural architecture search (NAS), pursues the most appropriate architecture to solve a given task by optimizing, with respect to a set of decision variables determined by the representation of the model architecture, one or more objectives that evaluate model performance from different aspects, such as accuracy and time efficiency. This optimization problem typically has a very large discrete search space due to a wide variety of model architectures. Also, it has a black-box nature because it usually lacks explicit mathematical functions that directly formulate the mapping from architecture to performance. Furthermore, the performance of a specific model architecture depends on the associated model parameters. As a result, this optimization problem is inherently bilevel, where architecture optimization is the upper level task, and parameter optimization is the lower level task (which is nested within the upper level task).

Model parameter optimization seeks the best parameters (i.e., connection weights and biases) for a model with a prespecified architecture to best solve a given task by optimizing, with respect to a set of decision variables determined by the representation of model parameters, an objective that evaluates the effectiveness of model parameters based on model's performance on training data, the so-called training loss. This optimization problem typically has a prohibitively large continuous search space due to the extremely large number of model parameters. Also, it is highly nonconvex, featuring a very challenging search landscape full of local optima.

Model (*architecure and parameter*) *learner optimization* targets at finding the most effective learners (intrinsically optimizers) to best solve the above two optimization problems, respectively. It belongs to the problem of optimizing optimizers, i.e., searching the best optimizer [together with its best-calibrated parameters, also known as (a.k.a.) model hyperparameters] to maximize its performance on solving an optimization problem. In fact, most of the existing studies on automated DNN construction did not explicitly consider solving this problem because it will lead to a multilevel optimization problem, which is seldom feasible to be solved in practice due to the prohibitively expensive computational cost.

By considering all the above aspects, we formulate automated DNN construction, provided a training set D^{trn} and a validation set D^{val} (used to measure the model's generalization performance [10], [17]) for the task to be solved, into the following multilevel multiobjective optimization problem defined in (1), shown at the bottom of p. 4. M_a, M_p, M_{la}, and M_{lp} denote the set of decision variables corresponding to the representations of model architecture, model parameters, model architecture learner, and model parameter learner, respectively. The constraints applied to each of them at different optimization level are used to define the feasible search space of decision variables. This optimization problem has four levels with one to four representing the recursively

nested upper to lower levels that correspond to model architecture learner optimization, model architecture optimization, model parameter learner optimization, and model parameter optimization, respectively.

Both model architecture learner optimization and model architecture optimization may have multiple objectives defined on the validation set D^{val}, e.g., validation accuracy (generalization performance estimation) and time efficiency. Model parameter learner optimization and model parameter optimization have one objective defined on the training set D^{trn}, which is typically the training loss.

The solution to this optimization problem is a Pareto optimal set, denoted by $\mathcal{P}(M_a, M_p, M_{la}, M_{lp})$, which contains multiple nondominated solutions.[1] Eventually, one of these nondominated solutions is chosen.

Solving this multilevel multiobjective optimization problem with constraints is seldom feasible in practice even by using the most cutting-edge modern computing facilities due to its large-scale, nonconvex, nondifferentiable (in some cases of model architecture optimization), and black-box nature. Accordingly, existing research works have focused on solving some simplified versions of this problem, e.g., optimizing model parameters M_p given the fixed model architecture M_a and model parameter learner M_{lp}, optimizing model architecture M_a together with model parameters M_p given fixed model learners M_{la} and M_{lp}, and optimizing model learners M_{la} and M_{lp} inside the model parameter and/or architecture optimization process. Furthermore, traditional mathematical optimization techniques such as gradient-descent-based methods become incompetent to handle this problem due to its nonconvex, nondifferentiable, and black-box nature. This fact makes EAs to gain increasing attention and popularity as a promising solver because of their featured capabilities to solve nonconvex, nondifferentiable, and black-box optimization problems, as well as the rapid advance in high performance computing, which mitigates the notorious computational bottleneck of EAs.

This work intends to provide the taxonomy and survey of the existing evolutionary DNN construction techniques applied to model parameter optimization (Section IV), model architecture optimization (Section V), and model learner optimization (Section VI), respectively, which are the simplified versions of the optimization problem defined in (1). Furthermore, Section VII discusses miscellaneous relevant to evolutionary DNN construction, e.g., optimization in terms of objectives and speedup.

III. Overview of Evolutionary DNN Construction

Evolutionary DNN construction studies the use of EAs for automated DNN construction. To help better understand such techniques, we provide an overview of DNNs and EAs followed by a short introduction to the fundamentals and motivations of evolutionary DNN construction techniques.

[1]Considering minimization problems, two candidate solutions x_1 and x_2, x_1 dominates x_2: $O^i(x_1) \leq O^i(x_2), \quad \forall i \in (1, \ldots, m)$ and $O^j(x_1) < O^j(x_2)$, $\exists j \in (1, \ldots, m)$. When x_1 and x_2 cannot dominate each other, they are nondominated solutions.

A. DNNs

DNNs are characterized with the powerful feature representation learning capability due to their sophisticated hierarchical architectures, which can extract a hierarchy of feature sets at different representational levels to address a specific task, such as classification and regression [4].

Different types of DNN architectures have been designed to deal with different types of data. For the same type of architectures, there exist many different types of models. The deep belief network (DBN) is good at data with independent features [142]. CNN is designed for data with local structures, such as images and videos [4]. RNN is apt at sequence data, such as texts and time series [4]. To configure these models for use, the number of different types of layers (e.g., fully connected layers, pooling layers, and convolution layers) and the parameters in each layer (e.g., the number of neurons in a layer, kernel size, stride size, and the padding value in pooling and convolution layers) need to be determined. Also, for some DNN architectures such as ResNet, topological connections across layers need to be determined.

Some DNN architectures are designed for special tasks. For example, stacked autoencoder (SAE) [143] with the symmetrical architecture, i.e., encoder and decoder, is designed for learning feature representations via encoding and decoding, where the architectures of the encoder and the decoder need to be designed. Generative adversarial network (GAN) [144], [206] is designed to produce new plausible data via co-learning two DNNs, i.e., the generator and the discriminator, which may have different architectures.

Besides the model architectre, DNN's performance also depends on model parameters. Typically, model parameters are optimized via gradient-based methods, such as SGD, Moment, Adagrad, and Adam [145]. However, such methods are prone to get stuck into local optima due to the high nonconvexity of the loss function they optimize. Furthermore, their performance is sensitive to their own parameters (a.k.a. model hyperparameters), such as learning rate, batch size, and decay rate.

B. EAs

EAs are the population-based metaheuristic search algorithms inspired by the nature and biology [16]. This article does not distinguish between EAs and swarms intelligence algorithms strictly because they typically follow a general framework composed of initialization I, evaluation E, reproduction R, and selection S modules. Specifically, to solve an optimization problem, a population of candidate solutions $P = \{x_1, \ldots, x_N\}$ is initialized at first. Then, each candidate solution x_i is evaluated to calculate its fitness value f. New candidate solutions (a.k.a. children) are generated via reproduction from the candidate solutions chosen from the population based on fitness elitism (a.k.a. parents). The reproduced candidate solutions are evaluated to obtain their fitness values. Finally, a new population is formed by selecting elicit candidate solutions with higher fitness values from the combination of the reproduced candidate solutions and the old population. This process is repeated until some termination criteria are met.

Then, the individual in the population, which has the best fitness value, is selected as the output. A general framework of EAs is described in Algorithm 1.

Different EAs have been proposed under different natural or biological inspirations, e.g., genetic algorithm (GA) [60], genetic programming (GP) [75], evolution strategy (ES) [146], differential evolution (DE) [92], particle swarm optimization (PSO) [147], ant colony optimization (ACO) [65], and artificial bee colony (ABC) [42], which mainly differ in one or more modules, i.e., I, E, R, and S, in the framework. For initialization, I is applied to initialize candidate solutions. There are two commonly used types of initialization methods, i.e., random initialization [52] and knowledge-based ones [56]. For evaluation, E is applied to calculate the fitness value of the candidate solution [82]. For reproduction, R is applied to generate new candidate solutions based on some candidate solutions chosen from the current population. For example, in GA, recombination is used to exchange some parts of two candidate solutions to produce two new ones [52]. In ES, mutation is used to alter some parts of a candidate solution to a prespecified degree [148]. In PSO, a candidate solution (a.k.a. particle) is changed via some position and velocity updating formula [57]. For selection, S is used to select promising candidate solutions to form the new population, typically based on fitness values. For example, by the tournament selection [72], two candidate solutions are chosen randomly from the population and the one with the higher fitness is added to the new population. There are also other selection methods like roulette wheel selection [59].

Algorithm 1 General Framework of EAs

Input: : The population size N.
Output: : The best-found candidate solution x^*
1: **/*Initialization*/**
2: $P \leftarrow I\{(x_1, x_2, ..., x_N\})$
3: **/*Evaluation*/**
4: $F \leftarrow E(P)$
5: **while** termination criteria are not met **do**
6: **/*Reproduction*/**
7: $C \leftarrow R(\{P, F\})$
8: **/*Evaluation*/**
9: $F_c \leftarrow E(C)$
10: **/*Selection*/**
11: $\{P, F\} \leftarrow S(\{P, F\} \bigcup \{C, F_c\})$
12: **end while**
13: Selecting from P the individual with the best fitness

C. Evolutionary Construction of DNNs

The evolutionary construction of DNNs applies EAs to automatically construct DNNs. Given the insights into the automated DNN construction process, EAs are inherently suitable for solving some optimization problems involved in the process. For example, EAs are well known for the capability to solve black-box nonconvex optimization problems [17]. Also, multiobjective EAs (MOEA) have been intensively studied to deal with multiple conflicting objectives in an effective way [70], [149]. Moreover, EAs are highly parallelization and thus may benefit from the rapid advance in high-performance computing to accelerate computational speed [19], [20].

EAs have a long history of being applied to design and configure neural networks, dated back to the 1990s [150]. With the renaissance of neural networks in the form of deep learning, where the starting point is commonly regarded as the invention of AlexNet in 2012, EAs have been applied to automatically construct DNNs in terms of DNN parameters [151], DNN architecture [59], and DNN learners [131]. Particularly, for DNN architecture design (commonly known as NAS), EAs have demonstrated promising performance [13], [69], achieving the state-of-the-art accuracy on many benchmark test problems, e.g., CIFAR-10, CIFAR-100, and ImageNet.

In the following sections, we will review and categorize existing works on the applications of EAs to different optimization problems involved in automated DNN construction, as analyzed in Section II. Also, we will discuss the pros and cons of EA-based techniques compared to non-EA-based ones when solving different optimization problems.

IV. Evolutionary DNN Construction: Model Parameter Optimization

A. Problem Statement

Model parameter optimization, a.k.a. DNN training, searches the best model parameters M_p^* for a DNN with fixed architecture $M_a^\dagger$ to best solve a given task via optimizing, with respect to (w.r.t.) model parameters M_p, an objective function L that measures the performance of the model (defined by $M_a^\dagger$ and M_p) on training data D^{trn} by using a manually specified and configured model parameter learner $M_{lp}^\dagger$. This optimization problem, simplified from that defined in (1), can

$$
\begin{cases}
\mathcal{P}\left(M_a, M_p, M_{la}, M_{lp}\right) = \arg\min_{M_a, M_p, M_{la}, M_{lp}} \left(O^1_{D^{\text{val}}}\left(M_a, M_p, M_{la}, M_{lp}\right), \ldots, O^m_{D^{\text{val}}}\left(M_a, M_p, M_{la}, M_{lp}\right)\right) \\
\text{s.t. constraint}_i(M_{la}), \ \ i = 1, \ldots, q \\
\left(M_a, M_p, M_{lp}\right) \in \arg\min_{M'_a, M'_p, M'_{lp}} \left(O^1_{D^{\text{val}}}\left(M'_a, M'_p, M_{la}, M'_{lp}\right), \ldots, O^m_{D^{\text{val}}}\left(M'_a, M'_p, M_{la}, M'_{lp}\right)\right) \\
\text{s.t. constraint}_i\left(M'_a\right), \ \ i = 1, \ldots, r \\
\left(M'_p, M'_{lp}\right) = \arg\min_{M''_p, M''_{lp}} \ L_{D^{trn}}\left(M'_a, M''_{lp}, M''_p\right) \\
\text{s.t. constraint}_i\left(M''_{lp}\right), \ \ i = 1, \ldots, t \\
M''_p = \arg\min_{M'''_p} \ L_{D^{trn}}\left(M'_a, M''_{lp}, M'''_p\right) \\
\text{s.t. constraint}_i\left(M'''_p\right), \ \ i = 1, \ldots, k
\end{cases}
\tag{1}
$$

be formulated as

$$\begin{cases} M_p^* = \arg\min_{M_p} \; L_{D^{trn}}\left(M_p; M_a^\dagger, M_{lp}^\dagger\right) \\ \text{s.t. constraint}_i\left(M_p\right), \; i = 1, \ldots, k. \end{cases} \tag{2}$$

It is a large-scale (likely with millions of model parameters) and highly nonconvex continuous optimization problem.

