Surrogate-Assisted Hybrid-Model Estimation of Distribution Algorithm for Mixed-Variable Hyperparameters Optimization in Convolutional Neural Networks

Jian-Yu Li, *Student Member, IEEE*, Zhi-Hui Zhan, *Senior Member, IEEE*, Jin Xu, Sam Kwong, *Fellow, IEEE*, and Jun Zhang, *Fellow, IEEE*

***Abstract*— The performance of a convolutional neural network (CNN) heavily depends on its hyperparameters. However, finding a suitable hyperparameters configuration is difficult, challenging, and computationally expensive due to three issues, which are 1) the mixed-variable problem of different types of hyperparameters; 2) the large-scale search space of finding optimal hyperparameters; and 3) the expensive computational cost for evaluating candidate hyperparameters configuration. Therefore, this article focuses on these three issues and proposes a novel estimation of distribution algorithm (EDA) for efficient hyperparameters optimization, with three major contributions in the algorithm design. First, a hybrid-model EDA is proposed to efficiently deal with the mixed-variable difficulty. The proposed algorithm uses a mixed-variable encoding scheme to encode the mixed-variable hyperparameters and adopts an adaptive hybrid-model learning (AHL) strategy to efficiently optimize the mixed-variables. Second, an orthogonal initialization (OI) strategy is proposed to efficiently deal with the challenge of large-scale search space. Third, a surrogate-assisted multi-level evaluation (SME) method is proposed to reduce the expensive computational cost. Based on the above, the proposed algorithm is named *s*urrogate-assisted hybrid-model EDA (SHEDA). For experimental studies, the proposed SHEDA is verified on widely used classification benchmark problems, and is compared with various state-of-the-art methods. Moreover, a case study on aortic dissection (AD) diagnosis is carried out to evaluate its performance. Experimental results show that the proposed SHEDA is very effective and efficient for hyperparameters optimization, which can find a satisfactory hyperparameters configuration for the CIFAR10, CIFAR100, and AD diagnosis with only 0.58, 0.97, and 1.18 GPU days, respectively.**

***Index Terms*— Aortic dissection (AD) diagnosis, convolutional neural network (CNN), deep learning, estimation of distribution algorithm (EDA), evolutionary computation (EC), hybrid model, hyperparameters optimization, mixed variable.**

Manuscript received 29 December 2020; revised 15 June 2021 and 3 August 2021; accepted 12 August 2021. Date of publication 20 September 2021; date of current version 3 May 2023. This work was supported in part by the National Key Research and Development Program of China under Grant 2019YFB2102102; in part by the Outstanding Youth Science Foundation under Grant 61822602; in part by the National Natural Science Foundations of China (NSFC) under Grant 62176094, Grant 61772207, and Grant 61873097; in part by the Key-Area Research and Development of Guangdong Province under Grant 2020B010166002; in part by Guangdong Natural Science Foundation Research Team under Grant 2018B030312003; in part by Hong Kong GRF-RGC General Research Fund under Grant 9042816 (CityU 11209819); and in part by the Project from Tencent. *(Corresponding authors: Zhi-Hui Zhan; Jun Zhang.)*

Jian-Yu Li and Zhi-Hui Zhan are with the School of Computer Science and Engineering, South China University of Technology, Guangzhou 510006, China, also with the Pazhou Laboratory, Guangzhou 510330, China, and also with Guangdong Provincial Key Laboratory of Computational Intelligence and Cyberspace Information, South China University of Technology, Guangzhou 510006, China (e-mail: zhanapollo@163.com).

Jin Xu is with the Data Quality Team, WeChat, Tencent Inc., Shenzhen 518052, China.

Sam Kwong is with the Department of Computer Science, City University of Hong Kong, Hong Kong.

Jun Zhang is with Hanyang University, Ansan 15588, South Korea, also with Zhejiang Normal University, Jinhua 321004, China, and also with Chaoyang University of Technology, Taichung 413310, Taiwan.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TNNLS.2021.3106399.

Digital Object Identifier 10.1109/TNNLS.2021.3106399

I. Introduction

CONVOLUTIONAL neural network (CNN), as one of the most efficient deep learning models [1], plays a vastly important role in various artificial intelligence applications like Go playing [2]. By using convolution operations [3], CNNs can extract meaningful features from the input data [4]. Such powerful learning and expression abilities result in the great success of CNNs in various learning tasks and application problems from different fields, such as learning physical properties of liquid crystals [5], electroencephalography signal analysis [6], medical diagnostic [7], and image recognition [8]. Hence, with the rapid development of artificial intelligence in these years, researches into CNN models and applications have attracted increasing attention [9].

Although CNNs have obtained promising results, the structure of CNN is actually very complex, and how to design a suitable CNN model for solving a specific problem remains a challenging issue [10]. Moreover, most reported efficient CNN models are designed by experienced network designers and researchers, and are fine-tuned through tedious trial-and-error experiments, which are very inefficient and computationally expensive.

Therefore, recent studies have started to consider more intelligent, automatic, and efficient ways of obtaining better CNN models, which result in the CNN optimization researches [11], i.e., consider finding the best CNN hyperparameters as an optimization problem and then design powerful algorithms to solve it. In this direction, many algorithms have been proposed and obtained promising results [12], such as using reinforcement learning [11], Bayesian optimization [12], and

evolutionary computation (EC) approaches like large-scale evolution [13] and hierarchical evolution [14]. As a branch of the artificial intelligence for knowledge discovering [15], the EC algorithms, including genetic algorithm (GA) [16], particle swarm optimization (PSO) [17], differential evolution [18], genetic programming (GP) [19], and estimation of distributed algorithm (EDA) [20], are efficient for finding optimal solutions for high-complexity problems. Furthermore, the EC algorithms can also cooperate with network science [21] to obtain better optimization ability [22]. Therefore, among the approaches for solving CNN optimization problems [23], the EC-based approaches have obtained great success and attracted increasing attention [24]. For example, Real *et al.* [13] proposed a large-scale neuro-evolutionary algorithm to evolve network structures for finding the best CNN model. Suganuma *et al.* [19] introduced the Cartesian GP approach to automatically find CNN architectures, which has shown to be efficient on widely used datasets. Sun *et al.* [23] proposed a novel GA-based algorithm called CNN-GA to automatically discover the best architecture of CNN. Besides, other EC-based approaches have also been proposed for CNN optimization and obtained promising results [25].

However, solving the CNN hyperparameters optimization problem is still challenging due to the following three difficulties, including the mixed-variable problem, large-scale search space, and expensive computational cost. First, CNN hyperparameters always consist of different variable types [26]. For example, the numbers of convolutional kernels are integer; the probability of dropout is a real number; while the kinds of activation functions are discrete. Such mixed-variable characteristic poses great challenges on the search efficiency of EC algorithms. For instance, the EC operators designed for continuous problems may not work well for the optimization of the kinds of activation function, while the operators for discrete optimization may be not efficient for finding suitable dropout probability. Therefore, how to efficiently handle different variable types simultaneously becomes an essential issue for improving the CNN optimization results. Second, the search space of hyperparameter configurations can be very large, especially when CNNs are deep and with many layers. As a result, it is greatly difficult to evenly explore the potential regions of the whole search space. In such cases, the optimization results may only find local optimal solutions. Third, as the fitness evaluation (FE) of candidate CNNs, i.e., the training and validation procedures, is very computationally expensive, the optimization efficiency of EC approaches may greatly deteriorate when the required expensive FEs increase [27].

Therefore, this article focuses on these three challenging issues in CNN hyperparameters optimization and proposes a novel EDA to deal with the three difficulties. The motivations and major contributions of this article are summarized and justified as Fig. 1 and are described as follows.

First, a hybrid-model EDA is proposed to efficiently deal with the mix-variable difficulty of the CNN hyperparameter. This is inspired by the fact that the EDA usually samples new solutions via the probabilistic model, where different probabilistic models are suitable for different types of variables. Therefore, this article proposes a mixed-variable encoding scheme to encode the CNN hyperparameters and proposes an adaptive hybrid-model learning (AHL) strategy to efficiently learn the hybrid-model for the mixed-variable optimization. It should also be noted that, to the best of our knowledge, this article is the first that uses EDA with a hybrid-model

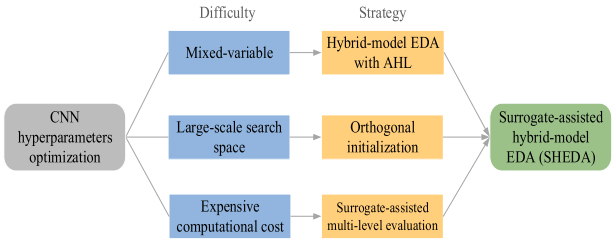

Fig. 1. Motivations and contributions of the proposed algorithm, which incorporates three strategies to handle three difficulties in the CNN hyperparameters optimization.

for CNN hyperparameters optimization. Although recently Zheng *et al.* [28] introduced distribution learning for CNN optimization, they only learn a discrete model for optimizing the best combination of pre-defined network cells (i.e., blocks containing several layers with fixed settings based on prior knowledge). Therefore, the proposed hybrid-model EDA, which learns a hybrid model for optimizing different types of hyperparameters for each network layer, can be more efficient and flexible for CNN optimization, which can inspire and benefit the development of evolutionary deep learning community.

Second, an orthogonal initialization (OI) strategy is proposed to efficiently deal with the large-scale search space challenge of CNN hyperparameters optimization. The OI strategy can help initialize solutions to cover all possible choices of each variable evenly in the initialization procedure, so that the algorithm can explore the whole large search space more evenly and efficiently to find the global optimal solution.

Third, a surrogate-assisted multi-level evaluation (SME) method is proposed to reduce the expensive computational cost in optimizing the CNN hyperparameters. In SME, two levels of evaluation are combined to strike a better balance between optimization accuracy and computational cost. In addition, a modified localized data generation (MLDG) method is designed to perform data generation for building more accurate surrogates.

With the above strategies and methods, the proposed algorithm is named surrogate-assisted hybrid-model EDA (SHEDA) for simplicity. To verify the efficiency and effectiveness of SHEDA, it is investigated on challenging and difficult classification problems, i.e., the 10-category classification problem (CIFAR10) [29] and the 100-category classification problem (CIFAR100) [29]. Moreover, a great deal of state-of-the-art methods, including manually designed, non-EC-based optimization, and EC-based optimization, are adopted as the competitors to compare with the SHEDA. In addition, ablation experiments and parameter analysis are also performed to provide in-depth observations of the SHEDA. Lastly, a case study on aortic dissection (AD) diagnosis has also been carried out to verify the effectiveness of SHEDA.

The rest of this article is organized as follows: Section II briefly introduces the CNN, EDA, and related work, while Section III details the proposed SHEDA and its components. Experiments, including the settings, comparisons, and discussions, are provided in Section VI. Finally, Section V gives the conclusion.

