Particle Swarm Optimization for Efficiently Evolving Deep Convolutional Neural Networks Using an Autoencoder-based Encoding Strategy

Gonglin Yuan, *Graduate Student Member, IEEE,* Bin Wang, *Graduate Student Member, IEEE,* Bing Xue, *Senior Member, IEEE,* and Mengjie Zhang, *Fellow, IEEE*

***Abstract*—Deep convolutional neural networks (DCNNs) have achieved surpassing success in the field of computer vision, and a number of elaborately designed networks refresh the performance records in benchmark datasets. Recently, evolutionary neural architecture search (ENAS) has become an emerging area, which could employ an evolutionary computation (EC) technique to automatically construct promising network architectures without human intervention. However, existing algorithms still have limitations: most standard EC approaches cannot process flexible-length architecture representations directly, and most fitness evaluation processes suffer from the exhibitive computational cost and the unreliable prediction results. To overcome these shortcomings, we propose an efficient particle swarm optimization (PSO)-based neural architecture search algorithm to search for appropriate dense blocks on image classification tasks, named EAEPSO. EAEPSO addresses the first limitation by designing an autoencoder to encode variable-length network representations as fixed-length latent vectors, which converts the original search space to a latent space that can facilitate the downstream search. Besides, an efficient and effective hierarchical fitness evaluation method is designed to guide the search process to address the second limitation. The experimental results show that EAEPSO is a very competitive ENAS algorithm that achieves an error rate of 2.74% on CIFAR-10 and 16.17% on CIFAR-100, and reduces the computational cost from hundreds or thousands of GPU-days to only 2.2 and 4 GPU-days, respectively. Further analyses investigate the reduced training data's effect and confirm the effectiveness of both the proposed autoencoder and the proposed hierarchical fitness evaluation method.**

***Index Terms*—Evolutionary neural architecture search, autoencoder, particle swarm optimization, image classification.**

I. Introduction

DEEP Convolutional Neural Networks (DCNNs) have demonstrated their superiority over most traditional image processing methods in recent years, and have become main-stream approaches in the field of computer vision. They achieved extraordinary performance in various tasks, such as object detection, image classification, and semantic segmentation. A number of well-performed networks were proposed, such as ResNet [1], DenseNet [2], and SENet [3]. The architectures of networks are usually task-specific, i.e., the architectures need to be redesigned accordingly when the distribution of the data in the specific task changes, and the design process is labor-intensive and error-prone. In addition, the architectures need to be designed by experts who know well about deep learning and the specific task or domain information.

To address the shortcomings mentioned above, neural architecture search (NAS) has made great progress and attracted great attention. NAS refers to searching for promising network architectures automatically for a given task. Existing NAS algorithms can be generally divided into three categories based on the search methods: gradient-based NAS algorithms [4], reinforcement learning (RL)-based NAS algorithms [5]–[7], and evolutionary computation (EC)-based NAS (ENAS) algorithms [8]–[10]. Gradient-based algorithms often require designing a big supernet in advance, which may involve human expertise and reduce automation. They can measure the importance of the weights using back propagation with explicit mathematical expressions. RL-based algorithms regard the construction of network architectures as the agent's action, which often consumes immense computational costs and is not affordable for most researchers [11], while these methods can often reflect the reasoning process of the agents, potentially benefiting interpretability. ENAS algorithms search for architectures by imitating the evolutionary process of nature, requiring less computational resources compared with RL-based algorithms but usually achieving good results [12], [13].

Particle swarm optimization (PSO) [14], an effective and efficient EC algorithm, attracts a lot of attention because of its simple implementation, high efficiency, and robustness [15]. Besides, there are fewer parameters needed to be tuned in PSO, which helps improve the automation in NAS algorithms, making PSO widely using in ENAS. Each particle represents a potential solution/network architecture. Generally, a standard PSO algorithm is used for processing fixed-length particles, and the length of the particles is usually associated with the depth of the corresponding network, but the optimal depth of the network is not easy to determine. To tackle this problem, some existing works [16], [17] use vectors of the same length to represent the network architectures, where some bits of the vectors can be disabled to represent no layer. However, the disabled bits are usually set to be 0, and the gap between them and valid bits is big, which may harm the evolutionary process. Besides, PSO is good at processing continuous dec-

This work was supported in part by the Marsden Fund of New Zealand Government under Contracts VUW1913, VUW1914 and VUW2115, the Science for Technological Innovation Challenge (SfTI) fund under grant E3603/2903, MBIE Data Science SSIF Fund under the contract RTVU1914, and National Natural Science Foundation of China (NSFC) under Grant 61876169.

The authors are with the School of Engineering and Computer Science, Victoria University of Wellington, Wellington 6140, New Zealand (e-mail: gonglin.yuan@ecs.vuw.ac.nz; bin.wang@ecs.vuw.ac.nz; bing.xue@ecs.vuw.ac.nz; mengjie.zhang@ecs.vuw.ac.nz). (Corresponding author: Gonglin Yuan.)

imal vectors. However, because the neural architectures are discrete in nature, the representations are usually composed of integers, which cannot suit the PSO algorithm well. In addition, it is likely to bring good results by using fixed-length continuous representations along with PSO. However, it is not easy to represent complicated network architectures using different lengths by vectors of the same length. In this paper, the architecture representations are transformed to fixed-length decimal vectors to suit PSO well with the help of an autoencoder.

Autoencoders are used for unsupervised representation learning [18], where the encoder part learns intrinsic features of the input data and outputs a corresponding encoded latent representation; the decoder part reconstructs the input data from the latent representation. The encoded latent representations are continuous and usually have fewer dimensions than the original input. Therefore, in this paper, the encoder part of the autoencoder is used to map original discrete architecture representations to the latent space, which can obtain low-dimensional, fixed-length, and continuous representations that are suitable for the PSO algorithm. Because similar networks tend to have similar performance [19], [20], it will further facilitate the search process if similar architectures can be mapped to nearby regions in the latent space. Therefore, both 'architecture similarity' and 'scale similarity' are considered while training the autoencoder.

Furthermore, the computational cost of most NAS algorithms is high. One reason is that the encoding space, i.e., the space that contains all the valid candidate architectures, is too large. Some existing works alleviate this by using block-based encoding space [21], and the encoding space is essentially shrunk by constricting the search to possible block architectures instead of network architectures. The final network is built by repeatedly stacking the blocks. Another usage of the block-based method is to reduce the computational cost on large, complicated datasets by taking advantage of the transferability of the searched block architecture, i.e., transferring the block architecture searched on a small dataset to other large datasets. Human-designed dense blocks [2] achieved good performance, but the block architecture is fixed and hard to adjust to different tasks. In this work, appropriate dense block architectures are automated searched by the proposed algorithm and they are transferred to other tasks.

Another reason for the prohibitive computational cost in NAS is that all the candidate networks need to be trained and evaluated to get their estimated performance during the search process. Researchers have been trying to alleviate this problem. Some researchers adopt an early stopping policy, i.e., only training the networks for a predefined small number of epochs [22]; some algorithms employ a subset of the whole training data to train the candidates, with the assumption that the subset maintains a similar data distribution as the original dataset [23], [24]. A few works use a partial network as the proxy to estimate the performance [25], [26]. While these methods reduce the computational cost, they might also lead to an inaccurate performance estimation. Zhou *et al*. [27] analyzed the inconsistency of different reduction factors systematically using a model pool containing 50 networks. However, the candidate architectures may become more similar and well-performing along the search process. In this regard, assessing the candidates under proxies may not precisely identify the actual promising candidates. We explore the effect of reducing training data used during the evolutionary process and propose an effective hierarchal fitness evaluation method accordingly.

The overall goal of this paper is to design an effective and efficient PSO-based ENAS algorithm, employing a newly designed autoencoder to represent the architectures of candidate networks effectively and adopting a dynamic hierarchical fitness evaluation method to identify well-performing candidates during the search process. To achieve this goal, there are four objectives as follows:

1) Design an encoding scheme to represent the candidate block architectures to suit the search algorithm. An autoencoder could compress the variable-length discrete integer block vectors to fixed-length continuous decimal latent vectors. The latent space is expected to be smooth and continuous, facilitating the downstream PSO search process.
2) Propose a new loss function for the autoencoder that not only considers the reconstruction loss but also takes the architecture similarity and the model scale similarity into account. In this way, the networks with a similar architecture or a similar model scale can be embedded to neighborhood regions in the latent space.
3) Investigate the effect of using different sizes of training data to estimate the fitness values and the impact on different search stages.
4) Present a dynamic hierarchical fitness evaluation method to efficiently and effectively estimate the performance of individuals during different stages of the search process.

The remainder of this paper is organized as follows. Section II introduces the background of the autoencoder and dense blocks, and summarizes related literature. The framework and details of the proposed algorithm are described in Section III. Then, Section IV documents the experiment design information, and Section V exhibits the experiment results and corresponding analysis. At last, the conclusions and the future work are summarized in Section VI.

II. Background

A. Autoencoder

Autoencoder was proposed in the later 1980s as an unsupervised learning algorithm, which was used for feature extraction and dimensionality reduction. An autoencoder is composed of two parts: the encoder and the decoder. The encoder maps the input samples to the latent space, which is the encoding process; the decoder maps the extracted features to the original space to reconstruct the input samples, which is the decoding process. An autoencoder could learn a compressed representation for the input samples by minimizing the distance between the original input and the output. This process does not rely on additional labels, making the autoencoder unsupervised and improving the versatility.

Most traditional autoencoder algorithms only consider the reconstruction distance and ignore the relationship between the different input samples. Generally, we hope the similar input samples can lead to similar encoded latent vectors. This

paper employs both the architecture similarity loss and the model scale similarity loss to address this problem, which could facilitate the downstream search process.

B. Dense Blocks

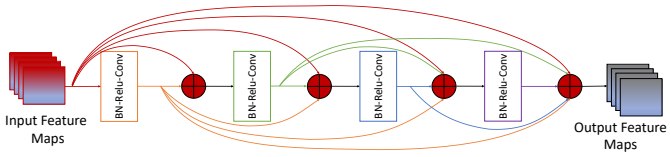

Fig. 1. An example of a four-layer dense block (adapted from [2]).