In the following, we will provide a systematical review of EA-based methods applied to solve this problem and then discuss the pros and cons of EA-based methods in comparison to non-EA-based ones.

B. Taxonomy and Survey of Existing EA-Based Approaches

1) Taxonomy: The existing EA-based approaches for model parameter optimization mainly differ in the representations of parameters and the search paradigm of EAs. In this part, we will categorize these works from two aspects, i.e., solution representations and search paradigms.

2) Solution Representations: For EAs, the solution is often represented via a certain encoding scheme. In the following, we are going to review the commonly used solution representation schemes under two categories, i.e., direct encoding and indirect encoding.

1) *Direct Encoding:* In this scheme, each model parameter is directly represented as it is. All model parameters are represented as a vector [29], [33], [40], [43], where each element in the vector denotes one specific model parameter.
2) *Indirect Encoding:* In this scheme, each model parameter is represented in an encoded form obtained via a certain mapping [25], [27], [37], [52]. For example, binary encoding is often used in GA [25], [27], where each model parameter is represented via a bit string. In [37], network weights are represented by a set of coefficients, which are applied to linearly combine some predefined basis vectors to produce network weights. In [52], the encoding is via the mean and variance of Gaussian distribution, and the weights of DNN are sampled from this distribution.

3) Search Paradigms: Search paradigms refer to the frameworks of search algorithms. From this view, EA-based approaches can be divided into two categories, i.e., pure EAs and hybrid EAs, according to whether gradient-based methods are incorporated.

Pure EAs merely rely on EA-based approaches to solve the model parameter optimization problem.

1) *Basic Evolution:* In this framework, model parameters are optimized by following the general EA framework depicted in Algorithm 1 [25]–[30].
2) *Cooperative Co-Evolution:* In this framework, the model parameters are decomposed into several groups where each group is optimized by the EA in a separate but cooperative way. Finally, the optimal model parameters found in each group are combined to produce the solutions [21]–[24]. In [22], two approaches were proposed to decompose model parameters, i.e., synapse based and neuron based, where groups correspond to the single connection weight and all connection weights for a neuron, respectively. Model parameters in different groups are optimized by EAs separately and when an individual in a group is to be evaluated, cooperative evaluation is applied to concatenate this individual with the best individuals from the other groups for the assembled model parameters. After optimizing model parameters in all the groups, the finally obtained model parameters are typically defined as the combination of the best individuals from each group. An example for the neuron-based cooperative co-evolution is shown in Fig. 1 for readers to better understand the framework.

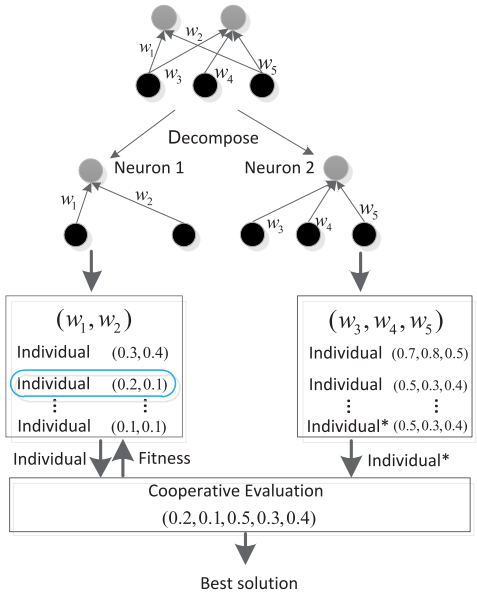


Fig. 1. Neuron-based cooperative co-evolution, where the connection weights for neuron 1 and neuron 2 are partitioned into two separate groups (w_1 and w_2) and (w_3, w_4, and w_5). Each group is evolved by the EA separately while evaluation of any individual in a group's population is performed in a cooperative way. For example, when evaluating the individual (0.2 and 0.1) from the group corresponding to neuron 1, it will be combined with the best individual found so far, i.e., (0.5, 0.3, and 0.4), from the group corresponding to neuron 2, and evaluated as a whole. The combination of the best individuals from each group forms the final output.

Hybrid EAs define combining EAs with the gradient-based methods to optimize model parameters. There are three common frameworks for making hybridization.

1) The first is applying the EA to find the best model parameters, which are then further optimized by using the gradient-based method to generate the final solution [31]–[36].
2) The second is applying the gradient-based method to first produce sets of model parameters, which are then used to initialize the population of the EA to let the EA keep searching for the best model parameters [39], [40].
3) The final is using the EA and the gradient-based method alternatively where the output of one method servers the start point for another, and this procedure is iterated until some stopping criteria are met. Existing works mainly differ in the order of applying these two methods in the iterative procedure and the individuals chosen from being continually optimized by using the gradient-based method [38], [41]–[43], [47]. For example, in [42], the EA is applied first, and the top 10% individuals with

higher fitness in the final population of the EA are further optimized by the gradient-based methods. In [38], the model is trained by using the gradient-based method until the performance improvement is below a certain threshold. Then, the EA is applied to further optimize model parameters. The best solution produced by the EA will be used as the starting point for applying the gradient-based method again. This process is repeated until certain criteria are met.

C. Discussion

Gradient-based methods [145] and EAs [152] are the two commonly used techniques for model parameter optimization. The gradient-based method is computationally efficient but may easily get stuck into inferior local optima due to its local search nature. The EA may mitigate the issue of getting stuck into inferior local optima due to its global search nature but suffer from high computational cost [153].

Three suitable optimization scenarios for EA-based methods are summarized as follows.

1) EAs are suitable for training DNNs with small size, such as RNNs and DBNs. For example, in [21]–[24], EAs are applied to train the RNN, where direct encoding is used to represent model parameters and the cooperative coevolution is used as the search paradigm. Because of the global search nature of EAs, the models trained by EAs achieve better performance than the ones trained by the gradient-based methods.
2) EAs can be combined with the gradient-based methods to train large-scale DNNs. To deal with a DNN with millions of parameters the pure EA search paradigm becomes less competent. In this situation, the hybrid EA search paradigm becomes promising, where the EA is combined with the gradient-based method in some way to train the DNN. The improvements contributed by EAs can be seen from two aspects: a) EAs can explore the promising regions in the search space, which assist the gradient-based method to more effectively exploit the best solution and b) the global search nature of EAs can help the gradient-based method to jump out of inferior local optima.
3) When the exact gradient information of the loss function is hard to be obtained, EAs can be used to train the DNN. A typical scenario is in the deep RL (DRL) tasks, where due to the sparse rewards, exact gradient information often cannot be obtained to update the policy network's model parameters. Therefore, in recent years, EAs have been applied to train the policy network following the pure EA search paradigm. From the experiment results reported in [27]–[30], [45], [154], and [155], the policy networks trained by EAs have demonstrated promising performance in various test scenarios.

V. Evolutionary DNN Construction: Model Architecture Optimization

A. Problem Statement

Model architecture optimization, a.k.a. NAS, is a bilevel optimization problem [10], [156], where the lower level task of model parameter optimization is nested within the upper level task of model architecture optimization. It may involve more than one objective functions $O^i,\ i = 1, \ldots, m$ that measure the performance of the model on validation data D^{val} from different aspects, such as accuracy and speed, leading to multiobjective optimization.

Assuming a manually specified and configured model architecture learner $M_{la}^{\dagger}$ and a manually specified and configured model parameter learner $M_{lp}^{\dagger}$ are used, this optimization problem, simplified from that defined in (1), can be formulated as

$$\begin{cases} \mathcal{P}(M_a, M_p) = \arg\min_{M_a, M_p} \\ \times \left(O^1_{D^{\text{val}}}\left(M_a, M_p; M_{la}^{\dagger}\right), \ldots, O^m_{D^{\text{val}}}\left(M_a, M_p; M_{la}^{\dagger}\right)\right) \\ \text{s.t. constraint}_i(M_a),\ i = 1, \ldots, r \\ M_p = \arg\min_{M_p'}\ L_{D^{trn}}\left(M_a, M_p'; M_{lp}^{\dagger}\right) \\ \text{s.t. constraint}_i\left(M_p'\right),\ i = 1, \ldots, k. \end{cases} \tag{3}$$

It is a very challenging bilevel optimization problem where the upper level architecture optimization is multiobjective, nondifferentiable (in many cases), and black-box while the lower level parameter optimization is large-scale and nonconvex. The solution to this problem is a Pareto optimal set $\mathcal{P}(M_a, M_p)$ composed of nondominated solutions. Eventually, one of these nondominated solutions will be chosen and deployed in use.

In the following, we will provide a systematical review of EA-based methods applied to solve this problem and then discuss the pros and cons of EA-based methods in comparison to non-EA-based ones.

B. Taxonomy and Survey of Existing EA-Based Approaches

1) Taxonomy: Similar to the model parameters optimization, existing EA-based approaches mainly differ from the solution representations and search paradigms. Also, the customized operators are designed to search the complex model architecture more effectively. Therefore, in this section, from the three aspects, i.e., solution representations, search paradigms, and customized search operators, EA-based approaches for the model architecture optimization are categorized.

2) Solution Representations: DNNs have a wide variety of architectures suitable for dealing with the tasks of different types and complexity. In general, DNN's architecture is determined by two factors, i.e., architectural units and topological patterns that define the connection between different units. Once these two factors are specified, a certain architecture is determined. Therefore, existing solution representations in model architecture optimization are often defined with respect to these two factors, where the architectural unit and the topological pattern are encoded as decision variables (in the representation) in certain ways. Then, some constraints, e.g., search ranges, are set for them, to define the search space within which model architecture optimization is carried out. Existing architectural units can be categorized into microunits and macrounits, where the former denotes the basic operational unit in the model, e.g., a convolutional layer, and the latter denotes the complex operation unit composed of multiple

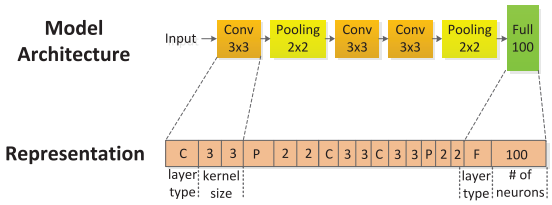


Fig. 2. Solution representation of a simple CNN using the fixed topological pattern, where the layers from the front to the end of the CNN are encoded via the type (C: convolutional, P: pooling, and F: fully connected) and parameters (the kernel size and # of neurons in a layer) of the layer and concatenated orderly to form a vectorial representation.