II. Background and Related Work

This section provides a brief introduction about the background knowledge and related work. To begin with, CNN and

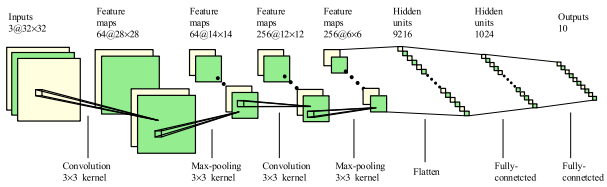

Fig. 2. Example of CNN with two convolutional layers, two pooling layers, and two fully connected layer for a CIFAR10.

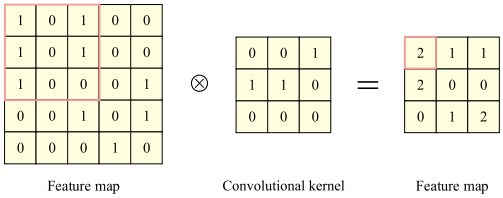

Fig. 3. Illustration of a convolution operation with a 3 × 3 kernel on a 5 × 5 feature map to output a 3 × 3 feature map.

EDA will be introduced in part A and B of this section. After this, part C will review some representative related work.

A. Convolutional Neural Network

Generally speaking, CNN is a kind of forward neural network, which includes an input layer, some convolutional layers, pooling layers, fully connected layers, and an output layer [1]. Fig. 2 shows a CNN example with two convolutional layers, two pooling layers, and two fully connected layers. In Fig. 2, the input image is 32 × 32 with three channels. After the sequential transformation of the first convolutional layer with 64 3 × 3 kernels, the first pooling layer with size 2 × 2, the second convolutional layer with 256 3 × 3 kernels, and the second pooling layer with size 2 × 2, the input map accordingly becomes a 28 × 28 map with 64 channels, then a 14 × 14 map with 64 channels, a 12 × 12 map with 256 channels, and a 6 × 6 map with 256 channels. After that, each pixel of the final 6 × 6 map in each channel can be regarded as a useful exacted feature value. Consequently, the pixels in the 6 × 6 map with 256 channels are flatted to be a 9216 (i.e., 6 × 6 × 256) dimensional vector as the input of the fully connected layer. Finally, the fully connected layer with 1024 hidden neurons will learn the best mapping from the 9216-dimensional vector to a 10-D output for the ten-category classification problem.

The key characteristic of CNN when compared with other network models is its convolutional layer. The convolutional layer can extract useful image features through convolution operation with various convolutional kernels. In detail, the convolution operation performs the bitwise multiplication of the convolutional kernels and the local regions of the input feature map, and then sums up the results as the corresponding value in the output feature map. For example, Fig. 3 shows a convolution operation with a 3 × 3 kernel on a 5 × 5 feature map, which outputs a 3 × 3 feature map.

In the literature, many interesting and efficient network architectures about the convolutional layer have been proposed manually. For example, one of the most widely used architecture units is the residual block proposed in ResNet [8]. In the residual blocks, the input can have two paths connected to the output, one is traditionally through the convolutional layer and the other is directly connected to the output. Due to the high efficiency, many variants and improved versions of the residual block have been proposed and studied, e.g., wider residual block [9] and DenseBlock [10]. Considering the verified high efficiency of ResNet-like architectures, this article adopts the ResNet as the base network and then investigates whether the proposed SHEDA can find CNNs that are better than those manually designed in the ResNet family.

B. Estimation of Distributed Algorithm

EDA is a kind of efficient EC algorithms based on statistical learning theory [20]. The main procedures of EDA are as follows. To begin with, EDA initializes a population of individuals, where each individual is a candidate solution to the optimization problem. After the evaluation of all the individuals, EDA selects N_b individuals with better fitness to estimate the probabilistic model through the statistical learning method, so as to obtain the distribution of potential optimal solutions. Then the EDA can sample a population of SN new individuals by the learned probabilistic model. Generally speaking, the number of better individuals for learning model, i.e., N_b, and the number of new sampling individuals, i.e., SN, will be set with different values. After sampling new individuals, EDA evaluates the fitness of new individuals and goes to the next generation. The above procedures will be performed repeatedly until the stop criteria are met. Finally, the EDA outputs the best-found solution. As the learning method and probabilistic model can be adjusted to fit different types of variables, the EDA may be a suitable choice for solving the mixed-variable problem in CNN optimization, which is also a motivation of this article.

C. Related Work

This part briefly reviews some representative-related work about using EC algorithms for solving the CNN optimization problem [26].

As GA is a kind of conventional and efficient EC algorithms for optimization problems, many GA-based approaches have been proposed and studied. Sun *et al.* [23] proposed a novel algorithm called CNN-GA to automatically discover the best architecture of CNN. Considering that the network blocks designed in ResNet and DenseNet are useful for extracting image features, enhanced GA has also been proposed to automatically construct CNNs based on blocks [24], which has obtained promising results. In addition, Yang *et al.* [30] proposed to use a multiobjective GA algorithm for obtaining smaller and more accurate CNNs.

Besides GA, other powerful EC algorithms, e.g., GP [19], differential evolution [31], and their combinations can be also used [3]. For example, Suganuma *et al.* [19] introduced the Cartesian GP approach to automatically find CNN architectures, which has shown to be efficient on widely used datasets. Xue *et al.* [32] proposed a hybrid algorithm with PSO and two differential evolutions for evolving CNN-LSTM models for inventory time series prediction. Wang *et al.* [33] proposed variable-length PSO for optimizing CNNs, while Fielding *et al.* [34] used PSO to find different CNNs for model ensemble. In addition, Wu *et al.* [35] proposed a multiobjective PSO to prune the CNN networks, which can reduce 80% of the parameters of the CNN without losing too much accuracy. Besides, Guo *et al.* [36] proposed a distributed PSO with multiple GPU cards for efficiently optimizing the hyperparameters of CNNs.

TABLE I
SETTINGS OF MIXED-VARIABLE ENCODING FOR WIDELY SEEN HYPERPARAMETERS WHEN DESIGNING CNNS

Hyperparameters	Available Choices	Encoding Way	Search Space After Encoding	Initialization Range Space
Number of kernels in convolution layer	{1, 2, 3, …, +∞}	continuous variable	[1, +∞)	[64, 512]
Number of neurons in fully connected layer	{1, 2, 3, …, +∞}	continuous variable	[1, +∞)	[64,1024]
Kernel size of the convolution layer	{3×3, 5×5, 7×7, 9×9}	discrete variable	{0,1,2,3}	{0,1,2,3}
Kind of activation function	{Relu, Sigmoid, Tanh}	discrete variable	{0,1,2}	{0,1,2}
Kind of pooling layer	{Max-pooling, Average-pooling}	discrete variable	{0,1}	{0,1}

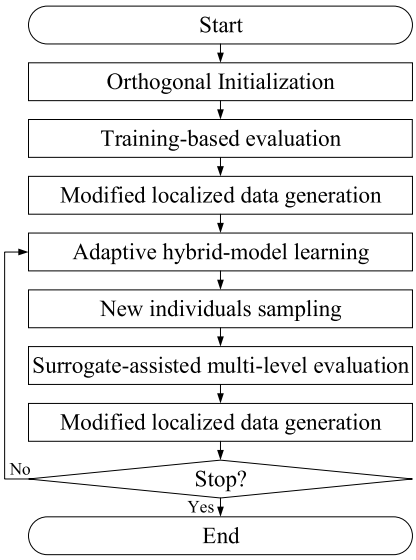

Fig. 4. General flowchart of the proposed algorithm.

Also, some researchers have proposed and studied the neuro evolution for optimizing CNNs. For instance, Real *et al.* [13] proposed a large-scale neuro-evolutionary algorithm to evolve network structures for finding the best CNN model. The proposed algorithm has nearly no restrictions on the search space, e.g., with arbitrary skip connections and no fixed depth, and it begins with simple and poorly performing models and then evolves to more complex CNN models. The experimental results show that this approach can find CNNs with great performance when given enough computational budgets. Also, with the idea of neuro evolution, Hadjiivanov and Blair [37] proposed a neuro evolution framework with an extended genome encoding method and corresponding crossover operator for evolving CNN models.

III. PROPOSED ALGORITHM

By encoding the CNN hyperparameters into a mix-variable vector, the SHEDA will optimize the hyperparameters following the procedure shown in Fig. 4. With the mixed-variable encoding scheme, the SHEDA can use the AHL strategy to learn different probabilistic models to optimize mixed-variable problem efficiently. Moreover, the SHEDA adopts the OI strategy in the initialization to cover the large search space more evenly, and utilizes the SME method with MLDG in the FE to reduce the expensive computational cost. In the following, the mixed-variable encoding scheme, the AHL strategy, the OI strategy, and the SME with the MLDG method will be described one by one in detail.

A. Mixed-Variable Encoding Scheme

In EDA, each sampled individual represents the hyperparameters of a CNN, where each dimension of the individual corresponds to a hyperparameter of the CNN. As each hyperparameter has its own meaning, different dimensions of an individual naturally have different search ranges and constraints. For example, some hyperparameters should be set as integers with a large range, e.g., the number of convolutional kernels, while some hyperparameters only have several discrete choices, e.g., the kinds of activation functions. In such a mixed-variable optimization problem, learning a probabilistic model for describing the distribution of promising individuals is difficult. Therefore, the mixed-variable encoding scheme is adopted to handle the mixed-variable problem.

In detail, the encoding scheme encodes the hyperparameters of CNN into two types of variables, i.e., continuous and discrete variables. By doing so, the EDA can build probabilistic models for different variable types separately. For better illustrations, Table I provides the settings of the mixed-variable encoding scheme for widely seen CNN hyperparameters. As given in Table I, during the optimization, the variables that have many available choices will be encoded as continuous variables, while the variables with only several choices will be regarded as discrete variables. For instance, the number of kernels in the convolutional layer and the number of neurons in the fully connected layer will be encoded as continuous variables because they have many possible choices, while the kernel size of the convolutional layer, the kind of activation functions, and the kind of pooling layer will be regarded as discrete variables because they only have several choices.

Although the variables that represent the number of kernels or hidden neurons are required to be integers, processing them as continuous variables can be more flexible and efficient in the evolution process. This has two reasons. First, when searching for the optimal number of kernels and neurons, the possible search space is in fact infinite without a prior-known upper bound. In other words, the optimal choice can be any integer in $[1, 2, \ldots, +\infty)$, which is too large and very inefficient to be encoded by a set of finite integers. Second, by processing the number of kernels and neurons as continuous variables, the SHEDA can estimate a corresponding continuous probabilistic model easily and accurately, which benefits the individual sampling and improves optimization efficiency. Therefore, these variables will be processed as continuous numbers during the optimization and they will be rounded off to be integers when setting the corresponding hyperparameters of the CNN. As for the hyperparameters with only several choices, this articles encodes them as discrete variables so that the EDA can build discrete probabilistic models to estimate the probability distribution of the promising choices.

For simplicity, the following contents denote the continuous and discrete variables in the entire individual solution as cx and dx, respectively. In addition, to make the encoding easier to understand, a simple example is given here. Considering the optimization of five hyperparameters of a CNN, including the number of kernels in two convolutional layers, the number

of neurons in one fully connected layer, and the kernel size of the two convolutional layers, the cx is 3-D and the dx is 2-D. Then, according to Table I, an individual with $cx = (32, 64, 256)$ and $dx = (1, 3)$ means the number of kernels in the first and second convolutional layers are 32 and 64, the number of neurons in the fully connected layer is 256, and the kernel sizes of the first and second convolutional layers are 5×5 and 9×9 (i.e., the first and third choice for the corresponding hyperparameters of kernel size), respectively.