Dense blocks are the core units in DenseNet [2]. They are used to build deep convolutional neural network architectures in our research because they can extract meaningful features and could alleviate the vanishing gradient problem during the training. Since searching for the effective dense block architecture given a specific dataset is the core task of this paper, Fig. 1 shows the details of a standard dense block.

In the example, the feature maps generated by the previous layers are input to the dense block, and the dense block could further extract powerful features from them. There are four composite layers inside the block in the example, and each layer is composed of three consecutive layers — a Batch Normalization layer, a rectified linear unit (ReLU) layer, and a convolutional layer, which work together to extract features. The input of each composite layer is the output of the previous directly connected layer and the outputs from all the previous layers inside the same dense block, which is different from some other conventional convolutional neural networks.

Hyper-parameters of dense blocks are essential and worth exploring. Firstly, the dense block's length (the number of the composite layers) affects the ability to extract features. Secondly, the growth rate of each composite layer is the number of feature maps generated by the corresponding convolutional layer. One of the core tasks of this research is to effectively and efficiently search for these hyper-parameters.

C. Related Work

In NAS algorithms, how to represent the network architecture is almost the first thing that researchers need to determine, as the encoding strategies significantly impact the search space. Many existing encoding strategies are designed to represent the chained-structured search space: Sun *et al.* [28] use a variable-length chromosome to encode the convolutional layers, the pooling layers, and the fully-connected layers inside one network; Wang *et al.* [29] propose an IP-based encoding strategy which uses a single IP address to represent one layer in the network; Sun *et al.* [30] encode the encoder part of a convolutional autoencoder to a variable-length particle to improve the flexibility and reduce the computational complexity. Some algorithms define a complex search space, and the candidate networks may contain multi-branches and skip-connections, which are described as directed acyclic graphs (DAGs). The encoding methods for these algorithms may be based on adjacency matrixes [31], [32] or paths from the input to the output [33], [34].

Recently, some researchers introduce unsupervised learning to the architecture representations. Yan *et al.* [35] adopt a variational graph isomorphism autoencoder to transform the representations to a continuous space without considering the similarity among architectures. Similar candidate architectures may be embedded into different regions in the latent space. Cheng *et al.* [36] consider the graph similarity to avoid the large inconsistency between similar architectures but ignore the different node operations. Autoencoder plays an important role in unsupervised learning, and it is worth trying to combine it with ENAS algorithms because the low-dimensional latent-space representation may better suit specific EC algorithms. This article employs an autoencoder to help represent the network architectures in a form, which is more suitable for the PSO search process.

The computational cost is also a main focus for many researches on NAS, as many NAS algorithms are usually expensive. For example, on the CIFAR-10 dataset, Zoph *et al.* [31] spend 22,400 GPU-days, AmoebaNet-B [21] costs 3,150 GPU-days, and Real *et al.* [37] spend 2,750 GPU-days. The exhibitive computational cost is not affordable for many researchers, and researchers have been trying to design and employ efficient performance assessment methods to reduce the computational cost. For example, Fan *et al.* [38] dynamically reduce the population size along with the evolution — a large population size is employed at the early stage for the global search, and a small population is used at the final stage to save computing resources. Chu *et al.* [39] train a big supernet, and the candidate networks inherit the weights from the corresponding layers in the supernet instead of being trained from scratch. Unlike the works mentioned above, this paper proposed an efficient and effective hierarchical fitness evaluation method to reduce the computational cost, which uses different amounts of training data along with the evolutionary process.

III. The Proposed Method

In this section, the overall framework and the details of the proposed method will be illustrated and explained. The proposed algorithm is referred to as EAEPSO (efficient autoencoder-based PSO) for convenience.

A. Overall Framework

Fig. 2 illustrates the overall framework of the proposed method. We employ a standard PSO search algorithm, which is shown in the yellow box in the flowchart. To facilitate the evolutionary process of PSO, the particles are expected to be fixed-length vectors of decimles. How to use fixed-length vectors to represent the variable-length candidate dense block architectures is a key focus here, and the procedure is represented in the upper green part. Besides, the procedure of the proposed efficient fitness evaluation method is represented in the left circle part.

For the network architecture representation, the proposed method encodes dense blocks of different lengths into variable-length integer vectors, called block vectors (see Section III-B). The block vectors are transformed by the autoencoder to fixed-length decimal-represented vectors called the latent vectors

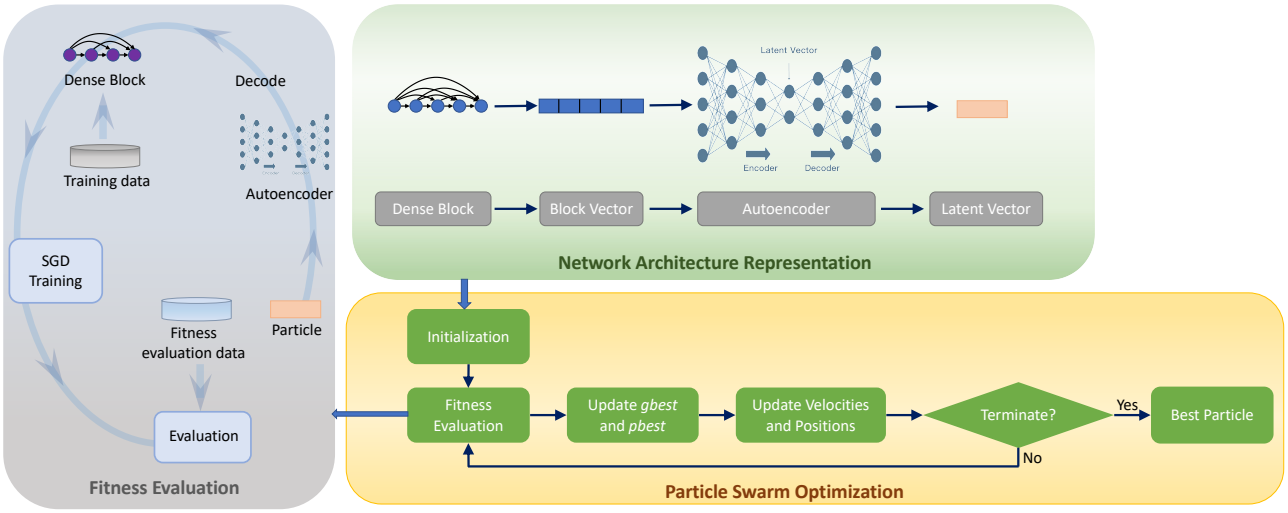

Fig. 2. Overview: an autoencoder is used to represent the dense block architectures, and the variable-length block representations can be compressed into fixed-length latent representations. A PSO algorithm is employed to implement the search process: the particles are initialized and then evaluated and updated until meeting the termination criterion. A new-designed fitness evaluation method is applied: the particle is decoded to dense block and trained by the SGD method on the predefined part of the training dataset and is then evaluated to receive the fitness.

(details in Section III-C). The details of the autoencoder training is illustrated in Section III-D, which includes how to build the autoencoder, prepare the training samples, and the details of the loss function. Latent vectors are the particles in the PSO algorithm and are initialized randomly for the first iteration (see Section III-E).

The proposed hierarchical fitness evaluation method is used to estimate the particles' performance (see Section III-F for details). Specifically, each candidate particle will be decoded to the block vector with the help of the decoder part of the autoencoder and then to the corresponding dense block, which is a reverse procedure of the encoding process. Next, a certain proportion of the training dataset is selected to train the dense block with the stochastic gradient descent (SGD) method. Finally, the dense block is measured on the fitness evaluation dataset, and the accuracy is recorded as its fitness value.

During PSO search process, the particles are updated and evolved, which is kind of global evolutionary search/training. Section III-G exhibits the details, and this process will continue until it meets a stopping criterion. The particle with the highest fitness will be chosen and decoded as the output dense block architecture. At last, the dense block is repeated and stacked to build the final network, and Section III-H shows how to determine the promising number of blocks.

B. Block Vectors to Encode Dense Blocks

Since the proposed method explores the number of layers and the growth rate of each layer of variable-length dense blocks [2], the hyper-parameters of the dense block need to be encoded into a vector, named block vector. Considering the common network scale and the specific hardware condition, a maximum number of layers, l_{max}, of a dense block should be defined. Besides, the minimum number of layers, l_{min}, should also be defined because too few layers in a block would not be able to learn powerful feature maps. The length of the block vector represents the length/number of layers in the dense block, so l_{max} and l_{min} are also the maximum and minimum length of a block vector. A block vector can be any length between l_{min} and l_{max}, where each element represents the growth rate of a corresponding layer.

C. Latent Vectors Transformed by Autoencoder

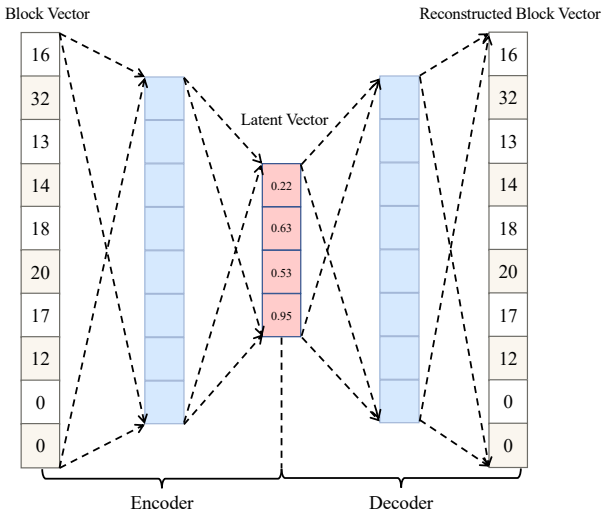

Fig. 3. Autoencoder to transform a block vector to a latent vector.

The block vector representations form a discrete search space, similar to most existing NAS algorithms. The reason is that the network architectures are discrete in nature. However, if we can map the discrete search space to a continuous latent space, the search in the latent space would be more efficient. In addition, if the dimension of the representation in the latent space can be reduced over the block vector representation, it will further improve the search efficiency.