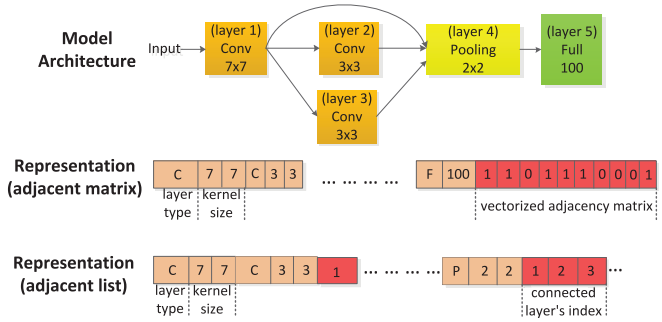


Fig. 3. Solution representation of a simple CNN of the similar Inception network type by using the adjacency matrix and adjacency list for representing its connections, respectively, denoted in red. The adjacency matrix is encoded for the entire network as a binary vector via serialization, where 0 and 1 represent the nonexistence and existence of a connection. The adjust list is encoded for each unit as a list of the indices of the units connecting to it.

basic ones, e.g., a residual block. Existing topological patterns can be categorized into fixed and nonfixed ones. For the fixed pattern, once architecture units for a model are specified in an ordered way, their connections are directly determined. In contrast, the connections between architectural units need to be specified for the nonfixed pattern. In the following, we categorize the solution representations used in existing works about model architecture optimization according to fixed and nonfixed topological patterns, where architectural units will be discussed therein. We do not make categorization for architectural units because there are too many of them, and the choice of them for defining the search space is problem dependent and needs to consider the balance between accuracy and computational time.

Fixed Topological Patterns: The solution representations following this pattern are merely determined by architectural units. Fig. 2 illustrates an example solution representation following this pattern, where a simple CNN is represented in the vectorial form via the concatenation of the representations of multiple basic architectural units, i.e., convolutional, pooling, and fully connected layers, and each basic unit is encoded via its type and parameters such as the kernel size. Sometimes, a same unit is repeated multiple times and the number of repeated times can be encoded in the presentation. For such solution representations, as long as the order of a unit is determined, i.e., its position in the representation, its connections to other units are determined.

Many existing DNN architectures can be represented in this way, e.g., DBNs [142], AlexNet [5], and VGG [6], which involves different numbers and types of architectural units. For example, DBNs [49], [157] and SAE [84], [92] can be represented via a concatenation of the encodings of multiple fully connected layers, where the encoding is simply the number of neurons in the layer. For CNNs [18], [51]–[57], [71]–[74], [80]–[82], [95], [102]–[104], [108]–[110], the encoded parameters for a convolutional layer may include the kernel size, stride size, padding value, etc.

Nonfixed Topological Patterns: The solution representations following this pattern are determined by both architectural units and topological patterns. In other words, the connections between architectural units, e.g., the skip connection [58], [60], [83], [98], need to be encoded and optimized. The adjacent matrix [58], [83], [98] has been commonly used to represent the connections between architectural units, where a matrix element with the value of 1 denotes the existence of the connection between the two units indexed by the row and column indices of that matrix element. In most works, the adjacency matrix is encoded as a binary vector via serialization [59], [60]. One limitation of using the adjacency matrix is that the total number of architectural units needs to be prefixed so that the matrix can be formed. Therefore, it is not suitable for the cases where the number of architectural units is a decision variable to be optimized. To address this issue, the adjacent list [61]–[63], [158] was proposed as a list of units connecting to a specific unit. It allows easily adding or removing units while considering the connections to them. Fig. 3 illustrates the solution representation of a simple CNN of the similar inception network type by using the adjacency matrix and adjacency list for representing the connections, respectively, where the representation of the unit follows the way described in the part of fixed topological patterns.

For both fixed and nonfixed topological patterns, the architectural units can be macrounits, which are more commonly known as cells in existing works. In cell-based solution representations [15], [53], [64]–[67], [67]–[69], [71], [79], [159], the encodings of connections among cells follow the ways described in the parts of fixed and nonfixed topological patterns. The cell itself can be treated as a "small" DNN model, which can also be represented in the way following either fixed or nonfixed topological patterns. This kind of solution representation may reduce the solution space by simply stacking the same cell in a certain way and merely optimizing this cell, which has achieved very promising performance on CIFAR-10 and CIFAR-100 [67]. Fig. 4 illustrates the solution representation of a simple cell-based CNN.

In addition to solution representations in the vectorial form as described above, there are other forms of representations. For example, decision variables can be partitioned into multiple levels and different levels are optimized in a hierarchical and cooperative way [13], [66], [86], [89], [125]. The tree-structure representations [75]–[77] are commonly used in model optimization methods based on GP [75], where the leaf node receives the input data and the other nodes perform some basic operations, such as convolution, max pooling, and summation.

3) Search Paradigms: Model architecture optimization requires evaluating the quality of candidate architectures,

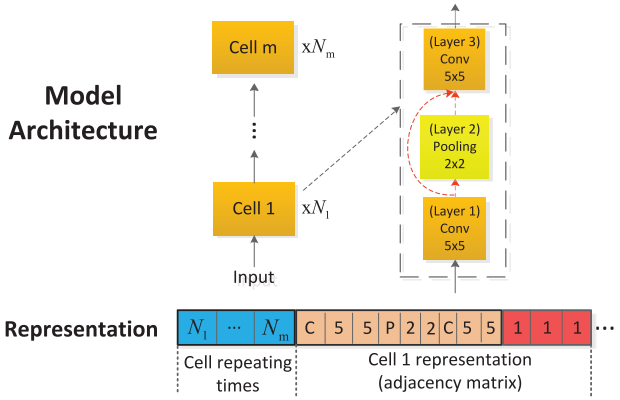


Fig. 4. Solution representation of a cell-based CNN, which are formed by stacking m types of cells with each cell, denoted by Cell i ($i = 1, \ldots, m$), repeating N_i times, where the connections in the cell are encoded via the adjacent matrix-based representation.

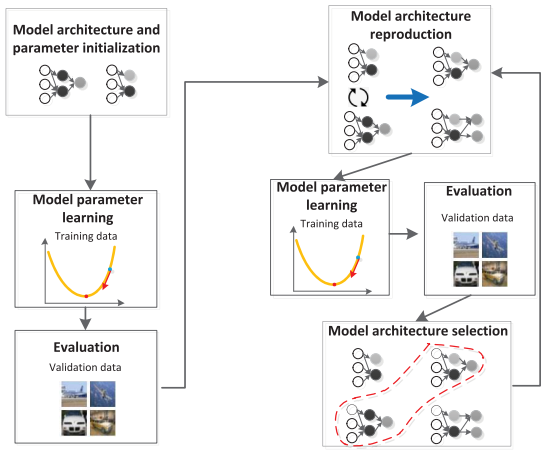


Fig. 5. Basic search paradigm.

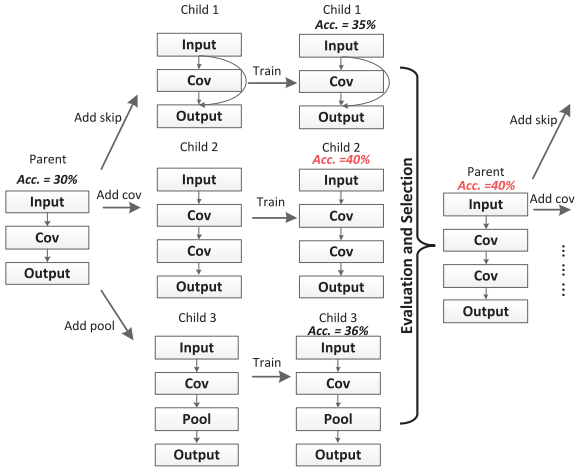


Fig. 6. Example of using the incremental search paradigm to construct a simple CNN via the ES.

where learning model parameters is typically indispensable when carrying out evaluation. Therefore, it usually needs to consider both architectures and parameters during the model architecture optimization process. Existing EA-based approaches in this regard often employ EAs for architecture optimization while using gradient-based methods for parameter optimization, differing in the ways to search for the best architecture. In the following, we describe the commonly employed search paradigms in existing works about applying EAs for model architecture optimization.

1) *Basic Search Paradigm:* This paradigm, as illustrated in Fig. 5, follows the general framework of EAs in Algorithm 1 but incorporates a gradient-based parameter learning process before evaluation. Specifically, a population of candidate model architectures are initialized at first. Then, model parameters for these architectures are learned via the gradient-based method on the training data, followed by evaluating the quality of candidate architectures on the validation data. After that, reproduction is applied to some selected existing candidate architectures to generate new candidature architectures, which are trained by using the gradient-based method and then evaluated. Next, a new population is formed by selecting elicit candidate architectures from both the current population and the new candidature architectures generated from it. This new population will be evolved by repeating the above steps until certain termination criteria are met. Many existing works [18], [74], [80]–[82] follow this search paradigm.
2) *Incremental Search Paradigm:* This paradigm is similar to the basic search paradigm in terms of the operational pipeline. The major difference is that it incrementally constructs the model by gradually adding components (i.e., different types of layers and connections) to it as the population evolves [13], [48], [160]. Fig. 6 illustrates an example of using this search paradigm to construct a simple CNN via a simple evolutionary strategy [160]. In this example, the initial population contains one candidature architecture with one convolutional layer. During reproduction, different adding operators, i.e., adding the convolutional layer, pooling layer, and skip connection, are applied to incrementally construct the architecture. Then, the newly generated candidate architectures are trained and evaluated. Next, the best of them will be selected to form the new population, which will be evolved by repeating the above steps until certain termination criteria are met. Compared to the basic search paradigm, the incremental one allows searching for part of the model architecture in different search stages, and thus, reduces the computational cost [160]. However, it can only generate the complete architecture at the end of the search process while the candidate architectures in the basic search paradigm are always complete during the entire search process.
3) *Cooperative Co-Evolution Paradigm:* In this search paradigm, the model architecture is decomposed into subcomponents (i.e., macrounits) according to a "blueprint," which defines the topological patterns of subcomponents. It is equivalent to the cell-based solution representation. Then, the blueprint and its associated subcomponents are optimized together in a certain way to search for the optimal model architecture. For example, in [86] and [87], the model architecture is

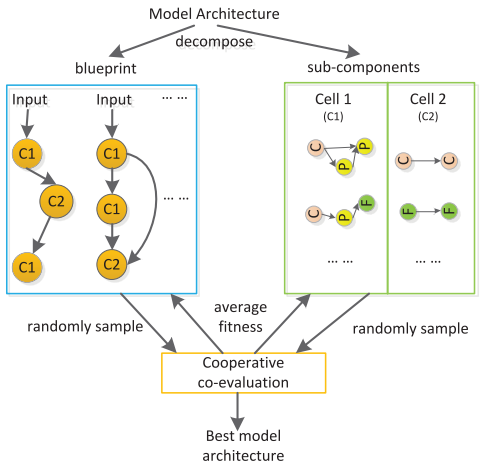


Fig. 7. Example of the decomposition-based search paradigm, where a population of blueprints and a population of its subcomponents are co-evolved [86].

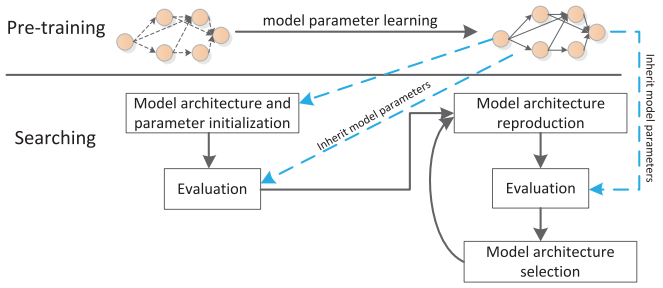


Fig. 8. Example of the hypernet-based search paradigm.

decomposed into two parts, i.e., the blueprint and its subcomponents, which are co-evolved in a cooperative way to search for the optimal architecture. Specifically, a population of blueprints and a population of subcomponents are evolved separately. To evaluate the quality of an individual in either population, a collection of model architectures is generated by randomly sampling from both populations and assembling the sampled blueprints and subcomponents. Then, these model architectures are trained and evaluated. Next, the quality of those evaluated model architectures that contain a blueprint or subcomponent will be averaged to estimate the quality of that blueprint or subcomponent. Fig. 7 illustrates this process.

