Evolving Deep Convolutional Variational Autoencoders for Image Classification

Xiangru Chen, Yanan Sun, *Member, IEEE*, Mengjie Zhang, *Fellow, IEEE*, and Dezhong Peng, *Member, IEEE*

***Abstract*—Variational autoencoders (VAEs) have demonstrated their superiority in unsupervised learning for image processing in recent years. The performance of the VAEs highly depends on their architectures, which are often handcrafted by the human expertise in deep neural networks (DNNs). However, such expertise is not necessarily available to each of the end users interested. In this article, we propose a novel method to automatically design optimal architectures of VAEs for image classification, called evolving deep convolutional VAE (EvoVAE), based on a genetic algorithm (GA). In the proposed EvoVAE algorithm, the traditional VAEs are first generalized to a more generic and asymmetrical one with four different blocks, and then a variable-length gene encoding mechanism of the GA is presented to search for the optimal network depth. Furthermore, an effective genetic operator is designed to adapt to the proposed variable-length gene encoding strategy. To verify the performance of the proposed algorithm, nine variants of AEs and VAEs are chosen as the peer competitors to perform the comparisons on MNIST, street view house numbers, and CIFAR-10 benchmark datasets. The experiments reveal the superiority of the proposed EvoVAE algorithm, which wins 21 times out of the 24 comparisons and outperforms the best competitors by 1.39%, 14.21%, and 13.03% on the three benchmark datasets, respectively.**

***Index Terms*—Convolutional variational autoencoder, evolving deep learning, genetic algorithm (GA), neural architecture search (NAS).**

Manuscript received May 11, 2020; revised August 12, 2020 and November 3, 2020; accepted December 22, 2020. Date of publication December 24, 2020; date of current version October 1, 2021. This work was supported in part by the National Natural Science Foundation of China under Grant 61971296, Grant U19A2078, and Grant 61836011; in part by Sichuan Science and Technology Planning Project under Grant 2020YFG0319 and Grant 2020YFH0186; in part by the Fundamental Research Funds for the Central Universities under Grant YJ201934; and in part by Chengdu Key Research and Development Support Plan under Grant 2019-YF08-00264-GX. *(Corresponding author: Yanan Sun.)*

Xiangru Chen is with the College of Computer Science, Sichuan University, Chengdu 610065, China (e-mail: cxrasdfg@gmail.com).

Yanan Sun is with the College of Computer Science, Sichuan University, Chengdu 610065, China, and also with Sichuan Eface Technology Company Ltd., Chengdu 610093, China (e-mail: ysun@scu.edu.cn).

Mengjie Zhang is with the School of Engineering and Computer Science, Victoria University of Wellington, Wellington 6140, New Zealand (e-mail: mengjie.zhang@ecs.vuw.ac.nz).

Dezhong Peng is with the Machine Intelligence Laboratory, College of Computer Science, Sichuan University, Chengdu 610065, China, also with the Shenzhen Peng Cheng Laboratory, Shenzhen 518052, China, and also with Chengdu Sobey Digital Technology Company Ltd., Chengdu 610041, China (e-mail: pengdz@scu.edu.cn).

This article has supplementary material provided by the authors and color versions of one or more figures available at https://doi.org/10.1109/TEVC.2020.3047220.

Digital Object Identifier 10.1109/TEVC.2020.3047220

I. Introduction

In RECENT years, there has been significant progress in computer vision, largely owing to the promising performance of the deep neural networks (DNNs) [1]. The DNNs are designed to learn the meaningful representations with higher level features formed by the nonlinear combination of lower level features. Such deep architectures can significantly outperform the competitors with the shallow architectures and other traditional methods, and have been the state-of-the-art models for image processing tasks.

The DNNs are often used as supervised learning models and heavily rely on a large number of labeled samples for the training. Compared to the unlabeled samples that exist more widely in practice and are also often easy to obtain, it is obvious that such a rich supervision signal for DNNs may be hard to complete. This motivates the adoption of unsupervised leaning [2] with the DNNs, which could leverage a large number of unlabeled samples to make the network fully trained with the better representations. Recently, researchers have found that unsupervised pretraining does help supervised training of the DNNs [3]. Autoencoders (AEs) [4] are such fundamental unsupervised methods, which employ DNNs to transform the raw input data to the meaningful representations and boost the training of the subsequent machine learning tasks. Different from the classical AEs aiming at learning a distance function, the variational AE (VAE) proposed by Kingma and Welling [5] is a powerful generative AE, which could provide high-quality representations for a large number of unlabeled samples and generate virtual instances that we desire in the controllable smooth latent space [6].

The VAEs have strong capability of approximating the intractable posterior density in a more general manner, and can effectively perform the inference compared with the classical AEs [5]. These merits collectively make the VAEs widely used and successful for multiple different machine learning tasks, such as natural image processing [7], text caption [8], and audio data recognition [9]. Furthermore, because of the variational lower bound in the optimization of VAEs, the latent representations can be controlled more precisely, and the precise modeling could provide better representations compared with other AE variants. This could improve the performances of the certain downstream tasks, such as image classification [10]. Additionally, because the VAEs learn the parameters of the specific probabilistic distribution, the generation of new instances can be easily controlled by varying the distribution parameters. For example, Kulkarni *et al.* [11]

leveraged a convolutional VAE to learn an interpretable representation of images, and the model can generate new images of the same object with variations in pose and lighting, which means that we can directly employ a VAE to render original static image with different pose directions. Besides, compared with another generative paradigm, such as generative adversarial networks (GANs) [12], VAEs are more stable during the training phases owing to its closed-formed objective function [13], and have the potential to produce crisp samples that are comparable to GANs [14]. Another important advantage is that compared with the variational inference, VAEs are typically more efficient for large datasets [6], because VAEs are still reasonably fast using a single feedforward neural networks to be a stochastic function of the input variables, while the variational inference usually encounters a huge computational complexity as the number of the samples increases.

Although VAEs have shown the promising performance in the diverse applications, the current works based on VAEs have two important limitations. First, the architectures of VAEs in the above works are all handcrafted by the human experts, which means that designing the architectures of DNNs requires rich domain-specific knowledge. The barrier between the researchers and the effective architectures also prevents the end users without network design expertise from utilizing VAEs. Even for those researchers with design expertise, finding an effective architecture tuned manually is also arduous since the multiple trial-and-error rounds are needed. These challenges come from the characteristics of the architectures of the VAEs themselves. One of the challenges is the optimal depth of the VAE is unknown to the researchers, therefore it is hard to choose the appropriate number of the convolutional layers, pooling layers, and dense layers. Another challenge lies on the hyperparameters of each type of layers, e.g., the kernel sizes of the convolutional and deconvolutional layers. It is worth noting that the performance of a VAE is highly affected by the dimensionality of the underlying manifold.

Second, most VAEs are originally designed with the symmetrical architecture, which requires the encoder and the decoder to be with the same number of the layers. However, for the image classification problems, the decoder portion is deprecated after the unsupervised pretraining phase has been completed and will never be used when fine tuned. Thus, retaining a symmetrical architecture is almost completely unnecessary [15]. Unfortunately, new problems arise when the asymmetrical manner is employed, although the adoption of the asymmetry is a very attractive decision. This means that if we want to optimize the architectures of the asymmetrical convolutional VAEs, we have to separately consider the subarchitectures of the different partitions in VAEs. Besides, guaranteeing the decoder to correctly fit the resolution of the raw input data is essential to the correct forwarding of the asymmetrical convolutional VAEs. This also increases the burden of the manual design for the architectures of the VAEs.

In this article, we propose the evolving deep convolutional VAE (EvoVAE) algorithm to effectively address the above two limitations of the VAEs. Particularly, the first limitation, i.e., the architecture design heavily relying on human expertise, is formalized as a neural architecture search (NAS) problem, and then we solve it by designing an algorithm based on genetic algorithms (GAs) [16]. The second limitation is addressed by disentangling the symmetrical constrain, i.e., each block in the VAEs could freely choose the number of the layers, whose optimal configurations are finalized by solving the first limitation. To the best of our knowledge, this is the first work to achieve the automatic architecture design of the VAEs. The contributions of this articles are highlighted as follows.

1) Design a flexible architecture of the VAEs that can address the limitation to the symmetrical architectures of the traditional VAEs. Particularly, four blocks have been proposed for constructing the architectures of the VAEs, namely, h-block, μ-block, σ-block, and t-block. Each block is flexible and any of the architectures can be different from the others. To this end, the proposed architectures are not necessarily limited to be symmetrical, and the optimized architectures are often asymmetrical.
2) Develop a new variable-length gene encoding strategy that can represent optimal architectures of the VAEs with arbitrary depths. Most of the existing work often employs the fixed-length encoding strategy to find the "best" architectures, whose depth cannot be changed during the optimization. However, the depth is a key hyperparameter affecting the performance of the VAEs, and the best depth is unknown before the optimal architectures have been found. If the adopted length is not proper in the fixed-length encoding, the optimal architecture cannot be found. The proposed encoding strategy has the potential to search the optimal depth of the network automatically.
3) Develop associated crossover and mutation operations to cope with the proposed variable-length encoding strategy. Although the proposed variable-length encoding strategy has the potential to find the optimal architectures of VAEs automatically, the original genetic operators work only for the fixed-length encoding. To this end, we have developed effective crossover and mutation operators to enhance the ability to search for the optimal depth of the VAEs.
4) Demonstrate the superiority of the proposed algorithm compared with the peer competitors by conducting the essential experiments. In order to evaluate the performance of the proposed EvoVAE in this article, we have chosen nine peer competitors, including AE, convolutional AE (CAE), denoising AE (DAE), and SAE, and compared them on the MNIST, street view house numbers (SVHN), and CIFAR-10 datasets. The experimental results demonstrate that the proposed EvoVAE can automatically find the architectures of the VAEs and achieve better performances.

