Action Command Encoding for Surrogate-Assisted Neural Architecture Search

Ye Tian, Shichen Peng, Shangshang Yang, Xingyi Zhang, *Senior Member, IEEE*, Kay Chen Tan, *Fellow, IEEE*, and Yaochu Jin, *Fellow, IEEE*

***Abstract*—With the development of neural architecture search, the performance of deep neural networks has been considerably enhanced with less human expertise. While the existing work mainly focuses on the development of optimizers, the design of encoding scheme is still in its infancy. This article thus proposes a novel encoding scheme for neural architecture search, termed action command encoding (ACEncoding). Inspired by the gene expression process, ACEncoding defines several action commands to indicate the addition and clone of layers, connections, and local modules, where an architecture grows from empty according to multiple action commands. ACEncoding provides a compact and rich search space that can be explored by various optimizers efficiently. Furthermore, a surrogate-assisted performance evaluator is tailored for ACEncoding, termed sequence-to-rank (Seq2Rank). By integrating the Seq2Seq model with RankNet, Seq2Rank embeds the variable-length encoding of ACEncoding into a continuous space, and then predicts the rankings of architectures based on the continuous representation. In the experiments, ACEncoding brings improvement to neural architecture search with existing encoding schemes and Seq2Rank shows better accuracy than existing performance evaluators. The neural architectures obtained by ACEncoding and Seq2Rank have competitive test errors and complexities on image classification tasks, and also show high transferability between different data sets.**

***Index Terms*—Encoding scheme, neural architecture search, optimizer, performance evaluator.**

Manuscript received 27 January 2021; revised 30 April 2021; accepted 10 July 2021. Date of publication 25 August 2021; date of current version 9 September 2022. This work was supported in part by the National Key Research and Development Program of China under Grant 2018AAA0100100; in part by the National Natural Science Foundation of China under Grant 61822301, Grant 61876123, and Grant 61906001; in part by the Hong Kong Scholars Program under Grant XJ2019035; in part by the Anhui Provincial Natural Science Foundation under Grant 1908085QF271; in part by the Research Grants Council of the Hong Kong Special Administrative Region, China, under Grant PolyU11202418 and Grant PolyU11209219; and in part by the Royal Society International Exchanges Program under Grant IEC\NSFC\170279. *(Corresponding author: Xingyi Zhang.)*

Ye Tian is with the Key Laboratory of Intelligent Computing and Signal Processing of Ministry of Education, Institutes of Physical Science and Information Technology, Anhui University, Hefei 230601, China (e-mail: field910921@gmail.com).

Shichen Peng and Shangshang Yang are with the Key Laboratory of Intelligent Computing and Signal Processing of Ministry of Education, School of Computer Science and Technology, Anhui University, Hefei 230601, China (e-mail: severus.peng@gmail.com; yangshang0308@gmail.com).

Xingyi Zhang is with the Key Laboratory of Intelligent Computing and Signal Processing of Ministry of Education, School of Artificial Intelligence, Anhui University, Hefei 230601, China (e-mail: xyzhanghust@gmail.com).

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

Yaochu Jin is with the Department of Computer Science, University of Surrey, Guildford GU2 7XH, U.K. (e-mail: yaochu.jin@surrey.ac.uk).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCDS.2021.3107555.

Digital Object Identifier 10.1109/TCDS.2021.3107555

I. Introduction

RECENT advances in computer vision tasks are mostly driven by the success of convolutional neural networks (CNNs), such as image classification [1], image segmentation [2], and object detection [3]. However, the design of an effective CNN for specific tasks is a labor-intensive process requiring repetitive trial and error performed by professionals. To address this challenge, a number of neural architecture search methods were suggested to reduce the human expertise [4]–[8].

Neural architecture search appeared in the 1980s [9] and manifested an explosive growth in recent years [10], where various optimizers have been adopted to automatically determine the desired hyperparameters and topologies of neural networks especially CNNs. For example, NEAT [11] optimizes both the topologies and weights of feedforward neural networks by a genetic algorithm; since it represents each neuron as a node in a graph that is only suitable for small-scale architectures, CoDeepNEAT [12] applies it to the architecture search of deep neural networks by encoding each layer as a node. The work in [13] adopts the tree-structured Parzen estimator (TPE) [14] to optimize the hyperparameters of CNNs, which represents the search space as an expression graph and optimizes the loss function by sampling hyperparameters in the graph. NAS [7] uses a controller recurrent neural network to generate the hyperparameters and topologies of CNNs, and trains the controller by a policy gradient method [15]. NAO [16] embeds the hyperparameters and topologies of CNNs into a continuous space by using an LSTM [17], and then optimizes it through gradient descent (GD).