4) *Hypernet-Based Search Paradigm:* This search paradigm typically has two stages, i.e., pretraining and searching stages [14], [83], [161], [162]. In the pretraining stage, a hypernet that subsumes all possible candidate model architectures is specified and trained, where the gradient-based methods with various tricks, such as single path [83] and random search [162], have been proposed and used for training. The pretrained hypernet will be used to guide the subsequent searching stage, where any search paradigm described previously can be used to search for the optimal model architecture. For example, a population of candidate model architectures are sampled from the pretrained hypernet, where the model parameters of the sampled model architectures are directly inherited from the hypernet. Then, these sampled architectures are evaluated, where the parameter learning process before evaluation is omitted. After that, reproduction is applied to some selected existing candidate architectures to generate new candidature architectures. The newly generated architectures may directly inherit their model parameters from the pretrained hypernet and then get evaluated. Next, a new population is formed by selecting elicit candidate architectures from both the current population and the newly generated candidature architectures. This new population will be evolved by repeating the above steps until certain termination criteria are met. Fig. 8 illustrates this example.

5) *Multiobjective Optimization:* In this search paradigm, multiple (often conflicting) objectives drive the search process and evolutionary multiobjective optimization techniques [163] are often employed to search for a set of Pareto optimal solutions. Finally, one of these Pareto optimal solutions will be chosen for deployment according to practical needs. This search paradigm can be combined with any of the previously discussed search paradigms. It becomes more and more commonly used in practice, where optimization of model architecture is subjected to some factors other than accuracy, e.g., inference time and energy consumption [15], [49], [60], [70], [73], [85], [91]–[94], [96], [159]. For example, in [15], MOEAs are used to optimize model architectures by considering model accuracy and model size that is hostable by the hardware on which the model will be deployed. In [70], both the accuracy and inference time of the model are considered in the MOEA to optimize model architectures so that the obtained optimal architecture may satisfy the practical requirement on latency.

4) Customized Search Operators: For model architecture optimization, most of the search operators, e.g., initialization, evaluation, reproduction, and selection, in EAs can be well applied to the solution representations described previously. For example, in [95], the architectures with different layers are randomly generated as initial candidate solutions. In [95] and [52], recombination is applied to swap parts of the two selected models to generate two new architectures. In [13], mutation is applied to alter the configuration of a layer, e.g., the kernel size and stride. In [72], architectures with higher accuracy will be selected into the population of the next generation. However, merely relying on the basic search operators in EAs may not be enough due to the nature of architecture search, e.g., varying length solution representation, invalid architecture, and model parameter inheritance. In the following, the customized operators widely used in existing works about EA-based model architecture optimization are categorized and described.

1) *Customized Recombination:* The basic recombination operator assumes the solution representations of two parents have an equal length. However, it is typical that EA's population used for model architecture

optimization contains individuals of various lengths, denoting architectures of various sizes. Therefore, many existing works [52], [54]–[57], [80], [82], [95], [96] have designed customized recombination operators that can handle varying length solution representations. For example, in [57] and [52], only the layers with the same positions in the two parents can be swapped. In [54], the cutting point is randomly selected in each of the two parents and then the right parts of two cutting points are swapped. Some quality control methods can be applied to the recombination operator to improve its effectiveness. For example, in [55], the cutting points are only allowed to be inserted into the predefined positions to prevent invalid architectures from being generated.

2) *Customized Mutation:* The basic mutation operator is used to alter the values of select decision variables within their feasible ranges, e.g., changing the kernel size. To enable search for architectures of different sizes, many existing works [13], [48], [52], [55], [56], [67], [80], [97] have designed customized mutation operators that can change the size of the architecture by adding or removing its parts (encoded as decision variables). For example, the mutation operator was used in [13] to insert/remove convolution layers and skip connections into/from the parent architecture. In [55], the mutation operator was used to replicate a specific layer. Some quality control methods can be applied to the mutation operator to improve its effectiveness. For example, in [48] and [97], if a new architecture generated by the mutation operator performs worse than its parent(s), the added or removed components will be revoked.
3) *Repairment:* When a new architecture is generated via recombination and/or mutation, it may be invalid, and thus, needs to be discarded or repaired. For example, when a new convolutional layer is added into an existing architecture, its input and/or output might be inconsistent with the output of the preceding and/or the input of the succeeding layers, resulting in an invalid new architecture [72]. The repairment operator is designed to modify an invalid architecture to make it become valid. For the previous example, it can be used to adjust the input and/or output of the newly added convolutional layer [72]. Invalid architectures may have different kinds and degrees of invalidity due to different ways to generate them, leading to various repairment operators. Notably, repairment operators allow designing more flexible search operators without having to strictly consider the validity of their outputs.
4) *Inheritance:* During search, any newly generated architecture (child) typically reserves part of its architecture from its parent(s). In addition, the model parameters in the reserved part of its architecture may also be inherited from its parent(s). For example, when two selected parents exchange parts of their architecture during recombination, both the architecture and its associated model parameters will be swapped [46], [97], [125]. When a selected parent undergoes mutation to generate a child, the child will inherit the unmutated part of the architecture as well as its associated parameters from its parent [48]. Model parameter inherence may prevent training newly generated models from scratch and thus, reduce computational costs [125].

Among the above-discussed customized search operators, both recombination and mutation intend to generate new architectures based upon the old ones. In comparison, recombination typically leads to significant architectural changes from parents to children and thus, suits exploration of innovative architectures that may result in much improved performance [98]. Mutation typically leads to incremental changes to the old architecture and thus, suits exploitation of existing architectures. Furthermore, recombination is more heavily dependent on the encoding scheme and the repairment operator. Therefore, mutation is more widely used in existing works [13], [48], [67], [68].

C. Discussion

In addition to EA-based approaches, RL-based [11], [12], [158], [164], [166], [167] and gradient-based [10], [145], [165] ones have also been widely used for model architecture optimization.

RL-based approaches typically follow the incremental search paradigm, where the policy used for incremental architecture generation is designed and learned via feedback (rewards) to gradually search for the best architecture. Similar to EA-based ones, RL-based approaches also have a vast architecture search space and require time-consuming parameter learning, leading to demanding computational costs. Furthermore, they typically employ a certain policy network (DNN), which involves many hyperparameters and is not easy to train. Moreover, although the two types of approaches can be both applied to multiobjective scenarios, RL-based ones usually need to first convert the multiobjective problem to the single-objective one in some ways [166], [167], and thus, cannot produce the Pareto optimal set as EA-based approaches do.

In recent years, gradient-based approaches have been proposed, aiming to reduce computational costs, where model architecture optimization is formulated as a continuous optimization problem. For example, the DARTs approach proposed in [10] relaxes the discrete architecture search space to a continuous one, by mixing possible candidate operations, so that the architecture can be optimized via gradient descent in an efficient way. In comparison to EA-based ones, such approaches are much more efficient, but they are less component to explore novel architectures because all possible architectures that could be found need to be manually predefined via the solution representation.

EA-based approaches have been notorious for their high computational costs. For example, it may take about 3000 GPU days to find a desirable architecture [13]. In recent years, many computational speedup strategies have been proposed from the perspectives of algorithmic design and computing power, which will be discussed in Section VII. They allow EA-based approaches to achieve satisfactory performance at much reduced computation costs like several GPU days [15], [83]. In addition, EA-based approaches can easily and effectively deal

with various constraints and multiple objectives, and inherently allow model ensemble due to its population-based nature to achieve better generalization [49], [101].

VI. Evolutionary DNN Construction: Model Learner Optimization

A. Problem Statement

Model architecture and parameter learner optimization seeks the most effective learners (intrinsically optimizers), aiming to best solve model architecture optimization and model parameter optimization problems, respectively. Specifically, model architecture learner optimization searches the best architecture learner (including its best-calibrated parameters, a.k.a. model hyperparameters) by optimizing, w.r.t. architecture learner's representation M_{la}, the performance of the learner on solving the model architecture optimization problem defined in Section V. This is equivalent to the optimization problem defined in (1), which is prohibitively challenging. Model parameter learner optimization searches the best parameter learner (including its best-calibrated parameters) by optimizing, w.r.t. parameter learner's representation M_{lp}, the performance of the learner on solving the problem of model parameter optimization for a DNN model with fixed architecture $M_a^\dagger$. It can be formulated as the following bilevel optimization problem:

$$\begin{cases} \left(M_{lp}^*, M_p^*\right) = \arg\min_{M_{lp}, M_p} \; L_{D^{trn}}\left(M_{lp}, M_p; M_a^\dagger\right) \\ \text{s.t. constraint}_i\left(M_{lp}\right), \; i = 1, \ldots, t \\ M_p = \arg\min_{M_p'} \; L_{D^{trn}}\left(M_p', M_{lp}; M_a^\dagger\right) \\ \text{s.t. constraint}_i\left(M_p'\right), \; i = 1, \ldots, k. \end{cases} \tag{4}$$

Most of the existing studies did not consider model architecture learner optimization due to its expensive computational cost. In the following, we will focus on model parameter learner optimization and provide a review of EA-based methods applied to solve it, and then discuss the pros and cons of EA-based methods in comparison to non-EA-based ones.

B. Taxonomy and Survey

In model parameter learner optimization, the learners used for model parameter optimization, e.g., gradient-based methods, are optimized in terms of both its types and parameters. The decision variables involved in such an optimization problem are often not many and may take either discrete (for types) or continuous (for parameters) values. Also, there exists no explicit mathematical formulation of the objective function of decision variables in such an optimization problem. Therefore, EAs are a good choice for solving this problem, where direct encoding is often used for solution representations.

EAs have been applied to optimize the parameters of gradient-based methods [19], [20], [132]. For example, in [132], they are applied to optimize the learning rate, momentum, and batch size for the gradient-based approach Adam. EAs have also been used to both choose the most appropriate learner and search the best parameters for the learner. For example, in [131], ES is applied to select the learner from Adam and Adadelta and optimize the parameters of the chosen learner. In some works, optimization of learner's parameters is integrated with the model architecture optimization process [51], [53], [55], [127]–[130], [133], [134], [168], [169]. For example, in [127], the EA is applied to design a VGG model, where the parameters of a prespecified model parameter learner are encoded together with the model architecture for solution representations.

C. Discussion

In existing works, the model architecture learner is typically prespecified instead of being optimized due to the practically prohibitive computational cost, even higher than that of NAS, for solving the model architecture learner optimization problem. As for model parameter learner optimization, it is a black-box, mixed-variable optimization problem as discussed previously. Bayesian optimization (BO) and EAs, as the two most representative derivative-free optimization methods, have been applied to solve it. Among them, BO does not rely on heavy trial-and-error exploration but is less effective for handling mixed decision variables and constraints. In contrast, EAs are more suitable for solving the mixed-variable optimization problem with constraints [131], [168]. Although EAs are very time consuming, many computational speedup strategies have been proposed in recent years from the perspectives of both algorithmic design and computing power, which will be discussed in Section VII. They allow EA-based approaches to achieve satisfactory performance at much reduced computation costs [19], [20], [134].

VII. Evolutionary DNN Construction: Miscellaneous

In addition to the three major optimization problems discussed in the previous sections, this section describes how EAs are applied to solve other optimization tasks involved in the automatic DNN construction process. Furthermore, existing works about two key factors relevant to optimization, i.e., speedup and objectives, are summarized.

A. Other Optimization Tasks

Besides model parameter, architecture, and learner optimization, evolutionary DNN construction involves some other optimization tasks. For example, in [170], the EA is used in the data preprocessing step to select more useful features for the input images to improve the performance of the DNN on edge detection. Furthermore, a DNN may employ different loss functions, leading to different performance. The loss function used by a DNN is typically designed or specified according to human expertise instead of in a problem-driven way. To address this issue, EAs have been applied to optimize DNN's loss functions [171]–[173], [206], [208]. For example, in [171], the EA is applied to optimize the misclassification cost to improve DNN's performance for solving imbalanced classification problems. In [173], a specific loss function is designed by the EA to speed up model parameter optimization via fewer training steps to achieve higher accuracy. In [208],

the EA is applied to optimize the loss function to make policy learning in RL suitable for the dynamic environment.

B. Optimization Speed-Up

Most optimization problems involved in automated DNN construction are computationally demanding, mainly due to the vast and complex search spaces for DNN architecture and parameters. Existing works aiming at optimization speedup can be categorized from the aspects of software and hardware, which are described below in terms of algorithm design and computation power, respectively.

1) Algorithm Design: From the viewpoint of the algorithm design, the commonly used speed-up techniques in literature are as follows.