Since PSO can easily process fixed-length vectors, the block vector should be transformed into a fixed-length latent vector. Although we can use fixed-length discrete strings or vectors to represent dense blocks of different lengths by padding zeros to represent the non-existing layers, this will cause significant gaps between the representations of existing layers and non-existing layers, i.e., the large difference between the existing layer's growth rate and zero, which may harm the downstream search process. Besides, PSO performs well in a continuous search space, so it would be better if the latent vector can be represented by decimal numbers. The proposed encoding strategy employs an autoencoder to extract features in the middle layer as the latent vector to achieve a fixed-length decimal-represented vector. Fig. 3 illustrates an example of how the autoencoder transforms a variable-length block vector to a fixed-length latent vector. First of all, the maximum length of the input block vector l_{max} is set to 10 in the example in Fig. 3, but it will be determined according to the available hardware conditions in the experiments and the applications. Secondly, to feed the variable-length block vector to the autoencoder, each variable-length block vector is padded with 0s to reach the length of l_{max}. Two 0s are padded into the block vector in the example. Thirdly, the block vector is processed by the encoder part, composed of several fully-connected layers, and the number of nodes gets smaller. The encoder finally outputs a fixed-length latent vector, consisting of four decimals in the example. It can be directly fed into the PSO algorithm, i.e., latent vectors encoded by the autoencoder will be the particles in PSO. The gaps between different integers and between existing and non-existing layers (represented by 0s) in the block vector representations no longer exist in the latent vector. Finally, with regard to the decoding from the latent vector to the block vector, the decoder part marked in Fig. 3 is used to perform the translation to obtain blocks.

There are several advantages of the transformation from block vectors to latent vectors. Firstly, since the autoencoder is a well-known approach for feature construction [40], it has the ability to extract meaningful features from the input vector. Therefore, the feature variables could well represent the block vector. Secondly, the autoencoder in the proposed encoding strategy has fewer dimensions in the feature variables than that of the input vector, so the dimensionality has been reduced in the transformation. Thirdly, the discrete values of growth rates in the block vector are transformed to continuous values in the latent vector, which suits the PSO algorithm better.

D. Training of the Autoencoder

1) Building the Autoencoder:

Generally, an autoencoder is built following the common convention, i.e., fully-connected layers are employed in both the encoder and decoder parts [18]. In particular, we consider the latent vectors' distribution, i.e., the particles' distribution in the PSO process, while designing the architecture of the autoencoder, so a batch-normalization layer is added to the end of the encoder part. This layer would adjust all the elements of the same place in a batch to obtain a Gaussian distribution with a mean of 0 and a variance of 1. In this way, the architecture representation in the latent space will become smooth, which may facilitate the downstream search on the latent space [35].

2) Sampling Training Data:

Before the PSO search process, the autoencoder is independently trained. Specifically, while training the weights of the autoencoder, we consider the relationship between the similarity among block vectors, and that among latent vectors. Considering the similarity, the autoencoder is supposed to be trained by pair-wised vectors. So we need to create two datasets of the same size. During training, the same number of block vectors are randomly selected from the two datasets to compose pair-wised training data for each batch.

Algorithm 1: Generating Block Vectors for Autoencoder Training

Input: The minimal and maximal number of layers l_{min}, l_{max}; the minimal and maximal growth rates g_{min}, g_{max}; the number of instances n.
Output: The block vector dataset D_b.

1 $D_g \leftarrow \emptyset$;
2 **for** $i = 1;\ i \leq n;\ i \leftarrow i + 1$ **do**
3 $\quad block\ vector \leftarrow \emptyset$;
4 $\quad n_{layer} \leftarrow$ Randomly generate an integer between $[l_{min}, l_{max}]$;
5 $\quad$ **for** $j = 1;\ j \leq n_{layer};\ j \leftarrow j + 1$ **do**
6 $\quad\quad n_{growth} \leftarrow$ Randomly generate an integer between $[g_{min}, g_{max}]$;
7 $\quad\quad block\ vector \leftarrow block\ vector \cup n_{growth}$;
8 $\quad$ **end**
9 $\quad$ **for** $k = n_{layer} + 1;\ k \leq l_{max};\ k \leftarrow k + 1$ **do**
10 $\quad\quad block\ vector \leftarrow block\ vector \cup 0$;
11 $\quad$ **end**
12 $\quad D_b \leftarrow D_b \cup block\ vector$
13 **end**
14 **Return** D_b.

Each dataset is composed of a number of block vectors, and the block vector data generating process is shown in Algorithm 1. First of all, block vectors need to be defined by a few parameters. Besides the maximum and minimun number of layers l_{max} and l_{min} mentioned in Section III-B, the maximum growth rate g_{max} and the minimum growth rate g_{min} are also needed. The convolutional layer will not extract enough meaningful features if the corresponding growth rate is too small. While the maximum growth rate mainly depends on the hardware resource, a too-large maximum growth rate may result in the issue of out of memory in graphic processing units (GPUs) and also increase the computation time. A less complex dataset would typically require a lower maximum growth rate.

The block vectors are generated repeatedly until reaching the predefined number of instances in the dataset n (lines 2-13). Specifically, for a $block\ vector$, the specific number of layers is randomly chosen from the predefined range (line 4), and the number of feature maps of each layer is within the minimal and maximal growth rates to give the same possibilities to each possible value (line 6). Furthermore, the proposed algorithm sets the last few layers' growth rates to 0 by looping from $n_{layer} + 1$ to l_{max} (lines 9-10). Finally, all of the generated data are stored (line 12) and will be used to train the autoencoder.

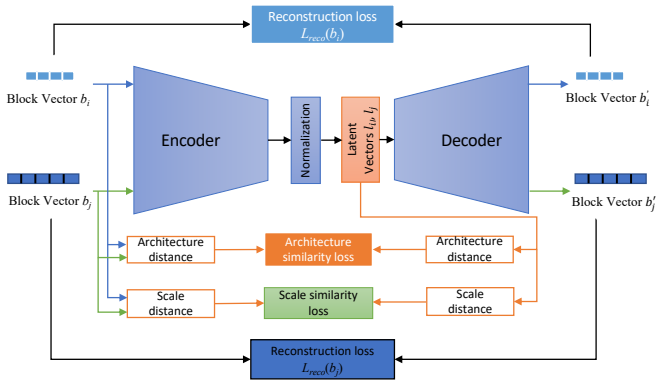

Fig. 4. An example of the autoencoder training: Two block vectors are corresponding to two different dense blocks, and they are input into the autoencoder. The encoder part generates two latent vectors, respectively. The decoder part outputs two reconstructed block vectors. The reconstruction loss between the input block vectors and the output reconstructed block vectors are calculated. The architecture distance between two input block vectors and the architecture distance between two latent vectors are used to evaluate the architecture similarity loss. The scale distance between the two block vectors and the scale distance between the two corresponding latent vectors are used to calculate the scale similarity loss.

3) The Loss Function:

A new loss function is designed, and Fig. 4 exhibits four different kinds of loss when using a pair of block vectors to train the autoencoder. The overall loss function of two block vectors b_i and b_j is shown as:

$$L(b_i, b_j) = L_{reco}(b_i) + L_{reco}(b_j) + L_{arch}(b_i, b_j) + L_{scale}(b_i, b_j) , \quad (1)$$

where $L_{reco}(b_i)$ and $L_{reco}(b_j)$ are the reconstruction loss of b_i and b_j, respectively. They can be calculated according to Equation (2). $L_{arch}(b_i, b_j)$ is the arechitecture similarity loss, which is calculated according to Equation (6), and $L_{scale}(b_i, b_j)$ represents the scale similarity loss between b_i and b_j, which is calculated according to Equation (9).

Specifically, the original target of an autoencoder is to minimize the reconstruction loss. The reconstruction loss of a block vector b_m is shown as:

$$L_{reco}(b_m) = \sum_{k=1}^{l_{max}} (b_{m,k} - \mathcal{A}(b_{m,k}))^2 , \quad (2)$$

where $\mathcal{A}(\cdot)$ refers to the autoencoder's processing, $b_{m,k}$ is the k-th bit in the m-th block vector, and l_{max} is the maximal length of block vectors.

To project the similar architecture to a neighborhood latent space, we also designed the architecture similarity loss L_{arch} and the scale similarity loss L_{scale} considering the similarity/difference between different dense blocks. If the dense blocks are similar, they usually lead to similar performance [19]. In this regard, their corresponding latent vectors are also expected to be similar. If the particles of similar fitness are distributed in the neighborhood space, the search could be more efficient and effective. To measure the similarity among dense blocks, we consider both the architectures and the model scales. In the employed block representation strategy, the differences among the candidate block architectures are the number of layers and the number of feature maps of each layer; we use the difference in each layer's number of feature maps to define the architecture difference and employ the sum of all layers' feature maps to approximate the model scale. In this way, both the depth and the width of the networks are taken into account when approximating the model scale.

To calculate the architecture similarity loss L_{arch}, we need to calculate the architecture distance first. As the element values in latent vectors are between 0 and 1, we also nomalize the element values in block vectors to the same range according to Equation (3) for the convenience of calculation, where $b^*_{m,k}$ is the k-th element value in the m-th block vector after normalization. g_{min} and g_{max} indicate the minimal and maximal growth rates, representing the minimal and maximal number of feature maps in dense block layers, respectively. Equation (4) and Equation (5) calculate the architecture distance between two block vectors b_i and b_j and between two latent vectors l_i and l_j, where l_{max} and h_{max} are the maximal length of the block vectors and latent vectors, respectively. $b^*_{i,k}$ and $b^*_{j,k}$ are the k-th element values of the i-th and j-th normalized block vectors, and $l_{i,k}$ and $l_{j,k}$ are the k-th element values of the i-th and j-th latent vectors. The architecture similarity loss is calculated according to Equation (6).

$$b^*_{m,k} = \frac{b_{m,k} - g_{min}}{g_{max} - g_{min}} \quad (3)$$

$$D_{arch}(b_i, b_j) = \frac{\sum_{k=1}^{l_{max}} | b^*_{i,k} - b^*_{j,k} |}{l_{max}} \quad (4)$$

$$D_{arch}(l_i, l_j) = \frac{\sum_{k=1}^{h_{max}} | l_{i,k} - l_{j,k} |}{h_{max}} \quad (5)$$

$$L_{arch}(b_i, b_j) = [D_{arch}(b_i, b_j) - D_{arch}(l_i, l_j)]^2 \quad (6)$$

For the scale similarity loss L_{scale}, the model scale distance between b_i and b_j and two corresponding latent vectors' scale distance are given by Equation (7) and Equation (8). The overall scale loss between b_i and b_j is represented by Equation (9).

$$D_{scale}(b_i, b_j) = \sum_{k=1}^{l_{max}} b^*_{i,k} - \sum_{k=1}^{l_{max}} b^*_{j,k} \quad (7)$$

$$D_{scale}(l_i, l_j) = \sum_{k=1}^{h_{max}} l_{i,k} - \sum_{k=1}^{h_{max}} l_{j,k} \quad (8)$$

$$L_{scale}(b_i, b_j) = [D_{scale}(b_i, b_j) - D_{scale}(l_i, l_j)]^2 \quad (9)$$

Algorithm 2: Particle Initialization

Input: The population size N, the trained autoencoder.
Output: Initialized population P_0.