Although these search methods can generate better architectures than hand-crafted ones, the performance is improved at the expense of computational resources, where many search methods are criticized for the high computational cost of the search process or the large complexity of the found architectures. For the classification task on CIFAR-10 [18], CoDeepNEAT obtains a test error of 7.3% by searching for 174 GPU days, and TPE obtains a test error of 21.2% by searching for six GPU days. NAS consumes 3150 GPU days to find an architecture with 3.65% test error and 37.4 M parameters, and NAO consumes 200 GPU days to find an architecture with 2.11% test error and 128 M parameters. Hence, NAS and

NAO exhibit better performance than CoDeepNEAT and TPE by consuming higher computational cost, while the architecture obtained by NAO has a lower test error but much more parameters than that obtained by NAS. In other words, it is difficult to strike a balance among the search performance, the complexity of architectures, and the computational cost of search process.

There have been many optimizers adopted in neural architecture search, such as evolutionary algorithms (EAs) (e.g., CoDeepNEAT), Bayesian optimization (e.g., TPE), reinforcement learning (RL) (e.g., NAS), and GD (e.g., NAO). However, only a few encoding schemes have been designed to provide an effective search space to be explored by these optimizers. In this article, we would like to take a step forward along this line of research, aiming to improve the performance of the neural architecture search by presenting a novel encoding scheme, termed action command encoding (ACEncoding). Inspired by the gene expression that describes the process of protein synthesis, we rethink the construction of neural network architectures as a similar process to gene expression. In the gene expression process, the protein is composed of a chain of amino acids and each amino acid is specified by three adjoined nucleotides in RNA [19]. While in the proposed ACEncoding, an architecture is constructed by some action commands and each action command is specified by three adjoined integers. By predefining a series of action commands, including adding a layer, adding a connection, and cloning a local module, ACEncoding can use multiple action commands to construct complex architectures in a generative manner. In comparison to many existing methods using direct encoding schemes, such as adjacent matrix [20] and graph [4], ACEncoding is an indirect encoding scheme that is argued to be biologically more plausible, as genetic information encoded in chromosomes cannot specify the whole nervous system [21].

On the other hand, many optimizers evaluate the generated architectures via a training and validation process, which is very time consuming as hundreds or thousands of architectures should be generated during the search process. To reduce the computational cost in performance evaluation, a surrogate-assisted performance evaluator is tailored for the proposed encoding scheme, termed sequence-to-rank (Seq2Rank). Considering that ACEncoding represents architectures as a variable-length sequence of integers within a finite range, Seq2Rank converts the sequence to a fixed-length vector by a Seq2Seq model [22], and then uses a RankNet [23] to predict the rankings of architectures. This way, the performance of most architectures can be estimated by Seq2Rank.

In short, the proposed ACEncoding aims to provide a compact and rich search space enabling optimizers to find architectures with good performance and small complexities, and the proposed Seq2Rank aims to reduce the computational cost consumed by optimizers. In the experiments, the superiority of ACEncoding over some representative encoding schemes is verified on an EA, RL, and GD in searching for the architectures of CNNs, and the effectiveness of Seq2Rank is verified by comparing it with existing performance evaluators

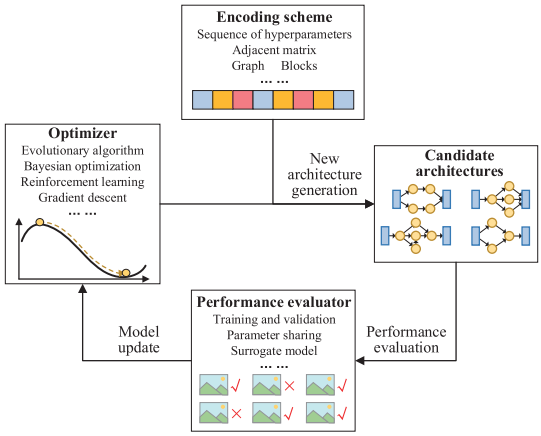


Fig. 1. General procedure of neural architecture search.

for predicting the rankings of architectures. Then, a complete neural architecture search method is established by integrating an EA with ACEncoding and Seq2Rank, which exhibits competitive performance and high transferability on image classification tasks.