1) *Parameter Sharing:* This kind of techniques is often used in transfer learning [64], [174]–[176], which uses the knowledge (i.e., model parameters) learned from solving one task to help solve another problem. For EA-based approaches, through the inheriting operator described in Section V, the offspring model can inherit model parameters from its parents. The inherited model parameters can be used as a warm start to prevent the offspring model from being trained from scratch [13], [125], [177]. Besides the inheriting operator, a more general parameter-sharing technique is the hypernet framework [14], [15], [83], [161], [162]. Specifically, a hypernet, which subsumes all the candidate model architectures in the search space, is trained first. Then, subnetworks sampled from the hypernet will inherit its model parameters, and these models can be evaluated directly. As a result, the time cost is reduced significantly.
2) *Training Cost Reduction:* Early stopping is a widely used strategy to reduce the training cost. For example, in [88], the number of training epochs is set as 5 for each candidate models to reduce training time. Another commonly used strategy is to first perform model optimization on a small dataset at the expense of the small cost, and then refine the obtained model on a large dataset. It can much save the computational cost compared to directly performing model optimization on the large dataset. For example, in [10] and [67], the model architecture is first optimized on CIFRA-10, and then, the obtained best architecture is applied with possible further refinement on ImageNet. This kind of speed-up techniques may sacrifice performance to some degree because the model optimized in this way is often suboptimal.
3) *Performance Prediction:* Predicting the performance of a DNN may help speed up model architecture and/or parameter optimization. For example, different machine learning models [102], [160], [178] have been used to predict the performance of a model under training to determine whether the model still deserves to be further trained [103]. In recent years, proxy models have been proposed to learn a mapping from model architecture to model performance [126], [179], [180]. As such, the huge computational burden of parameter learning for a specific model architecture can be avoided to reduce the computational cost in NAS.
4) *Cell-Based Framework:* Many existing works search for the entire DNN at once, which is often computationally demanding due to the very large size of the DNN. The cell-based framework is used to speed up the construction of DNNs. In this framework, a DNN is supposed to be composed, in a certain pattern, of multiple small components with the same architectures, the so-called cell [64], [104]. As a result, the automated construction of an entire DNN is transformed to optimize a cell in a much reduced search space, leading to significant computational speedup.

2) Computation Power: Taking advantage of the available computational resources is also an effective way to speed up optimization. A straightforward method is applying the free or cheap resources, such as the cloud [181] and volunteer computers [182]. But these resources are often limited, and considering the security of private information, access to these available computational resources is also constrained.

Parallel computation is an effective way to take the most advantages of the computational resources [20], [105], [183]. EAs as the population-based methods are nature for the parallel framework. Recently, in [20], an asynchronous parallel framework is proposed upon two parts, i.e., the controller and the workers. The controller is responsible for search operators to produce offspring and update the population. The workers are applied to train and evaluate these individuals, which require a considerable amount of compute resources. When a worker finishes the training and evaluation of one individual, it will send back the fitness and receive a new model from the controller. Meanwhile, the controller will update the global information and population according to the information it has received. Compared with the parallel process, in the asynchronous parallel framework, the devices can continually deal with the population rather than keep pace with each other, improving the efficiency of limited compute resources.

C. Optimization Objectives

Accuracy is the most intuitive optimization objective for the evolutionary construction of DNNs. In addition, there are other objectives that need to be considered in practical applications. This section will discuss these other objectives studied in the literature.

1) *Inference Time:* It measures the time required for the input propagating forwards through the network to produce the output, which is a crucial factor to be considered in real-time applications such as velocity prediction in autopilot [184]. This measurement is relevant to the model architecture and the computing device that the model is deployed [70], [97].
2) *Computational Complexity:* It estimates the computing speed of an algorithm in terms of the number of floating-point operations (FLOPs) [70], [94] or multiply adds operators [73], [159].
3) *Space Complexity:* It measures the amount of working storage required by an algorithm, and can be roughly estimated via the number of model

parameters [15], [94], the number of connections, or the sparsity of the model [92]. This measurement is related to the overfitting and underfitting of the model.

4) *Energy Consumption:* It measures the average energy consumption for the model to make inference on the input data, which can be roughly estimated via the peak power, average power, and the running time of CUDA kernel functions [167], [184]. This measurement is vital for the scenarios where the construction of DNNs is for devices with the limited energy capacity such as mobile phones.

VIII. Applications, Challenges, and Trends

A. Applications

Evolutionary DNN construction approaches, as elaborated in Section IV–VII, have demonstrated successes in various applications. In the following, we summarize these applications from three dimensions, i.e., data types, application fields, and deployment scenarios.

Data Types: Evolutionary DNN construction approaches have been applied to various data types, such as images [13], [59], [108], [109], speech [128], [133], [148], and texts [15], [110]. In particular, tremendous research effort has been devoted to solving the image classification problem. Specifically, CIFAR-10, CIFAR-100, and ImageNet are the three most commonly used benchmark image datasets in the study of evolutionary DNN construction for image classification. On CIFAR-10, the DNN model constructed by the EA achieved the test error of 2.5% [15], outperforming the mainstream handcrafted models, such as ResNet-50, VGG, and DenseNet. On CIFAR-100 and ImageNet, the DNN models constructed by EAs have achieved competitive performance in comparison to handcrafted DNN models [13], [15]. Besides image classification, the DNN models designed by EAs have demonstrated great successes in object identification [17], [185], speech recognition [128], and emotion recognition [40].

Application Fields: The automated DNN construction opens the door for researchers and engineers from different fields and with little expertise and experience in DNNs to better utilize the DNN to resolve the problems arising in their fields. For example, in biomedical engineering, EAs have been applied to design DNNs for disease diagnosis [81], [116]–[119], protein structure prediction [186], and sleep study [187]. In mechanical engineering, EAs have been used to design DNNs for failure detection [114], [115], robot control [188], and remaining useful life prediction [49]. In addition, evolutionary DNN construction has been applied to gamma-ray detection [110], traffic flow prediction [189], and electricity price forecasting [79].

Deployment Scenarios: With the advancement of IoT and 5G, more and more devices, e.g., smartphones, vehicles, and drones, have benefited from various DNN-powered applications. The DNNs deployed on such devices typically need to carefully consider model size and complexity to meet real-world constraints on computational latency, energy consumption, etc. This poses extra challenges to evolutionary DNN construction approaches, e.g., to solve a multiobjective, instead of single-objective, optimization problem. For example, EAs have been applied for model compression so that the compressed model can work well on the device without suffering from computational resource and memory storage issues [73], [184], [190], [207]. Also, MOEAs have been applied to design the DNN by considering multiple conflicting objectives (e.g., model size and accuracy) at the same time, and then users can select, from the finally obtained Pareto optimal set, the most suitable model according to the hardware environment in practice [70], [97], [107].

B. Challenges

Evolutionary DNN construction has achieved great successes, but also come with some unsolved challenges to be further addressed. In the following, some key challenges are summarized and discussed.

Tradeoff Between Optimization Cost and Model Performance: Evolutionary DNN construction approaches typically have high computational costs. In recent years, many speedup strategies have been proposed from the perspectives of both algorithmic design and computing power, as discussed in Section VII. However, they may impose some side effects on model performance. For example, when the search space is purposely limited, the chance of finding novel architectures becomes small. The one-short design paradigm cannot avoid unreliable model ranking, which may lead to undesirable design outcomes. Performance prediction approaches may wrongly discard promising candidate models due to the inaccuracy of predictive models, leading to undesirable results. As such, making a good tradeoff between optimization cost and model performance remains a challenge to be further studied.

Effectiveness of Model Architecture Optimization Methods: Recent studies revealed that many mainstream optimization methods do not differ much from random search in their performance of solving the model architecture optimization problem [191]–[194]. On some tasks, random search even outperforms the others. Therefore, it becomes necessary to comprehensively and systematically evaluate and compare existing model architecture optimization methods, aiming to reveal its suitable and unsuitable application scenarios. This endeavor is more important than keeping proposing new but less understood methods.

More Challenging Tasks: In existing works, many DNN models constructed by EAs are based on the datasets of small size, e.g., MNIST and CIFAR-10, because of low computational costs. These constructed models may achieve nearly perfect accuracy of 99% and 97% on MNIST and CIFAR-10, respectively, due to the low difficulty of these tasks. As a result, further improvement becomes more technically difficult but less practically useful. On the other hand, when the same approach is applied to construct the DNN on large-scale datasets, such as ImageNet and CIFAR-100, its computational cost may become prohibitively high. Furthermore, there is no guarantee that the effectiveness of the approach demonstrated on the small-scale dataset can be retained on the large-scale dataset. As a result, datasets of small size but

high difficulty are demanded. Also, the performance sensitivity of an approach to the size of its used dataset should be investigated.

Model Architecture Learner: The effectiveness of model architecture optimization depends upon the employed optimization method, a.k.a. model architecture learner. To seek the best performance of model architecture optimization, it is an intuitive to think of finding the best model architecture learner, e.g., finding the most suitable population size and maximum generation number for EA-based learners. However, as discussed in Section VI, this task corresponds to a multilevel optimization problem, which is computationally prohibitive, much higher than model architecture optimization per se. This big challenge calls for further investigations.

C. Trends

In this part, some popular research trends in the evolutionary DNN construction are discussed.

Benchmark Platform for Model Comparison and Development: Existing works usually involve comparison of different evolutionary DNN construction methods. However, these compared methods often have distinct search spaces and thus, their intrinsic capabilities for DNN construction are different, i.e., the best model obtained by one method can never be found by another method. To address this issue, a special benchmark for NAS, named Nas-bench-101, was proposed in [195], where all possible DNN architectures with respect to a prespecified search space for solving a specific task, i.e., image classification on CIFAR-10, are fully trained in advance. When methods are compared on this benchmark, they can employ the same search space to guarantee fairness of comparison. Furthermore, there is no need to invoke the time-consuming parameter learning process during architecture optimization because all candidate architectures have been pretrained. Therefore, this kind of benchmarks removes the computational bottleneck and allows the researchers to be able to focus on designing and developing more effective optimization methods. However, one benchmark merely corresponds to a certain search space and a certain task, which cannot cover problems at various difficulty levels. To deal with this issue, more and more benchmarks of different search spaces and tasks being solved have been proposed, e.g., Nas-bench-201 [196], NASBench-301 [197], TransNAS-Bench-101 [198], and HW-NASBench [199].

Design of Architecture Search Space: The intrinsic capability of an evolutionary DNN construction method heavily depends on the search space, which is often manually specified. Recently, there emerges a growing interest in automatically designing the search space of model architecture [200]–[202]. On the one hand, a simpler search space may facilitate the optimization process carried out therein. For example, in [201], by fitting a linear function between decision variables, the size of the search space is reduced, leading to better search results. On the other hand, a more powerful search space that contains more effective architectures therein may inherently boost the effectiveness of the optimization process carried out therein. For example, in [202], EAs are applied to design the search space automatically. In this work, the search space itself is formulated as the candidate solution which is evolved by the EA, where the quality of a specific search space is estimated via the average quality of DNNs sampled from it. The best search space eventually found by the EA is expected to allow any optimization method performed therein to produce high-quality models.

Handling Insufficient Annotated Data: Evolutionary DNN construction typically requires a fairly large amount of annotated data to enable the optimization process, e.g., the training set for model parameter optimization and the validation set for model architecture optimization. However, in practice, the amount of available annotated data is often limited. It poses challenges to many existing works. Furthermore, the unannotated data are usually available but not well utilized. Recent years have seen many works on addressing these issues, e.g., the metalearning [203], unsupervised [204], and self-supervised [205] NAS techniques, which have demonstrated very promising results and deserve further investigations.

IX. Conclusion

In this work, we formulated automated DNN construction into a multilevel multiobjective optimization problem with constraints, analyzed this problem to gain deep insights, and provided a comprehensive review of EA-based approaches to solving this problem, mainly from the aspects of model parameter optimization, model architecture optimization, and model learner optimization. Furthermore, we discussed the pros and cons of EA-based approaches in comparison with other commonly used approaches in different optimization scenarios as well as two essential factors in optimization, i.e., computational speedup and optimization objectives. Moreover, we summarized the applications, challenges, and trends in this area of study. As discussed in Section I, this work is different from existing survey works. It aims to help DNN researchers to better understand why, where, and how to use EAs for automated DNN construction and also help EA researchers to better understand the task of automated DNN construction so that they may focus more on EA-favored optimization scenarios to devise more effective techniques. Furthermore, we summarized the publicly available datasets and code used in relevant studies and provided them in the online supplementary document.