B. AHL Strategy

After solution encoding, a suitable learning strategy should be designed to estimate the optimal continuous and discrete distributions of promising variables, i.e., the cx and dx, respectively. To improve estimation and resampling efficiency, the AHL strategy is therefore proposed. The AHL utilizes a fitness-weighted method to adaptively learn models for cx and dx separately and then combine them to get the hybrid model for sampling new individuals. To be specific, the fitness-weighted value corresponding to the individual i, say w_i, can be calculated as

$$w_i = \frac{fit_i}{\sum_{j=1}^{N_b} fit_j} \tag{1}$$

where N_b is the number of the individuals (with better fitness value) for learning probabilistic models, and fit_i is the corresponding fitness value (i.e., classification accuracy) of individual i. With these weights, the probabilistic model can be adaptively adjusted to pay more attention to individuals with better fitness to better capture the distribution of promising variables adaptively and efficiently. Moreover, with the fitness-weighted value used in every generation, the new improvement on the fitness of any individual can be adaptively considered to update the hybrid model adaptively and efficiently.

For cx, the weights are combined with the Gaussian distribution to learn the most suitable continuous probabilistic model. In detail, for the jth variable dimension of cx, the mean value and standard deviation value of the probabilistic model, i.e., μ_j and σ_j, can be calculated as

$$\mu_j = \sum_{i=1}^{N_b} w_i \cdot cx_{ij} \tag{2}$$

$$\sigma_j = \sqrt{\frac{\sum_{i=1}^{N_b} (cx_{ij} - \mu_j)^2}{N_b}} \tag{3}$$

where cx_{ij} is the jth dimension of the continuous part of individual i (i.e., cx_i). Then, the jth dimension of the continuous part of a new individual (i.e., cx_{nj}) can be sampled according to the corresponding Gaussian distribution as

$$cx_{nj} \sim N\left(\mu_j, \sigma_j^2\right). \tag{4}$$

For discrete variables, the probabilistic models are also updated with the weighted value but are different from that for continuous variables. To be specific, for the jth dimension of the discrete part of a new individual, the probability of choosing the kth choice (say $prob_{jk}$) is calculated as

$$\begin{aligned} prob_{jk} &= \sum_{i=1}^{N_b} w_i \cdot I(dx_{ij}, \text{choice}_{jk}) \\ &= \frac{\sum_{i=1}^{N_b} fit_i \cdot I(dx_{ij}, \text{choice}_{jk})}{\sum_{i=1}^{N_b} fit_i} \end{aligned} \tag{5}$$

where dx_{ij} is the value of the jth dimension of the discrete part of individual i, and $I(a, b)$ is the function that checks whether a equals to b, which can be written as

$$I(a, b) = \begin{cases} 1, & \text{if } a == b \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

It should be noted that for any discrete variable (e.g., the jth discrete variable) with K available choices, the probability of all choices satisfies the following equation:

$$\sum_{k=1}^{K} prob_{jk} = 1. \tag{7}$$

Then, the discrete variables of the new individual (i.e., dx_n) can be sampled by the roulette based on probability prob. For example, assuming that the jth variable in dx_n is with three choices, e.g., $prob_{j_1} = 0.2$, $prob_{j_2} = 0.3$, and $prob_{j_3} = 0.5$, the jth discrete variable of newly sampled individual will select the first choice with a probability of 0.2, the second choice with a probability of 0.3, and the third choice with a probability of 0.5.

C. OI Strategy

As the search space can be very large when searching for the optimal CNN hyperparameters configuration, the random initialization strategy may be not efficient to cover the whole search space evenly. For example, a variable may have five choices, while there may be only three choices that have been adopted by the random initialization. This may mislead the EDA to learn an incomplete or even wrong probabilistic model for describing the distribution of promising choices. Therefore, the OI strategy is proposed.

Algorithm 1 Orthogonal Initialization Strategy

Input: Dc-the dimension of continuous variables;
D_d- the dimension of discrete variables;
U-the upper bound of continuous variables;
L-the lower bound of continuous variables.
Output: cx-the initialized continuous variables;
dx-the initialized discrete variables.

Begin
  Configure the levels for each dimension in Dc and D_d;
  $\boldsymbol{OA} \leftarrow$ orthogonal array generated by orthogonal experimental design;
  $Row \leftarrow$ the number of rows in $\boldsymbol{OA}$;
  For $i = 1$ to Row **Do**
    For $j = 1$ to Dc **Do**
      If $\boldsymbol{OA}(i, j) == 1$ **Then**
        $cx_{ij} \leftarrow$ uniform distribution over $[L, (U + L)/2]$;
      Else
        $cx_{ij} \leftarrow$ uniform distribution over $[(U + L)/2, U]$;
      End If
    End for
    For $j = 1$ to D_d **Do**
      $dx_{ij} \leftarrow$ the value of $\boldsymbol{OA}(i, Dc+j)$;
    End for
  End for
End

The OI strategy is based on the orthogonal experimental design [38]. For an experiment with D factors and each factor has several available levels, the orthogonal experimental design can generate an orthogonal array (denoted as $\boldsymbol{OA}$) with Row rows and D columns, where each row is a different combination of levels for D factors and each column value of each row represents the adopted level for the corresponding factor. It should be noted that the orthogonal experimental design can determine the Row value automatically according to the number of factors and the number of available levels for each factor, where the Row value can be reasonably small. For example, if there are four factors and each factor has three available levels, the Row can be set as nine [38]. Furthermore, the generated $\boldsymbol{OA}$ has a useful characteristic that all the available levels of each factor will occur evenly in $\boldsymbol{OA}$. Therefore, the experiment design according to $\boldsymbol{OA}$ can use a small number of test samples to evenly cover all available levels of different factors. Then, considering the dimensions as factors and the choices of each dimension as levels, the OI strategy can perform initialization to cover all choices of different dimensions evenly with a small number of individuals, as described in the following. More information about the orthogonal experimental design can refer to [38].

In the OI strategy, to initialize the continuous variables evenly in the large space, the continuous variables are temporarily considered as discrete variables with two choices. To be specific, the initialization range of each continuous variable is divided into two equal parts, i.e., the search range $[L, U]$ will be divided into $[L, (L+U)/2]$ and $[(L+U)/2, U]$, where each part corresponds to one choice. The first choice means that the variable will be initialized within the lower search range $[L, (L+ U)/2]$, while the second choice means that the initialization is performed in the upper search range $[(L + U)/2, U]$. For example, assume that the whole initialization range for a continuous variable is [64, 512], the first choice will randomly initialize the variable in [64, 288], while the second choice will be in [288, 512]. Then the two choices for each continuous variable can be considered as two levels of each dimension factor in orthogonal experimental design.

As for discrete variables dx, the number of levels can be directly set as the number of their available choices. For example, if a discrete variable has three choices, the corresponding level numbers for this dimension factor in the orthogonal experimental design are set as three. Note that the number of levels for different dimensions may be different because the number of available choices for different discrete variables may differ.

By doing the above level configurations, the orthogonal experimental design can generate the orthogonal array $\boldsymbol{OA}$ for all the continuous and discrete variables of the CNN hyperparameters together, where the jth column value of the ith row in $\boldsymbol{OA}$ is the initialization choice of the jth dimension of the ith individual. Therefore, by generating the $\boldsymbol{OA}$ via orthogonal experimental design, OI can initialize Row individuals according to each row of the $\boldsymbol{OA}$, whose pseudo code is given in Algorithm 1. In the implementation, without loss of generality, this article uses the open-source Python implementation of orthogonal experimental design to generate $\boldsymbol{OA}$ (see: https://github.com/lovesoo/OrthogonalArrayTest).

D. SME Method

As the training procedure of a CNN should be repeated until its classification accuracy converges, evaluating its fitness (i.e., classification accuracy) is very computationally expensive. Moreover, even though there are enough computational resources and time to train the CNN with many epochs until convergence, the CNN with the best fitness value (i.e., performance) on the validation set is not guaranteed to still have the best performance when testing on the test set because the validation data and the test data are different.

Therefore, instead of training the CNN with a large number of epochs until it converges, the SME method is proposed to evaluate and compare different CNN models more efficiently. In fact, in EDA or other EC algorithms, it is not necessary to know the actual fitness of individuals during the evolution. Instead, what EC algorithms need in the optimization procedure is how to figure out which individuals are better and can survive into the next generation. Therefore, a relaxation evaluation method is also sufficient if it can distinguish the performance of different individuals.

In the proposed SME, there are two evaluation methods with different accuracies and time costs, one is to train CNN with several (e.g., T) epochs and the other is to employ the surrogate instead of the training. The surrogates in this articles refer to the machine learning models that receive individuals as input and output the predicted fitness of these individuals. Commonly used surrogates in the surrogate-assisted EC community include the Kriging model [39] (also called Gaussian process model [40]), radial basis function neural network for approximation [41] and classification [42], and random forest [43]. To be simple, this article adopts the Kriging surrogate model. The detailed configurations of adopted surrogates in the implementation will be described in the experimental part.

For the above two evaluation methods, this article for simplicity denotes them as training-based evaluation and surrogate-based evaluation, respectively. The advantages of cooperating training-based evaluation and surrogate-based evaluation are as follows. It has been shown that training CNN with several epochs is enough to figure out which CNN is better [44]. Therefore, the training-based evaluation method can achieve nearly accurate FE with much less time cost. However, it should also be noted that training a CNN with serval epochs will still be a bit time consuming, especially when the CNN model is deep and large. Therefore, the surrogate-based evaluation is also needed. Although the surrogate-based evaluation may be not as accurate as that of the training-based evaluation, it requires much less computational resources and time costs, providing a computationally efficient method to complement the training-based evaluation. Therefore, with these two evaluation methods, the SME can make a great balance between the comparison accuracy and computational time cost.

The main idea of the SME is to first evaluate CNNs with surrogate-based evaluation and second only those promising CNNs will be evaluated by the training-based evaluation, resulting in a two-level evaluation. The pseudo code of SME is given in Algorithm 2. First, a surrogate is built based on all the evaluated individuals stored in the archive (denoted as arch). Note that the archive arch stores all the individuals that have been evaluated by the training-based evaluation. Second, the surrogate predicts the fitness values of the newly generated individuals. Third, if the predicted fitness value of a new

Algorithm 2 Surrogate-Assisted Multi-Level Evaluation

Input: $\mathcal{D}_{train}$ -the dataset for training CNNs;
$\mathcal{D}_{valid}$ - the dataset for validating CNNs.
P_{arch}-the population of individuals in *arch*;
fit_{arch}-the fitness of individuals in *arch*;
P-the population of individuals to be evaluated;
NP-the number of individuals in population P
T-the number of epochs for training.

Output: P_{eval}-the set of evaluated individuals;
fit_{eval}-the fitness of evaluated individuals.