1 $P_0 \leftarrow \emptyset$;
2 **for** $i = 1;\ i \leq N;\ i \leftarrow i + 1$ **do**
3 $\quad b_i \leftarrow$ Randomly generate a block vector;
4 $\quad l_i \leftarrow$ Encode b_i by the encoder of the trained autoencoder;
5 $\quad P_0 \leftarrow P_0 \cup l_i$;
6 **end**
7 **Return** P_0.

E. Particle Initialization

Algorithm 2 exhibits how to initialize the particles at the beginning of the PSO searching process. Particles are generated until reaching the predefined population size N (lines 2-6). For each particle, a block vector is randomly generated according to lines 3-12 in Algorithm 1 (line 3 in Algorithm 2). It is then encoded by the trained autoencoder to the corresponding latent vector (line 4). Finally, the first population with randomly generated particles will be generated.

F. Fitness Evaluation

Fitness evaluation is the most time-consuming process in ENAS algorithms. We investigated the use of reduced training data to propose an efficient and effective hierarchical fitness evaluation method. In most cases, training the candidate networks with a reduced dataset is likely to lead to lower test accuracy than training using the whole dataset. However, the fitness evaluation in the EC technique is usually used to guide the selection, which aims at evaluating how good each individual is to identify which network is better, so we care more about the relative goodness of the individuals, i.e., the relative fitness ranking relationship among individuals. This could be achieved by a (surrogate) measure if the measure can correctly identify the relative goodness of the individuals. There are three main considerations:

(1) Reducing the training dataset often reduces the test accuracy of the candidate, along with harming the ranking consistency between the ranking based on estimated performance and the ranking based on true performance. The less training data is used, the lower ranking consistency may lead to.
(2) The approximate ranking performance received by using reduced training data could also guide the evolutionary process, primarily at the beginning of the evolution because there is considerable variation among candidates.
(3) The best candidate found by more training data usually also performs well if trained by less training data. However, the best candidate found by less training data may not be the best one when given more training data.

Based on the above three considerations, we designed a new efficient and effective hierarchical fitness evaluation method for the PSO algorithm, which dynamically increases the amount of the training data along with the evolutionary process. At the beginning of the evolution, in order to reduce the computational cost, we develop a basic fitness evaluation method, which only employs a small number of training data instances and consumes cheap computing resources. As there is a lot of variation among particles at the beginning of evolution, and the performance is not promising, the basic fitness evaluation method could guide the evolution effectively. As the evolution goes, when the basic fitness evaluation cannot further guide the searching process effectively, the developed progressive fitness evaluation method will be used, and its main idea is to use more training data to improve the rank consistency and select local and global best particles more accurately.

Algorithm 3: Basic Fitnees Evaluation

Input: The population P, the maximum training epochs k_{bmax}, the training loss criterion l_{bt}, the proportion of the training data r_b, the whole training dataset D_{train}, the fitness evaluation dataset D_{fit}.
Output: The population with fitness.

1 $D_{btrain} \leftarrow$ Randomly select r_b of D_{train};
2 **for** p *in* P **do**
3 $\quad reconstructed\ block\ vector \leftarrow$ Decode p by the decoder of the autoencoder;
4 $\quad block \leftarrow$ Decode $reconstructed\ block\ vector$ to the dense block;
5 $\quad epoch,\ loss \leftarrow 0,\ +\infty$;
6 $\quad$ **while** $epoch \leq k_{bmax}$ and $loss \geq l_{bt}$ **do**
7 $\quad\quad$ Apply SGD to train $block$ on D_{btrain};
8 $\quad\quad loss \leftarrow$ Update by the training loss;
9 $\quad$ **end**
10 $\quad acc \leftarrow$ Evaluate $block$ on D_{fit};
11 $\quad p \leftarrow$ Update the fitness of p;
12 **end**
13 **Return** P with the updated fitness.

Specifically, the basic fitness evaluation method is presented in Algorithm 3. According to the predefined proportion r_b, the training data is randomly selected from the whole training dataset (line 1). Next, every particle in the population is evaluated (lines 2-12). Specifically, the particle is decoded to the corresponding dense block vector by the decoder part of the autoencoder (line 3) and then is interpreted to the dense block (line 4). The dense block will be trained until the training epoch reaches the predefined maximal epoch, or the training loss is lower than the predefined criterion (lines 6-9). Note that the dense block will not be evaluated on the fitness evaluation dataset until the training is finished. This is different from some existing performance estimation methods in NAS, which evaluate the candidate's performance after each training epoch and choose the highest accuracy as the corresponding fitness. Because at the early stage of the training, the candidate's performance on the evaluation dataset is relatively poor, but it will get better along with the training process. So it will consume much unnecessary time if the evaluation is performed for each epoch.

The weight parameters and the training loss will be updated during the training (lines 7-8). After training, the candidate block will be evaluated (line 10), and its fitness is updated by the accuracy on the evaluation dataset (line 11).

Algorithm 4: Progressive Fitnees Evaluation

Input: The population P, the population size N, the maximum progressive training epochs k_{pmax}, the progressive training loss criterion l_{pt}, the proportion of the progressive training data r_{prog}, the whole training data D_{train}, the fitness evaluation dataset D_{fit}.

Output: The population with both basic fitness and progressive fitness.

1 $D_{ptrain} \leftarrow$ randomly select r_{prog} of D_{train};
2 **for** $i = 1;\ i \leq N;\ i \leftarrow i + 1$ **do**
3 Employ Algorithm 3 to achieve the acc_i on $Dfit$;
4 $p_i \leftarrow$ Update the basic fitness with acc_i;
5 $block_i \leftarrow$ Save the model with the trained weights;
6 **end**
7 **for** $i = 1;\ i \leq N;\ i \leftarrow i + 1$ **do**
8 **if** acc_i is in the top third of P **then**
9 Load $block_i$;
10 $epoch, loss \leftarrow 0, +\infty$;
11 **while** $epoch \leq k_{pmax}$ and $loss \geq l_{pt}$ **do**
12 Apply SGD to train $block_i$ on D_{ptrain};
13 $loss_p \leftarrow$ Update by the training loss;
14 **end**
15 $acc_p \leftarrow$ Evaluate $block$ on D_{fit};
16 $p_i \leftarrow$ Update the progressive fitness with acc_p;
17 **else**
18 $p_i \leftarrow$ Update the progressive fitness with 0;
19 **end**
20 **end**
21 **Return** P with the updated basic and progressive fitness.

Along with the evolution, the basic fitness evaluation method may not accurately evaluate the individuals' performance, and a progressive fitness evaluation method will then be employed. The details of the proposed progressive fitness evaluation method is presented in Algorithm 4. As mentioned earlier, the candidates that perform well on a larger training dataset usually also achieve good performance on a less training dataset. Two training datasets with different scales are used to evaluate the fitness to balance efficiency and effectiveness. The progressive training dataset D_{ptrain} is built according to the predefined portion r_{prog} (line 1), and the scale is larger than the basic training dataset mentioned in Algorithm 3. Then, all the particles are trained by the basic training dataset (lines 2-4) to get their basic fitness, which is similar to Algorithm 3, but their corresponding model and trained weights will be saved (line 5). The basic fitness is used for selecting the local best particle. Next, each particle's progressive fitness is evaluated (lines 7-20). If the particle's basic fitness is good, it is more likely to perform well when training by the progressive training dataset. To save the computational resources, we only select the top third of particles (line 8), which will be further trained (lines 9-16). Specifically, the model trained by the basic training dataset is loaded (line 9) and is further trained by the larger progressive training dataset (lines 10-14). Then the network is measured by the fitness evaluation dataset (line 15), and the performance is recorded as the progressive fitness (line 16). For the particles with poor basic fitness, their progressive fitness is directly set to 0 (line 18), which can improve the search efficiency.

G. Evolving Dense Blocks

After the encoding transformation, fixed-length latent vectors are achieved, which can feed to the PSO algorithm. Then the dense blocks evolve in the PSO process based on the basic and progressive fitness evaluation methods mentioned in Section III-F. Please note that there is no fixed training data proportion for the progressive fitness evaluation method. A set of gradually increasing progressive training dataset proportions will be provided as $[r_{prog1}, ..., r_{progn}]$.

Specifically, the population is initialized according to Section III-E first. After that, all the particles are evaluated based on Algorithm 3 to get their basic fitness, and then select/update the personal best particle for each one and select/update the global best one for the population, and all particles are then updated. All the particles will be evaluated by the basic fitness evaluation method and updated repeatedly until the global best fitness has not increased for five consecutive iterations, when the current evolution is considered not to be able to guide the search further. The particles will be evaluated by the proposed progressive fitness evaluation method according to Algorithm 4 with r_{prog1} as the training data proportion, and then personal best particles and the global best one are updated based on the basic fitness and progressive fitness respectively, and all the particles are then updated. Similarly, particles will be repeatedly evaluated and updated until global best fitness has not been updated for five consecutive iterations. Then, the next progressive training proportion r_{prog2} is used to fit the progressive evaluation method and help to evaluate and update the particles. The evolutionary process will continue until the last progressive training proportion r_{progn} is used. Finally, the global best particle $gBest$ is selected and decoded to the network architecture as the evolved dense block.

H. Stacking Dense Blocks

The evolved dense block needs to be stacked to get the whole network, and the promising number of blocks is hard to be determined manually. The entire training dataset is employed, and a simple grid search process is designed to determine how many blocks should be stacked together.