References

[1] H. Bagherinezhad, M. Rastegari, and A. Farhadi, "LCNN: Lookup-based convolutional neural network," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 7120–7129.

[2] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *J. Mach. Learn. Res.*, vol. 12, pp. 2493–2537, Nov. 2011.

[3] J. Janai, F. Güney, A. Behl, and A. Geiger, "Computer vision for autonomous vehicles: Problems, datasets and state-of-the-art," 2017. [Online]. Available: arXiv:1704.05519.

[4] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

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

[6] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014. [Online]. Available: arXiv:1409.1556.

[7] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 1–9.

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

[9] W. De Mulder, S. Bethard, and M.-F. Moens, "A survey on the application of recurrent neural networks to statistical language modeling," *Comput. Speech Lang.*, vol. 30, no. 1, pp. 61–98, 2015.

[10] H. Liu, K. Simonyan, and Y. Yang, "DARTs: Differentiable architecture search," 2018. [Online]. Available: arXiv:1806.09055.

[11] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," 2016. [Online]. Available: arXiv:1611.01578.

[12] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," 2016. [Online]. Available: arXiv:1611.02167.

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

[14] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," 2018. [Online]. Available: arXiv:1802.03268.

[15] Z. Yang *et al.*, "CARs: Continuous evolution for efficient neural architecture search," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 1829–1838.

[16] M. Črepinšek, S.-H. Liu, and M. Mernik, "Exploration and exploitation in evolutionary algorithms: A survey," *ACM Comput. Surveys*, vol. 45, no. 3, pp. 1–33, 2013.

[17] Y. Chen, T. Yang, X. Zhang, G. Meng, X. Xiao, and J. Sun, "DetNas: Backbone search for object detection," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 6642–6652.

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

[19] M. Jaderberg *et al.*, "Population based training of neural networks," 2017. [Online]. Available: arXiv:1711.09846.

[20] A. Li *et al.*, "A generalized framework for population based training," 2019. [Online]. Available: arXiv:1902.01894.

[21] R. Chandra and M. Zhang, "Cooperative coevolution of elman recurrent neural networks for chaotic time series prediction," *Neurocomputing*, vol. 86, pp. 116–123, Jun. 2012.

[22] R. Chandra, "Competition and collaboration in cooperative coevolution of Elman recurrent neural networks for time-series prediction," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 12, pp. 3123–3136, Dec. 2015.

[23] R. Chandra, Y.-S. Ong, and C.-K. Goh, "Co-evolutionary multi-task learning with predictive recurrence for multi-step chaotic time series prediction," *Neurocomputing*, vol. 243, pp. 21–34, Jun. 2017.

[24] S. Chand and R. Chandra, "Cooperative coevolution of feed forward neural networks for financial time series problem," in *Proc. IEEE Int. Joint Conf. Neural Netw. (IJCNN)*, 2014, pp. 202–209.

[25] Y. Li, Z. Zhu, D. Kong, H. Han, and Y. Zhao, "EA-LSTM: Evolutionary attention-based LSTM for time series prediction," *Knowl. Based Syst.*, vol. 181, Oct. 2019, Art. no. 104785.

[26] S. Khadka, J. J. Chung, and K. Tumer, "Neuroevolution of a modular memory-augmented neural network for deep memory problems," *Evol. Comput.*, vol. 27, no. 4, pp. 639–664, 2019.

[27] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune, "Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning," 2017. [Online]. Available: arXiv:1712.06567.

[28] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, "Evolution strategies as a scalable alternative to reinforcement learning," 2017. [Online]. Available: arXiv:1703.03864.

[29] P. Chrabaszcz, I. Loshchilov, and F. Hutter, "Back to basics: Benchmarking canonical evolution strategies for playing Atari," 2018. [Online]. Available: arXiv:1802.08842.

[30] G. Morse and K. O. Stanley, "Simple evolutionary optimization can rival stochastic gradient descent in neural networks," in *Proc. ACM Genet. Evol. Comput. Conf.*, 2016, pp. 477–484.

[31] S. S. Tirumala, S. Ali, and C. P. Ramesh, "Evolving deep neural networks: A new prospect," in *Proc. IEEE 12th Int. Conf. Nat. Comput. Fuzzy Syst. Knowl. Disc. (ICNC-FSKD)*, 2016, pp. 69–74.

[32] E. P. Ijjina and K. M. Chalavadi, "Human action recognition using genetic algorithms and convolutional neural networks," *Pattern Recognit.*, vol. 59, pp. 199–212, Nov. 2016.

[33] A. Banharnsakun, "Towards improving the convolutional neural networks for deep learning using the distributed artificial bee colony method," *Int. J. Mach. Learn. Cybern.*, vol. 10, no. 6, pp. 1301–1311, 2019.

[34] M.-R. Chen, B.-P. Chen, G.-Q. Zeng, K.-D. Lu, and P. Chu, "An adaptive fractional-order BP neural network based on extremal optimization for handwritten digits recognition," *Neurocomputing*, vol. 391, pp. 260–272, May 2020.

[35] J. Zhang and F. B. Gouza, "GADAM: Genetic-evolutionary ADAM for deep neural network optimization," 2018. [Online]. Available: arXiv:1805.07500.

[36] G. Rosa, J. Papa, K. Costa, L. Passos, C. Pereira, and X.-S. Yang, "Learning parameters in deep belief networks through firefly algorithm," in *Proc. IAPR Workshop Artif. Neural Netw. Pattern Recognit.*, 2016, pp. 138–149.

[37] 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. 2019.

[38] M. Gong, J. Liu, A. Qin, K. Zhao, and K. C. Tan, "Evolving deep neural networks via cooperative coevolution with backpropagation," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 1, pp. 420–434, Jan. 2021.

[39] S. Khadka and K. Tumer, "Evolution-guided policy gradient in reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 1188–1200.

[40] M. Wu, W. Su, L. Chen, Z. Liu, W. Cao, and K. Hirota, "Weight-adapted convolution neural network for facial expression recognition in human–robot interaction," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 51, no. 3, pp. 1473–1484, Mar. 2021.

[41] H. M. Albeahdili, T. Han, and N. E. Islam, "Hybrid algorithm for the optimization of training convolutional neural network," *Int. J. Adv. Comput. Sci. Appl.*, vol. 1, no. 6, pp. 79–85, 2015.

[42] H. Badem, A. Basturk, A. Caliskan, and M. E. Yuksel, "A new efficient training strategy for deep neural networks by hybridization of artificial bee colony and limited–memory BFGS optimization algorithms," *Neurocomputing*, vol. 266, pp. 506–526, Nov. 2017.

[43] X. Cui, W. Zhang, Z. Tüske, and M. Picheny, "Evolutionary stochastic gradient descent for optimization of deep neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 6048–6058.

[44] J. Bergstra, D. Yamins, and D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in *Proc. Int. Conf. Mach. Learn.*, 2013, pp. 115–123.

[45] J. Won, J. Park, K. Kim, and J. Lee, "How to train your dragon: Example-guided control of flapping flight," *ACM Trans. Graph.*, vol. 36, no. 6, pp. 1–13, 2017.

[46] S. Lander and Y. Shang, "EVoAE—A new evolutionary method for training autoencoders for deep learning networks," in *Proc. IEEE 39th Annu. Comput. Softw. Appl. Conf.*, vol. 2, 2015, pp. 790–795.

[47] A. Pourchot and O. Sigaud, "CEM-RL: Combining evolutionary and gradient-based methods for policy search," 2018. [Online]. Available: arXiv:1810.01222.

[48] T. Elsken, J.-H. Metzen, and F. Hutter, "Simple and efficient architecture search for convolutional neural networks," 2017. [Online]. Available: arXiv:1711.04528.

[49] 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.

[50] J.-B. Yu, "Evolutionary manifold regularized stacked denoising autoencoders for gearbox fault diagnosis," *Knowl. Based Syst.*, vol. 178, pp. 111–122, Aug. 2019.

[51] A. Baldominos, Y. Saez, and P. Isasi, "Evolutionary convolutional neural networks: An application to handwriting recognition," *Neurocomputing*, vol. 283, pp. 38–52, Mar. 2018.

[52] 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. 2019.

[53] D. Laredo, Y. Qin, O. Schütze, and J.-Q. Sun, "Automatic model selection for neural networks," 2019. [Online]. Available: arXiv:1905.06010.

[54] 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.

[55] F. Assunçao, N. Lourenço, P. Machado, and B. Ribeiro, "DENSER: Deep evolutionary network structured representation," *Genet. Program. Evol. Mach.*, vol. 20, no. 1, pp. 5–35, 2019.

[56] Y. Zhou and G. G. Yen, "Evolving deep neural networks for movie box-office revenues prediction," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, 2018, pp. 1–8.

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

[58] 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)*, 2015, pp. 4979–4983.

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

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

[61] M. Suganuma, M. Ozay, and T. Okatani, "Exploiting the potential of standard convolutional autoencoders for image restoration by evolutionary search," 2018. [Online]. Available: arXiv:1803.00370.

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

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

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

[65] T. Desell, S. Clachar, J. Higgins, and B. Wild, "Evolving deep recurrent neural networks using ant colony optimization," in *Proc. Eur. Conf. Evol. Comput. Comb. Optim.*, 2015, pp. 86–98.

[66] A. Rawal and R. Miikkulainen, "Evolving deep lstm-based memory networks using an information maximization objective," in *Proc. ACM Genet. Evol. Comput. Conf.*, 2016, pp. 501–508.

[67] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical representations for efficient architecture search," 2017. [Online]. Available: arXiv:1711.00436.

[68] Y. Chen *et al.*, "RENAS: Reinforced evolutionary neural architecture search," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 4787–4796.

[69] 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.

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

[71] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang, "Efficient architecture search by network transformation," in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2018, pp. 2787–2794.

[72] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Completely automated CNN architecture design based on blocks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 4, pp. 1242–1254, Apr. 2020.

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

[74] B. Ma, X. Li, Y. Xia, and Y. Zhang, "Autonomous deep learning: A genetic DCNN designer for image classification," *Neurocomputing*, vol. 379, pp. 152–161, Feb. 2020.

[75] 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)*, 2018, pp. 1–6.

[76] A. Rawal and R. Miikkulainen, "From nodes to networks: Evolving recurrent neural networks," 2018. [Online]. Available: arXiv:1803.04439.

[77] L. Rodriguez-Coayahuitl, A. Morales-Reyes, and H. J. Escalante, "Evolving autoencoding structures through genetic programming," *Genet. Program. Evol. Mach.*, vol. 20, pp. 1–28, May 2019.

[78] E. Byla and W. Pang, "DeepSwarm: Optimising convolutional neural networks using swarm intelligence," 2019. [Online]. Available: arXiv:1905.07350.

[79] 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.

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

[81] G. L. F. da Silva, T. L. A. Valente, A. C. Silva, A. C. de Paiva, and M. Gattass, "Convolutional neural network-based PSO for lung nodule false positive reduction on CT images," *Comput. Methods Progr. Biomed.*, vol. 162, pp. 109–118, Aug. 2018.

[82] 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.

[83] Z. Guo *et al.*, "Single path one-shot neural architecture search with uniform sampling," 2019. [Online]. Available: arXiv:1904.00420.

[84] 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)*, 2018, pp. 1–8.

[85] N. Chouikhi, B. Ammar, A. Hussain, and A. M. Alimi, "Bi-level multi-objective evolution of a multi-layered echo-state network autoencoder for data representations," *Neurocomputing*, vol. 341, pp. 195–211, May 2019.

[86] J. Liang, E. Meyerson, and R. Miikkulainen, "Evolutionary architecture search for deep multitask networks," in *Proc. ACM Genet. Evol. Comput. Conf.*, 2018, pp. 466–473.