The remainder of this article is organized as follows. In Section II, we review existing architecture search methods and classify them according to the optimizers, encoding schemes, and performance evaluators. Then, we describe the proposed ACEncoding in Section III and the proposed Seq2Rank in Section IV. Afterward, we present the experimental results in Section V. Finally, we conclude this article in Section VI.

II. Related Work

As summarized in [10], neural architecture search usually repeats the following three steps until a termination state is reached: 1) generating candidate architectures in the search space; 2) evaluating the performance of these architectures; and 3) updating the model in the optimizer according to these architectures. As illustrated in Fig. 1, three main components should be designed for this aim, including the optimizer for generating candidate architectures, the encoding scheme for defining the search space, and the performance evaluator for evaluating the performance of generated architectures. Table I lists some representative optimizers, encoding schemes, and performance evaluators in existing architecture search methods, which are reviewed in the rest of this section.

A. Optimizers

Neural architecture search is in fact an optimization problem aiming to find the desired architecture for the best training or validation performance; hence, many popular optimizers have been employed for this task. EAs have been applied to the evolution of neural networks as early as in the 1980s [47], [48], which can directly handle the discrete search space without learning any model. The flexibility of EAs enables them to optimize the weights [49], hyperparameters [28], and topologies [20] of neural networks by designing specific crossover and mutation operators. Besides, multiobjective EAs have

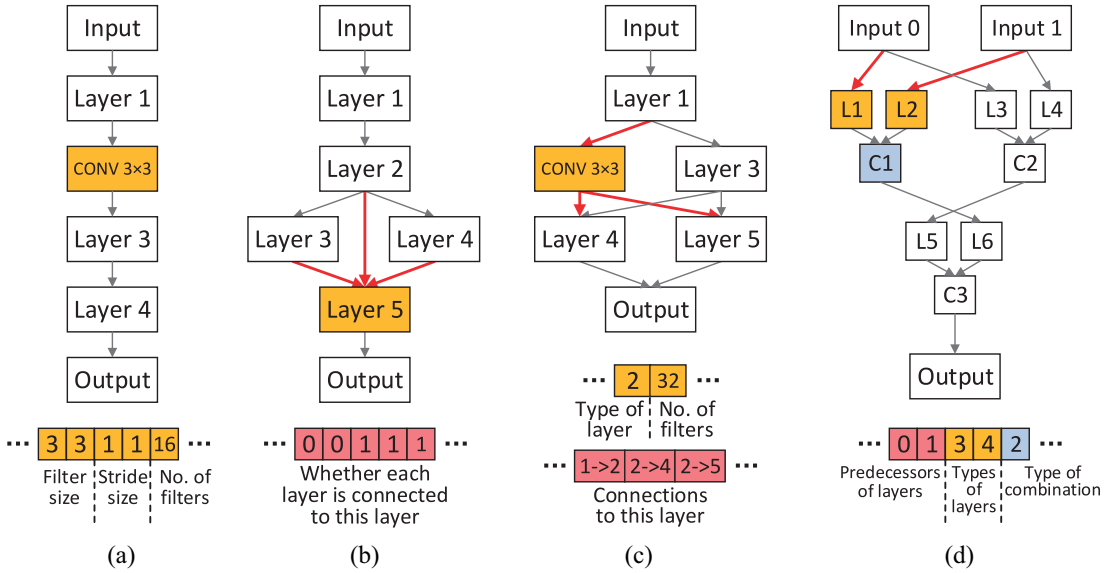


Fig. 2. Four representative encoding schemes used in neural architecture search. (a) Hyperparameter-based encoding (without skip connection) [7]. (b) Adjacent matrix-based encoding [6]. (c) Graph-based encoding [41]. (d) Block-based encoding [5].