1: **Begin**
2: Build a surrogate model with P_{arch} and fit_{arch}.
3: $sfitness \leftarrow$ fitness of P predicted by surrogate;
4: $afitness \leftarrow$ the average fitness of fit_{arch};
5: $P_{eval} \leftarrow$ empty set;
6: $fit_{eval} \leftarrow$empty set;
7: $ri \leftarrow$ a random integer in $\{1, 2, \ldots, NP\}$;
8: **For** each individual P_i in P **Do**
9: **If** $sfitness_i > afitness$ or $ri == i$ **Then**
//The ">" means "better than"
10: $Net \leftarrow$ build a CNN according to the variables in P_i;
11: $Net \leftarrow$ train Net with T epochs on $\mathcal{D}_{train}$;
12: $accuracy \leftarrow$ validate Net on $\mathcal{D}_{valid}$;
13: $fit_{eval} \leftarrow fit_{eval} \cup \{accuracy\}$;
14: $P_{eval} \leftarrow P_{eval} \cup \{P_i\}$;
15: **End If**
16: **End for**
17: **End**

individual is better than the average fitness of the individuals in the arch, this new individual will be regarded as a promising individual and will be evaluated by the training-based evaluation. To ensure that there is at least one individual will be evaluated by training-based evaluation in every generation, a random individual will be also selected for the training-based evaluation, as shown in line 7 of Algorithm 2. After the training-based evaluation, the SME finally outputs those individuals evaluated by the training-based evaluation, where these individuals will be stored to update the arch.

E. MLDG Method

As the number of evaluated individuals may be not enough for building an accurate surrogate, the MLDG method is proposed to increase data amount. The MLDG is inspired by the localized data generation method in the surrogate-assisted EC community [16]. By generating synthetic data within the localized neighborhood of evaluated individuals, the localized data generation is originally proposed to alleviate the data shortage and build more accurate surrogate models. At the same time, it can reduce the need for expensive evaluations to some extent. However, the localized neighborhood in original localized data generation is calculated based on the fixed upper bound, while the CNN optimization problem considered in this article does not have predefined upper bounds for some continuous variables. Therefore, a suitable localized neighborhood should be designed in this article, which is named relatively localized neighborhood in the following. To be specific, for a continuous variable cx_{ij}, its relative localized neighborhood is defined as $[cx_{ij} \times 0.99, cx_{ij} \times 1.01]$. By doing so, when cx_{ij} increases, its localized neighborhood can also increase adaptively, and vice versa. As for the discrete variables, it is difficult to define the localized neighborhood because two different choices for a discrete variable may have significantly different influences. Therefore, this article does not consider their neighborhood herein, and the discrete variables of the generated synthetic individual will keep the same values as those in the corresponding original evaluated individual. That is, for the jth dimension of the generated discrete variable. say dx_{gj}, it can be set as the same as the original discrete variable, say dx_{ij}, i.e., $dx_{gj} = dx_{ij}$.

With the above settings, the synthetic individual generated randomly in the relatively localized neighborhood will be similar to the original individual. Therefore, the fitness of the synthetic individual can be set the same as that of the original individual. By doing so, the data for training surrogates can increase without accessing the time-consuming training process. In addition, it is suggested in the original localized data generation that the data generation should be performed to about half of the evaluated data, so as to control the number of the generated data [16]. Therefore, for each evaluated individual, there will be a possibility as 0.5 that a new individual will be generated within its localized neighborhood. The complete pseudo code of MLDG is provided in Algorithm 3.

Algorithm 3 Modified Localized Data Generation

Input: P-the population before data generation;
fit- the corresponding fitness of individuals in P.
Dc-the dimension of continuous variables;

Output: P_{after}-the population after data generation;
fit_{after}-the corresponding fitness after data generation.

1: **Begin**
2: $P_{new} \leftarrow$ empty set; $fit_{new} \leftarrow$empty set;
3: **For** each individual P_i in P **Do**
4: **If** rand$(0, 1) < 0.5$ **Then**
5: $P_{gen} \leftarrow$a generated data in the localized neighborhood of P_i;
6: $fit_{gen} \leftarrow$ the fitness of P_i;
7: $P_{new} \leftarrow P_{new} \cup P_{gen}$; $fit_{new} \leftarrow fit_{new} \cup fit_{gen}$;
8: **End If**
9: **End for**
10: $P_{after} \leftarrow P \cup P_{new}$; $fit_{after} \leftarrow fit \cup fit_{new}$;
11: **End**

F. Complete Algorithm

The pseudo code of the complete SHEDA is shown in Algorithm 4. After encoding the CNNs, the SHEDA mainly has the following five steps.

Step 1: SHEDA initializes individuals through OI strategy (Algorithm 1) and evaluates them by the training-based evaluation. Then the MLDG (Algorithm 3) is performed on these evaluated individuals to generate synthetic individuals. Both the original and generated individuals will be stored in the arch.

Step 2: The N_b best individuals in the arch (including the generated synthetic individuals) are used to update the probabilistic models, i.e., to compute the μ and σ for continuous variables [i.e.,(2) and(3)] and calculate the *prob* for discrete variables [i.e., (5)].

Algorithm 4 Complete Algorithm

Input: $\mathcal{D}_{train}$-the training dataset;
Dc-the dimension of continuous variables;
D_d- the dimension of discrete variables;
U-the upper bound of continuous variables;
L-the lower bound of continuous variables;
SN-the sampling number of new individuals;
$MAX\_UNCHANGE$ -the maximum generation for stop criterion.

Output: Net -the best-found CNN network with trained weights.

1: **Begin**
2: $\mathcal{D}_{train}$, $\mathcal{D}_{valid}$ ← randomly split $\mathcal{D}_{train}$;
3: cx, dx ← perform OI strategy; //**Algorithm 1**
4: P ← combine cx and dx;//each P_i is generated by cx_i and dx_i
5: $fitness$ ←empty set;
6: **For** each individual P_i in P **Do**
7: Net ← build a CNN according to the variables in P_i;
8: Net ← train Net with T epochs on $\mathcal{D}_{train}$;
9: $accuracy$← validate Net on $\mathcal{D}_{valid}$;
10: $fitness$ ←$fitness$∪{$accuracy$ };
11: **End for**
12: P_{best}, fit_{best} ← the best individual and its fitness;
13: P, $fitness$ ← perform MLDG with P, $fitness$, and Dc;//**Algorithm 3**
14: P_{arch} ← P; fit_{arch} ← $fitness$;
15: $best\_unchage$← 0; //record how long P_{best} has not been updated
16: **While** $best\_unchange < MAX\_UNCHANGE$ **Do**
17: //update probabilistic models by Eq.(1)-(5)
18: μ, σ, $prob$ ← update model with N_b best individuals in P_{arch};
19: // sample SN new individuals
20: cx ← sample SN continuous part by $N(\mu, \sigma)$;
21: dx ← sample SN discrete part by $prob$ with roulette;
22: P ← combine cx and dx;//each P_i is generated by cx_i and dx_i
23: //evaluation new individuals
24: P_{eval}, fit_{eval} ← perform SME on P; //**Algorithm 2**
25: //update $arch$
26: P_{eval}, fit_{eval} ←perform MLDG with P_{eval}, fit_{eval}, Dc;//**Algorithm 3**
27: P_{arch} ← $P_{arch} \cup P_{eval}$; fit_{arch} ← fit_{arch}∪fit_{eval};
28: P_{best}, fit_{best} ← the best individual and its fitness;
29: **If** P_{best} has been updated **Then**
30: $best\_unchange$ ← 0;
31: **Else**
32: $best\_unchange$←$best\_unchange$ +1;
33: **End If**
34: **End While**
35: Net ← Train the best CNN with $\mathcal{D}_{train} \cup \mathcal{D}_{valid}$;
36: **End**

Step 3: SN new individuals are sampled by corresponding probabilistic models with μ, σ, and $prob$ obtained in Step 2.

Step 4: Newly sampled individuals will be evaluated by SME (Algorithm 2). Among these individuals, the individuals evaluated by training-based evaluation will be collected to update the arch. Also, the MLDG will be performed on these individuals and the generated synthetic individuals will be also stored in the arch.

Step 5: If the stop criteria are not met, the algorithm goes back to Step 2 and repeats the procedure. Otherwise, the algorithm trains the CNN model corresponding to the best-found individual by using all the training data and then outputs the CNN model with well-trained weights finally.

G. Time Complexity of the Complete Algorithm

This part analyzes the time complexity of the complete SHEDA. As the most computationally expensive operation is the training of CNN, this article analyzes the time complexity with respect to the total number of training epochs.

First, the initialization process of SHEDA is considered, i.e., the lines 2–14 of Algorithm 4. As can be seen in line 8 of Algorithm 4, each training-based FE of one individual requires T training epochs, which means that the time complexity of the evaluation of each initialized individual is $O(T)$. As the initialization is based on OI strategy (refer to Algorithm 1 and Section III-C), where the number of initialized individuals depends on the search space. Therefore, without loss of generality, this part considers N_init individuals are initialized and require training-based FEs. Based on this, the time complexity for initialization is $O(N\_init \times T)$.

Second, the evolutionary process of SHEDA is considered, i.e., the lines 15–35 of Algorithm 4. In the evolutionary process, as shown in line 24 of Algorithm 4, the evaluation of the individuals in the population P is performed based on SME (i.e., Algorithm 2). As shown in Algorithm 2, only some of the total SN individuals in P will be evaluated by the training-based evaluation with T training epochs. Without loss of generality, this part considers SN_avg individuals in average in each generation will be evaluated by training-based evaluation. Then, the time complexity of the evolutionary process for one generation is $O(SN\_avg \times T)$. As shown in line 16 of Algorithm 4, the stop criterion is that the algorithm cannot find better solutions in $MAX\_UNCHANGE$ generations. That is, the stop generation can vary from problem to problem. For simplicity, this part considers that the algorithm will run Gen generations. Then, the time complexity of the evolutionary process is $O(Gen \times SN\_avg \times T)$.

Based on the above, the total time complexity is $O(SN\_init \times T + Gen \times SN\_avg \times T)$. As can be seen, given a fixed training cost for evaluating individual as $O(T)$, the time complexity of the proposed algorithm mainly depends on the number of generations and the number of evaluated individuals in each generation, which is a common time complexity in the field of evolutionary deep learning.

IV. Experimental Studies

This part conducts experiments to study the efficiency of proposed algorithms. To begin with, parts A to C introduce the benchmark and evaluation metrics, the compared algorithms, and the algorithm settings in the experiments, respectively. Then, the proposed SHEDA is compared with state-of-the-art algorithms to investigate its effectiveness and with its variants for component analysis and parameter studies. Also, a case study is conducted on an AD diagnosis application. Finally, a discussion is provided.

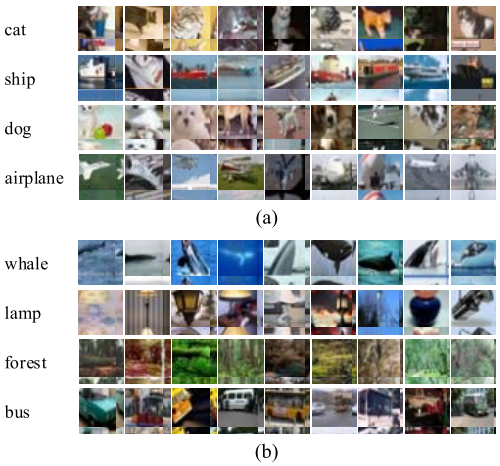

Fig. 5. Image instances from (a) CIFAR10 and (b) CIFAR100. The left column shows the categories while the corresponding instances are on the right.