[87] J. Liang, E. Meyerson, B. Hodjat, D. Fink, K. Mutch, and R. Miikkulainen, "Evolutionary neural automl for deep learning," 2019. [Online]. Available: arXiv:1902.06827.

[88] 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," 2019. [Online]. Available: arXiv:1903.03893.

[89] Q. Song, D. Cheng, H. Zhou, J. Yang, Y. Tian, and X. Hu, "Towards automated neural interaction discovery for click-through rate prediction," in *Proc. 26th ACM SIGKDD Int. Conf. Knowl. Disc. Data Min.*, 2020, pp. 945–955.

[90] S. S. Tirumala, "Evolving deep neural networks using coevolutionary algorithms with multi-population strategy," *Neural Comput. Appl.*, vol. 32, pp. 13051–13064, Feb. 2020.

[91] Y.-H. Kim, B. Reddy, S. Yun, and C. Seo, "NEMO: Neuro-evolution with multiobjective optimization of deep neural network for speed and accuracy," in *Proc. ICML AutoML Workshop*, 2017, pp. 1–9.

[92] M. Gong, J. Liu, H. Li, Q. Cai, and L. Su, "A multiobjective sparse feature learning model for deep neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 12, pp. 3263–3277, Dec. 2015.

[93] 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. 2018.

[94] Z. Lu *et al.*, "Multi-objective evolutionary design of deep convolutional neural networks for image classification," *IEEE Trans. Evol. Comput.*, vol. 25, no. 2, pp. 277–291, Apr. 2020.

[95] B. Wang, Y. Sun, B. Xue, and M. Zhang, "Evolving deep convolutional neural networks by variable-length particle swarm optimization for image classification," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, 2018, pp. 1–8.

[96] X. Xiao, M. Yan, S. Basodi, C. Ji, and Y. Pan, "Efficient hyperparameter optimization in deep learning using a variable length genetic algorithm," 2020. [Online]. Available: arXiv:2006.12703.

[97] T. Elsken, J. H. Metzen, and F. Hutter, "Efficient multi-objective neural architecture search via lamarckian evolution," 2018. [Online]. Available: arXiv:1804.09081.

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

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

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

[101] S. Zaidi, A. Zela, T. Elsken, C. Holmes, F. Hutter, and Y. W. Teh, "Neural ensemble search for performant and calibrated predictions," 2020. [Online]. Available: arXiv:2006.08573.

[102] 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.

[103] F. Mattioli, D. Caetano, A. Cardoso, E. Naves, and E. Lamounier, "An experiment on the use of genetic algorithms for topology selection in deep learning," *J. Elect. Comput. Eng.*, vol. 2019, Jan. 2019, Art. no. 3217542.

[104] 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. Genet. Evol. Comput. Conf.*, 2017, pp. 481–488.

[105] P. R. Lorenzo, J. Nalepa, L. S. Ramos, and J. R. Pastor, "Hyper-parameter selection in deep neural networks using parallel particle swarm optimization," in *Proc. ACM Genet. Evol. Comput. Conf. Companion*, 2017, pp. 1864–1871.

[106] S. Liu, D. C. Mocanu, and M. Pechenizkiy, "On improving deep learning generalization with adaptive sparse connectivity," 2019. [Online]. Available: arXiv:1906.11626.

[107] D. C. Mocanu, E. Mocanu, P. Stone, P. H. Nguyen, M. Gibescu, and A. Liotta, "Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science," *Nat. Commun.*, vol. 9, no. 1, pp. 1–12, 2018.

[108] 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)*, 2018, pp. 1–8.

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

[110] F. Assunção *et al.*, "Automatic design of artificial neural networks for gamma-ray detection," 2019. [Online]. Available: arXiv:1905.03532.

[111] O. Çaylak, A. Yaman, and B. Baumeier, "Evolutionary approach to constructing a deep feedforward neural network for prediction of electronic coupling elements in molecular materials," *J. Chem. Theory Comput.*, vol. 15, no. 3, pp. 1777–1784, 2019.

[112] W. Mao, W. Ding, Y. Xing, and H. Gong, "AmoebaContact and GDFold as a pipeline for rapid de novo protein structure prediction," *Nat. Mach. Intell.*, vol. 2, no. 1, pp. 25–33, 2020.

[113] A. E. Olsson, A. Björkman, and C. Antfolk, "Automatic discovery of resource-restricted convolutional neural network topologies for myoelectric pattern recognition," *Comput. Biol. Med.*, vol. 120, May 2020, Art. no. 103723.

[114] Q. Li, Z. Y. Wu, and A. Rahman, "Evolutionary deep learning with extended kalman filter for effective prediction modeling and efficient data assimilation," *J. Comput. Civil Eng.*, vol. 33, no. 3, 2019, Art. no. 04019014.

[115] A. L. Ellefsen, E. Bjørlykhaug, V. Æsøy, S. Ushakov, and H. Zhang, "Remaining useful life predictions for turbofan engine degradation using semi-supervised deep architecture," *Rel. Eng. Syst. Safety*, vol. 183, pp. 240–251, Mar. 2019.

[116] T. Y. Tan, L. Zhang, and C. P. Lim, "Intelligent skin cancer diagnosis using improved particle swarm optimization and deep learning models," *Appl. Soft Comput.*, vol. 84, Nov. 2019, Art. no. 105725.

[117] P. Liu, M. D. E. Basha, Y. Li, Y. Xiao, P. C. Sanelli, and R. Fang, "Deep evolutionary networks with expedited genetic algorithms for medical image denoising," *Med. Image Anal.*, vol. 54, pp. 306–315, May 2019.

[118] N. Qiang *et al.*, "Modeling task-based fMRI data via deep belief network with neural architecture search," *Comput. Med. Imag. Graph.*, vol. 83, Jul. 2020, Art. no. 101747.

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

[120] S. Khadka *et al.*, "Collaborative evolutionary reinforcement learning," 2019. [Online]. Available: arXiv:1905.00976.

[121] G. Bingham, W. Macke, and R. Miikkulainen, "Evolutionary optimization of deep learning activation functions," 2020. [Online]. Available: arXiv:2002.07224.

[122] P. Cui, B. Shabash, and K. C. Wiese, "EVODNN—An evolutionary deep neural network with heterogeneous activation functions," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, 2019, pp. 2362–2369.

[123] X. Song *et al.*, "ES-ENAS: Combining evolution strategies with neural architecture search at no extra cost for reinforcement learning," 2021. [Online]. Available: arXiv:2101.07415.

[124] S. Hu, R. Cheng, C. He, and Z. Lu, "Multi-objective neural architecture search with almost no training," 2020. [Online]. Available: arXiv:2011.13591.

[125] H. Zhang, Y. Jin, R. Cheng, and K. Hao, "Sampled training and node inheritance for fast evolutionary neural architecture search," 2020. [Online]. Available: arXiv:2003.11613.

[126] Z. Lu, K. Deb, E. Goodman, W. Banzhaf, and V. N. Boddeti, "NSGANetV2: Evolutionary multi-objective surrogate-assisted neural architecture search," in *Proc. Eur. Conf. Comput. Vis.*, 2020, pp. 35–51.

[127] A. K. Anaraki, M. Ayati, and F. Kazemi, "Magnetic resonance imaging-based brain tumor grades classification and grading via convolutional neural networks and genetic algorithms," *Biocybern. Biomed. Eng.*, vol. 39, no. 1, pp. 63–74, 2019.

[128] T. Moriya, T. Tanaka, T. Shinozaki, S. Watanabe, and K. Duh, "Evolution-strategy-based automation of system development for high-performance speech recognition," *IEEE/ACM Trans. Audio, Speech, Language Process.*, vol. 27, no. 1, pp. 77–88, Sep. 2018.

[129] N. Dong, J.-F. Chang, A.-G. Wu, and Z.-K. Gao, "A novel convolutional neural network framework based solar irradiance prediction method," *Int. J. Elect. Power Energy Syst.*, vol. 114, Jan. 2020, Art. no. 105411.

[130] F. Assuncao, D. Sereno, N. Lourenco, P. Machado, and B. Ribeiro, "Automatic evolution of autoencoders for compressed representations," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, 2018, pp. 1–8.

[131] I. Loshchilov and F. Hutter, "CMA-ES for hyperparameter optimization of deep neural networks," 2016. [Online]. Available: arXiv:1604.07269.

[132] T. Silhan, S. Oehmcke, and O. Kramer, "Evolution of stacked autoencoders," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, 2018, pp. 1–8.

[133] M. U. Anwar and M. L. Ali, "Boosting neuro evolutionary techniques for speech recognition," in *Proc. IEEE Int. Conf. Elect. Comput. Commun. Eng. (ECCE)*, 2019, pp. 1–5.

[134] P. Balaprakash, M. Salim, T. Uram, V. Vishwanath, and S. Wild, "DeepHyper: Asynchronous hyperparameter search for deep neural networks," in *Proc. IEEE 25th Int. Conf. High Perform. Comput. (HiPC)*, 2018, pp. 42–51.

[135] Y.-Q. Hu and Y. Yu, "A technical view on neural architecture search," *Int. J. Mach. Learn. Cybern.*, vol. 11, no. 4, pp. 795–811, 2020.

[136] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *J. Mach. Learn. Res.*, vol. 20, no. 55, pp. 1–21, 2019.

[137] P. Ren *et al.*, "A comprehensive survey of neural architecture search: Challenges and solutions," 2020. [Online]. Available: arXiv:2006.02903.

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

[139] Y. Jaafra, J. L. Laurent, A. Deruyver, and M. S. Naceur, "Reinforcement learning for neural architecture search: A review," *Image Vis. Comput.*, vol. 89, pp. 57–66, Sep. 2019.

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

[141] 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, 2020.

[142] 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.

[143] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P.-A. Manzagol, and L. Bottou, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *J. Mach. Learn. Res.*, vol. 11, no. 12, pp. 3371–3408, 2010.

[144] I. J. Goodfellow *et al.*, "Generative adversarial networks," 2014. [Online]. Available: arXiv:1406.2661.

[145] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning," *SIAM Rev.*, vol. 60, no. 2, pp. 223–311, 2018.

[146] X. Zhang, J. Clune, and K. O. Stanley, "On the relationship between the openai evolution strategy and stochastic gradient descent," 2017. [Online]. Available: arXiv:1712.06564.

[147] Z. Beheshti, S. M. H. Shamsuddin, E. Beheshti, and S. S. Yuhaniz, "Enhancement of artificial neural network learning using centripetal accelerated particle swarm optimization for medical diseases diagnosis," *Soft Comput.*, vol. 18, no. 11, pp. 2253–2270, 2014.

[148] T. Moriya, T. Tanaka, T. Shinozaki, S. Watanabe, and K. Duh, "Automation of system building for state-of-the-art large vocabulary speech recognition using evolution strategy," in *Proc. IEEE Workshop Autom. Speech Recognit. Understand. (ASRU)*, 2015, pp. 610–616.

[149] M. Gong, H. Li, E. Luo, J. Liu, and J. Liu, "A multiobjective cooperative coevolutionary algorithm for hyperspectral sparse unmixing," *IEEE Trans. Evol. Comput.*, vol. 21, no. 2, pp. 234–248, Apr. 2017.

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

[151] A. Bhardwaj and A. Tiwari, "Breast cancer diagnosis using genetically optimized neural network model," *Exp. Syst. Appl.*, vol. 42, no. 10, pp. 4611–4620, 2015.

[152] V. K. Ojha, A. Abraham, and V. Snášel, "Metaheuristic design of feedforward neural networks: A review of two decades of research," *Eng. Appl. Artif. Intell.*, vol. 60, pp. 97–116, Apr. 2017.

[153] A. Auger and N. Hansen, "Theory of evolution strategies: A new perspective," in *Theory of Randomized Search Heuristics: Foundations and Recent Developments*. Singapore: World Sci., 2011, pp. 289–325.

[154] E. Conti and V. Madhave, "Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents," 2017. [Online]. Available: arXiv:1712.06560.