TABLE I
REPRESENTATIVE OPTIMIZERS, ENCODING SCHEMES, AND PERFORMANCE EVALUATORS IN ARCHITECTURE SEARCH METHODS

Optimizers	Evolutionary algorithm [4], [6], [12], [20], [24]–[30], Reinforcement learning [5], [7], [31]–[36], Gradient descent [8], [16], [37]–[40], Bayesian optimization [14], [41], Random search [42], [43], Hill climbing search [44], Monte Carlo tree search [45], . . .
Encoding scheme	Sequence of hyperparameters [4], [7], [14], [24], [26], [28], [30]–[32], [34], [36], [38], [44], [45], Blocks [5], [16], [25], [35], [37], [46], Graph [12], [41], Hypernetwork [8], [27], [33], [39], [40], [42], [43], Adjacent matrix [6], [20], Hierarchical graph [29], . . .
Performance evaluator	Training and validation [4]–[7], [12], [20], [25], [26], [28]–[32], [34], [35], [37], [38], [41], [44], [45], Parameter sharing [8], [14], [27], [33], [39], [40], [43], Surrogate model [16], [24], [36], [42], [46], . . .

been adopted to find multiple diverse architectures for ensemble [30], [50]. Bayesian optimization has also demonstrated high effectiveness in neural architecture search, which uses a probabilistic model to sample promising hyperparameters in a continuous space [51], [52]. In order to enable Bayesian optimization to optimize the topologies of neural networks represented in a discrete space, it can cooperate with EAs [41] or represent a topology by hyperparameters [37]. RL makes great contributions to the prevalence of neural architecture search [5], [33]. In RL-based neural architecture search, the search space, encoding, and performance of an architecture are regarded as the action space, agent's action, and agent's rewards, respectively. To determine the agent's policy, some methods adopt a Q-table and update it by Q-learning [31], [34], and some others adopt a recurrent neural network and train it by the policy gradient method [7] or proximal policy optimization [5]. Recently, GD has also shown good potential in neural architecture search [16]. To obtain the gradient direction for generating better architectures, some method maps the discrete sequence into a continuous space through an LSTM [16], some method relaxes the discrete search space to a continuous space by using a mixture model [8], and some method makes the discrete sequence differentiable by concrete distribution and reparametrization [40]. In addition, neural architecture search methods have employed some other optimizers, such as Monte Carlo tree search [45], hill climbing search [44], heuristic search [46], and random search [43].

B. Encoding Schemes

It is generally difficult to claim which optimizer is the best for neural architecture search, as the search performance also highly depends on the encoding scheme. The search space provided by an expressive encoding scheme has a simple and smooth landscape, enabling optimizers to find desired architectures easily. A naive encoding scheme is to encode the hyperparameters of each layer in a sequence, such as the filter size, stride size, number of filters, and type of layer (e.g., convolutional, pooling, or fully connected layer) [7], [34]. Besides, some methods also consider the hyperparameters in the training algorithm, such as the learning rate and momentum [12], [30]. As illustrated in Fig. 2(a), this encoding scheme stacks all the layers into a chain-like structure, which cannot represent many effective architectures having multibranch structures or skip connections, such as GoogLeNet [53], ResNet [54], and DenseNet [55]. Nevertheless, this encoding scheme contains only integers and is easy to be handled by different optimizers.

In order to represent complex architectures, some methods add a binary string to each layer indicating whether each previous layer is connected to it [7]. While the architectures represented by this encoding scheme still have a chain-like structure with skip connections, some other methods suggest a more general way to represent arbitrary topologies, i.e., using the adjacent matrix shown in Fig. 2(b) [6], [20]. However, this

encoding scheme provides a large search space as the length of encoding is the square of the number of layers, and there are many duplicated architectures in the search space since the adjacent matrix may lead to many isolated nodes. In addition, the maximum number of layers and the hyperparameters of each layer should be predefined, as the adjacent matrix has a fixed size and only encodes the connections between layers.

To optimize both the hyperparameters and topologies of neural networks, some methods represent an architecture as a graph, in which the nodes store the hyperparameters of each layer and the edges store the connections between layers [12], [41]. As shown in Fig. 2(c), this encoding scheme encodes a topology in an adjacency list rather than an adjacent matrix; hence, the number of layers is alterable and the encoding is shortened for representing sparse architectures. However, the encoding becomes very long for representing dense architectures. To solve this problem, some methods represent a topology by a subgraph of a predefined hypernetwork [8], [43], and some others compress the encoding by using a hierarchical graph [29] or fractal graph [56].

The block-based encoding has received increasing attention in recent work [5], [46], which has obtained some architectures with better performance than those obtained by other encoding schemes. The block-based encoding represents an architecture as a number of stacked and identical cells. For each cell, it consists of several blocks with each block having two parallel layers and a combination operation as shown in Fig. 2(d). In terms of topology, this encoding scheme only specifies one predecessor for each layer; hence, it can encode complex architectures with multibranch structures in a very short sequence. However, it is apparent that this encoding scheme is inefficient to encode some other architectures, such as those having dense connections or a chain-like structure.