II. Background and Motivation

A. Variational Autoencoder

As have discussed in Section I, VAE is a type of generative model. Particularly, for a given dataset $\boldsymbol{X} = \{\boldsymbol{x}_i\}_{i=1}^{N}$ consisting of N independent identically distributed samples of

continuous variable $\boldsymbol{x}$, the generative model is designed to estimate the distributions $P(\boldsymbol{X})$, where the real samples should get a high probability while the random noise should get a low probability [17]. In a more general case, we assume that the generation of the real data follows a random two-step process: 1) a continuous latent variable $\boldsymbol{z}$ is generated from some prior distributions $p(\boldsymbol{z};\boldsymbol{\theta})$ and 2) the observed data $\boldsymbol{x}$ are generated by the conditional distribution $p(\boldsymbol{x}|\boldsymbol{z};\boldsymbol{\theta})$. The true parameters $\boldsymbol{\theta}$ and the latent variable $\boldsymbol{z}$ are invisible to us, and the posterior $p(\boldsymbol{z}|\boldsymbol{x})$ is intractable in the general case. The VAEs can perform efficient posterior inference on the data whose posterior is intractable by maximizing a variational lower bound $\mathcal{L}(\boldsymbol{\theta},\boldsymbol{\phi};\boldsymbol{x}_i)$, which can be derived by

$$\begin{aligned}\mathcal{L}(\boldsymbol{\theta},\boldsymbol{\phi};\boldsymbol{x}_i) = &-\mathrm{KL}(q(\boldsymbol{z}|\boldsymbol{x}_i;\boldsymbol{\phi})\|p(\boldsymbol{z};\boldsymbol{\theta})) \\ &+ \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x}_i;\boldsymbol{\phi})}\big[\log p(\boldsymbol{x}_i|\boldsymbol{z};\boldsymbol{\theta})\big] \end{aligned} \tag{1}$$

where the parameters $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$ are the learned weights of the VAE.

The prior distribution $p(\boldsymbol{z};\boldsymbol{\theta})$, conventionally, is a multivariate Gaussian distribution [5], and we also assume the true posterior $p(\boldsymbol{z}|\boldsymbol{x}_i;\boldsymbol{\theta})$ is an approximate Gaussian with approximately diagonal covariance. In this case, $\mathcal{L}(\boldsymbol{\theta},\boldsymbol{\phi};\boldsymbol{x}_i)$ can be the form formulated by

$$\begin{aligned}\mathcal{L}(\boldsymbol{\theta},\boldsymbol{\phi};\boldsymbol{x}_i) \simeq &\frac{1}{2}\sum_{j=1}^{J}\Big(1+\log\sigma_{i,j}^2-\mu_{i,j}^2-\sigma_{i,j}^2\Big) \\ &+ \frac{1}{L}\sum_{l=1}^{L}\log p(\boldsymbol{x}_i|\boldsymbol{z}_l;\boldsymbol{\theta}) \end{aligned} \tag{2}$$

where the distribution parameters mean $\boldsymbol{\mu}_i$ and variance $\boldsymbol{\sigma}_i^2$ are the outputs of the encoder of the VAEs. In addition, the symbol J refers to the dimensionality of the underlying manifold, and L denotes the sample size of the Monte Carlo method sampling from the approximate posterior distributions.

B. Brief Review of Neural Architecture Search

The NAS [18] aims to automatically find the optimal neural network architecture by exploring the search space such that the automatically optimized architecture could achieve a superior predictive performance on the unseen data for a specific task. As a result, the automation of the NAS helps to make the deep learning methods more accessible to the end users, especially for those practitioners with the poor domain-specific knowledge or DNN training knowledge. The research of NAS can be tracked back to 1990s of the last century [19], and the early methods usually evolved the topologies of the network architectures along with their weights and hyperparameters simultaneously [20]. With the renaissance of the artificial intelligence and deep learning, there have been an insurgence in research efforts by the NAS community that seeks to automate the search process of network architectures within the last few years [18]. Generally, existing NAS algorithms can be divided into four different types based on their base optimizers: 1) the Bayesian-based [21] algorithms; 2) the gradient-based [22] algorithms; 3) the reinforcement learning (RL)-based [23] algorithms; and 4) the evolutionary algorithms (EA)-based [24] algorithms.

Specifically, the Bayesian-based algorithms assume the performances of the architectures are determined by the hyperparameters, and often model the relationship with the Gaussian process [25]. For example, Kandasamy *et al.* [26] proposed the NASBOT method, which is a Gaussian process-based Bayesian optimization framework for NAS. However, specifying the distance function between network architectures is a cumbersome task for the Bayesian-based algorithms, and it could also be quite challenging to achieve highly prediction results with the Gaussian process [27].

The gradient-based algorithms make the whole architecture optimization be fully differentiable. DARTS [22] is generally viewed as a typical work to leverage the gradient-based method to evolve the network architecture, where a continuous relaxation of the architecture representation is employed. Unfortunately, the gradient-based algorithms often consume excessive GPU memory to update the numerous parameters, and would cause competition between different operations when jointly optimizing them [28].

The RL-based algorithms consider the process of the architecture designing as the actions. Baker *et al.* [29] proposed the MetaQNN method using the RL algorithms, where the states represented the premature architectures, and the architectures were updated by employing the ϵ-greedy strategy [30]. An alternative approach based on the policy gradient method was proposed in [23]. In this approach, they used the controller formed by a recurrent neural network (RNN) to predict the actions, which are used to construct a network based on the previous actions. Cai *et al.* [31] employed an RL agent as the metacontroller, whose action is to grow the network depth or layer width with a function-preserving transformations. However, the RL-based NAS relies on hyperparameters to guarantee the stability, which would make the controllers take tens of actions to get a positive reward and is generally ineffective [32].

Compared with the RL-based algorithms, the EA-based algorithms are developed to discover the novel architectures by the production of the new individuals [24], [33]. The experimental evidence shows that the RL-based algorithms typically require more extensive computational resources than the EA-based ones [34]. In addition to ordinary EAs, the GAs and the particle swarm optimization (PSO) [35] are the other two popular algorithms. For the PSO-based algorithms, the network architectures are encoded by the particles, and the velocity and position of each particle are updated toward the optimal network architecture in each iteration, such as SFCAE [36], SOBA [37], and BQPSO [38]. Recently, many GA-based algorithms for the NAS have emerged, due to the property that the GAs can handle problems that are difficult to directly solve. Sun *et al.* [39] proposed a new method using the GA to evolve the architectures and the connection weight initialization values of the DNNs to solve the image classification problems. Assuncao *et al.* [40] leveraged the GA to generate the architecture of the AEs, which intended to get the compressed representations on a particular dataset. Lu *et al.* proposed NSGA-NET [41] and NSGANetV1 [42] to approximate the entire Pareto frontier between the computational budget and the predictive performance, where NSGANetV1 extended the

earlier proof-of-principle NSGA-NET algorithm to take into account the limitations of existing algorithms in terms of the multiobjective optimization and the vast computational resources.

However, existing works related to the automatically architecture design for the AEs are mainly focused on the ordinary AEs [40], [43], where their strategies cannot be directly applied to the VAE. This is because VAEs are more complicated than ordinary AEs. For example, a VAE has two independent branches in the encoder to generate the parameters $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$. While in an ordinary AE, the encoder directly generates a latent variable $\boldsymbol{z}$. Hence, it is necessary to redesign some components of the classical GA to cope with the NAS problems for encoding VAEs.

C. Motivation of Asymmetrical Convolutional VAE

Keeping a symmetrical architecture of AE may not be necessary due to the deprecation of decoder after the unsupervised pretraining phase. On the contrary, following an asymmetrical manner could enable each partition in the asymmetrical convolutional VAE to freely choose their own optimal subarchitectures. Therefore, in the proposed EvoVAE algorithm, the encoder (including the mean branch and variance branch) and decoder of the VAE could have different numbers of layers and neurons.

Many researches related to DNNs have shown the essential of sharing weights on the DNNs [44]. Compared with the separated networks, sharing weighs of the networks has two advantages: 1) reduce the number of parameters and decrease the computational complexity and 2) enhance network robustness and generalization capabilities by detecting effective features. Thus, it is necessary to embed the weight sharing mechanism to the new method, before the computation of $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$.

Thus, the concern has been naturally raised that the architecture of VAE should be revised accordingly, which motivates the designing of the asymmetrical convolutional VAE where the nonidentical numbers of the convolutional layers, pooling layers, and deconvolutional layers are allowed.

III. Proposed Algorithm

In this section, the details of the proposed EvoVAE algorithm are documented. Specifically, the overview of the EvoVAE is provided in Section III-A, and then the main components of the EvoVAE are introduced in Sections III-B–III-F.

A. Algorithm Overall

Algorithm 1 shows the framework of the proposed EvoVAE algorithm. First, the population P_0 is initialized with the proposed gene encoding strategy (line 1). In order to obtain the fitness value of each individual, the individuals in the population P_0 are evaluated by the proposed fitness evaluation method (line 2). Then, the algorithm will start the evolutionary process from generation to generation to expectedly create individuals with better performance until the number of generations reaches the predefined maximum value T (lines 3–8). Finally, the individual with the best performance

Algorithm 1: Framework of EvoVAE

Input: The number of generations T
Output: The best individual p_{best}

1 $P_0 \leftarrow$ Initialize the population with the ***proposed gene encoding strategy***;
2 Evaluate the fitness of individuals in P_0 by using ***the proposed fitness evaluation approach***;
3 **for** $t = 1$ **to** T **do**
4 $\quad C_t \leftarrow$ Select the parent solutions with the binary tounamental selection;
5 $\quad O_t \leftarrow$ Generate the offsprings with the ***designed genetic operators*** from C_t;
6 $\quad$ Evaluate the fitness of individuals in O_t;
7 $\quad P_t \leftarrow$ Environmental selection from $P_{t-1} \cup O_t$ by using ***the proposed strategy***;
8 **end**
9 $p_{best} \leftarrow$ Select the best individual from P_T and decode it to the corresponding VAE for the final deep training;
10 **return** p_{best};

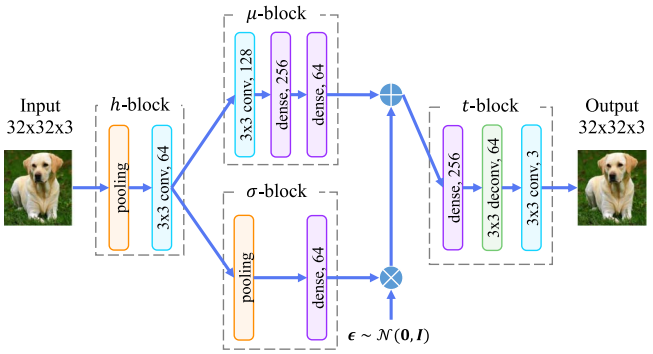


Fig. 1. Skeleton of asymmetrical convolutional VAE.

is selected for the final deep training (line 9). This is because that the training of individuals during the evolutionary process is often not sufficient due to a small number of epochs for acceleration [36].

The evolutionary process contains five steps (lines 3–8). The binary tournament selection [45] is employed to construct the mating pool from P_{t-1} (line 4). Then, a population of offspring is generated by the proposed genetic operators (line 5), and their fitness is evaluated (line 6). Finally, the current population is combined with the offspring for the environmental selection, and the selected individuals are used as the parent solutions for the next generation (line 7).