A. Benchmark Datasets and Evaluation Metrics

To evaluate the proposed SHEDA, popular and widely used benchmark datasets are adopted as the test suite in this article, which are CIFAR10 [29] and CIFAR100 [29]. The CIFAR10 and CIFAR100 are the datasets with different levels of difficulties for classification tasks. Both of them contain 5×10^4 training images and 1×10^4 test images, where each image is with 32×32 pixels and each pixel is with three channels. Moreover, the CIFAR10 contains images of ten categories that need to be correctly classified and the CIFAR100 contains up to 100 categories, such as whale, lamp, forest, and bus. To better illustrate the datasets, Fig. 5 shows some image samples from CIFAR10 and CIFAR100. It can be seen that even the images in the same category will have different resolutions, object shapes, and object locations, imposing great classification difficulties on the classification models. Therefore, these benchmarks can provide in-depth observations on how the CNN found by SHEDA may behave for different tasks.

In the experimental comparisons, three aspects are considered, which are network performance, network size, and the time cost for finding and training this network. For this, three popular metrics, i.e., the classification accuracy on the test dataset, the number of parameters, and the GPU days [23], are adopted for measuring the above three aspects, respectively. In detail, the GPU day is a unit that reflects how many resources and running time are needed for an algorithm to finish. For example, four GPU days can mean that the algorithm requires about one day with four GPUs or four days with one GPU to terminate, where the GPUs usually refer to the NVIDIA GeForce GTX 1080Ti GPU card or more powerful GPU cards.

B. Compared State-of-the-Art Methods

To show the strength of SHEDA, popular and state-of-the-art deep neural networks (DNNs), including manually designed and automatically found network models, are adopted for comparisons in terms of the classification accuracy, the number of parameters, and GPU days. To be specific, these state-of-the-art methods can be divided into three categories: manually designed, non-EC-based optimization, and EC-based optimization methods.

To be specific, the manually designed methods include the famous network models ResNet [8] with 20, 110, and 1202 depths, DenseNet [10] (a powerful enhanced variant of the ResNet), VGG network [46], Maxout [47], Network in Network [48], Highway Network [49], All-CNN [50], and FractalNet [51]. These manually designed CNNs have shown their effectiveness on classification tasks, and therefore they are suitable for investigating whether the SHEDA can find better network models than these CNNs designed by experienced and skillful researchers. As for the non-EC-based optimization methods, this article adopts the representative automatic algorithms like NAS [11] and MetaQNN [52], and semi-automatic algorithms with partial manually designed procedures like EAS [53] and Block-QNN-S [54], for comparisons. In addition, the EC-based optimization methods include genetic CNN [55], Large-scale Evolution [13], Hierarchical Evolution [14], CGP-CNN [19], CNN-GA using GA [23], AE-CNN [24], and AE-CNN + E2EPP [25], where AE-CNN + E2EPP is also a surrogate-assisted method. As these EC-based optimization methods have different characteristics, they are ideal for evaluating the advantages of the proposed SHEDA. Considering that the adopted algorithms require expensive computational cost (e.g., NAS requires 22400 GPU days for one run) and the implemented version may produce different results due to the randomness of the initial weights, the best results in their original paper are cited directly in this article for the comparisons. Also, it should be noted that using the classification results from the corresponding papers is a convention in the evolutionary deep learning community for algorithm comparisons [23]. As for the ablation experiment and parameter studies, the Wilcoxon's rank sum test (significant level of $\alpha = 0.05$) is adopted to compare the results. According to the Wilcoxon's rank sum test, the symbols "+", "$\approx$", and "$-$" are used to show whether the original SHEDA performs significantly better than, similar to, and significantly worse than its variants, respectively.

C. Algorithm Settings

In the experiment, SHEDA used the ResNet with depth = 20, denoted as ResNet-20, as the base network for experimental studies. The ResNet-20 has 17 convolutional layers and two pooling layers for feature extraction and one linear layer for the final classification. By adopting the mixed-variable encoding scheme, the network is encoded into 17 continuous variables (i.e., the number of convolutional kernels in 17 convolutional layers) and 36 discrete variables (i.e., the kernel size of 17 convolutional layers, the kind of 17 activation functions, and the kind of two pooling layers). The initialization and search range are defined as those in Table I. It should be noted that, in ResNet-20, the parameters of the linear layer depend on the last convolutional layer and the number of classification categories, which do not require additional manual configurations.

As for the parameters of SHEDA, the N_b, i.e., the number of selected individuals for learning the probabilistic model, is set as 45% of the size of the current arch, i.e., $N_b = 0.45 \times N_{\text{arch}}$, while the sampling number of new individuals, i.e., SN, is configured as 300, as suggested in the literature [45]. In addition, the stop criterion *MAX_UNCHANGE* is set as ten generations. That is, the SHEDA will finish when it cannot find a better CNN within ten generations. During the training-based evaluation in SME, the new CNNs will be trained with $T = 5$ epochs using the widely used and conventional optimizer in

TABLE II
COMPARISONS WITH STATE-OF-THE-ART METHODS ON CIFAR10 AND CIFAR100

Methods	Network Models	CIFAR10	CIFAR100	# Parameters	GPU days
Manually-design CNNs	ResNet(depth=20) [8]	91.25	-	0.27M	-
	ResNet(depth=110) [8]	93.57	74.84	1.7M	-
	ResNet(depth=1202) [8]	92.07	72.18	10.2M	-
	DenseNet-BC [10]	95.49	77.72	0.8M	-
	VGG-Net [46]	93.34	71.95	20.04M	-
	Maxout [47]	90.70	61.40	-	-
	Network in Network [48]	91.19	64.32	-	-
	Highway Network [49]	92.40	67.66	-	-
	All-CNN [50]	92.75	66.29	1.3M	-
	FratalNet [51]	94.76	77.51	22.9M	-
Non-EC-based CNN optimization approaches	NAS [11]	93.99	-	2.5M	22400
	MetaQNN [52]	93.08	72.86	-	100
	EAS [53]	95.77	-	23.4M	10
	Block-QNN-S [54]	95.62	79.35	6.1M	90
EC-based CNN optimization approaches	Large-scale Evolution [13]	94.60	-	5.4M	2750
	Large-scale Evolution [13]	-	77.00	40.4M	2750
	Hierarchical Evolution [14]	96.37	-	-	300
	CGP-CNN [19]	94.02	-	1.68M	27
	CNN-GA [23]	96.78	-	2.9M	35
	CNN-GA [23]	-	79.47	4.1M	40
	AE-CNN [24]	95.70	-	-	27
	AE-CNN [24]	-	79.15	-	36
	AE-CNN+E2EPP [25]	94.70	77.98	-	8.5
	Genetic CNN [55]	92.90	70.97	-	817
	SHEDA-CNN	96.36	-	10.88M	**0.58**
	SHEDA-CNN	-	78.84	18.64M	**0.97**

the deep learning community, i.e., stochastic gradient descent with the momentum of 0.9 and the learning rate of 0.01 [24]. As for the surrogate-assisted evaluation, the employed surrogate model is the Kriging model with default settings in the Scikit-learn Python toolbox [56], which is an efficient and widely used surrogate method. The experimental studies of these parameters are provided in Section IV-F–IV-H.

As for the datasets, the conventional data augmentation method is adopted [23]. That is, each 32×32 image will be randomly padded by four zero pixels on each side, resulting in a 40×40 image, and then randomly cropped back into a 32×32 image. Finally, the horizontal flip with a probability of 0.5 will be randomly performed on the cropped images. Moreover, when performing the SHEDA, the 5×10^4 training dataset will be further split into two non-overlapped parts, one contains 90% images for training and the other with 10% images for validation. The best solution found by SHEDA will be decoded into the corresponding CNN and then will be trained together with the cutout method [24], which is a conventional and widely used regularization training strategy. This training procedure will repeat 350 epochs using all the 5×10^4 training images (including the validation data), and then the trained CNN will be tested on the test dataset. In the training with 350 epochs, the optimizer is also the SGD with a momentum of 0.9 while the learning rate begins at 0.01 but reduces to 1×10^{-3} and 1×10^{-4} at the 150th and 250th epoch, as suggested in the literature [24]. Due to the expensive computational cost, the SHEDA and its variants will run five times independently and the best results are reported, which is also the convention in the literature [24]. In the experiments, the algorithms are run on the machine with an Intel (R) Core (TM) i7-5930K CPU and three NVIDIA Titan X GPU cards (each with 12 GB RAM) in the Ubuntu 16.04 system. It should be noted that the Titan X GPU cards are not as powerful as the GTX 1080Ti GPU cards. Therefore, the required GPU days of SHEDA will be less than those reported in this articles when the more powerful GTX 1080Ti GPU cards are used. As three GPU cards are used, the time cost of the proposed algorithm will be multiplied by 3 and then used as GPU days for comparisons.

D. Comparisons With State-of-the-Art Methods

In the experiment, the SHEDA required 16578 s to find the CNN model with 96.36% accuracy and 10.88 million parameters for CIFAR10, and 27973 s to find the CNN model with 78.84% accuracy and 18.64 million parameters for CIFAR100. As three GPU cards are used, the corresponding GPU days are about 0.58 and 0.97, respectively. The comparison results with state-of-the-art methods are given in Table II, which shows the great performance of SHEDA. Herein, the CNN found by SHEDA is denoted as SHEDA-CNN.

First, when compared with manually designed models, the SHEDA-CNN produced better results. As given in Table II, the SHEDA-CNN can achieve at least 1% higher accuracy than manually designed CNNs on both CIFAR10 and CIFAR100 datasets, and at most 5.66% on CIFAR10 and 17.44% on CIFAR100, respectively. More importantly, when compared with the ResNet with depths $= 1202$ and its more outstanding variant DenseNet, the SHEDA can optimize the simplest ResNet architecture with depths $= 20$ to obtain better classification results. This suggests that instead of trying deeper and more complex network models, optimizing the hyperparameters of existing well-performing CNN models by SHEDA can obtain more satisfactory results more efficiently.