Before the search process, a maximum number of blocks s_{max} needs to be predefined considering the hardware resources and the image size. If s_{max} is too big, the hardware storage limit may be overloaded. Besides, a pooling layer is usually attached to each dense block, which means the size of the feature maps will be reduced to half after processing by a dense block. The minimum limit of the size of the feature maps is 1×1, so the initial input image size constrains s_{max}.

Specifically, several networks are built using the repeated dense blocks, and the numbers of blocks are increased from one to s_{max}. For each network, 80% of the training dataset is selected as the training part, and the other 20% is selected as the evaluation part. Each network is trained by the training part and further measured on the evaluation part, and the performance on the evaluation part is used to evaluate the

network. The network with the best performance on the evaluation part is selected as the final solution to the specific task.

IV. Experiment Design

In this section, we offer the design of the experiment. The employed benchmark datasets are introduced first. The selected peer competitors are then discussed. Finally, the specific parameter settings of EAEPSO are exhibited.

A. Benchmark Datasets

Considering the hardware resources, we choose three widely-used image classification benchmark datasets: CIFAR-10, CIFAR-100 [41], and Street View House Number (SVHN) [42] to verify the effectiveness and efficiency of the proposed EAEPSO algorithm. We also employ CIFAR-100 and ImageNet to test the performance of the network searched on CIFAR-10, which could verify the good transferability of the architecture searched by EAEPSO. Another reason to choose these three datasets is that many famous manual-designed networks and NAS algorithms are also tested on them, which can offer a good comparison.

There are 60,000 color images in the CIFAR-10 dataset. The image size is 32×32, and there are three channels: R, G, and B. 50,000 images are used for training, and the other 10,000 images are for assessing the model's performance. It contains ten categories, such as plane, automobile, bird, cat, and deer. There are objects in the real world; the noise in the images is loud, and the scales and features of the objects are also various, which brings great difficulties to classify. The number of images in these ten categories is evenly distributed, i.e., for each category, there are 5,000 training images and 1,000 test images. As for CIFAR-100, the image size and the number of images are the same as CIFAR-10. However, there are 100 categories, i.e., there are only 600 images for each category, and 500 of them are used for training. More classes and fewer training examples for each class make the classification task on CIFAR-100 much more challenging than on CIFAR-10. ImageNet is a large-scaled dataset: about 15,000,000 images belong to about 1,000 classes.

The data augmentation technique is not employed during the NAS process but is performed for the post-search training process, because the fitness evaluation method used in the NAS process could effectively compare the performance of different candidate networks even without data augmentation. The augmentation method used in post-search training not only includes cropping, but also includes *cutout* [43]. However, we did not employ *dropout* or *scheduled path dropout* [44] techniques to furtehr improve the performance.

B. Peer Competitors

To verify the effectiveness and efficiency of the proposed algorithm, we select some state-of-the-art algorithms and compare them with EAEPSO. These peer competitors could be broadly categorized into two categories according to whether the CNN is designed manually. In the first category, the CNNs are designed by human experts. They are FractalNet [45], Maxout [46], ResNet [1], DenseNet [2], Highway Network [47], VGG [48], SENet [3], and WRN-18 [49]. Most competitors are trained and tested on both CIFAR10 and CIFAR100. The second category automatically searches and constructs the network architectures, such as CGP-CNN [50], NAS [31], Large-scale Evolution [37], Block-QNN [51], MetaQNN [5], EIGEN [52], CNN-GA [53], PNASNet [7], AmoebaNet [21], EAS [6], NASNET [54], AECNN [55], DENSER [56], GeNet [9], CoDeepNEAT [57], Hier. repr-n, evolution [8], EffPnet [17], DARTS [4], NSGA-Net [58], LEMONADE [59], NSGANetV1-A1 [60], Proxyless NAS [61], AE-CNN+E2EPP [62], EffPnet [17], and MobileNet [63].

C. Parameter Settings

The parameter settings in EAEPSO follow the convention of the PSO algorithm and deep learning, and we also consider the computational capability of the resources available — all the experiments are implemented on the GPU cards of NVIDIA A6000.

Specifically, while employing the autoencoder to represent the dense block architecture, the minimal and maximum length of block vectors are set to 10 and 20 considering the hardware limit and the model scales of the prevalent networks [2], [17], [49]. Considering the autoencoder's scale and the convention of dimension reduction [18], the latent vectors' length is set to 8. In block vectors, the growth rates are between 10 and 32, which is similar to the width of the networks that work well. When building the dense block, following the convention, all the convolutional layers' kernel sizes are set to 3, and the stride is 1. As sampling block vectors is relatively cheap, we sample 300,000 block vectors as the training data for the autoencoder training, and they are trained by an Adam optimizer [64] with a learning rate of 5×10^{-3} for 500 epochs.

For the evolutionary process, the population size is 30. The regular parameters of PSO follow the convention: the inertia weight is 0.7298, and the two acceleration coefficients are both 1.49618 [65]. The velocity range is between -0.1 and 0.1. In the proposed dynamic fitness evaluation method, the proportion for the basic evaluation is 10%, and the maximal number of training epochs is set to 60. In progressive fitness evaluation, the number of training epochs is 50, and there are two predefined training dataset proportions: 20% and 40%. When stacking the block, the maximum number of blocks is 5.

For the post-search training process, the settings are followed [2]. Specifically, the searched network is trained for 300 epochs, and the initial learning rate is set to 0.1, which changes to 0.01 and 0.001 when the training epoch achieves 150 and 225, respectively.

V. Results and Analysis

In this section, the experimental results of EAEPSO against peer competitors are presented in Section V-A. Then, the comparision results and corresponding analyses between the proposed autoencoder and the comparative one are presented in Section V-B. At last, the comparison among different fitness evaluation methods and different training dataset scales' impact on the rank consistency during the evolutionary process are investigated in Section V-C.

A. Overall Results

We run EAEPSO on both the CIFAR-10 dataset and the CIFAR-100 dataset five times with different initial random seeds, respectively. We present the experimental results with the best error rate, the median error rate, and the average performance. Besides, the block architecture searched on CIFAR-10 (with the median error rate) is transferred to CIFAR-100 and ImageNet to prove its transferability.

The results of EAEPSO are compared with manually designed methods from two aspects, i.e., the predictive error rate and the model size. In order to demonstrate the efficiency of EAEPSO, we also compare the computational cost with the NAS peer competitors, which is measured by the GPU-days[1] of the searching process. The results on SVHN are presented in supplementary materials.

TABLE I
THE COMPARISONS ON THE **CIFAR-10** DATASET.

Model	#Parameters	Error Rate	GPU-Days
FractalNet [45]	38.6M	5.22%	—
Maxout [46]	—	9.3%	—
ResNet-101 [1]	**1.7M**	6.43%	—
DenseNet (k=24) [2]	27.2M	3.74%	—
Highway Network [47]	—	7.72%	—
VGG [48]	20.04M	6.66%	—
CGP-CNN [50]	**2.64M**	5.98%	27
NAS [31]	**2.5M**	6.01%	22,400
Large-scale Evolution [37]	5.4M	5.4%	2,750
Block-QNN-S [51]	6.1M	4.38%	90
MetaQNN [5]	—	6.92%	100
EIGEN [52]	**2.6M**	5.4%	**2**
CNN-GA [53]	**2.9M**	4.78%	35
PNASNet-5 [7]	3.2M	3.41%	150
AmoebaNet-B [21]	34.9M	2.98%	3,150
EAS [6]	23.4M	4.23%	<10
NASNET-A [54]	27.6M	2.97%	2,000
AECNN [55]	**2.0M**	4.3%	27
DENSER [56]	10.81M	5.87%	—
GeNet from WRN [9]	—	5.39%	100
CoDeepNEAT [57]	—	7.3%	—
Hier. repr-n, evolution [8]	—	3.63%	300
EffPnet [17]	**2.68M**	3.58%	<3
DARTS [4]	3.4M	2.82%	**1**
NSGA-Net [58]	3.3 M	2.75%	4
LEMONADE [59]	13.1M	**2.58%**	90
NSGANetV1-A1 [60]	**0.5M**	3.49%	27
Proxyless NAS [61]	5.7M	**2.08%**	1,500
EAEPSO (Best)	3.6M	2.51%	3
EAEPSO (Median)	2.94M	2.74%	2.2
EAEPSO (Average)	3.17M	2.75%	2.8

1) Performance on CIFAR-10: Table I lists the number of parameters, the predictive error rate, and the GPU-days of the searching process of both EAEPSO and the other baseline methods on the CIFAR-10 dataset. EAEPSO (Median) achieves an error rate of 2.74% on CIFAR-10, and two peer competitors outperform it, whose error rates are in bold font in Table I. Specifically, for LEMONADE [59] and Proxyless NAS [61], the model scales are 4.46 times and 1.94 times of EAEPSO (Median), and the computational costs are 41 and 682 times of that of EAEPSO (Median). The best error rate is only 2.51%, and the average error rate is similar to the median value. As EAEPSO is based on the DenseNet structure, we further implement a One-Sample T-Test between EAEPSO and DenseNet, and the P-value is 0.003, showing the proposed algorithm is statistically significantly better than the original DenseNet. In terms of the model scale, EAEPSO (Median)'s 2.94M ranks ninth, and eight competitors are with a slightly smaller number of parameters. Nevertheless, their predictive error rates are all worse than EAEPSO (Median)'s. The average model scale is 3.17M, which is a little larger than EAEPSO (Median). Concerning the computational cost of the searching process, EAEPSO (Median)'s 2.2 GPU-days ranks third over the NAS methods, and EIGEN's [52] 2 GPU-days and DARTS's [4] 1 GPU-day outperform it. However, EIGEN's predictive accuracy is 1.78% worse than EAEPSO (Median), and DARTS's model scale is 0.46M bigger than EAEPSO (Median) along with a worse predictive accuracy. The average computational cost is 2.8 GPU-days, which is still significantly smaller than most of the NAS methods, as many of them spend hundreds or even thousands of times of EAEPSO's GPU-days. Overall, EAEPSO demonstrates it is a very competitive method by comparing it with the 28 peer competitors on the CIFAR-10 dataset.