In the following, we will detail the gene encoding strategy, population initialization, fitness evaluation, mating pool selection, genetic operators, and environment selection, which are the key components of the proposed algorithm.

B. Gene Encoding Strategy

The first step of using GA is to represent the potential solutions of the problem to be solved by chromosomes via the proper encoding strategy. In the proposed algorithm, each chromosome represents a potential architecture of the VAEs.

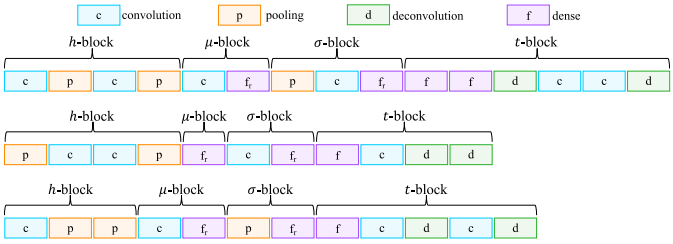


Fig. 2. Example of three chromosomes with different lengths in the proposed EvoVAE algorithm.

In order to more simply describe how to represent the architectures of asymmetrical VAEs, we divided the entire VAEs into four blocks. First, we denote the mean branch as μ-block and the variance branch as σ-block, which compute the parameter mean and variance of the distribution $p(\boldsymbol{z}|\boldsymbol{x})$, respectively. To perform the common feature extraction which is discussed in the motivation, h-block is added before the μ-block and the σ-block. In addition, we use t-block to represent the decoder, which outputs the reconstructed data. The h-block is a sequential network constructed by the convolutional layers and the pooling layers; the μ-block and the σ-block are the sequential networks constructed by the dense layers, convolutional layers, and the pooling layers, whose positions and settings are flexible. The decoder, t-block, is also a sequential network with the deconvolutional layers, instead of pooling layers, to recover the resolution of the features.

Fig. 1 illustrates the skeleton of such an asymmetrical convolutional VAE. Each block is contoured by the gray dotted boxes. The VAE receives the 32×32 RGB image as input, and finally output the reconstructed image. Particularly, when the image is fed into the h-block, it is converted to a $64 \times 16 \times 16$ tensor. Next, it is converted to a 64-D vector after forwarding the μ-block and σ-block. Note that the 3-D tensors should be flattened to a vector when they are fed into a dense layer. Finally, in the t-block, the 64-D vector is first converted to a 256-D vector, then is unflattened back to an $1 \times 16 \times 16$ tensor to continue the deconvolutional operations. The last convolutional layer outputs a $3 \times 32 \times 32$ tensor. Obviously, the numbers of layers in each block are not identical. With such an asymmetrical manner, we have the potential to obtain the optimal subarchitectures for different blocks of the VAEs.

Since the optimal depth of the network in dealing with such a particular problem is unknown in advance, a variable-length gene encoding strategy is necessary in this article for the evolution. Specifically, the variable length is realized by the different numbers and types of the layers, and the length of a chromosome could be changed during the optimization process, to adapt to the optimal depth of the VAEs. Furthermore, following the asymmetrical manner, we do not constrain each block of the VAEs for keeping a symmetrical architecture, such as keeping the number of the layers and the sizes of the output features the same for all the blocks. In order to give a better understanding of the proposed variable-length gene encoding strategy and the asymmetrical manner, Fig. 2 shows three chromosomes with different lengths. The figure clearly shows how the variable length and asymmetry are reflected in the proposed gene encoding strategy. Compared with the proposed variable-length encoding method, the fixed-length one cannot handle this problem because it assumes the depth of the whole network is a constant. In principle, the depth is a key hyperparameter in the network architectures. This is because the performance of a DNN is largely affected by the depth [46], [47]. When the global optimum is far away from the predefined network depth, such a fixed-length strategy would deteriorate the performance for evolving the architecture.

Algorithm 2: Encoding Strategy

Input: the VAE model m
Output: the chromosome p

h-block $\leftarrow \emptyset$, μ-block $\leftarrow \emptyset$, σ-block $\leftarrow \emptyset$, t-block $\leftarrow \emptyset$;
From m extract h-block, μ-block, σ-block and t-block as c_h, c_μ, c_σ and c_t, respectively;
for $l \in c_h$ **do**
  Create a unit u;
  if l *is the dense layer* **then**
    Encode the dense layer l in unit u;
  else
    if l *is the convolutional layer* **then**
      Encode the convolutional layer l in unit u;
    else
      if l *is the pooling layer* **then**
        Encode the pooling layer l in unit u;
      else
        Encode the deconvolutional layer l in unit u;
      end
    end
  end
  h-block $\leftarrow$ h-block $\cup$ u;
end
Update the μ-block, the σ-block and the t-block with the steps in analogy to the h-block;
$p \leftarrow h$-block $\cup$ μ-block $\cup$ σ-block $\cup$ t-block;
return p;

TABLE I
ENCODED INFORMATION IN GENETIC UNIT

Unit Type	Encoded Information
Dense	the number of features
Convolution	the number of feature maps, the kernel width, the kernel height
Pooling	the pooling type (max pooling or average pooling)
Deconvolution	the number of feature maps

A chromosome stores the complete information of constructing a VAE, and the genetic information of network layer is stored in the units of the chromosome. A genetic unit can be decoded as a certain layer in the network, and a certain layer in the network can also be encoded as a certain genetic unit in the chromosome. In the EvoVAE, there are four types of units: 1) the dense; 2) the convolutional; 3) the deconvolutional; and 4) the pooling units. Table I exhibits the details of the types of the units in EvoVAE, and the corresponding encoded information in the proposed encoding strategy.

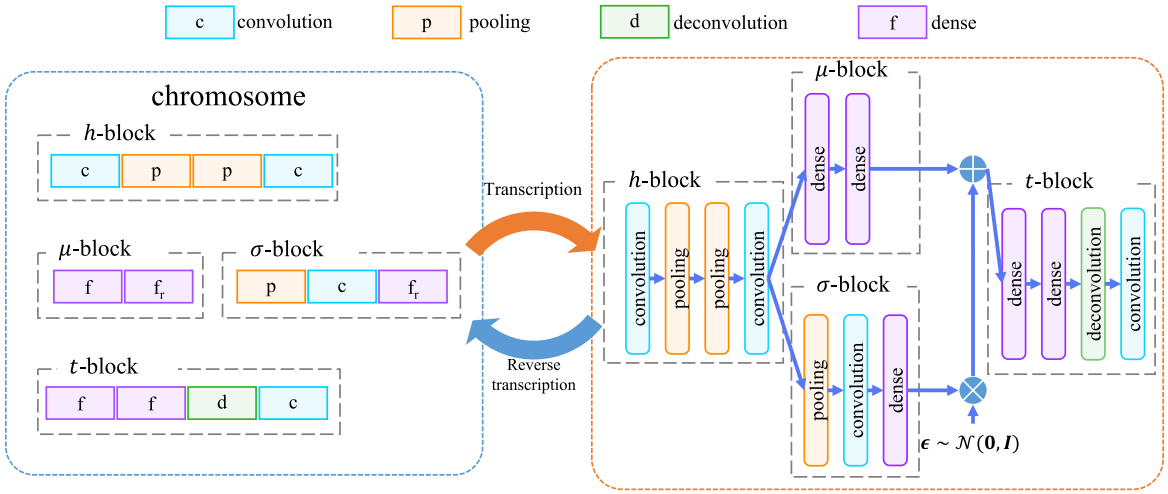


Fig. 3. Example of the chromosomes. The left part is the chromosome containing four different blocks, where "f_r" denotes the intrinsic dimensionality of the underlying manifold. The right part is the VAE model decoded by the chromosome, where the "latent" gene is decoded as the dense layers.

Particularly, the dense units correspond to the dense layers, and only encode the numbers of features. All the values of the attributes stored in the units are integers. For the pooling type in the pooling units, we use zero and one to distinguish the max pooling and average pooling units, respectively. Note that the ground-truth dimensionality of the manifold embedded in such high-dimensional measurement space is unknown to us. Thus, we design a "latent" gene plugged into the chromosome to estimate the intrinsic dimensionality of the underlying generative manifold of the input data.

The architecture of a particular VAE is encoded as a chromosome. Algorithm 2 shows how to encode a known VAE to a chromosome. Specifically, for a given VAE model m, we first separately extract each block in model m (line 2). Then, for each layer l in a block, we create a unit u and encode the type and the information of the layer in the genetic unit u (lines 3–20). Finally, the updated h-block, μ-block, σ-block, and t-block are combined to form a chromosome p (line 21). Note that for a valid model, a dense layer only appears in the tail of the encoder and the head of the decoder, while a deconvolutional layer only appears in the decoder. Fig. 3 illustrates an example of chromosome and the corresponding VAE model, where the left part of this figure is the chromosome and the right part is the decoded network model. In the chromosome, the h-block includes two convolutional units and two pooling units, the u-block includes two dense units, the σ-block includes one pooling unit, one convolutional unit, and one dense unit, and the t-block includes two dense units, one deconvolutional unit, and one convolutional unit. In the decoded network model, each block corresponds to the block of the chromosome. It is worth noting that we use "f_r" to represent the "latent" gene in the μ-block and the σ-block, since the "latent" gene works as a dense layer in the model.

C. Population Initialization

Based on the designed gene encoding strategy, a population of individuals is generated randomly. As discussed above, each individual is composed of four parts. For each of the parts, we first generate the number of each type of units, then randomly initialize the same number of units according to the chosen number.

Algorithm 3 shows the details of the population initialization. Specifically, for the generation of h-block, we uniformly sample an integer $n_{hc} \in [1 \dots N_c]$ and another integer $n_{hp} \in [1 \dots N_p]$ (line 3), where n_{hc} and n_{hp} denote the number of convolutional units and the pooling units in the h-block, respectively. In the next step, we randomly generate the n_{hc} convolutional layers and n_{hp} pooling layers represented by the genetic units, and then the units are concatenated into one list to form the h-block (line 4). The similar procedure can be easily adapted to obtain the network models for any other blocks, i.e., the μ-block (line 6), σ-block (line 7), and t-block (line 9).

After that, we randomly permute the convolutional, pooling, and deconvolutional units of each block so that different kinds of layers in the network model are mixed together. The motivation of doing so is that the order of layers in the DNN models could affect the final performance. It is worth noting that the h-block, μ-block, and σ-block are parts of the encoder. Thus, the maximal number of units of a particular type in the part of encoder is constrained by the predefined maximal number. Taking the maximal number of the convolutional units N_c as an example, the numbers of the convolutional units in the h-block, μ-block, and σ-blocks satisfy $\max(n_{hc} + n_{\mu c}, n_{hc} + n_{\sigma c}) \leq N_c$. To estimate the best dimensionality of the underlying manifold, we add an additional "latent" genetic fragment, a randomly chosen integer r from the predefined range R (line 11), into the chromosome of each individual (line 13). Finally, all the blocks including the "latent" size r are combined into a single individual (line 14). This procedure will be repeated until N individuals are initialized.