Second, when compared with non-EC-based CNN optimization approaches, the SHEDA has also shown great efficiency. On CIFAR10, the SHEDA-CNN outperforms the competitors in terms of accuracy and GPU days. This means that the SHEDA only requires much less time and computational cost

TABLE III
COMPARISONS AMONG DIFFERENT VARIANTS OF THE PROPOSED ALGORITHM

Variants	CIFAR10			CIFAR100		
	Accuracy	# Parameters	GPU days	Accuracy	# Parameters	GPU days
SHEDA-CNN	**96.36**	10.88M	0.58	**78.84**	18.64M	**0.97**
SHEDA-w/o-OI-CNN	95.85(+)	8.49M	0.60	77.06(+)	12.31M	1.05
SHEDA-w/o-AHL-CNN	96.27(≈)	8.43 M	**0.54**	78.65(≈)	11.20M	1.03
SHEDA-w/o-SME-CNN	96.15(≈)	**7.98M**	4	78.44(≈)	**10.23M**	5
SHEDA-w/o-MLDG-CNN	96.16(≈)	10.8M	0.62	78.49(≈)	11.07M	1.02

to find better CNN models. For example, when compared with the NAS, the SHEDA-CNN can have 2.37% higher classification accuracy but only cost about 0.25% computational costs (0.58 versus 22400 GPU days) on CIFAR10. This is an exciting significant improvement. Besides, although Block-QNN-S can have a little higher accuracy than SHEDA-CNN on CIFAR100, its computational costs are 180 times as large as SHEDA-CNN, which also suggests the high efficiency of SHEDA. This may be due to the fact that the surrogate-assisted strategy (i.e., the SME) can avoid the unnecessary evaluation of poor hyperparameter configurations and help SHEDA find a satisfactory solution efficiently.

Third, when compared with state-of-the-art EC-based optimization approaches, the SHEDA can still obtain compelling results. The results show that SHEDA-CNN can outperform Genetic CNN, Large-scale Evolution, CGP-CNN, and the surrogate-assisted approach AE-CNN + E2EPP in terms of both classification accuracy and GPU days. In addition, although CNN-GA and AE-CNN can have higher classification accuracy about 0.6% and 0.3% when compared with SHEDA-CNN on CIFAR10, they require 40 and 36 GPU days, respectively. But the SHEDA can obtain the CNN models using 0.58 GPU days, which is much more computationally efficient. Moreover, as the CNN-GA and AE-CNN aim to evolve the best CNN, the CNN produced by them can be adopted as the base network and then further optimized using the SHEDA.

In conclusion, the comparisons with state-of-the-art methods on CIFAR10 and CIFAR100 have shown the great efficiency of SHEDA.

E. Ablation Experiments for Contribution Analysis

To study the contributions of each component in SHEDA, the ablation experiment is performed in this part. In detail, the SHEDA is compared with its variants that do not use OI, AHL, SME, or MLDG, which are denoted as SHEDA-w/o-OI, SHEDA-w/o-AHL, SHEDA-w/o-SME, and SHEDA-w/o-MLDG, respectively. In SHEDA-w/o-OI, the random initialization is performed instead of OI, and for a fair comparison, the number of randomly initialized individuals is the same as the row number of $\boldsymbol{OA}$. In addition, the maximum generation of SHEDA-w/o-SME is set to 100 in each run due to its very expensive computational costs. Also, for a better illustration, the CNNs found by these variants are denoted with "X-CNN," e.g., SHEDA-w/o-OI-CNN and SHEDA-w/o-AHL-CNN.

The results of different SHEDA variants are provided in Table III. According to the Wilcoxon's rank sum test, the SHEDA can perform significantly better than SHEDA-w/o-OI, and perform similar to SHEDA-w/o-AHL, SHEDA-w/o-SME, and SHEDA-w/o-MLDG in terms of the classification accuracy on both CIFAR10 and CIFAR100. Moreover, considering the network parameters and GPU days, the SHEDA obtains similar results with the SHEDA-w/o-OI, SHEDA-w/o-AHL, and SHEDA-w/o-MLDG. When compared with SHEDA-w/o-AHL-CNN, SHEDA-CNN has higher accuracy but is with a larger number of parameters. This may be because the AHL will increase the mean values of continuous probabilistic models for controlling the number of kernels or neurons in CNNs more quickly through the fitness-based weights, and therefore the models are evolved to be larger quickly. As for SHEDA-w/o-SME-CNN, the SHEDA-CNN can have a similar accuracy but requite a much shorter time for optimization, which suggests the great computational efficiency of SME. Based on the above, the OI, AHL, SME, and MLDG have their contributions to the promising performance of SHEDA, and removing any of them will deteriorate the performance of SHEDA.

TABLE IV
COMPARISONS WITH SHEDA VARIANTS WITH DIFFERENT SAMPLING NUMBER

Variants	Accuracy	# Parameters	GPU days
SN=300 (original)	**96.36**	10.88M	0.58
SN=100	95.78(+)	**5.21M**	**0.47**
SN=200	95.82(+)	7.79M	0.52
SN=400	96.10(≈)	6.12M	0.58
SN=500	96.26(≈)	8.48M	0.61

F. Influence of Sampling Number

To investigate the influence of the sampling number (i.e., *SN*) of new individuals, the SHEDA is compared with variants using different *SN* on CIFAR10. The results are given in Table IV, where the Wilcoxon's rank sum test (significant level of $\alpha = 0.05$) is also adopted to compare the results. It can be seen that different *SN* settings obtain similar accuracy, among which the smaller *SN* produces worse results while the larger *SN* obtains better results. Moreover, according to the Wilcoxon's rank sum test, the original SHEDA (i.e., $SN = 300$) performs similar to the variants with $SN = 400$ and $SN = 500$, which show that the algorithm is not that sensitive to the *SN*. In addition, it seems that *SN* has a small influence on the number of network parameters. For example, the number of network parameters obtained by $SN = 400$ is smaller than $SN = 200$ while is larger than that of $SN = 100$. This may be because the optimal structure of CNNs has a multimodal characteristic and some very different structures may have similar accuracy. Concluding from the above, the *SN* can influence the search effectiveness of SHEDA, but the search effectiveness is not that sensitive to *SN*.

G. Influence of Individual Number for Learning Probabilistic Models

To investigate the influence of the N_b (i.e., the number of individuals selected for learning probabilistic model), the

TABLE V
COMPARISONS WITH SHEDA VARIANTS USING THE DIFFERENT NUMBER OF INDIVIDUALS FOR UPDATING PROBABILISTIC MODELS

Variants	Accuracy	# Parameters	GPU days
N_b =0.45×N_{arch}	**96.36**	10.88M	0.58
N_b =0.3×N_{arch}	96.22(≈)	8.43M	**0.48**
N_b =0.4×N_{arch}	96.13(≈)	8.91M	0.54
N_b =0.5×N_{arch}	95.86(+)	**7.90M**	0.58
N_b =0.6×N_{arch}	95.68(+)	9.11M	0.61
N_b =0.7×N_{arch}	95.65(+)	8.98M	0.58

SHEDA is compared with its variants using $N_b = 0.3 \times N_{\text{arch}}$, $N_b = 0.4 \times N_{\text{arch}}$, $N_b = 0.5 \times N_{\text{arch}}$, $N_b = 0.6 \times N_{\text{arch}}$, and $N_b = 0.7 \times N_{\text{arch}}$, where N_{arch} contains the number of evaluated individuals. The comparison results are given in Table V. The results show that smaller N_b is better than larger N_b in terms of accuracy and GPU days. According to the Wilcoxon's rank sum test, the original SHEDA ($N_b = 0.45 \times N_{\text{arch}}$) performs similar to its variants with $N_b = 0.3 \times N_{\text{arch}}$ and $N_b = 0.4 \times N_{\text{arch}}$, but significantly better than its variants with $N_b = 0.5 \times N_{\text{arch}}$, $N_b = 0.6 \times N_{\text{arch}}$, and $N_b = 0.7 \times N_{\text{arch}}$. This may be because stricter criterion for choosing the individuals (i.e., smaller N_b) can build a more accurate probabilistic model and accelerate the optimization convergence. Furthermore, in Table V, although $N_b = 0.3 \times N_{\text{arch}}$, $N_b = 0.4 \times N_{\text{arch}}$, and $N_b = 0.45 \times N_{\text{arch}}$ produce similar accuracy, the $N_b = 0.45 \times N_{\text{arch}}$ obtains the best results among them. This may be because too few individuals can be also difficult for learning the accurate probabilistic model. Therefore, $N_b = 0.45 \times N_{\text{arch}}$ is suitable and is recommended for SHEDA.

H. Influence of the Training Epochs

In training-based evaluation, candidate CNNs are only trained with T epochs ($T < 350$) for comparisons, so as to reduce the time cost. Hence, whether training T epochs is enough for figuring out better CNNs can be a key issue for the effectiveness of the SME. Therefore, to study the influences of T, an experiment is carried out as follows to measure the accuracy of CNN rankings by training different T epochs.

First, 100 individuals are randomly sampled and then transformed into 100 CNN models. Second, the 100 CNNs are trained with $T = 350$ epochs on the training dataset, and then are ranked based on their accuracy on the test dataset. Moreover, at epoch $T = 1, 2, \ldots, 9$, and 10, the 100 CNNs are also ranked according to their accuracy on the test dataset, respectively. Third, the errors between the rankings at different epochs and that at $T = 350$ are computed. Finally, the mean and standard values of ranking errors are obtained for comparisons. For example, at epoch $T = i$, the ranking error on j^{th} CNN, denoted as $\text{err}_{i,j}$, is computed as

$$\text{err}_{i,j} = \left|\text{rank}_{i,j} - \text{rank}_{350,j}\right| \tag{8}$$

where $\text{rank}_{i,j}$ is the ranking of jth CNN after training i epochs. Based on this, the mean and standard value of the 100 ranking errors at epoch $T = i$ can be calculated with $\text{err}_{i,1}, \text{err}_{i,2}, \ldots, \text{err}_{i,99}$, and $\text{err}_{i,100}$.

Fig. 6 plots the mean and standard value of ranking error at epoch $T = 1$ to $T = 10$. Moreover, the average time costs of training a CNN with different T epochs are also provided as Bar graph in the table. As can be seen, the ranking error reduces rapidly from $T = 1$ to $T = 3$, and begins to converge

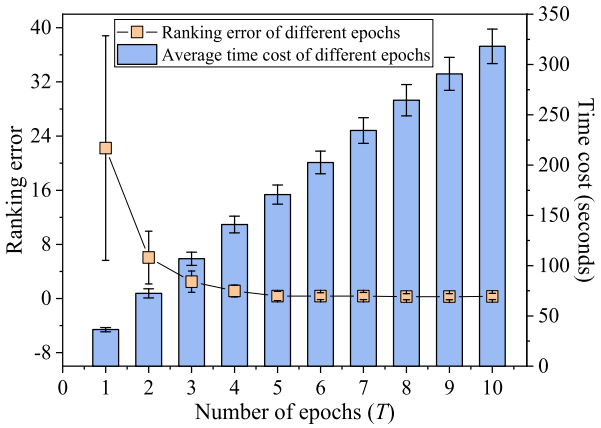

Fig. 6. Ranking error and required time cost when using different epochs. The figure shows that as T increases, the ranking error (i.e., the difference between the ranking of 100 randomly generated CNNs after training T epochs and the ranking of them after training 350 epochs) reduces, while the average time cost for training these CNNs increases nearly linearly.