To summarize, most existing encoding schemes encounter various difficulties in representing specific types of architectures. In short, the hyperparameter-based encoding cannot represent multibranch structures, the adjacent matrix-based encoding cannot define the hyperparameters in each layer, the graph-based encoding is inefficient to represent dense architectures, and the block-based encoding is inefficient to represent chain-like structures. Although the architectures represented by block-based encoding have achieved state-of-the-art performance on popular data sets [5], [35], [46], the topology of the block (i.e., two parallel layers and a combination operation) may not be suitable for other deep learning tasks. Hence, the purpose of ACEncoding is to represent as many different architectures as possible, which can flexibly adapt to various tasks. Meanwhile, it encodes each architecture in a sequence of integers, providing a compact and rich search space that can be easily explored by various optimizers.

C. Performance Evaluators

After generating a number of candidate architectures, the performance of them should be evaluated and the better ones are used to update the model used in optimizers (e.g., the controller recurrent neural network used in RL [7] and the LSTM used in GD [16]) for generating better architectures in further iterations. The simplest performance evaluator is to train each neural network from scratch and calculate the classification accuracy on a held-out validation set. However, this procedure is very time consuming for searching CNNs with millions of parameters since thousands of GPU days may be consumed for the whole search process [5], [7]. To alleviate this problem, many methods train each neural network on a small training set [57] or for few epochs [34], [51]; besides, the complexity of the neural networks can be reduced by setting fewer filters and cells [5], [39].

An efficient paradigm for performance evaluation is the parameter sharing mechanism in one-shot architecture search [43], [58], which is usually used together with the hypernetwork-based encoding. The hypernetwork stores the weights of all the potential connections, and the weights of each architecture can be directly obtained from the hypernetwork without training since each architecture is a subgraph of the hypernetwork. In practice, the weights can be optimized before the architecture [43], together with the architecture [8], or alternately with the architecture [27]. As a consequence, the parameter sharing mechanism can considerably reduce the computational cost required by neural architecture search. However, the storage requirement becomes very large for storing a massive number of weights in the GPU memory, which could further restrict the scale of the architectures that can be found. Besides, the weights in the hypernetwork are likely to be better adapted for the found architectures and further reinforce the bias of these architectures [10], which may result in premature convergence of the search process [59].

Another alternative way to estimate the performance of candidate architectures is the utilization of surrogate models, whose input is the encoding of an architecture and output is the validation accuracy. In comparison to the parameter sharing mechanism, the surrogate model is more flexible since it is independent of the optimizers and encoding schemes. But on the other hand, the true validation accuracies of some architectures should be obtained via training and validation to train the surrogate model. Various surrogate models have been adopted in existing neural architecture search methods, such as feedforward neural network [16], LSTM [60], and random forest [24].

In this work, a surrogate-assisted performance evaluator Seq2Rank is tailored for the proposed ACEncoding, which contains two main steps, i.e., architecture embedding and performance prediction. The former step is to extract features from the architecture represented by ACEncoding, which is achieved by converting the variable-length sequence of integers to a fixed-length sequence of real variables via a Seq2Seq model used in neural language processing. The later step is to predict the rankings of the validation accuracies of multiple candidate architectures, by means of training a RankNet used for pairwise sorting. The difference between the proposed Seq2Rank and existing surrogate models mainly lies in two aspects. First, existing surrogate models directly predict the performance of architectures; in contrast, Seq2Rank predicts the rankings of architectures; second, existing surrogate models directly feed the encodings of architectures and train the model in a supervised manner, whereas Seq2Rank first

embeds the encodings into a continuous space by training a Seq2Seq model in an unsupervised manner. The experiments in Section V-B will verify the superiority of Seq2Rank over some existing surrogate models.

III. Proposed Encoding Scheme

A. Action Commands in ACEncoding

The proposed ACEncoding is inspired by the process of gene expression, where each architecture is constructed by a series of action commands and each action command is encoded by three integers. Regarding an architecture as a directed acyclic graph $G = (V, E)$, ACEncoding does not directly encode the layer set V or connection set E like existing encoding schemes; instead, the architecture grows from empty according to the action commands in the encoding. In ACEncoding, the following two basic action commands are defined.

1) Adding a new layer to the architecture.
2) Adding a new connection to the architecture.