D. Evaluation

As introduced in Section II-A, the objective function for training a VAE is to minimize the reconstruction error with an extra term of the Kullback–Leibler (KL) divergence between

Algorithm 3: Population Initialization

Input: The population size N, the maximal number of dense layers N_f, the maximal number of convolutional N_c, the maximal number of pooling layers N_p, the maximal number of deconvolutional layers N_d and the maximal dimensionality of the latent representation R

Output: Seed individuals P_0

1 $P_0 \leftarrow \emptyset$;
2 **for** $i = 1$ **to** N **do**
3 Sample integers $n_{hc} \in [1 \dots N_c]$ and $n_{hp} \in [1 \dots N_p]$;
4 h-block$\leftarrow$ concatenate the randomly generated n_{hc} convolutional units and n_{hp} pooling units;
5 Sample integers $n_{\mu c} \in [0 \dots N_c - n_{hc}]$, $n_{\mu p} \in [0 \dots N_p - n_{hp}]$ and $n_{\mu f} \in [1 \dots N_f]$;
6 μ-block $\leftarrow$ Concatenate the randomly generated $n_{\mu c}$ convolutional units, $n_{\mu p}$ pooling units and $n_{\mu f}$ dense units;
7 Generate σ-block in the same manner as the μ-block generation process;
8 Sample integers $n_{tf} \in [1 \dots N_f]$, $n_{tc} \in [1 \dots N_c]$ and $n_{td} \in [1 \dots N_d]$;
9 t-block $\leftarrow$ Concatenate the randomly generated n_{tf} dense units, n_{tc} convolutional units, n_{td} deconvolutional units;
10 Shuffle the convoluional units, the pooling units and the deconvolutional units in each block;
11 Sample the dimensionality of the underlying manifold $r \in [2 \dots R]$;
12 Generate two dense units f_r^1 and f_r^2 with r neurons;
13 μ-block $\leftarrow \mu$-block $\cup f_r^1$, σ-block $\leftarrow \sigma$-block $\cup f_r^2$;
14 $p_i \leftarrow h$-block $\cup$ μ-block $\cup$ σ-block $\cup$ t-block;
15 $P_0 \leftarrow P_0 \cup p_i$;
16 **end**
17 **return** P_0;

the prior distribution and the recognition model. During the training process, when the reconstruction error is relatively large, the network will reduce the output of the variance part in order to better reconstruct the original image; and when the reconstruction error is reduced, the loss of the KL divergence part will increase, and the optimization will bias to allowing the output of that part close to the standard normal distribution. Therefore, the adversarial relationship between the two is potentially included in the training process of the network and can be regarded as a multiobjective optimization problem. Naturally, the loss function of training VAE is employed as the fitness to find a better network architecture so that the average loss is the lowest. That is, because our preferences in solving this multiobjective optimization problem are to highlight the best individual for the image classification, and the promising model with a lower loss often has better capability to model images.

Algorithm 4 depicts how to evaluate the given individuals in P_t and assign the final fitness to each individual. First, line 3 shows the details of decoding the chromosome to the network model m_i, and the initialization of the parameters of the VAE. Line 4 is the calculation of the number of iterations in each epoch based on the size of the unsupervised training dataset D_{ut} and the batch size b. Because the randomness is essential to the training, we shuffle D_{ut} before each epoch starts. The unsupervised training generally does not need any supervised signal, thus we only use the raw data itself, and the evaluation algorithm will return the lowest average loss on D_{ut} as the fitness value of an individual (line 14). Lines 7–11 show how to complete the unsupervised training of an epoch. First, we sample a minibatch data from D_{ut}, supposing the mini batch data are $\{\boldsymbol{x}_i\}_{i=1}^b$, where $\boldsymbol{x}_i \in \mathbb{R}^{h\times w\times c}$ denotes the ith image in the batch data. The corresponding reconstructed data are $\{\hat{\boldsymbol{x}}_i\}_{i=1}^b$. Derived from (2), the loss on the training minibatch can be obtained by

$$\frac{1}{2b}\sum_{i=1}^{b}\left(\sum_{m=1}^{h}\sum_{l=1}^{w}\sum_{k=1}^{c}\left(x_i^{m,l,k}-\hat{x}_i^{m,l,k}\right)^2 + \sum_{j=1}^{r}\left(\mu_{i,j}^2+\sigma_{i,j}^2-\log\sigma_{i,j}^2-1\right)\right) \tag{3}$$

where $x_i^{m,l,k}$ is the pixel at the position (m, l) of the kth channel of $\boldsymbol{x}_i$. Because we normalize the pixel values of each image to $[-1, 1]$, thus we utilize mean-square error (MSE) as the reconstruction error and divide it by the number of samples b. The loss function of one VAE model is calculated after feeding the minibatch into the model. Second, we compute the gradient of the unsupervised loss with respect to the parameters W_u contained in the encoder and the decoder. The parameters W_u related to the unsupervised pretraining include the parameters in the four blocks of the VAE. We can choose the standard batch-based stochastic gradient descent (SGD) to update the parameters. In principle, other prevailing optimizer such as Adam [48] is recommended and could also be applied.

E. Genetic Operators

Crossover and *mutation* are the two main genetic operators of GAs. Generally, the crossover operator plays the role of local search to provide the mixing of the dual chromosomes in the encoding space. The mutation changes part of a chromosome randomly to maintain the diversity of the population, and increases the ability to escape from the local minimal as the global search.

1) Crossover: The traditional GAs provide the crossover operation at hand for the fixed-length individuals, based on the biological phenomenon that the two parents must be from the same origin having the same number of genes. In order to find the optimal depth of VAEs, in this article, we propose the variable-length encoding strategy, giving rise to the requirement that a proper crossover operation should be redesigned accordingly. Specifically, the variable-length gene encoding strategy arises the nonidentical problem that the parent individuals to be mated may have different lengths. To solve this issue, Sun *et al.* [39] proposed the unit alignment method to the recombination of the dual individuals with variable-length chromosomes, where they force each chromosome to

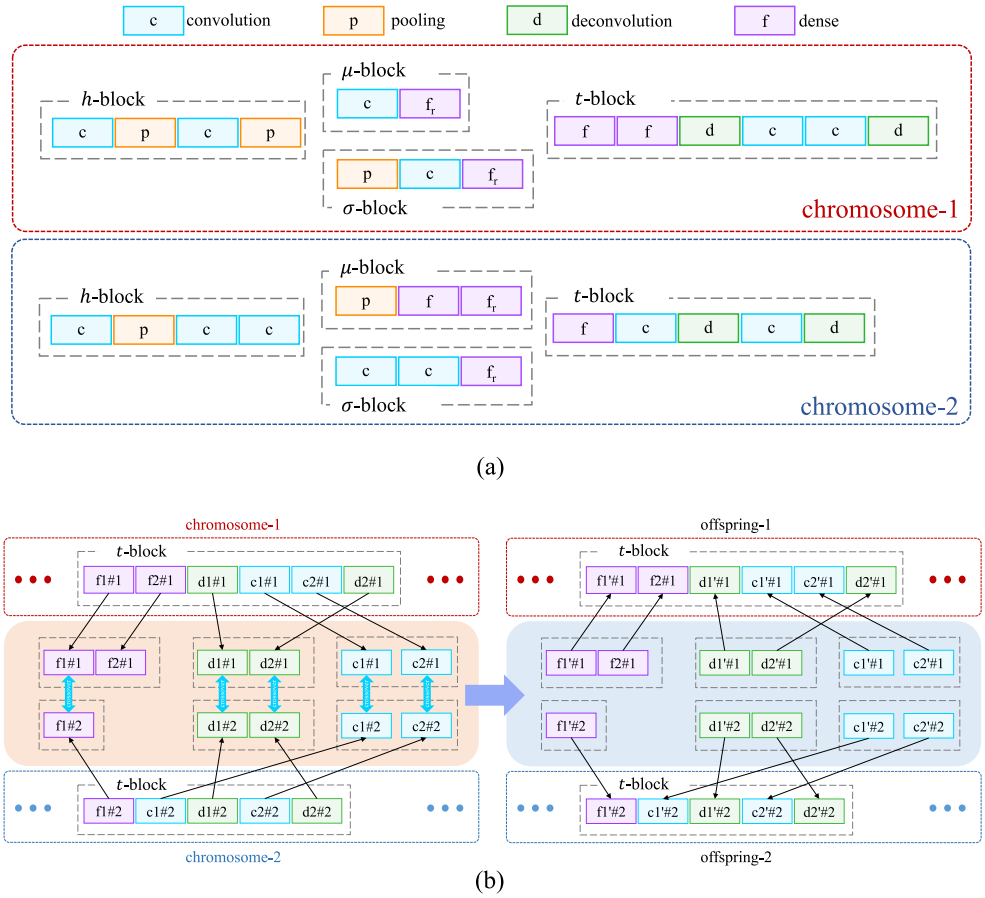


Fig. 4. Example to illustrate the proposed multiblock alignment strategy for the crossover. (a) Two examples of parent chromosomes (denoted as chromosome-1 and chromosome-2). (b) Diagram of the crossover on the t-block of the two chromosomes.

align at the head and exchange the gene information in the common segments, such a strategy is actually inspired by the crossover of human chromosomes. The studies have shown that two human chromosomes are interchanged in the homologous segments when crossover occurs, and the unique regions keep unchanged. In EvoVAE, each block in an individual may have different lengths, at the same time, the units on the same position in a block may be mismatched on the type of the layer. For example, Fig. 4(a) shows such a case, the type of first unit in the μ-block of the chromosome-1 is different from that of the chromosome-2. Thus, the hybridization of the information between the units with different types is infeasible, and the method cannot be directly applied to the proposed algorithm. To solve the emerged problem, we design the multiblock alignment strategy to achieve the chromosome crossover in the proposed EvoVAE.