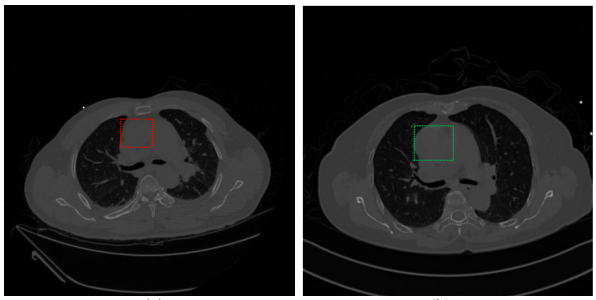

Fig. 7. Two patients' CT images: a) with AD in the red box area and b) without AD in the green box area, which can be very difficult for human beings including medical professionals to find out on normal CT images.

at $T = 4$ and $T = 5$. It should also be noted that obtaining the exact performance of a CNN is difficult, because the training results will be significantly influenced by its initial weights. Therefore, training five epochs may be enough and acceptable for comparing different CNNs. Moreover, as the number of epochs increases in Fig. 6, the time costs also increase nearly linearly, showing that each epoch requires about 30 s for training with three GPU cards. Considering the ranking error and time cost, $T = 5$ can make a better balance between these two factors, and therefore is recommended in this article.

I. Case Study on AD Diagnosis

To evaluate the performance of SHEDA in real-world problems, a case study on AD diagnosis [57] is carried out in this part. The ADs are high-risk and dangerous cardiovascular diseases that usually result in great harm and poor natural prognosis [58]. Moreover, it has been shown that the morbidity and mortality of ADs are strongly related to the late implementation of proper treatment [57]. Hence, early diagnosis is very important and crucial for saving patients.

However, although computerized tomography (CT) can provide images to help professional doctors to recognize ADs, the recognition procedures require doctors to scan the images throughout the patients' body one by one to find the potential ADs, which is very inefficient [57]. Moreover, CT images are not so clear and the AD shapes of different patients differ greatly [59]. For example, in Fig. 7, the aorta

TABLE VI
RESULTS OF THE AD DIAGNOSIS IN TERMS OF THE NUMBER OF TRUE POSITIVE, TRUE NEGATIVE, FALSE POSITIVE, AND FALSE NEGATIVE, ACCURACY, THE NUMBER OF PARAMETERS, AND GPU DAYS

Methods	#TP	#TN	#FP	#FN	Accuracy	# Para.	GPU days
ResNet (depth=20)	250	132	3	2	98.71	**2.93M**	-
ResNet (depth=110)	250	**133**	3	**1**	98.97	10.78M	-
SHEDA-CNN	**252**	**133**	**1**	**1**	**99.48**	4.18M	1.18

in the red box [Fig. 7(a)] has irregular regions with different shades, also called the true lumen and false lumen caused by the dissection, while the aorta in the green box [Fig. 7(b)] is normal and with a regular region. Actually, these two cases are difficult to be recognized and distinguished by human beings. As a result, even experienced doctors may have problems with diagnosis and miss some ADs, which will delay the needed treatments. Therefore, to achieve accurate AD diagnosis automatically and efficiently, deep learning approaches like CNN can been used [60]. As the resolution of CT images and various AD shapes can bring in great classification difficulties, this classification problem is suitable for testing the CNNs obtained by SHEDA.

To conduct the experiment, 3873 CT images, including 2635 images with positive AD diagnosis and 1238 images without ADs, are totally collected from patients in Guangdong General Hospital in Guangzhou, China. Each image has a size of 512×512 and is with three channels. In the experiments, the total data are randomly split into the training dataset with 3486 images and the test dataset with 387 images, which are about 90% and 10% of the total data size, respectively. The configurations of SHEDA are kept the same as before. For comparisons, due to the expensive computational cost, only the ResNet with 20 and 110 depths are adopted.

The comparison results are given in Table VI, where the number of True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN) are provided. To be specific, the TP and TN represent the number of predictions that correctly predicts positive and negative diagnoses, respectively, while the FP means that the model mistakes an image of negative diagnosis as a positive diagnosis and FN indicates that the model mistakes an image of positive diagnosis as a negative diagnosis. Considering the high-risk and dangerous characteristics of ADs, the FN will result in much more serious situations than FP. The results in Table VI have verified the effectiveness of SHEDA. As can be seen, SHEDA-CNN can produce higher classification accuracy than ResNet(depth $=$ 20) and ResNet(depth $=$ 110). Moreover, the SHEDA-CNN can have fewer FP and FN predictions than ResNet(depth $=$ 20) and ResNet(depth $=$ 110), showing the reliability of SHEDA. Therefore, this case study shows that the SHEDA is potential for solving real-world applications.

J. Further Discussion

The above experiments have shown the superiority of the proposed SHEDA for finding suitable hyperparameters for the CNN. In this artricle, the widely used CNN, i.e., the ResNet with depth as 20, is adopted as the basic network for hyperparameters optimization and the results obtained by SHEDA can outperform even the ResNet with depth as 1202 and other state-of-the-art CNNs. However, to be honest, the SHEDA also has its limitations. To be specific, the SHEDA is an efficient tool for optimizing the hyperparameters for a given basic network. Consequently, if the basic network is not suitable for the target tasks, the improvements brought by the SHEDA will be limited. In this article, the ResNet with depth as 20 is selected as the basic network for two reasons. First, it is a conventional and powerful CNN in the deep learning community, which can help observe the potential of the proposed algorithm for further improving the CNNs through hyperparameters optimization. Second, it is much easier and faster to train when compared with other more complex, larger, or deeper CNNs, which can help to get the experimental results in a more satisfactory time, especially when the time and resources (e.g., GPU cards) are limited. In fact, how to choose a suitable basic CNN for different tasks is still a challenging and open problem, because the relationship between the representative learning ability of CNNs and their depth and topology is still not clear, which requires further researches and is somewhat out of the scope of this article. Therefore, considering dealing with the difficulties for the hyperparameters optimization for a given CNN, the contribution of this article is justified. In addition, this also indicates that the evolution of network depth and topology should be further considered in the proposed SHEDA in the future, so as to improve its performance.

V. CONCLUSION

This article focuses on the three big difficulties in CNN hyperparameters optimization and proposes the SHEDA for efficiently optimizing the CNN hyperparameters. In detail, the SHEDA uses the AHL strategy based on the mixed-variable encoding scheme, the OI strategy, and the SME method with MLDG, for dealing with the mixed-variable problem, large-scale search space challenge, and the expensive computational cost, respectively. Experiments on two challenging classification problems CIFAR10 and CIFAR100 and the comparisons with various state-of-the-art algorithms have verified the superiority of SHEDA. In addition, experimental studies have also been conducted to investigate the component contributions and parameter sensitivities. A case study on AD diagnosis has also been carried out to further show the effectiveness of SHEDA.

For future work, more efficient EC algorithms will be studied for handling various kinds of important problems in CNN optimization, such as multi-task and multimodal problems. Furthermore, how to evolve the network depth and topology of CNNs is worthy of further research. Also, more efficient optimization paradigms can be considered for CNN optimization in different scenarios, where potential optimization paradigms include data-driven optimization [61], large-scale optimization [62], dynamic optimization [63], many-objective optimization [64], multi-modal optimization [65], expensive optimization [66], and parallel [67] and distributed optimization [68]. In addition, the SHEDA is potential for solving more challenging learning tasks in complex real-world applications, such as healthcare application [69] and autonomous robot application [70], which will be further explored and studied.

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, Feb. 2015.
[2] Z.-Q. Zhao, P. Zheng, S.-T. Xu, and X. Wu, "Object detection with deep learning: A review," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 30, no. 11, pp. 3212–3232, Nov. 2019.

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

[4] 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.

[5] H. Y. D. Sigaki, E. K. Lenzi, R. S. Zola, M. Perc, and H. V. Ribeiro, "Learning physical properties of liquid crystals with deep convolutional neural networks," *Sci. Rep.*, vol. 10, no. 1, pp. 1–11, May 2020.

[6] Z. Gao *et al.*, "Complex networks and deep learning for EEG signal analysis," *Cognit. Neurodyn.*, vol. 15, no. 3, pp. 369–388, Jun. 2021.

[7] F. E. Fernandes and G. G. Yen, "Automatic searching and pruning of deep neural networks for medical imaging diagnostic," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Oct. 13, 2020, doi: 10.1109/TNNLS.2020.3027308.

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

[9] S. Zagoruyko and N. Komodakis, "Wide residual networks," in *Proc. Procedings Brit. Mach. Vis. Conf.*, 2016, p. 87.

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

[11] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *Proc. Int. Conf. Learn. Represent.*, 2017, pp. 1–16.

[12] K. Kandasamy, W. Neiswanger, J. Schneider, B. Póczos, and E. P. Xing, "Neural architecture search with Bayesian optimisation and optimal transport," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 2016–2025.

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

[14] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical representations for efficient architecture search," in *Proc. Int. Conf. Learn. Represent.*, 2018, pp. 1–13.

[15] Z.-H. Zhan *et al.*, "Matrix-based evolutionary computation," *IEEE Trans. Emerg. Topics Comput. Intell.*, early access, Jan. 21, 2021, doi: 10.1109/TETCI.2020.3047410.

[16] J.-Y. Li, Z.-H. Zhan, C. Wang, H. Jin, and J. Zhang, "Boosting data-driven evolutionary algorithm with localized data generation," *IEEE Trans. Evol. Comput.*, vol. 24, no. 5, pp. 923–937, Oct. 2020.

[17] X. Xia *et al.*, "Triple archives particle swarm optimization," *IEEE Trans. Cybern.*, vol. 50, no. 12, pp. 4862–4875, Dec. 2020.

[18] Z.-H. Zhan *et al.*, "Cloudde: A heterogeneous differential evolution algorithm and its distributed cloud version," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 3, pp. 704–716, Mar. 2017.

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

[20] Z.-G. Chen, Y. Lin, Y.-J. Gong, Z.-H. Zhan, and J. Zhang, "Maximizing lifetime of range-adjustable wireless sensor networks: A neighborhood-based estimation of distribution algorithm," *IEEE Trans. Cybern.*, early access, Apr. 1, 2020, doi: 10.1109/TCYB.2020.2977858.

[21] W. Du *et al.*, "The networked evolutionary algorithm: A network science perspective," *Appl. Math. Comput.*, vol. 338, pp. 33–43, Dec. 2018.

[22] U. Alvarez-Rodriguez, F. Battiston, G. F. de Arruda, Y. Moreno, M. Perc, and V. Latora, "Evolutionary dynamics of higher-order interactions in social networks," *Nature Hum. Behav.*, vol. 5, no. 5, pp. 586–595, May 2021.

[23] 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.

[24] 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.

[25] 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.

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

[27] X. F. Liu, Z. H. Zhan, and J. Zhang, "Resource-aware distributed differential evolution for training expensive neural-network-based controller in power electronic circuit," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, May 7, 2021, doi: 10.1109/TNNLS.2021.3075205.

[28] X. Zheng, R. Ji, L. Tang, B. Zhang, J. Liu, and Q. Tian, "Multinomial distribution learning for effective neural architecture search," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 1304–1313.

[29] A. Krizhevsky and G. Hinton. (2009). *Learning Multiple Layers of Features From Tiny Images*. [Online]. Available: https://www.cs.toronto.edu/ kriz/cifar.html

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

[31] Z.-H. Zhan, Z.-J. Wang, H. Jin, and J. Zhang, "Adaptive distributed differential evolution," *IEEE Trans. Cybern.*, vol. 50, no. 11, pp. 4633–4647, Nov. 2020.