Obviously, these two action commands are enough to represent arbitrary topologies, which is the same as the graph-based encoding.

However, the encoding will be very long when representing large architectures with dense connections; hence, the following two action commands are also considered in ACEncoding.

3) Adding a new layer in parallel with an existing one, i.e., the two layers have the same predecessors and successors.
4) Adding a new layer after an existing one, i.e., the successors of the new layer are set to the same as the existing layer, then the successor of the existing layer is set to the new layer.

These two action commands can generate some complex topologies like the widely used block-based topology consisting of two parallel layers [5]; hence, we believe that they are helpful to find desired architectures more efficiently. Note that the combination operation is omitted here, which is fixed to the elementwise addition as suggested in many existing encoding schemes [16], [39], [46]. It is obvious that the graph-based encoding should use many operations (i.e., adding a new layer and adding multiple connections) to achieve each of these two action commands, especially when the existing layer is connected to many other layers.

Moreover, it has been well recognized that many successful architectures, including GoogLeNet, ResNet, and those generated by block-based encoding, are composed of repeated modules [12]. Therefore, the following two action commands are designed to repeat the local modules in the architecture.

5) Cloning an existing local module and locating the two local modules in parallel, i.e., the two modules have the same predecessors and successors.
6) Cloning an existing local module and locating the two local modules in sequence, i.e., the successors of the new module are set to the same as the existing module, then the successor of the existing module is set to the new module.

TABLE II
Codon of Each Action Command Defined in ACEncoding, Where Each Action Command Is Represented by Three Integers. The Variable T Denotes the Type of Layer, I Denotes the Index of an Existing Layer, and L Denotes the Length of Local Module

Codon	Action command
$\{1, T, I\}$	1) Adding a layer T with predecessor I
$\{2, I, I'\}$	2) Adding a connection from I to I'
$\{3, T, I\}$	3) Adding a layer T in parallel with I
$\{4, T, I\}$	4) Adding a layer T after I
$\{5, I, L\}$	5) Cloning a module ($L+1$ layers starting from I) and locating the two modules in parallel
$\{6, I, L\}$	6) Cloning a module ($L+1$ layers starting from I) and locating the two modules in sequence
$\{7, I, L\}$	7) Making a module ($L+1$ layers starting from I) a densely connected block

Here, a local module is defined as a set of layers connected in a chain-like structure with no branching. Besides, another action command is designed to represent the dense block like those in DenseNet.

7) Making an existing local module a densely connected block, i.e., each layer is connected to all the others in the module.

By using these seven action commands, the proposed ACEncoding can represent arbitrary topologies and encode some popular blocks in a short sequence. Since these action commands are designed according to the topologies of some popular architectures, ACEncoding is free to be extended by more action commands for representing the topologies of other effective architectures. Note that each action command does not necessarily appear in the encoding, as an action command will be weeded out during the search process if it always brings a bad performance to the architecture. Besides, it is noteworthy that some basic action commands (e.g., adding a layer or connection) have been adopted in the mutation operator of EA-based search methods [4], [41], but these action commands cannot be achieved by other optimizers. In contrast, ACEncoding encodes these action commands in a sequence that can be explored by different optimizers, where an EA, RL, and GD are employed in the experiments.

B. Decoding Process of ACEncoding

Table II lists the *codon* of each action command, where each action command is represented by three integers with different meanings. The first integer in each codon varies from 1 to 7 denoting the index of an action command, while the second and third integers can be one of the three variables T, I, and L. T denotes the type of layer to be added, which varies from 1 to 10 denoting one of the following ten operations as suggested in [16]: 1) 1×1 convolution; 2) 3×3 convolution; 3) $1 \times 3 + 3 \times 1$ convolution; 4) $1 \times 7 + 7 \times 1$ convolution; 5) 2×2 max pooling; 6) 3×3 max pooling; 7) 5×5 max pooling; 8) 2×2 average pooling; 9) 3×3 average pooling; and 10) 5×5 average pooling. Note that all the operations are tailed by batch normalization. I denotes the index of an existing layer; note that the first two indices denote the input layer and output layer rather than the generated convolutional or pooling layers. L denotes the length of a local module with