An example for crossover with two variable-length chromosomes of different individuals is exhibited in Fig. 4, where each block is contoured with a gray dotted box in the figure, and the units representing different layers in the block are indicated with different colors, such as the convolutional layers and the pooling layers. Fig. 4(a) shows an example of the parents' chromosomes, the length of each chromosome, and the type of the corresponding genes are not the same or do not meet the one-to-one correspondence. This situation is common during the evolution, and the proposed multiblock alignment strategy can solve this problem well. In this strategy, we first extract the individual blocks, separately; all crossover operations are based on the same block. This also means that there is no information exchange between different blocks of the parents. Subsequently, we put the different types of genetic units in the blocks into a set of lists according to the type. Specifically, Fig. 4(b) shows how to perform the crossover on the t-blocks of the two chromosomes.

1) We take out the t-blocks of the parents, and let different types of the units in the t-block be in the different lists, then we align the list of the same unit type located on the two chromosomes according to the head.
2) The blue two-way arrow on the left part in Fig. 4(b) represents the crossover between the two genetic units, using the simulated binary crossover (SBX) [49]. If there is no corresponding unit in the list, no information will be exchanged.
3) After the crossover in the unit list is completed, we put each unit in the list back to the position of the original unit on the corresponding chromosomes. As a result, we generate two offspring from the chromosome-1 and the chromosome-2, as shown in the right part of Fig. 4(b).

Clearly, the multiblock alignment has realized the exchange of two chromosomes without changing the lengths. The other three blocks can be deduced by analogy.

Algorithm 4: Performance Evaluation

Input: The individual list P_t, unsupervised training epoch T_u, the unsupervised training dataset D_{ut}, the learning rate γ, and batch size b
Output: The population with fitness P_t

for $i = 1$ **to** $|P_t|$ **do**
  $f \leftarrow -\infty$;
  Decode the VAE represented by the individual P_t^i as m_i and initialize its weight;
  $K = \lceil |D_{ut}|/b \rceil$;
  for $t = 1$ **to** T_u **do**
    Shuffle the training dataset D_{ut};
    for $k = 1$ **to** K **do**
      $X \leftarrow$ Fetch the k-th mini-batch data in D_{ut};
      Calculate the loss l_k by (3) after feed X into the m_i;
      $W_u \leftarrow W_u - \gamma * \frac{\partial l_k}{\partial W_u}$;
    end
    $f \leftarrow \max(-\frac{1}{K}\sum_{k=1}^{K} l_k, f)$;
  end
  Assign f to the i-th individual P_t^i;
end
return P_t;

2) Mutation: We design three mutation operations in the proposed EvoVAE, *addition*, *deletion*, and *modification*, which act upon the offspring obtained by the crossover. When the mutation begins, we use a uniform distribution to determine which mutation method we could use. The addition and deletion operations achieve the alternation of the chromosome lengths. In the following, the three operations are introduced.

Addition: The addition operation aims to add a layer in a particular chosen block. First, we randomly choose a block x in the given chromosome, and randomly generate the unit u which represents a particular layer, whose type and parameters are generated from a uniform distribution with the predefined sets. Recall that there exist some constraints on a valid architecture of the VAE: the deconvolutional units only exist in the t-block; the pooling units only exist in the blocks representing the encoder of the VAE; and the dense units only exist in the tail of the μ-block and σ-block. Thus, the type of u must be treated differently according to the block x. Finally, u is randomly inserted into the block x with a randomly selected position, which obeys the mentioned constraints.

Deletion: The deletion operation is designed to remove a layer from a particular block. Specifically, we randomly choose a block x in the given chromosome. Then, one of the units in the block x will be removed.

Modification: The modification operation is designed to change the values of the attributes in the genetic unit of the given chromosome: the kernel sizes and the number of the features in the convolutional units, the number of the features in the dense units, the kernel sizes and the number of the features in deconvolutional units, and the type of the pooling units. For each genetic unit u in the chromosome, the polynomial mutation [50] is applied to the values of the attributes for u, due to the promising performance in processing mutations on the real numbers.

F. Environmental Selection

We use environmental selection to produce the parent population for the next generation from the combination of the current population and their offspring. The environmental selection is achieved by collectively employing the elitist selection and the civilian selection [39]. For doing so, we first select the best partition from them as the elitism, so as to ensure the convergence of the optimization results. At the same time, in order to ensure the diversity of the next generation of individuals, among the remaining individuals who have not been selected as the elite, the binary tournament competition is used to select the remaining civilians. Finally, the elitism and civilians are merged as the population for the next generation.

IV. Experiment Design

To demonstrate the effectiveness of the proposed EvoVAE algorithm, a group of well-designed experiments are performed. In this section, the chosen benchmark datasets, implemented details, and peer competitors for the experiments are elaborated in the following sections.

A. Benchmark Datasets

Based on the conventions of the community of AEs, three widely used benchmark datasets are chosen for the experiments, they are MNIST, SVHN, and CIFAR-10.

MNIST: The MNIST benchmark dataset [51] contains 60 000 training and 10 000 test examples with ten categories, aiming to classify the numerical number from 0 to 9. Each sample in the MNIST is a small 28×28 pixel grayscale image of the handwritten single digit, and each category that such a digit belonging to has the same number of the samples.

SVHN: The SVHN dataset [52] is a real-world image dataset, obtained from house numbers in Google Street View images. The official website of the SVHN provides two formats of the datasets. For simplicity, we use the dataset with format-2, which comprises of 73 257 training and 26 032 test images. In the dataset, each image is resized to a fixed resolution of 32×32 pixels. The format-2 dataset is an MNIST-like dataset, containing categories from 0 to 9. But the SVHN dataset is significantly harder since it requires the classifier to recognize digits in natural scene images.

CIFAR-10: The CIFAR-10 benchmark dataset [53] contains 50 000 training images and 10 000 test images with ten categories of natural objects (i.e., airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck). Each one is an RGB image with resolution of 32×32, where the objects with the different categories occupy different areas. The numbers of the samples in each category are roughly equal.

B. Parameter Setting

In the proposed EvoVAE algorithm, the parameters related to evolution are set according to the convention in the community of the GAs [54]. Specifically, the probabilities of crossover

and mutation are set to be 0.8 and 0.1, respectively. The distribution index of both SBX [49] and polynomial mutation [55] is set to be 20. The size of population and the number of generations are both set to be 20, as suggested in [56]. In addition, the proportion of elitism is set to be 20% according to the Pareto principle [39].

Unsuitable settings for the convolutional, deconvolutional, and dense layer parameters will deteriorate the performance of the EvoVAE, and also arise additional computational overhead. The number of the convolutional layers is set to be in $[1 \ldots 6]$; the channels of the convolutional and deconvolutional layer is restricted to be in $[20 \ldots 128]$; the number of the dense layers is set to be in $[1 \ldots 2]$, whose number of the neurons is set to be in $[64 \ldots 256]$; the number of the pooling layers is set to be in $[1 \ldots 2]$; the size of the kernel is set to be an integer in $[2 \ldots 5]$; and the size of the latent representation r is set to be in $[2 \ldots 128]$. Furthermore, according to the prior knowledge of parameter setting in mainstream network architectures, such as VGG-Net [46] and Res-Net [57], we set the convolution kernel and stride to be square, i.e., their length and width are the same. For the regular convolutional layers, the "same" padding operation is employed to ensure that the resolutions of input and output are the same. Thus, the reduction of resolution only depends on the pooling operation, where the kernel size and stride size are fixed to be two. This setting not only follows the conventions of the deep learning community but also facilitate to control the network parameters and the output resolution during the evolution, while reducing the complexity of the gene decoding.

C. Peer Competitors

Because the proposed algorithm focuses on the AEs, the state-of-the-art AEs are chosen as the peer competitors for the comparisons, including the AE, CAE [58], DAE [59], Sparse AE (SAE) [60], and their stacked forms, i.e., the stacked AE (sAE), stacked SAE (sSAE), stacked DAE (sDAE), stacked CAE (sCAE), and stacked convolutional DAE (sCDAE) are also chosen for the extensive comparisons. We did not introduce the convolutional neural network (CNN) [51] for this comparison. On the one hand, we are more concerned about the AEs and VAEs. On the other hand, the second fine-tuning phase for the CAEs is actually equivalent to the supervised training of the CNNs.

Because the proposed algorithm focuses on the AEs, the state-of-the-art AEs are chosen as the peer competitors for the comparisons, including AE, CAE [58], DAE [59], Sparse AE (SAE) [60], and their stacked forms, i.e., sAE, sSAE, sDAE, sCAE, and sCDAE are also chosen for the extensive comparisons. We did not introduce CNN [51] for this comparison. On the one hand, we are more concerned about the AEs and the VAEs. On the other hand, the second fine-tuning phase for the CAEs is actually equivalent to the supervised training of the CNNs. Because the architectures of the chosen peer competitors are manually designed, for a fair comparison, an equal number of candidates as populations in EvoVAE for each AE is generated, their architectures are designed by performing the random search using the same ranges of the architecture parameters and the same candidate numbers defined for performing the EvoVAE, and the one having the best performance is selected for the comparison, i.e., the one that has the smallest reconstruction error on the entire unsupervised training data. This is because the reconstruction error is the widely used criterion for training AEs. Note that the maximal number of pooling layers of the proposed algorithm is set to two. Based on our design, once a pooling layer is added, two convolutional layers will be added to the encoder, and a deconvolutional layer and a convolutional layer are added to the decoder accordingly. Therefore, the maximal number of layers for the proposed algorithm is 10. For a fair comparison, the number of hidden layers of all the staked AEs in the experiments is also set to 10.

In addition, the semisupervised learning model (SSL) [10], which is based on VAEs, is also chosen as the peer competitor because it has achieved the best performance very recently. We adopt the "M1" model proposed in SSL, and we rename it as "SSL(M1)" in the following experiments. To obtain the particular experimental results, we reimplement the network architecture according to the original paper. The architecture of SSL(M1) is handcrafted, whose encoder and decoder are constructed with two hidden dense layers, each with 600 neurons, using a softplus $\log(1+e^x)$ activation function. They set the dimensionality of the underlying manifold to be 50.

Except for the above AEs, some other existing solutions in machine learning, such as k-nearest neighbors (KNN) where we let k be one, support vector machine (SVM) [61], and learning with local and global consistency (LLGC) [62] are also employed as the baselines to solve the image classification problems in the following experiments, due to their widespread applications and the fast reproduction. Note that LLGC is a semisupervised learning method. The results of these three algorithms could help the reader to have an intuitive understanding about the difficulty of a particular task.

D. Characteristics of Performance Evaluation

It is hard to directly employ the generic AEs for the image classification. We follow the mechanism that builds a classifier on the top of the encoder of an AE [36], [63]. To complete the image classification with the AE, the whole process often includes two phases: 1) the unsupervised pretraining phase and 2) the supervised fine-tuning phase. Specifically, in the unsupervised pretraining phase, we leverage the massive unlabeled samples to train the encoder and the decoder of the AE. After that, we use the labeled samples to train the classifier added on the AE in the second phase. Note that the decoder is usually deprecated after the first phase and never used in the second phase. The details of the training mechanism that are used in EvoVAE and the other AE-like competitors are discussed in the following paragraphs.