[32] N. Xue, I. Triguero, G. P. Figueredo, and D. Landa-Silva, "Evolving deep CNN-LSTMs for inventory time series prediction," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jun. 2019, pp. 1517–1524.

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

[34] B. Fielding, T. Lawrence, and L. Zhang, "Evolving and ensembling deep CNN architectures for image classification," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2019, pp. 1–8.

[35] T. Wu, J. Shi, D. Zhou, Y. Lei, and M. Gong, "A multi-objective particle swarm optimization for neural networks pruning," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jun. 2019, pp. 570–577.

[36] Y. Guo, J.-Y. Li, and Z.-H. Zhan, "Efficient hyperparameter optimization for convolution neural networks in deep learning: A distributed particle swarm optimization approach," *Cybern. Syst.*, vol. 52, no. 1, pp. 36–57, Jan. 2021.

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

[38] Z.-H. Zhan, J. Zhang, Y. Li, and Y.-H. Shi, "Orthogonal learning particle swarm optimization," *IEEE Trans. Evol. Comput.*, vol. 15, no. 6, pp. 832–847, Dec. 2011.

[39] J. Tian, Y. Tan, J. Zeng, C. Sun, and Y. Jin, "Multiobjective infill criterion driven Gaussian process-assisted particle swarm optimization of high-dimensional expensive problems," *IEEE Trans. Evol. Comput.*, vol. 23, no. 3, pp. 459–472, Jun. 2019.

[40] X. Cai, L. Gao, and X. Li, "Efficient generalized surrogate-assisted evolutionary algorithm for high-dimensional expensive problems," *IEEE Trans. Evol. Comput.*, vol. 24, no. 2, pp. 365–379, Apr. 2020.

[41] F. Li, X. Cai, L. Gao, and W. Shen, "A surrogate-assisted multiswarm optimization algorithm for high-dimensional computationally expensive problems," *IEEE Trans. Cybern.*, vol. 51, no. 3, pp. 1390–1402, Mar. 2021.

[42] Z. Yang, H. Qiu, L. Gao, X. Cai, C. Jiang, and L. Chen, "Surrogate-assisted classification-collaboration differential evolution for expensive constrained optimization problems," *Inf. Sci.*, vol. 508, pp. 50–63, Jan. 2020.

[43] H. Wang and Y. Jin, "A random forest-assisted evolutionary algorithm for data-driven constrained multiobjective combinatorial optimization of trauma systems," *IEEE Trans. Cybern.*, vol. 50, no. 2, pp. 536–549, Feb. 2020.

[44] I. Radosavovic, R. P. Kosaraju, R. Girshick, K. He, and P. Dollar, "Designing network design spaces," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 10428–10436.

[45] W. Shi, W.-N. Chen, Y. Lin, T. Gu, S. Kwong, and J. Zhang, "An adaptive estimation of distribution algorithm for multipolicy insurance investment planning," *IEEE Trans. Evol. Comput.*, vol. 23, no. 1, pp. 1–14, Feb. 2019.

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

[47] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio, "Maxout networks," in *Proc. Int. Conf. Mach. Learn.*, 2013, pp. 1319–1327.

[48] M. Lin, Q. Chen, and S. Yan, "Network in network," in *Proc. Int. Conf. Learn. Represent.*, 2014, pp. 1–10.

[49] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Highway networks," in *Proc. Int. Conf. Learn. Represent.*, 2015, pp. 1–6.

[50] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for simplicity: The all convolutional net," in *Proc. Int. Conf. Learn. Represent.*, 2015, pp. 1–14.

[51] G. Larsson, M. Maire, and G. Shakhnarovich, "FractalNet: Ultra-deep neural networks without residuals," in *Proc. Int. Conf. Learn. Represent.*, 2016, pp. 1–11.

[52] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," in *Proc. Int. Conf. Learn. Represent.*, 2017, pp. 1–18.
[53] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang, "Efficient architecture search by network transformation," in *Proc. 32nd Conf. Artif. Intell.*, 2018, pp. 2787–2794.
[54] Z. Zhong, J. Yan, W. Wu, J. Shao, and C.-L. Liu, "Practical block-wise neural network architecture generation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 2423–2432.
[55] L. Xie and A. Yuille, "Genetic CNN," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 1388–1397.
[56] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, no. 85, pp. 2825–2830, Oct. 2011.
[57] C. A. Nienaber *et al.*, "Aortic dissection," *Nat. Rev. Dis. Prim.*, vol. 2, pp. 1–18, Mar. 2016.
[58] R. J. Harris *et al.*, "Classification of aortic dissection and rupture on post-contrast CT images using a convolutional neural network," *J. Digit. Imag.*, vol. 32, no. 6, pp. 939–946, Sep. 2019.
[59] C. Xu *et al.*, "Multifunctional cationic nanosystems for nucleic acid therapy of thoracic aortic dissection," *Nature Commun.*, vol. 10, no. 1, pp. 1–15, Jul. 2019.
[60] S. S. Sumit, D. R. A. Rambli, and S. Mirjalili, "Vision-based human detection techniques: A descriptive review," *IEEE Access*, vol. 9, pp. 42724–42761, Mar. 2021.
[61] J.-Y. Li, Z.-H. Zhan, H. Wang, and J. Zhang, "Data-driven evolutionary algorithm with perturbation-based ensemble surrogates," *IEEE Trans. Cybern.*, vol. 51, no. 8, pp. 3925–3937, Aug. 2021.
[62] Z.-J. Wang, Z.-H. Zhan, S. Kwong, H. Jin, and J. Zhang, "Adaptive granularity learning distributed particle swarm optimization for large-scale optimization," *IEEE Trans. Cybern.*, vol. 51, no. 3, pp. 1175–1188, Mar. 2021.
[63] X.-F. Liu *et al.*, "Neural network-based information transfer for dynamic optimization," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 5, pp. 1557–1570, May 2020.
[64] X.-F. Liu, Z.-H. Zhan, Y. Gao, J. Zhang, S. Kwong, and J. Zhang, "Coevolutionary particle swarm optimization with bottleneck objective learning strategy for many-objective optimization," *IEEE Trans. Evol. Comput.*, vol. 23, no. 4, pp. 587–602, Aug. 2019.
[65] Z.-G. Chen, Z.-H. Zhan, H. Wang, and J. Zhang, "Distributed individuals for multiple peaks: A novel differential evolution for multimodal optimization problems," *IEEE Trans. Evol. Comput.*, vol. 24, no. 4, pp. 708–719, Aug. 2020.
[66] S. H. Wu, Z. H. Zhan, and J. Zhang, "SAFE: Scale-adaptive fitness evaluation method for expensive optimization problems," *IEEE Trans. Evol. Comput.*, vol. 25, no. 3, pp. 479–491, Jun. 2021.
[67] J.-Y. Li, Z.-H. Zhan, R.-D. Liu, C. Wang, S. Kwong, and J. Zhang, "Generation-level parallelism for evolutionary computation: A pipeline-based parallel particle swarm optimization," *IEEE Trans. Cybern.*, early access, Nov. 4, 2020, doi: 10.1109/TCYB.2020.3028070.
[68] Y. F. Ge *et al.*, "Distributed memetic algorithm for outsourced database fragmentation," *IEEE Trans. Cybern.*, Nov. 4, 2020, doi: 10.1109/TCYB.2020.3027962.
[69] J. Too and S. Mirjalili, "A hyper learning binary dragonfly algorithm for feature selection: A COVID-19 case study," *Knowl.-Based Syst.*, vol. 212, pp. 1–16, Jan. 2021.
[70] S. M. J. Jalali, S. Ahmadian, A. Khosravi, S. Mirjalili, M. R. Mahmoudi, and S. Nahavandi, "Neuroevolution-based autonomous robot navigation: A comparative study," *Cognit. Syst. Res.*, vol. 62, pp. 35–43, Aug. 2020.

Jian-Yu Li (Student Member, IEEE) received the B.S. degree in computer science and technology from South China University of Technology, Guangzhou, China, in 2018, where he is currently pursuing the Ph.D. degree.

His research interests mainly include computational intelligence, data-driven optimization, and their applications in real-world problems.

Zhi-Hui Zhan (Senior Member, IEEE) received the bachelor's and Ph.D. degrees in computer science from Sun Yat-sen University, Guangzhou, China, in 2007 and 2013, respectively.

He is currently a Changjiang Scholar Young Professor with the School of Computer Science and Engineering, South China University of Technology, Guangzhou. His current research interests include evolutionary computation, swarm intelligence, and their applications in real-world problems and in environments of cloud computing and big data.

Dr. Zhan was a recipient of the IEEE Computational Intelligence Society (CIS) Outstanding Early Career Award in 2021, the Outstanding Youth Science Foundation from National Natural Science Foundations of China (NSFC) in 2018, and Wu Wen-Jun Artificial Intelligence Excellent Youth from the Chinese Association for Artificial Intelligence in 2017. His doctoral dissertation was awarded the IEEE CIS Outstanding Ph.D. Dissertation and China Computer Federation Outstanding Ph.D. Dissertation. He is listed as one of the Highly Cited Chinese Researchers in Computer Science. He is currently an Associate Editor of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, the *Neurocomputing*, and the *Memetic Computing*.

Jin Xu received the Ph.D. degree from the Department of Electrical and Computer Engineering, Stevens Institute of Technology, Hoboken, NJ, USA, in 2012.

He is currently a Principal Researcher with WeChat, Tencent Inc., Shenzhen, China. He has published more than 30 peer-reviewed publications in AI research. His current research interests include machine learning, data mining, and their applications in internet product.

Sam Kwong (Fellow, IEEE) received the Ph.D. degree from the University of Hagen, Hagen, Germany, in 1996.

He is currently a Chair Professor with the Department of Computer Science, City University of Hong Kong, Hong Kong. His research interests include pattern recognition, evolutionary computations, and video analytics.

Prof. Kwong was elevated to an IEEE Fellow for his contributions to optimization techniques for cybernetics and video coding in 2014. He is the President-Elect of the IEEE SYSTEMS, MAN, AND CYBERNETICS (SMC). He was also appointed as an IEEE Distinguished Lecturer of the IEEE SMC Society in March 2017. He is currently an Associate Editor of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION.

Jun Zhang (Fellow, IEEE) received the Ph.D. degree from the City University of Hong Kong, in 2002.

He is currently a Korea Brain Pool Fellow Professor with Hanyang University, Seoul, South Korea, and a Visiting Professor with Chaoyang University of Technology, Taichung, Taiwan. His current research interests include computational intelligence, cloud computing, operations research, and power electronic circuits. He has published over more than 150 IEEE TRANSACTIONS articals in his research areas.

Dr. Zhang was a recipient of a Changjiang Chair Professor from the Ministry of Education, China, in 2013, the National Science Fund for Distinguished Young Scholars of China in 2011, and the First-Grade Award in Natural Science Research from the Ministry of Education, China, in 2009. He is currently an Associate Editor of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION and the IEEE TRANSACTIONS ON CYBERNETICS.