[1]Strictly speaking, GPU-days are not accurate since GPUs can be different, but GPU-days can be used as a good indicator, like many other papers [9], [17], [21], [53].

TABLE II
THE COMPARISONS ON THE **CIFAR-100** DATASET.

Model	#Parameters	Error Rate	GPU-Days
FractalNet [45]	38.6M	22.3%	—
Maxout [46]	—	38.6%	—
ResNet-101 [1]	**1.7M**	25.16%	—
DenseNet (k=40) [2]	25.6M	17.2%	—
Highway Network [47]	—	32.39%	—
VGG [48]	20.04M	28.05	—
SENet [3]	34.4M	**15.41**	—
Large-scale Evolution [37]	40.4M	23%	>2730
Block-QNN-S [51]	6.1M	20.65%	90
MetaQNN [5]	—	27.14%	100
EIGEN [52]	11.8M	21.9%	5
CNN-GA [53]	**4.1M**	20.03%	40
NSGA-Net [58]	**3.3M**	20.74%	8
PNASNet-5 [7]	**3.2M**	19.53%	150
ENAS [66]	**4.6M**	19.43%	**0.5**
AmoebaNet-A [21]	**3.1M**	18.93%	3,150
DARTS [4]	**3.4M**	17.54%	**1**
NSGANetV1-A1 [60]	**0.7M**	19.23%	27
AE-CNN+E2EPP [62]	20.9M	22.02%	10
EffPnet [17]	—	18.70%	—
EAEPSO	5.42M	16.17%	4
EAEPSO(Transfer)	4.32M	16.94%	2.2

2) Performance on CIFAR-100: Table II exhibits the performance of EAEPSO, EAEPSO(Transfer), and some peer competitors on the CIFAR-100 dataset. EAEPSO's error rate ranks second among eleven peer competitors, and only the error rate of manually-designed SENet [3] is 0.76% lower than it. Nevertheless, the number of parameters of SENet is 6.35 times of EAEPSO. In regard to the number of parameters, eight peer competitors have more minor model scales than EAEPSO, but they are also with higher error rates. As for the computational cost, EAEPSO's 4 GPU-days ranks third among the 13 NAS methods. ENAS [66] and DARTS [4] have less searching time, however, they also have higher error rates. So we can say EAEPSO achieves very competitive performance

on CIFAR-100 considering the model scale, predictive error rate, and computational cost.

3) Transferability Performance: In order to justify the transferability of the searched block architecture, we transfer the block searched on CIFAR-10 to other datasets — CIFAR-100 and ImageNet. The blocks sharing the same CIFAR-10 architecture are stacked together to explore the appropriate number of blocks on the new dataset.

The experimental results on CIFAR-100 are showed in Table II, denoted as 'EAEPSO (Transfer)'. The error rate of EAEPSO (Transfer) is 16.94%, which is only 0.77% higher than directly searching the block architecture on CIFAR-100, but the model size is only 4.32M, which is smaller than EAEPSO on CIFAR-100. Please note that the GPU-days of EAEPSO (Transfer) contains two parts: the computational cost of searching for the promising block architecture on CIFAR-10 and the time-consuming of stacking the block to determine the promising number of blocks on CIFAR-100. EAEPSO (Transfer)'s GPU-days is only 2.2 GPU-days, which is the second lowest among all the competitors.

TABLE III
THE COMPARISONS ON THE **IMAGENET** DATASET.

Model	#Parameters	Error Rate	GPU-Days
NAGANetV1-A1 [60]	**3.0M**	29.1%	27
MobileNet-V2 [63]	**3.4M**	28.0%	—
NSGANet-A [54]	5.3M	**26.0%**	1,575
AmoebaNet-A [21]	5.1M	**25.5%**	3,150
WRN-18 [49]	11.7M	30.4%	—
PNASNET-5 [7]	5.1M	**25.8%**	150
EffPnet [17]	—	27.0%	—
EAEPSO(Transfer)	4.9M	26.9%	**4**

Table III presents the results on ImageNet. ImageNet is more complex than CIFAR-10 and CIFAR-100, because the input size is much larger; the number of classes is 1000 which is harder to perform classification and needs 1000 output nodes in the classification layer, so the model scale and the classification error rate are larger on ImageNet. Specifically, in terms of the error rate, three peer competitors outcome EAEPSO(Transfer), but their numbers of parameters are all a little larger than EAEPSO(Transfer). Their computational cost is much larger than EAEPSO(Transfer). One reason is that they directly search the architecture on ImageNet; on the other hand, EAEPSO(Transfer) searches the block structure on CIFAR-10.

The experimental results indicate the block architecture evolved by EAEPSO has a very good transferability, which can lead to a satisfactory predictive accuracy on the target domain. Besides, we can take advantage of the transferability to improve the search efficiency when encountering a huge dataset by transferring the block architecture searched on a smaller domain dataset to the target large dataset, reducing the computational cost and help people solve larger problems.

B. Analysis on Autoencoder

The objective of this section is to investigate the effectiveness of the designed autoencoder, which contains a special normalization layer at the end of the encoder part, and employs pair-wised training samples and the proposed loss function during the training process. To achieve this objective, a comparative autoencoder is built, which has the same architecture as the proposed one except for not including the specific normalization layer. The comparative autoencoder is called CAE, which is also trained with the same training dataset for the same number of training epochs, but the samples do not need to be input pair-wisely, and its loss function only contains the reconstructive error without the architecture similarity loss or the scale similarity loss.

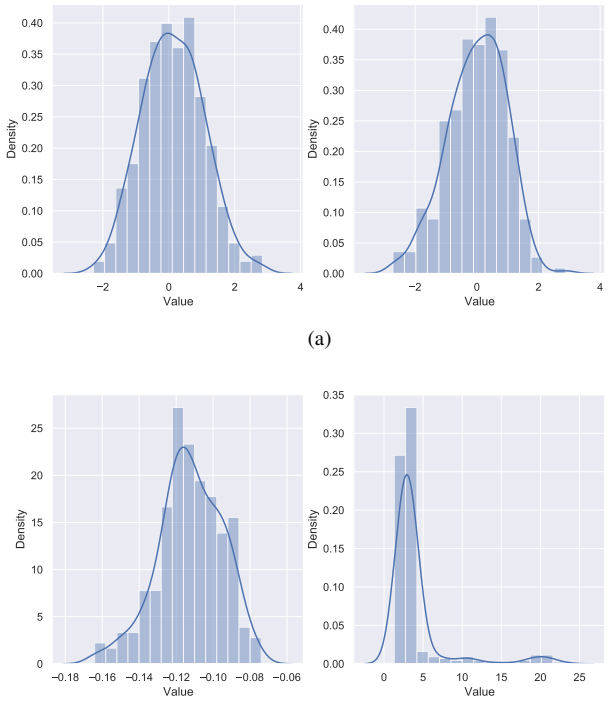

Fig. 5. The probability density curve of two element values of latent vectors. (a) Produced by the proposed autoencoder, which contains a normalization layer at the end of the encoder part. (b) Produced by CAE.

We compare the latent space produced by the autoencoder in EAEPSO and by CAE. Specifically, we sample some random dense blocks and use the two autoencoders to represent them into 8-element latent vectors. We visualize two randomly selected element values' probability density in Fig. 5. As shown in the figure, for the autoencoder used in EAEPSO, the distribution of the values in latent blocks is very smooth and follows the Gaussian distribution, which may facilitate the downstream search [35]. On the contrary, the latent space is not so smooth for CAE. The first probability density is relatively smooth, and most values also follow almost a Gaussian distribution. However, the second one is not smooth at all. Most values are concentrated around three and four and accompanied by a long tail distribution, which may be hard to make an effective and efficient search.

Because similar candidates are more likely to have comparable performance, we hope similar candidates could also be close in the latent space. We plot the relationship between the L_1 distance in the latent space and that in the normalized block vectors in Fig. 6. Similar dense block architectures tend

to have a small latent block distance in EAEPSO. On the other hand, the relationship between the block vector distance and the latent vector distance is obviously non-linear for CAE, which means similar candidates may have a large latent vector distance, and this may harm to the downstream search process.

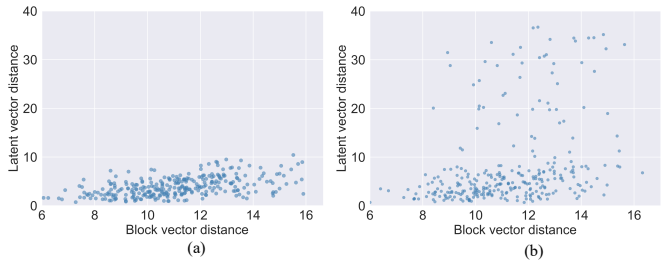

Fig. 6. The Relationship between the L_1 distance of block vectors and that of corresponding latent vectors. (a) Produced by the proposed autoencoder, which considers the architecture similarity loss. (b) Produced by CAE.

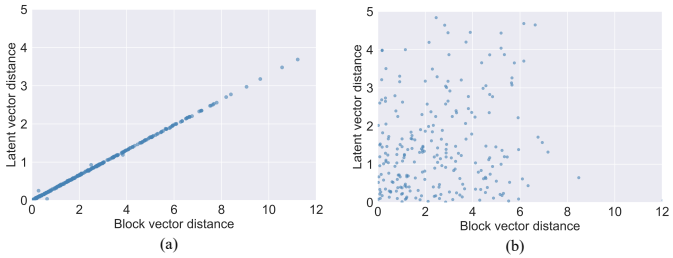

Fig. 7. The relationship between the L_1 distance of the sum of values in block vectors and that in corresponding latent vectors. (a) Produced by the proposed autoencoder, which considers the scale similarity loss. (b) Produced by CAE.

We use the sum of all the values in the block vector to approximate the model scale and compare the relationships between the sum of values in block vectors and that in corresponding latent vectors generated by the two autoencoders, which is exhibited in Fig 7. With the help of the autoencoder in EAEPSO, the dense blocks of a similar scale also have a similar sum of values in the latent vectors, and the linear relationship is apparent. In this way, the candidates with the similar model scales are tend to be in the same region in the latent space, which may be helpful to the search process. On the contrary, the sum of values in latent vectors cannot reflect the model scale in CAE.