In the first training phase of the VAE-like models, we follow the community convention to set the sampling frequency L of (2) to be one. At the same time, to speed up the convergence of the training, we adopt the Adam optimizer [48] in our experiments. The initial learning rate is set to be 0.001, the batch size is set to be 128, and the weight decay is set to be 0.00001. Note that we do not perform any special preprocessings on

TABLE II
EXPERIMENTAL RESULTS OF THE COMPARED ALGORITHMS ON THE MNIST DATASET

Methods	100	600	1K	2K	3K	5K	10K	60K
KNN	73.09	86.03	88.58	90.65	92.20	93.67	94.89	96.88
SVM	79.34	91.32	93.09	94.60	95.19	95.99	96.72	98.33
LLGC	**87.86**	94.00	94.35	95.28	95.37	95.83	96.24	96.93
AE	75.36 (2.43)	87.81 (0.13)	90.41 (0.45)	92.54 (0.17)	94.25 (0.06)	95.45 (0.15)	96.94 (0.18)	98.40 (0.07)
CAE	78.95 (0.91)	92.73 (0.43)	94.81 (0.35)	96.64 (0.16)	97.07 (0.11)	97.05 (0.28)	98.01 (0.09)	98.95 (0.02)
DAE	76.78 (3.19)	88.27 (0.23)	91.16 (0.42)	93.89 (0.20)	94.97 (0.27)	96.22 (0.18)	97.24 (0.14)	98.63 (0.04)
SAE	78.87 (0.86)	88.71 (0.33)	90.44 (0.17)	92.92 (0.21)	94.32 (0.27)	95.60 (0.16)	96.97 (0.02)	98.35 (0.06)
sAE	73.64 (1.30)	86.93 (0.40)	89.05 (0.37)	92.14 (0.26)	93.58 (0.40)	95.06 (0.32)	96.58 (0.11)	98.34 (0.06)
sCAE	77.29 (2.04)	93.04 (0.34)	95.03 (0.28)	96.51 (0.21)	97.31 (0.15)	97.89 (0.07)	98.51 (0.10)	99.37 (0.04)
sDAE	73.85 (1.37)	88.87 (0.28)	90.95 (0.26)	93.40 (0.22)	94.43 (0.18)	95.26 (0.26)	96.56 (0.18)	98.15 (0.06)
sSAE	22.91 (1.24)	80.72 (1.69)	84.93 (1.29)	90.74 (0.67)	92.02 (0.35)	94.52 (0.20)	96.03 (0.11)	98.10 (0.12)
sCDAE	81.33 (0.94)	94.53 (0.11)	96.05 (0.17)	97.26 (0.16)	97.78 (0.08)	98.29 (0.09)	98.74 (0.03)	99.41 (0.02)
SSL(M1+FC)*	72.78 (1.27)	88.21 (0.21)	90.64 (0.40)	93.42 (0.23)	94.51 (0.22)	95.51 (0.13)	96.95 (0.14)	98.43 (0.09)
EvoVAE(Ours)	87.65 (1.49)	**95.92 (0.44)**	**96.76 (0.31)**	**97.89 (0.27)**	**98.38 (0.20)**	**98.79 (0.16)**	**99.14 (0.13)**	**99.48 (0.06)**

∗ We adopt the network architecture of the VAE described in Kingma *et al.* [10] and evaluate it in our framework for the fair of comparison.

the input image data, such as the regular data augmentation operation, or other tricks.

In the practical experiments, it is generally necessary to train the network to the magnitude of 10^2 or 10^3 on the entire training set during the first unsupervised pretraining phase. However, this number of training epochs is unacceptable in terms of the computational complexity of the population-based EvoVAE. Actually, we do not need to iterate so many times as suggested in [36]. Hence, we set the number of the epochs to be 20 during the unsupervised pretraining phase, and this setting can greatly accelerate the proposed EvoVAE algorithm.

Once the best architecture of the VAE is determined by the proposed EvoVAE, we will deeply train the found network model with a longer time for the image classification. Specifically, we add the classifier, a subnetwork with two dense layers and a dropout [64] layer, on the top of the μ-block. For the deep training, we set the number of epochs to be 400 in the first phase, and set the number of epochs to be 100 in the fine-tuning phase. These two parameters are consistent among all the AE-like models in our experiments.

All the experiments are implemented with PyTorch, a prevailing open-source deep learning framework. Each copy of the codes runs on the machine equipped with two NVIDIA RTX 2080TI GPUs. For the acceleration and stability of the training, the batch normalization [65] layer is added after each convolutional layer in all the used models, and we also employ the widely used rectifier linear unit (ReLU) [66] as the activation function of the hidden layers. In the supervised fine-tuning phase, all the DNN models are initialized with five times of random seeds and trained independently. Then, the means and SDs of the five results are reported to reduce the disturbance caused by the random values. By the way, the whole time of the network architecture optimization on the above datasets need about three days on the two NVIDIA RTX 2080TI GPUs.

V. EXPERIMENTAL RESULTS AND ANALYSIS

In this section, we summarize the overall results and the evolutionary trajectories of the proposed EvoVAE on the three benchmark datasets, demonstrating the proposed EvoVAE can automatically find the best architectures of the VAEs with the fairly competitive performances.

A. Overall Results

In this section, we use the VAEs optimized by the proposed EvoVAE algorithm to solve the image classification problems, and compare the classification accuracy with other chosen competitors. As discussed in Section IV, the generic AEs are unsupervised learning models, which cannot be directly applied on the image classification tasks. In this article, the image classification is actually done in the second fine-tuning phase, and the first unsupervised pretraining phase can be regarded as an auxiliary process to make the model be with a better initial weight for the second phase, such that the fine-tuning phase could benefit from the first phase. A common criteria, which we use to evaluate the performance of the training mechanism, is to vary the number of labeled samples in the second fine-tuning phase, such as the work in [36]. In principal, a promising unsupervised learning model could achieve a better performance on the supervised tasks even when there are only a small number of labeled samples. Hence, we design the experiments under different numbers of the labeled samples to investigate the capabilities of these unsupervised learning algorithms. As for the KNN and the SVM algorithms, because they do not have the ability to first learn from the large number of unlabeled samples, both algorithms should be directly applied on the labeled samples. In this way, we can also observe how the unsupervised learning algorithms could boost the performance of the supervised learning algorithms.

Tables II–IV show the experimental results of the EvoVAE on the three benchmark datasets, with the architectures optimized by itself, where the best results are highlighted in bold. On the MNIST dataset, we vary the number of labeled samples from 100, 600, 1K, 2K, 3K, 5K, 10K, and 60K. On the SVHN dataset, we vary the number of labeled samples from 100, 600, 1K, 2K, 5K, 10K, and 40K. On the CIFAR-10 dataset, this changes to 100, 600, 1K, 2K, 3K, 5K, 10K, and 50K, respectively.

As can be seen from the results, the VAE optimized by the EvoVAE achieves the competitive results in most experiments with the varied numbers of the labeled samples on the chosen datasets. Overall, the proposed EvoVAE gains best performances among all the compared algorithms except the cases of 100 labeled samples, where the exceptions happen to LLGC, sSAE, and CAE, respectively. Especially on the

TABLE III
EXPERIMENTAL RESULTS OF THE COMPARED ALGORITHMS ON THE SVHN DATASET

Methods	100	600	1K	2K	5K	10K	40K	FULL[1]
KNN	13.80	18.36	19.92	23.12	27.69	31.44	37.54	42.37
SVM	12.54	12.97	17.22	20.31	29.52	36.74	51.68	54.34
LLGC	12.92	18.83	21.24	25.39	28.01	31.34	45.29	48.39
AE	16.39 (1.60)	40.69 (2.00)	50.15 (0.89)	59.45 (0.43)	68.07 (0.72)	72.92 (0.61)	79.13 (0.10)	82.70 (0.15)
CAE	17.26 (1.86)	18.86 (1.45)	18.15 (1.76)	19.59 (0.00)	17.88 (3.41)	19.59 (0.00)	19.59 (0.00)	19.59 (0.00)
DAE	16.42 (0.92)	38.93 (1.05)	48.97 (1.62)	59.39 (0.79)	63.67 (0.68)	73.10 (0.70)	79.83 (0.25)	82.78 (0.25)
SAE	18.49 (1.43)	43.55 (1.15)	52.11 (1.34)	60.02 (0.94)	68.19 (0.46)	72.67 (0.50)	80.17 (0.20)	83.17 (0.18)
sAE	15.72 (1.01)	38.97 (1.04)	47.87 (1.44)	57.16 (0.38)	65.84 (0.66)	70.71 (0.49)	78.52 (0.30)	81.98 (0.17)
sCAE	18.82 (1.54)	19.59 (0.01)	19.59 (0.00)	19.59 (0.00)	19.59 (0.00)	19.59 (0.00)	19.59 (0.00)	89.15 (0.10)
sDAE	15.08 (0.68)	42.66 (1.55)	51.85 (1.03)	61.11 (0.63)	68.42 (0.93)	73.71 (0.51)	81.01 (0.13)	83.73 (0.16)
sSAE	**19.59 (0.00)**	19.08 (1.53)	21.96 (4.74)	24.48 (2.56)	41.50 (2.38)	53.08 (4.33)	76.29 (0.43)	80.98 (0.18)
sCDAE	17.58 (2.45)	20.02 (1.29)	17.54 (3.26)	19.59 (0.00)	19.59 (0.00)	19.59 (0.00)	19.59 (0.00)	89.56 (0.19)
SSL(M1+FC)	16.00 (1.10)	42.58 (1.82)	49.71 (0.98)	57.99 (1.33)	66.41 (0.68)	71.53 (0.29)	78.92 (0.34)	82.38 (0.12)
EvoVAE(Ours)	16.74 (1.37)	**55.69 (2.09)**	**66.13 (2.55)**	**75.32 (2.73)**	**79.31 (1.97)**	**82.51 (1.56)**	**90.37 (0.97)**	**92.01 (0.70)**

[1] Use full labels in the training dataset.