[155] J. Lehman, J. Chen, J. Clune, and K. O. Stanley, "Safe mutations for deep and recurrent neural networks through output gradients," 2017. [Online]. Available: arXiv:1712.06563.

[156] B. Colson, P. Marcotte, and G. Savard, "An overview of bilevel optimization," *Ann. Oper. Res.*, vol. 153, pp. 235–256, Apr. 2007.

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

[158] Z. Zhong, J. Yan, W. Wu, J. Shao, and C.-L. Liu, "Practical block-wise neural network architecture generation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 2423–2432.

[159] X. Chu, B. Zhang, R. Xu, and H. Ma, "Multi-objective reinforced evolution in mobile neural architecture search," 2019. [Online]. Available: arXiv:1901.01074.

[160] C. Liu *et al.*, "Progressive neural architecture search," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 19–34.

[161] G. Bender, P.-J. Kindermans, B. Zoph, V. Vasudevan, and Q. Le, "Understanding and simplifying one-shot architecture search," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 550–559.

[162] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "SMASH: one-shot model architecture search through hypernetworks," 2017. [Online]. Available: arXiv:1708.05344.

[163] A. Zhou, B.-Y. Qu, H. Li, S.-Z. Zhao, P. N. Suganthan, and Q. Zhang, "Multiobjective evolutionary algorithms: A survey of the state of the art," *Swarm Evol. Comput.*, vol. 1, no. 1, pp. 32–49, 2011.

[164] S. Ebrahimi, A. Rohrbach, and T. Darrell, "Gradient-free policy architecture search and adaptation," 2017. [Online]. Available: arXiv:1710.05958.

[165] R. Luo, F. Tian, T. Qin, E. Chen, and T.-Y. Liu, "Neural architecture optimization," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 7816–7827.

[166] M. Tan *et al.*, "MnasNet: Platform-aware neural architecture search for mobile," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 2820–2828.

[167] C.-H. Hsu *et al.*, "MONAS: Multi-objective neural architecture search using reinforcement learning," 2018. [Online]. Available: arXiv:1806.10332.

[168] N. Awad, N. Mallik, and F. Hutter, "Differential evolution for neural architecture search," 2020. [Online]. Available: arXiv:2012.06400.

[169] R. Egele, P. Balaprakash, V. Vishwanath, I. Guyon, and Z. Liu, "Agebo-tabular: Joint neural architecture and hyperparameter search with autotuned data-parallel training for tabular data," 2020. [Online]. Available: arXiv:2010.16358.

[170] W. Zheng, C. Gou, L. Yan, and F.-Y. Wang, "Differential-evolution-based generative adversarial networks for edge detection," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. Workshop (ICCVW)*, 2019, pp. 2999–3008.

[171] C. Zhang, K. C. Tan, H. Li, and G. S. Hong, "A cost-sensitive deep belief network for imbalanced classification," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 1, pp. 109–122, Jan. 2018.

[172] J. Liang, S. Gonzalez, and R. Miikkulainen, "Population-based training for loss function optimization," 2020. [Online]. Available: arXiv:2002.04225.

[173] S. Gonzalez and R. Miikkulainen, "Improved training speed, accuracy, and data utilization through loss function optimization," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, 2020, pp. 1–8.

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

[175] H. Guo, X. Zhuang, D. Liang, and T. Rabczuk, "Stochastic groundwater flow analysis in heterogeneous aquifer with modified neural architecture search (nas) based physics-informed neural networks using transfer learning," 2020. [Online]. Available: arXiv:2010.12344.

[176] Y. Wu, G. Zhang, H. Xu, X. Liang, and L. Lin, "Auto-panoptic: Cooperative multi-component architecture search for panoptic segmentation," 2020. [Online]. Available: arXiv:2010.16119.

[177] A. G. Chung, P. Fieguth, and A. Wong, "Assessing architectural similarity in populations of deep neural networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, 2019, pp. 496–498.

[178] B. Baker, O. Gupta, R. Raskar, and N. Naik, "Accelerating neural architecture search using performance prediction," 2017. [Online]. Available: arXiv:1705.10823.

[179] M. S. Abdelfattah, A. Mehrotra, Ł. Dudziak, and N. D. Lane, "Zero-cost proxies for lightweight NAS," 2021. [Online]. Available: arXiv:2101.08134.

[180] C. Wei, Y. Tang, C. Niu, H. Hu, Y. Wang, and J. Liang, "Self-supervised representation learning for evolutionary neural architecture search," 2020. [Online]. Available: arXiv:2011.00186.

[181] Z. Chiba *et al.*, "Intelligent approach to build a deep neural network based IDs for cloud environment using combination of machine learning algorithms," *Comput. Security*, vol. 86, pp. 291–317, Sep. 2019.

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

[183] H. Jin, Q. Song, and X. Hu, "Auto-keras: An efficient neural architecture search system," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Disc. Data Min.*, 2019, pp. 1946–1956.

[184] L. Yang, W. Jiang, W. Liu, H. Edwin, Y. Shi, and J. Hu, "Co-exploring neural architecture and network-on-chip design for real-time artificial intelligence," in *Proc. IEEE 25th Asia South Pac. Design Autom. Conf. (ASP-DAC)*, 2020, pp. 85–90.

[185] H.-P. Cheng *et al.*, "ScaleNAS: One-shot learning of scale-aware representations for visual recognition," 2020. [Online]. Available: arXiv:2011.14584.

[186] H. Rakhshani, L. Idoumghar, S. Ghambari, J. Lepagnot, and M. Brévilliers, "On the performance of deep learning for numerical optimization: An application to protein structure prediction," 2020. [Online]. Available: arXiv:2012.09741.

[187] I. De Falco, G. De Pietro, A. D. 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.

[188] H.-T. L. Chiang, A. Faust, M. Fiser, and A. Francis, "Learning navigation behaviors end-to-end with autorl," *IEEE Robot. Autom. Lett.*, vol. 4, no. 2, pp. 2007–2014, Apr. 2019.

[189] 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.

[190] Z. Liu *et al.*, "Metapruning: Meta learning for automatic neural network channel pruning," 2019. [Online]. Available: arXiv:1903.10258.

[191] K. Yu, C. Sciuto, M. Jaggi, C. Musat, and M. Salzmann, "Evaluating the search phase of neural architecture search," 2019. [Online]. Available: arXiv:1902.08142.

[192] L. Li and A. Talwalkar, "Random search and reproducibility for neural architecture search," in *Proc. Uncertainty Artif. Intell.*, 2020, pp. 367–377.

[193] G. Bender *et al.*, "Can weight sharing outperform random architecture search? an investigation with tunas," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 14323–14332.

[194] S. You, T. Huang, M. Yang, F. Wang, C. Qian, and C. Zhang, "GreedyNAS: Towards fast one-shot NAS with greedy SuperNet," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 1999–2008.

[195] 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.

[196] X. Dong and Y. Yang, "NAS-bench-201: Extending the scope of reproducible neural architecture search," 2020. [Online]. Available: arXiv:2001.00326.

[197] J. Siems, L. Zimmer, A. Zela, J. Lukasik, M. Keuper, and F. Hutter, "NAS-bench-301 and the case for surrogate benchmarks for neural architecture search," 2020. [Online]. Available: arXiv:2008.09777.

[198] Y. Duan *et al.*, "Transnas-bench-101: Improving transferrability and generalizability of cross-task neural architecture search," in *Proc. Int. Conf. Learn. Represent.*, 2021, pp. 1–10.

[199] C. Li *et al.*, "HW-NAS-bench: Hardware-aware neural architecture search benchmark," in *Proc. Int. Conf. Learn. Represent.*, 2021, pp. 1–18.

[200] Y.-C. Chen, C. Gao, E. Robb, and J.-B. Huang, "NAS-DIP: Learning deep image prior with neural architecture search," 2020. [Online]. Available: arXiv:2008.11713.
[201] I. Radosavovic, R. P. Kosaraju, R. Girshick, K. He, and P. Dollár, "Designing network design spaces," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 10428–10436.
[202] Y. Ci, C. Lin, M. Sun, B. Chen, H. Zhang, and W. Ouyang, "Evolving search space for neural architecture search," 2020. [Online]. Available: arXiv:2011.10904.
[203] T. Elsken, B. Staffler, J. H. Metzen, and F. Hutter, "Meta-learning of neural architectures for few-shot learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 12365–12375.
[204] S. Yan, Y. Zheng, W. Ao, X. Zeng, and M. Zhang, "Does unsupervised architecture representation learning help neural architecture search?" in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp 12486–12498.
[205] S. Kaplan and R. Giryes, "Self-supervised neural architecture search," 2020. [Online]. Available: arXiv:2007.01500.
[206] C. Wang, C. Xu, X. Yao, and D. Tao, "Evolutionary generative adversarial networks," *IEEE Trans. Evol. Comput.*, vol. 23, no. 6, pp. 921–934, Dec. 2019.
[207] H. Shu *et al.*, "Co-evolutionary compression for unpaired image translation," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2019, pp. 3235–3244.
[208] R. Houthooft *et al.*, "Evolved policy gradients," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 5400–5409.

Xun Zhou received the B.Eng. degree in communication engineering and the M.Eng. degree in electronics and communications engineering from Xidian University, Xi'an, China, in 2015 and 2018, respectively. He is currently pursuing the Ph.D. degree with the Department of Computer Science, City University of Hong Kong, Hong Kong.

His current research interests include evolutionary algorithms and its application for deep learning.

A. K. Qin (Senior Member, IEEE) received the B.Eng. degree from Southeast University, Nanjing, China, in 2001, and the Ph.D. degree from Nanyang Technology University, Singapore, in 2007.

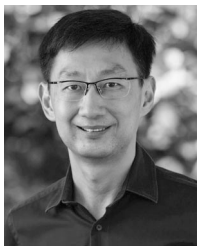
From 2007 to 2017, he was with the University of Waterloo, Waterloo, ON, Canada, with INRIA Grenoble Rhône-Alpes, Grenoble, France, and with RMIT University, Melbourne, VIC, Australia. He joined Swinburne University of Technology, Hawthorn, VIC, Australia, in 2017, where he is currently a Professor. He is currently the Director of Swinburne Intelligent Data Analytics Lab, the Deputy Director of Swinburne Space Technology and Industry Institute, and the Program Lead of Swinburne Data Science Research Institute. His major research interests include machine learning, evolutionary computation, computer vision, remote sensing, services computing, and pervasive computing.

Prof. Qin was the recipient of the 2012 IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION Outstanding Paper Award. He is currently the Chair of the IEEE Neural Networks Technical Committee and the Vice-Chair of the IEEE Emergent Technologies Task Forces on "Collaborative Learning and Optimization" and "Multitask Learning and Multitask Optimization."

Maoguo Gong (Senior Member, IEEE) received the B.Sc. degree in electronic engineering and the Ph.D. degree in electronic science and technology from Xidian University, Xi'an, China, in 2003 and 2009, respectively.

Since 2006, he has been a Teacher with Xidian University, where he was promoted as an Associate Professor and as a Full Professor in 2008 and 2010, with exceptive admission. His research interests are in the areas of computational intelligence with applications to optimization, learning, data mining, and image understanding.

Prof. Gong is currently the Vice Chair of the IEEE Computational Intelligence Society Task Force on Memetic Computing. He is an Executive Committee Member of the Chinese Association for Artificial Intelligence and a Senior Member of the Chinese Computer Federation.



Kay Chen Tan (Fellow, IEEE) received the B.Eng. degree (First Class Hons.) in electronics and electrical engineering and the Ph.D. degree from the University of Glasgow, Glasgow, U.K., in 1994 and 1997, respectively.

He is a Chair Professor with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong. He has published over 300 articles and seven books.

Dr. Tan is currently the Vice-President (Publications) of IEEE Computational Intelligence Society, USA. He has served as the Editor-in-Chief for the *IEEE Computational Intelligence Magazine* from 2010 to 2013 and for the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION from 2015 to 2020, and currently serves as the editorial board member for more than ten journals. He is also an IEEE Distinguished Lecturer Program (DLP) Speaker and the Chief Co-Editor of Springer Book Series on *Machine Learning: Foundations, Methodologies, and Applications*.