These comparative experiments show that the proposed autoencoder could transform the original dense blocks' block vectors to a smooth latent space, reflecting the architecture similarity and the model scale similarity among different dense blocks. In this way, the candidates with similar architecture and scale are more likely to concentrate together in the latent space.

C. Analysis on different training data scales

To prove the effectiveness and efficiency of the proposed hierarchical fitness evaluation method, we use different fitness evaluation methods to replace the proposed one in the PSO process on CIFAR-10. The performance of the population in the last iteration is estimated again using the same assessment method, and the final results are presented in Table IV.

TABLE IV
THE COMPARISONS OF DIFFERENT EVALUATION METHODS.

Method	Average Acc.	Highest Acc.	GPU-Days
10% training data	71.62%	72.56%	0.3
40% training data	73.57%	74.68%	4
100% training data	73.92%	74.89%	14
EAEPSO	73.50%	74.44%	2

Specifically, four different fitness evaluation methods are employed: one using only 10% training data, which is the same as the basic fitness evaluation in EAEPSO; one using 40% training data, whose training data portion is the same as the final portion of the progressive fitness evaluation in EAEPSO; one using 100% training data. In comparison, the proposed EAEPSO method employs both the basic fitness evaluation method and progressive fitness evaluation method, which use a variety of training data portions: 10%, 20%, and 40%.

The method using only 10% training data consumes only 0.3 GPU-days, which is much smaller than the other three methods, along with the lowest average accuracy and highest accuracy. Only using a small number of training data can save computational resources, but the performance of the candidates is not satisfying. On the other hand, the method using all of the training data achieves both the best average accuracy and the best highest accuracy, but the computational cost is also much more expensive than others. EAEPSO's average accuracy is slightly lower than directly using 40% training data, and the highest accuracy is 0.24% lower. The reason is probably only one-third particles are evaluated by the progressive method in EAEPSO, which may harm the final performance. However, EAEPSO's 2 GPU-days is only half of the comparative method's 4 GPU-days. So we can say the proposed fitness evaluation method achieves a good balance between effectiveness and efficiency.

We also investigate the impact on the rank consistency between the evaluated fitness and the true performance when reducing the training dataset. We employ a standard PSO algorithm to evolve 30 particles for 10 iterations, and the whole training dataset in CIFAR-10 is used for fitness evaluation. Specifically, 80% training dataset is selected as the SGD training part to train the candidates, and the other 20% is used for evaluating the candidates' fitness.

After the evolution, we record the candidates' architectures at each iteration and employ 10%, 20%, and 40% of the training dataset to re-evaluate the networks for each generation. The rank relationships of all the candidates' performance with the reduced training datasets are compared with each other. We employ Spearman Coefficient as the metric to evaluate the rank consistency of different training data scales. The Spearman Coefficient between two ranking strings r_p and r_q is calculated by:

$$\rho_s(p,q) = 1 - \frac{\sum_{i=1}^{n}(r_{p,i} - r_{q,i})^2}{n \times (n^2 - 1)}, \tag{10}$$

where n is the number of the candidates in each iteration, and

$r_{p,i}$ and $r_{q,i}$ refer to the i-th candidate's rank over the 30 candidates under two different conditions, respectively.

We calculate the Spearman Coefficient using different training data scales for each iteration. Specifically, we compare the ranking results using 40% training data with 10% and 20% training data, respectively. Fig. 8 shows the comparison results. Along with the evolution, the Spearman Coefficients of both training scales (10% and 20%) show a downward trend. The reason is probably that the candidates learn from each other and gather in the neighborhood space, making their fitness tend to be similar, which inspires us to use more training data along with the evolution process. Another observation is that the Coefficient of 20% training data is almost always better than that of 10% of the training data at the same iterations, indicating that using more training data can bring more reliable fitness evaluation results.

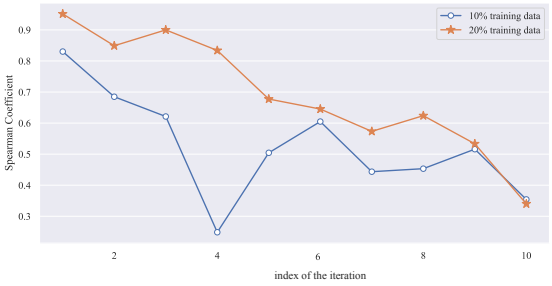

Fig. 8. The influence of different data scales.

One of the primary purposes of the fitness evaluation is to identify the best particle at each iteration to guide the downstream velocity and position update. If we can use a reduced training dataset to help identify each iteration's best particle, the searching time will be reduced. We investigate this by analyzing the performance of the best particle found by a bigger training dataset when trained by a smaller dataset. Fig. 9 shows that eight particles over ten top rank third when trained by a smaller dataset, indicating that the best particle found by a larger dataset usually also performs well when trained by a smaller dataset. This inspires us to use more minor training data to evaluate the candidates and then use more training data to further evaluate the well-performing candidates to identify the best particle precisely.

Fig. 9. The best candidate at each iteration is found by training on 40% training data, and their rank at each iteration when trained by 20% training data is shown.

VI. Conclusions

This work aimed to propose an effective and efficient ENAS method for image classification. The goal has been achieved by designing an autoencoder and developing an efficient hierarchical fitness evaluation method. In order to represent the blocks of different depths with fixed-length individuals to facilitate the search, we designed an autoencoder, which is able to encode the block vectors to fixed-length latent vectors. In addition, the proposed autoencoder can transfer the discrete search space to a continuous smooth latent space, and the designed loss function can make similar candidates concentrate together to facilitate the downstream search process. By investigating the reliability of using different training data scales, it is found that approximate ranking performance can also guide the search process, which inspired the proposed efficient hierarchical fitness evaluation method that could automatically change the scale of the training data considering the trade-off between effectiveness and efficiency. The proposed EAEPSO method was tested on three popular benchmark datasets — CIFAR-10, CIFAR-100, SVHN, and ImageNet, and was compared with 58 peer competitors. EAEPSO shows very good performance in terms of the searched network's performance and the computational cost. It achieves error rates of 2.74%, 16.17%, and 1.72% on CIFAR-10, CIFAR-100, and SVHN, outperforming most peer competitors. Besides, it only costs less than 3 GPU-days on CIFAR-10, much smaller than most peer competitors, costing hundreds or even thousands of GPU-days. To test the transferability, the block architecture searched by CIFAR-10 was also transferred to CIFAR-100 and ImageNet, which achieved competitive performance, indicating its good transferability. In addition, additional experiments prove the effectiveness of the proposed autoencoder and the proposed hierarchical fitness evaluation method.

In this paper, the original block architectures are represented by vectors, and each element represents a specific growth rate. However, some complicated network architectures are based on graphs; we could further investigate how to design an unsupervised learning method to convert the graph-based architecture representations to fixed-length vectors to be used by the standard EC methods. We predefined a series of candidate training data scales in the proposed hierarchical fitness evaluation method in this work, and we may further densign a new method to automatically determine the scale of the training data for the fitness evaluation to balance the estimated effectiveness and the training efficiency. In addition, we will investigate the use of EAEPSO in practical real-world problems, such as classifying fishes of different species using fish data from aquaculture. Furthermore, interpretability is essential for networks, and we plan to further explore evolving networks with high classification accuracy and interpretability.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[2] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[3] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7132–7141.

[4] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," *arXiv preprint arXiv:1806.09055*, 2018.

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

[6] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang, "Efficient architecture search by network transformation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[7] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, "Progressive neural architecture search," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 19–34.

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

[9] L. Xie and A. Yuille, "Genetic cnn," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 1379–1388.

[10] B. Evans, H. Al-Sahaf, B. Xue, and M. Zhang, "Evolutionary deep learning: A genetic programming approach to image classification," in *2018 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2018, pp. 1–6.

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

[12] Y. Liu, Y. Sun, B. Xue, M. Zhang, G. G. Yen, and K. C. Tan, "A survey on evolutionary neural architecture search," *IEEE Transactions on Neural Networks and Learning Systems*, Aug 06, 2021, doi: 10.1109/TNNLS.2021.3100554.

[13] Y. Bi, B. Xue, P. Mesejo, S. Cagnoni, and M. Zhang, "A survey on evolutionary computation for computer vision and image analysis: Past, present, and future trends," *IEEE Transactions on Evolutionary Computation*, pp. 1–1, 2022.

[14] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95-international conference on neural networks*, vol. 4. IEEE, 1995, pp. 1942–1948.

[15] Y. Shi, "Particle swarm optimization," *IEEE connections*, vol. 2, no. 1, pp. 8–13, 2004.

[16] B. Wang, Y. Sun, B. Xue, and M. Zhang, "Evolving deep convolutional neural networks by variable-length particle swarm optimization for image classification," in *2018 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2018, pp. 1–8.

[17] B. Wang, B. Xue, and M. Zhang, "Surrogate-assisted particle swarm optimization for evolving variable-length transferable blocks for image classification," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[18] G. E. Hinton and R. S. Zemel, "Autoencoders, minimum description length, and helmholtz free energy," *Advances in neural information processing systems*, vol. 6, pp. 3–10, 1994.

[19] J. You, J. Leskovec, K. He, and S. Xie, "Graph structure of neural networks," in *International Conference on Machine Learning*. PMLR, 2020, pp. 10 881–10 891.

[20] C. Wei, Y. Tang, C. N. C. Niu, H. Hu, Y. Wang, and J. Liang, "Self-supervised representation learning for evolutionary neural architecture search," *IEEE Computational Intelligence Magazine*, vol. 16, no. 3, pp. 33–49, 2021.

[21] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proceedings of the aaai conference on artificial intelligence*, vol. 33, 2019, pp. 4780–4789.

[22] A. A. Ahmed, S. M. S. Darwish, and M. M. El-Sherbiny, "A novel automatic cnn architecture design approach based on genetic algorithm," in *International Conference on Advanced Intelligent Systems and Informatics*. Springer, 2019, pp. 473–482.

[23] H. Shu and Y. Wang, "Automatically searching for u-net image translator architecture," *arXiv preprint arXiv:2002.11581*, 2020.

[24] D. Sapra and A. D. Pimentel, "Constrained evolutionary piecemeal training to design convolutional neural networks," in *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*. Springer, 2020, pp. 709–721.