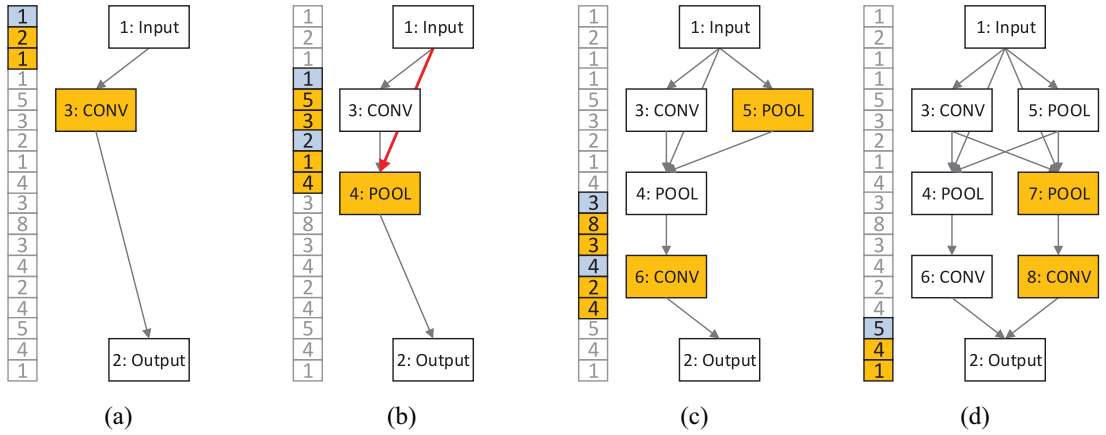


Fig. 3. Illustration of the decoding process of ACEncoding. (a) Adding a convolutional layer. (b) Adding a pooling layer and a connection. (c) Adding a pooling layer and a convolutional layer. (d) Cloning a module having two layers.

chain-like structure, which consists of $L+1$ layers starting from an existing layer.

Fig. 3 depicts an example to illustrate the decoding process of ACEncoding, where the sequence contains six action commands determined by 18 integers. The initial architecture has two inputs and an output, and it grows according to the following action commands to form an architecture having 9 layers and 12 connections: adding a convolutional layer, adding a pooling layer, adding a connection, adding a pooling layer, adding a convolutional layer, and cloning a module. It is worth noting that the action commands do not specify the successor when adding a layer, where the successor is set to the output layer by default [e.g., the connection between the convolutional layer and the output layer shown in Fig. 3(a)]. If this layer is specified as the predecessor of other layers by the subsequent action commands, the default connection between this layer and the output layer is omitted.

In order to provide a finite search space, the variables I and L in the codon are restricted to be less than a maximum value. This maximum value is set to 30 in the experiments, which is enough to represent a relatively large architecture. In addition, since some large values of I or L may lead to invalid action commands, they are revised by specific strategies in the decoding process. First, if the layer index I is larger than the number of existing layers N, I is changed to $\text{mod}(I-1,N)+1$. Second, if the module length L exceeds the maximum length of the module (i.e., reaching the output layer or a layer having multiple branches), L is changed to the maximum length of the module. Third, if the architecture becomes cyclic after adding a connection, the connection is reversed. The procedure of the decoding process of ACEnoding is summarized in Algorithm 1.

Algorithm 1: Decoding Process of ACEncoding

Input: S (an ACEncoding string)
Output: *Net* (a neural architecture)

Net ← Initialize the architecture with an input node and an output node;
for *each codon* ∈ S **do**
  $[action, p_1, p_2]$ ← Determine the action command of *codon* according to Table II;
  if p_1 *denotes the type T of layer* **then**
    $p_1 \leftarrow \text{mod}(p_1 - 1, 10) + 1$;
  else if p_1 *denotes the index I of a layer* **then**
    $p_1 \leftarrow \text{mod}(p_1 - 1, N) + 1$; `//N denotes the number of layers in Net`
  if p_2 *denotes the index I of a layer* **then**
    $p_2 \leftarrow \text{mod}(p_1 - 1, N) + 1$;
  else if p_2 *denotes the length L of a module* **then**
    $p_2 \leftarrow \min\{p_2, M\}$; `//M denotes the maximum length of the module starting from` p_1 `in Net`
  Net ← Update the architecture according to *action*, p_1, and p_2;
  Net ← Flip the connections in *Net* to remove loops;
return *Net*;