TABLE IV
EXPERIMENTAL RESULTS OF THE COMPARED ALGORITHMS ON THE CIFAR-10 DATASET

Methods	100	600	1K	2K	3K	5K	10K	50K
KNN	18.45	23.13	23.99	24.49	25.72	27.17	29.09	35.39
SVM	24.15	34.09	36.75	39.15	40.88	43.11	45.61	51.40
LLGC	17.18	22.15	22.62	26.02	26.86	28.08	29.81	36.06
AE	26.45 (1.02)	33.95 (0.44)	36.18 (0.32)	39.77 (0.43)	41.53 (0.27)	44.49 (0.40)	47.55 (0.19)	55.65 (0.28)
CAE	**27.35 (0.89)**	33.89 (3.89)	32.85 (6.40)	32.05 (7.94)	29.41 (4.64)	29.65 (4.30)	37.02 (3.54)	42.73 (16.37)
DAE	25.48 (1.73)	33.81 (0.40)	36.35 (0.67)	40.28 (0.65)	42.03 (0.25)	44.50 (0.21)	48.22 (0.26)	55.87 (0.13)
SAE	24.64 (0.69)	32.97 (0.81)	35.26 (0.38)	38.71 (0.33)	40.51 (0.46)	43.52 (0.53)	46.84 (0.40)	54.40 (0.36)
sAE	24.93 (1.01)	32.34 (0.93)	34.67 (0.98)	37.45 (0.33)	39.70 (0.22)	42.32 (0.35)	46.13 (0.15)	54.66 (0.33)
sCAE	25.76 (1.13)	27.83 (6.88)	31.73 (4.82)	26.27 (3.46)	26.08 (4.40)	35.95 (5.50)	41.84 (1.72)	63.42 (1.37)
sDAE	25.51 (1.15)	32.61 (0.44)	35.86 (0.22)	39.28 (0.45)	41.07 (0.25)	43.97 (0.45)	47.74 (0.16)	55.68 (0.21)
sSAE	15.84 (0.77)	21.46 (0.46)	23.83 (1.55)	28.27 (1.15)	29.82 (1.39)	33.87 (0.20)	38.74 (0.36)	49.34 (0.32)
sCDAE	26.20 (1.26)	37.42 (1.50)	38.51 (3.23)	45.35 (1.12)	44.04 (3.18)	46.01 (1.51)	54.79 (1.24)	66.80 (2.11)
SSL(M1+FC)*	24.05 (1.13)	33.13 (0.92)	35.14 (0.55)	38.29 (0.15)	40.26 (0.29)	43.16 (0.32)	46.54 (0.36)	54.55 (0.41)
EvoVAE(Ours)	26.81 (1.96)	**40.96 (2.03)**	**45.12 (1.33)**	**51.59 (1.09)**	**54.79 (0.95)**	**59.04 (0.88)**	**64.58 (1.35)**	**76.12 (1.94)**

CIFAR-10 dataset, the proposed algorithm outperforms the counterparts by a large margin, which demonstrates the superiority of the proposed EvoVAE. Additionally, it can be found that with an increase number of training samples, the promotion of the unsupervised learning weakens as the number of the samples increases. Particularly, in the case of the full samples on the SVHN dataset, the accuracy of the proposed EvoVAE is only 2.45% higher than the suboptimal sCDAE algorithm. It is worth noting that the training settings, such as the number of the training epochs and the initial learning rate, are set to be the same for all the models to keep consistent, but this strategy might cause that some models do not converge in Tables III and IV.

Compared with the baseline algorithms, we discover that the two-phase training mechanism is effective in the most cases. On the MNIST dataset, the best result of the EvoVAE is 8.31% better than the SVM algorithm when using 100 labeled samples. On the SVHN dataset, the accuracy gap is particularly prominent. In the case of the 2000 samples, this improvement expands to 55.01%. Besides, in some cases of a small number of samples, the two-phase training mechanism seems to be less noticeable. On the CIFAR-10 dataset with 100 labeled samples, the best AE only outperforms the best baseline algorithm by 3.2%. This limitation might be brought by the two-phase training mechanisms, thus an end-to-end semisupervised algorithm should be investigated to replace the naive two-phase training mechanism that we use. Additionally, the experimental results also reveal that LLGC is superior on the less complicated datasets with the small number of labeled samples. Specifically, with 100 labeled samples on the MNIST dataset, the LLGC method shows the best performance and outperforms EvoVAE and SVM by 0.21% and 8.52%, respectively. On the CIFAR-10 dataset, the performance of the LLGC is comparable to that of KNN.

In summary, the results exhibited in this section show the promising performance of the proposed EvoVAE on the varying numbers of the training samples. The competitive results demonstrate the scalability of the proposed EvoVAE in dealing with the image classification problems on the chosen datasets.

B. Evolution Trajectory of EvoVAE

In order to confirm that the proposed EvoVAE algorithm does effectively optimize the architectures of the VAEs and obtains promising architecture, we show the optimization trajectories of the EvoVAE on the three benchmark datasets in Fig. 5, where the *x*-axis represents the corresponding generation and the *y*-axis represents the loss of the best architecture of the VAE.

Compared with the other two datasets in Fig. 5, the CIFAR-10 dataset with real object is more complicated. It can also be seen from the loss curve in Fig. 5(c) that the corresponding VAE on the dataset has a loss greater than 100. At the same time, the losses on the MNIST and SVHN datasets are smaller than that of the CIFAR-10 dataset. Especially on the MNIST dataset, the loss changes around 1, and has converged from the 6th generation. On the SVHN dataset, the proposed EvoVAE converged since about 11th generation, then its loss was around 54.4. On the CIFAR-10 dataset, it converged at the

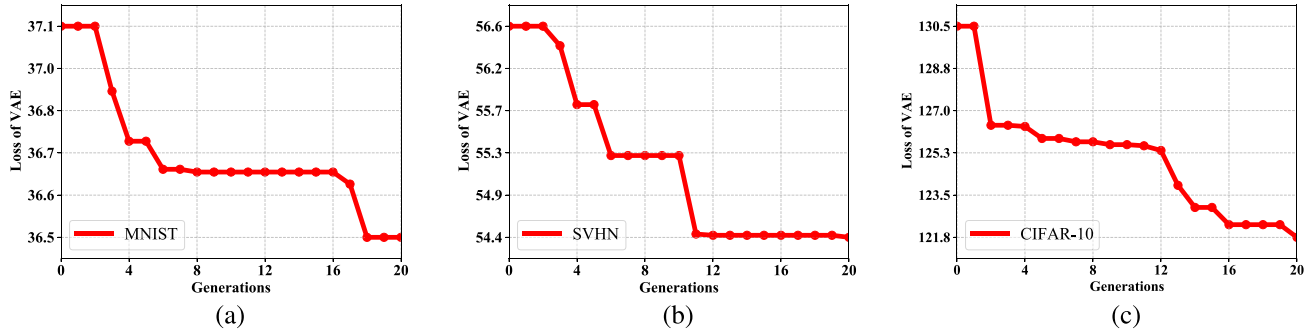


Fig. 5. Trajectory of evolved VAEs in the architecture discovering. (a)–(c) show the trajectories of the EvoVAE on MNIST, SVHN, and CIFAR-10 datasets, respectively.

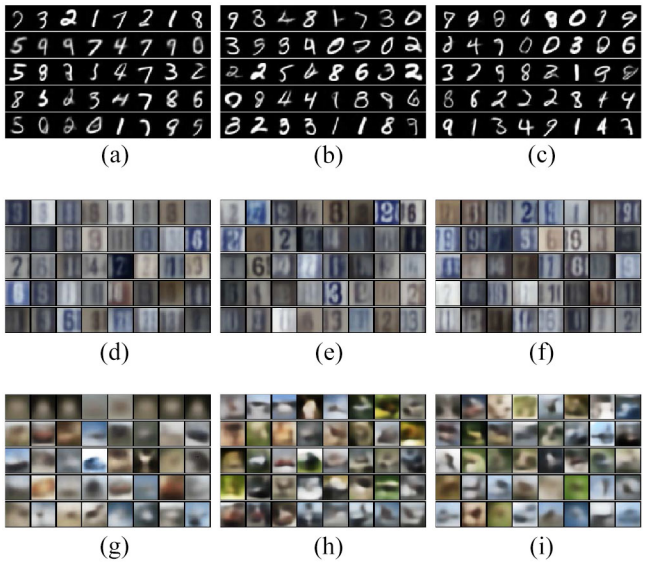


Fig. 6. Fig. 6. Randomly generated images, each row in the subfigure is generated by the individuals that are randomly selected during the evolution. (a-c) Generated images in the first, the 7-th and the last generations on the MNIST dataset. (d-f) Generated images in the first, the 12-th and the last generations on the SVHN dataset. (g-i) Generated images in the first, the 16-th and the last generations on the CIFAR-10 dataset.

16th generation, and there seems to be a tendency to decrease continuously.

Another function of the VAEs is to generate virtual images for the end users who want to obtain the synthetic data. By sampling from the multivariate normal distribution to obtain a random latent variable z', then feeding z' into the decoder of the VAE, a synthetic fake sample can be generated. In order to better clarify the effectiveness of the architecture search, we plot the generated images in Fig. 6 during the evaluation of each individual, where Fig. 6(a)–(i) represents the generated images on the three benchmark datasets, and the subfigures from the left to the right represent the images generated by the first, middle, and last generation of population, respectively.

It can be seen from the datasets such as SVHN and CIFAR-10, under the same small number of training epochs, there is a significant quality improvement in sharpness compared with the first generation. For example, in Fig. 6(f), we can observe that the number of the clear digits is larger than that of Fig. 6(d). However, on the MNIST dataset, there is no obvious difference of the image quality among the different generations. Meanwhile, compared with the GANs, the images generated by the VAEs are more blurred, especially for complex datasets such as CIFAR-10. There are also works leveraging the VAEs to synthesize high-quality images, such as human faces [14], but it is beyond the scope of this study.

VI. Conclusion

The goal of this article is to design an EA to automatically search the architectures of the deep convolutional VAEs (in short named as EvoVAE), aiming to effectively solve two limitations of the VAEs in the real applications: 1) the inefficient process of the handcrafted architecture designs and 2) the symmetrical constraint. To achieve this goal, we first designed a flexible architecture of the VAEs to address the limitation brought by the symmetrical constraint. Next, we developed a new variable-length gene encoding strategy to automatically search the optimal depth of the architectures. In addition, we developed the associated crossover and mutation operations to address the nonidentical problem caused by the asymmetrical manner and the variable-length encoding strategy. To justify the performances of the proposed EvoVAE for the image classification problems, we conducted extensional experiments on the MNIST, SVHN, and CIFAR-10 datasets to investigate the performances of the optimized architectures by the proposed EvoVAE. In the experiments, nine commonly used AEs, including CAE, sCAE, and etc., are chosen as the peer competitors. Overall, the proposed EvoVAE algorithm outperforms the best competitors by 1.39%, 14.21%, and 13.03% on the chosen datasets, which reveals the superiority of the proposed EvoVAE. In the future, we will try the semisupervised models in the deep learning, to design a better fitness evaluation, and to utilize the evolution-based frameworks to search their optimal architectures.