[25] B. Wang, B. Xue, and M. Zhang, "Particle swarm optimisation for evolving deep neural networks for image classification by evolving and stacking transferable blocks," in *2020 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2020, pp. 1–8.

[26] J. Liu, M. Gong, Q. Miao, X. Wang, and H. Li, "Structure learning for deep neural networks based on multiobjective optimization," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 6, pp. 2450–2463, 2017.

[27] D. Zhou, X. Zhou, W. Zhang, C. C. Loy, S. Yi, X. Zhang, and W. Ouyang, "Econas: Finding proxies for economical neural architecture search," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 11 396–11 404.

[28] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Evolving deep convolutional neural networks for image classification," *IEEE Transactions on Evolutionary Computation*, vol. 24, no. 2, pp. 394–407, 2019.

[29] B. Wang, Y. Sun, B. Xue, and M. Zhang, "A hybrid differential evolution approach to designing deep convolutional neural networks for image classification," in *Australasian Joint Conference on Artificial Intelligence*. Springer, 2018, pp. 237–250.

[30] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "A particle swarm optimization-based flexible convolutional autoencoder for image classification," *IEEE transactions on neural networks and learning systems*, vol. 30, no. 8, pp. 2295–2309, 2018.

[31] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.

[32] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter, "Nas-bench-101: Towards reproducible neural architecture search," in *International Conference on Machine Learning*. PMLR, 2019, pp. 7105–7114.

[33] G. Yuan, B. Xue, and M. Zhang, "A graph-based approach to automatic convolutional neural network construction for image classification," in *2020 35th International Conference on Image and Vision Computing New Zealand (IVCNZ)*. IEEE, 2020, pp. 1–6.

[34] C. Wei, C. Niu, Y. Tang, Y. Wang, H. Hu, and J. Liang, "Npenas: Neural predictor guided evolution for neural architecture search," *arXiv preprint arXiv:2003.12857*, 2020.

[35] S. Yan, Y. Zheng, W. Ao, X. Zeng, and M. Zhang, "Does unsupervised architecture representation learning help neural architecture search?" *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[36] H.-P. Cheng, T. Zhang, S. Li, F. Yan, M. Li, V. Chandra, H. Li, and Y. Chen, "Nasgem: Neural architecture search via graph embedding method," *arXiv preprint arXiv:2007.04452*, 2020.

[37] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin, "Large-scale evolution of image classifiers," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 2017, pp. 2902–2911.

[38] Z. Fan, J. Wei, G. Zhu, J. Mo, and W. Li, "Evolutionary neural architecture search for retinal vessel segmentation," *arXiv preprint arXiv:2001.06678*, 2020.

[39] X. Chu, B. Zhang, and R. Xu, "Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 12 239–12 248.

[40] X. Chen, D. P. Kingma, T. Salimans, Y. Duan, P. Dhariwal, J. Schulman, I. Sutskever, and P. Abbeel, "Variational lossy autoencoder," *arXiv preprint arXiv:1611.02731*, 2016.

[41] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[42] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011.

[43] T. DeVries and G. W. Taylor, "Improved regularization of convolutional neural networks with cutout," *arXiv preprint arXiv:1708.04552*, 2017.

[44] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8697–8710.

[45] G. Larsson, M. Maire, and G. Shakhnarovich, "Fractalnet: Ultra-deep neural networks without residuals," *arXiv preprint arXiv:1605.07648*, 2016.

[46] I. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio, "Maxout networks," in *International conference on machine learning*. PMLR, 2013, pp. 1319–1327.

[47] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Highway networks," *arXiv preprint arXiv:1505.00387*, 2015.

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

[49] S. Zagoruyko and N. Komodakis, "Wide residual networks," *arXiv preprint arXiv:1605.07146*, 2016.

[50] M. Suganuma, S. Shirakawa, and T. Nagao, "A genetic programming approach to designing convolutional neural network architectures," in *Proceedings of the genetic and evolutionary computation conference*, 2017, pp. 497–504.

[51] Z. Zhong, J. Yan, W. Wu, J. Shao, and C.-L. Liu, "Practical block-wise neural network architecture generation," in *Proceedings of the IEEE*

conference on computer vision and pattern recognition, 2018, pp. 2423–2432.
[52] J. Ren, Z. Li, J. Yang, N. Xu, T. Yang, and D. J. Foran, "Eigen: Ecologically-inspired genetic approach for neural network structure searching from scratch," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 9059–9068.
[53] Y. Sun, B. Xue, M. Zhang, G. G. Yen, and J. Lv, "Automatically designing cnn architectures using the genetic algorithm for image classification," *IEEE Transactions on Cybernetics*, 2020.
[54] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8697–8710.
[55] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Completely automated cnn architecture design based on blocks," *IEEE transactions on neural networks and learning systems*, vol. 31, no. 4, pp. 1242–1254, 2019.
[56] F. Assunção, N. Lourenço, P. Machado, and B. Ribeiro, "Evolving the topology of large scale deep neural networks," in *European Conference on Genetic Programming*. Springer, 2018, pp. 19–34.
[57] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy *et al.*, "Evolving deep neural networks," in *Artificial intelligence in the age of neural networks and brain computing*. Elsevier, 2019, pp. 293–312.
[58] Z. Lu, I. Whalen, V. Boddeti, Y. Dhebar, K. Deb, E. Goodman, and W. Banzhaf, "Nsga-net: neural architecture search using multi-objective genetic algorithm," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2019, pp. 419–427.
[59] T. Elsken, J. H. Metzen, and F. Hutter, "Efficient multi-objective neural architecture search via lamarckian evolution," *arXiv preprint arXiv:1804.09081*, 2018.
[60] Z. Lu, I. Whalen, Y. Dhebar, K. Deb, E. Goodman, W. Banzhaf, and V. N. Boddeti, "Multi-objective evolutionary design of deep convolutional neural networks for image classification," *IEEE Transactions on Evolutionary Computation*, 2020.
[61] H. Cai, L. Zhu, and S. Han, "Proxylessnas: Direct neural architecture search on target task and hardware," in *International Conference on Learning Representations*, 2018.
[62] 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 Transactions on Evolutionary Computation*, vol. 24, no. 2, pp. 350–364, 2019.
[63] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.
[64] K. DP and J. Ba, "Adam: A method for stochastic optimization," in *Proc. of the 3rd International Conference for Learning Representations (ICLR)*, 2015.
[65] F. Van den Bergh and A. P. Engelbrecht, "A study of particle swarm optimization particle trajectories," *Information sciences*, vol. 176, no. 8, pp. 937–971, 2006.
[66] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *International Conference on Machine Learning*. PMLR, 2018, pp. 4095–4104.

Gonglin Yuan (Graduate Student Member, IEEE) received the B.E. degree from Northwestern Polytechnical University, Xi'an, China, in 2016, and the M.S. degree from Nanjing Research Institute of Electronics Technology, Nanjing, China, in 2019. He is currently pursuing the Ph.D. degree in computer science at Victoria University of Wellington, Wellington, New Zealand.

His current research interests include evolutionary computation, computer vision, and neural architecture search.

Bin Wang (Student Member of IEEE) received B.E. degree from the Heilongjiang University of Science and Technology, Haerbin, Heilongjiang, China, in 2008, and BSc(Hons) degree from the Victoria University of Wellington, Wellington, New Zealand, in 2019. He's currently pursuing the Ph.D. degree in Computer Science with the School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand.

Bin Wang has been severing as a Reviewer for top international journals, such as IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS, IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, IEEE-TRANSACTIONS ON CYBERNETICS, IEEE COMPUTATIONAL INTELLIGENCE MAGAZINE.

Bing Xue (M'10) received the B.Sc. degree from the Henan University of Economics and Law, Zhengzhou, China, in 2007, the M.Sc. degree in management from Shenzhen University, Shenzhen, China, in 2010, and the PhD degree in computer science in 2014 at Victoria University of Wellington (VUW), New Zealand. She is currently an Professor in Computer Science, and Program Director of Science in School of Engineering and Computer Science at VUW. She has over 300 papers published in fully refereed international journals and conferences and her research focuses mainly on evolutionary computation, machine learning, classification, symbolic regression, feature selection, evolving deep neural networks, image analysis, transfer learning, multi-objective machine learning.

Dr Xue is currently a Vice-Chair of the IEEE Computational Intelligence Society (CIS) Evolutionary Computation Technical Committee, was the Chair of Data Mining and Big Data Analytics Technical Committee. She is also a Vice-Chair of IEEE Task Force on Evolutionary Feature Selection and Construction, Vice-Chair of IEEE CIS Task Force on Transfer Learning & Transfer Optimization, and of IEEE CIS Task Force on Evolutionary Deep Learning and Applications. She is also served as associate editor of several international journals, such as IEEE Computational Intelligence Magazine and IEEE Transactions on Evolutionary Computation.

Mengjie Zhang (M'04-SM'10-F'19) received the B.E. and M.E. degrees from Artificial Intelligence Research Center, Agricultural University of Hebei, Hebei, China, and the Ph.D. degree in computer science from RMIT University, Melbourne, VIC, Australia, in 1989, 1992, and 2000, respectively. He is currently Professor of Computer Science, Head of the Evolutionary Computation Research Group, and the Associate Dean (Research and Innovation) in the Faculty of Engineering. His current research interests include evolutionary computation, particularly genetic programming, particle swarm optimization, and learning classifier systems with application areas of image analysis, multi-objective optimization, feature selection and reduction, job shop scheduling, and transfer learning. He has published over 800 research papers in refereed international journals and conferences. Prof. Zhang is a Fellow of Royal Society of New Zealand and has been a Panel Member of the Marsden Fund (New Zealand Government Funding), a Fellow of IEEE, and a member of ACM. He was the chair of the IEEE CIS Intelligent Systems and Applications Technical Committee, and chair for the IEEE CIS Emergent Technologies Technical Committee and the Evolutionary Computation Technical Committee, and a member of the IEEE CIS Award Committee. He is a vice-chair of the Task Force on Evolutionary Computer Vision and Image Processing, and the founding chair of the IEEE Computational Intelligence Chapter in New Zealand. He is also a committee member of the IEEE NZ Central Section.