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436, 2015.
[2] G. E. Hinton *et al.*, *Unsupervised Learning: Foundations of Neural Computation*. Cambridge, MA, USA: MIT Press, 1999.
[3] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio, "Why does unsupervised pre-training help deep learning?" *J. Mach. Learn. Res.*, vol. 11, pp. 625–660, Feb. 2010.
[4] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Cogn. Model.*, vol. 5, no. 3, p. 1, 1988.
[5] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," in *Proc. Int. Conf. Learn. Represent.*, 2014, p. 9.

[6] D. P. Kingma and M. Welling, "An introduction to variational autoencoders," *Found. Trends Mach. Learn.*, vol. 12, no. 4, pp. 307–392, 2019.

[7] X. Chen, J. Song, and O. Hilliges, "Unpaired pose guided human image generation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR) Workshops*, Jun. 2019, pp. 46–55.

[8] Y. Pu *et al.*, "Variational autoencoder for deep learning of images, labels and captions," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 2352–2360.

[9] M. Blaauw and J. Bonada, "Modeling and transforming speech using variational autoencoders," in *Proc. Interspeech*, 2016, pp. 1770–1774.

[10] D. P. Kingma, S. Mohamed, D. J. Rezende, and M. Welling, "Semi-supervised learning with deep generative models," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 3581–3589.

[11] T. D. Kulkarni, W. F. Whitney, P. Kohli, and J. Tenenbaum, "Deep convolutional inverse graphics network," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 2539–2547.

[12] I. J. Goodfellow *et al.*, "Generative adversarial nets," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 2672–2680.

[13] S. M. Arjovsky and L. Bottou, "Wasserstein generative adversarial networks," in *Proc. 34th Int. Conf. Mach. Learn.*, Sydney, NSW, Australia, 2017, pp. 214–223.

[14] B. Dai and D. Wipf, "Diagnosing and enhancing VAE models," in *Proc. Int. Conf. Learn. Represent.*, 2018, pp. 1–12.

[15] K. H. Lee, S. J. Kang, W. H. Kang, and N. S. Kim, "Two-stage noise aware training using asymmetric deep denoising autoencoder," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP)*, 2016, pp. 5765–5769.

[16] J. H. Holland *et al.*, *Adaptation in Natural and Artificial Systems: An Introductory Analysis With Applications to Biology, Control, and Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1992.

[17] C. Doersch. (2016). *Tutorial on Variational Autoencoders*. [Online]. Available: https://arxiv.org/abs/1606.05908

[18] M. Wistuba, A. Rawat, and T. Pedapati. (2019). *A Survey on Neural Architecture Search*. [Online]. Available: https://arxiv.org/abs/1905.01392

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

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

[21] S. Xie, H. Zheng, C. Liu, and L. Lin, "SNAS: Stochastic neural architecture search," in *Proc. Int. Conf. Learn. Represent.*, 2019, pp. 1–17.

[22] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," in *Proc. Int. Conf. Learn. Represent.*, 2019, pp. 1–13.

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

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

[25] C. E. Rasmussenand and C. Williams, *Gaussian Processes for Machine Learning, Vol. 1*, vol. 39, MIT Press, 2006, pp. 40–43.

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

[27] C. White, W. Neiswanger, and Y. Savani. (2019). *Bananas: Bayesian Optimization With Neural Architectures for Neural Architecture Search*. [Online]. Available: https://arxiv.org/abs/1910.11858

[28] X. Dong and Y. Yang, "Searching for a robust neural architecture in four GPU hours," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 1761–1770.

[29] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," in *Proc. Int. Conf. Learn. Represent.*, 2017, pp. 1–15.

[30] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

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

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

[33] T. Elsken, J. H. Metzen, and F. Hutter, "Efficient multi-objective neural architecture search via Lamarckian evolution," in *Proc. Int. Conf. Learn. Represent.*, 2019, pp. 1–23.

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

[35] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. Int. Conf. Neural Netw. (ICNN)*, vol. 4, 1995, pp. 1942–1948.

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

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

[38] Y. Li, J. Xiao, Y. Chen, and L. Jiao, "Evolving deep convolutional neural networks by quantum behaved particle swarm optimization with binary encoding for image classification," *Neurocomputing*, vol. 362, pp. 156–165, Oct. 2019.

[39] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Evolving deep convolutional neural networks for image classification," *IEEE Trans. Evol. Comput.*, vol. 24, no. 2, pp. 394–407, Oct. 2020.

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

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

[42] Z. Lu *et al.*, "Multi-objective evolutionary design of deep convolutional neural networks for image classification," *IEEE Trans. Evol. Comput.*, early access, Sep. 21, 2020, doi: 10.1109/TEVC.2020.3024708.

[43] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "An experimental study on hyper-parameter optimization for stacked auto-encoders," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, 2018, pp. 1–8.

[44] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, pp. 2980–2988.

[45] B. L. Miller and D. E. Goldberg, "Genetic algorithms, tournament selection, and the effects of noise," *Complex Syst.*, vol. 9, no. 3, pp. 193–212, 1995.

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

[47] Y. Bengio and O. Delalleau, "On the expressive power of deep architectures," in *Proc. Int. Conf. Algorithmic Learn. Theory*, 2011, pp. 18–36.

[48] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. Learn. Represent.*, 2014.

[49] K. Deb and R. B. Agrawal, "Simulated binary crossover for continuous search space," *Complex Syst.*, vol. 9, no. 2, pp. 115–148, 1995.

[50] K. Deb, *Multi-Objective Optimization Using Evolutionary Algorithms*. vol. 16. Hoboken, NJ, USA: Wiley, 2001.

[51] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[52] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *Proc. NIPS Workshop Deep Learn. Unsupervised Feature Learn.*, 2011, pp. 1–9.

[53] A. Krizhevsky, "Learning multiple layers of features from tiny images," M.S. thesis, Dept. Comput. Sci., Univ. Toronto, Toronto, ON, Canada, 2009.

[54] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002.

[55] K. Deb and S. Agrawal, "A niched-penalty approach for constraint handling in genetic algorithms," in *Proc. Artif. Neural Nets Genetic Algorithms*, 1999, pp. 235–243.

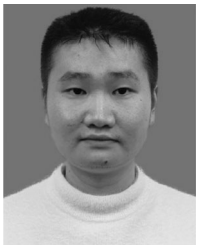
[56] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Completely automated CNN architecture design based on blocks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 4, pp. 1242–1254, Apr. 2020.

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

[58] J. Masci, U. Meier, D. Cireşan, and J. Schmidhuber, "Stacked convolutional auto-encoders for hierarchical feature extraction," in *Proc. Int. Conf. Artif. Neural Netw.*, 2011, pp. 52–59.

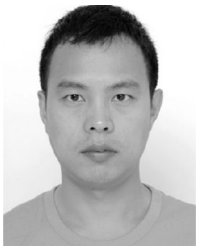
[59] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a

deep network with a local denoising criterion," *J. Mach. Learn. Res.*, vol. 11, pp. 3371–3408, Dec. 2010.
[60] H. Lee, A. Battle, R. Raina, and A. Y. Ng, "Efficient sparse coding algorithms," in *Proc. Adv. Neural Inf. Process. Syst.*, 2007, pp. 801–808.
[61] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, 1995.
[62] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Schölkopf, "Learning with local and global consistency," in *Proc. Adv. Neural Inf. Process. Syst.*, 2004, pp. 321–328.
[63] Y. Sun, G. G. Yen, and Z. Yi, "Evolving unsupervised deep neural networks for learning meaningful representations," *IEEE Trans. Evol. Comput.*, vol. 23, no. 1, pp. 89–103, Feb. 2019.
[64] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.
[65] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 448–456.
[66] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proc. 14th Int. Conf. Artif. Intell. Stat.*, 2011, pp. 315–323.



Xiangru Chen received the B.Eng. degree in computer science and technology from Sichuan University, Chengdu, China, in 2018, where he is currently pursuing the Ph.D. degree with the College of Computer Science.

His current research interests include deep learning and neural architecture search.



Yanan Sun (Member, IEEE) received the Ph.D. degree in engineering from the Sichuan University, Chengdu, China, in 2017.

He is currently a Professor (research) with the College of Computer Science, Sichuan University. He was a Research Fellow with the School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand. His research topics are evolutionary algorithms, deep learning, and evolutionary deep learning.

Prof. Sun is the leading organizer of the First Workshop on Evolutionary Deep Learning, the leading organizer of the Special Session on Evolutionary Deep Learning and Applications in CEC19, and the Founding Chair of the IEEE CIS Task Force on Evolutionary Deep Learning and Applications.

Mengjie Zhang (Fellow, IEEE) received the B.E. and M.E. degrees from the Artificial Intelligence Research Center, Agricultural University of Hebei, Hebei, China, in 1989 and 1992, respectively, and the Ph.D. degree in computer science from RMIT University, Melbourne, VIC, Australia, in 2000.

He is currently Professor of Computer Science, the Head of the Evolutionary Computation Research Group, and the Associate Dean (Research and Innovation) with the Faculty of Engineering. He has published over 350 research papers in refereed international journals and conferences. His current research interests include evolutionary computation, particularly genetic programming, particle swarm optimization, and learning classifier systems with application areas of image analysis, multiobjective optimization, feature selection and reduction, job shop scheduling, and transfer learning.

Prof. Zhang is currently the Chair of IEEE CIS Intelligent Systems and Applications Technical Committee, and the immediate Past 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 IEEE CIS Task Force on Evolutionary Feature Selection and Construction and 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. He is a Fellow of Royal Society of New Zealand and have been a Panel member of the Marsden Fund (New Zealand Government Funding). He is also a member of ACM.

Dezhong Peng (Member, IEEE) received the B.Sc. degree in applied mathematics, and the M.Sc. and Ph.D. degrees in computer software and theory from the University of Electronic Science and Technology of China, Chengdu, China, in 1998, 2001, and 2006, respectively.

From 2001 to 2007, he was with the University of Electronic Science and Technology of China as an Assistant Lecturer and a Lecturer. He was a Postdoctoral Research Fellow with the School of Engineering, Deakin University, Geelong, VIC, Australia, from 2007 to 2009. He is currently a Professor with the Machine Intelligence Laboratory, College of Computer Science, Sichuan University, Chengdu, China. His research interests include blind signal processing and neural networks.