C. Remarks

It can be found from Fig. 3 that ACEncoding constructs an architecture with a relatively complex topology by using 18 variables. On the contrary, the adjacent matrix-based encoding requires $[8 \times (7-1)]/2 = 28$ variables to represent the same topology, the graph-based encoding requires 6 variables to store the layers and $12 \times 2 = 24$ variables to store the connections, and the block-based encoding needs to use four blocks determined by $5 \times 4 = 20$ variables to represent a similar architecture with fewer connections. The shorter encoding of ACEncoding is owing to the delicate action commands that inherit the advantages of both graph-based encoding and block-based encoding, which provide a more compact search space that facilitates the optimization of architectures. The tradeoff between the length of encoding and the complexity of the represented architecture is automatically balanced by optimizers in the search process, where a bias toward the first four action commands in Table II leads to more complex architectures and a bias toward the last three action commands makes the encoding shorter. The main difference between ACEncoding and existing encoding schemes is summarized in the following.

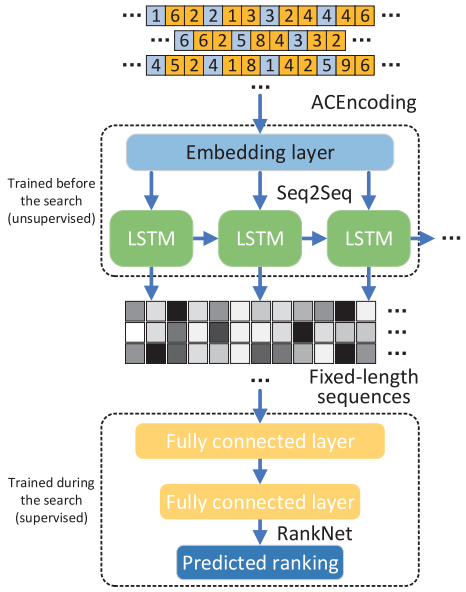


Fig. 4. Illustration of the Seq2Rank architecture.

ACEncoding Versus Hyperparameter-Based Encoding and Adjacent Matrix-Based Encoding: The hyperparameter-based encoding and adjacent matrix-based encoding can only represent the hyperparameters and topologies of architectures, respectively, while ACEncoding can represent both of them. Besides, ACEncoding specifies a topology by using a series of action commands stored in a sequence, which is much shorter than the adjacent matrix.

ACEncoding Versus Graph-Based Encoding: ACEncoding is equivalent to the graph-based encoding when using only the first two action commands, which are enough to represent arbitrary topologies. In contrast, ACEncoding can represent specific complex topologies by the other action commands, which cannot be represented by the graph-based encoding using a short sequence.

ACEncoding Versus Block-Based Encoding: Both ACEncoding and the block-based encoding can represent complex topologies using a short encoding. However, the topologies that can be represented by block-based encoding are limited by the topology of a single block, whereas arbitrary topologies can be represented by ACEncoding.

IV. Proposed Performance Evaluator

A. Architecture Embedding

The general architecture of Seq2Rank is illustrated in Fig. 4. As can be seen, the model for architecture embedding consists of a fully connected layer for word embedding and an LSTM layer used in the Seq2Seq model. To begin with, an architecture represented by ACEncoding is tokenized by regarding the codon of each action command as a token; besides, an <SOS> token and an <EOS> token are attached to the start and end of each sequence, respectively, and an <MOS> token is also required to split the action commands for normal cell and reduction cell if the cell structure is adopted. Then, each token is converted to one-hot representation and fed to the embedding layer to obtain a word vector. Afterward, a Seq2Seq model is trained by maximizing the following conditional probability:

$$p(y_1, \ldots, y_{T'} \mid x_1, \ldots, x_T) = \prod_{t=1}^{T'} p(y_t \mid h_1, \ldots, h_T, y_1, \ldots, y_{t-1}) \quad (1)$$

where $(x_1, \ldots, x_T)$ is an input sequence, $(y_1, \ldots, y_{T'})$ is the target output sequence, and $(h_1, \ldots, h_T)$ is the hidden states of the input sequence. The conditional probability depends on all the hidden states rather than the last one h_T since the attention mechanism is adopted.

In order to convert the variable-length sequence of ACEncoding to a fixed-length sequence, a large number of training samples should be generated to train the above model in an unsupervised manner. That is, the input sequence and target output sequence of each sample are the same architecture represented by ACEncoding. For this aim, we randomly generate three million architectures beforehand to train the model until convergence. Afterward, the fixed-length sequence s can be obtained by conducting mean pooling on all the hidden states $(h_1, \ldots, h_T)$ of an input sequence, i.e.,

$$s = \frac{1}{T} \sum_{i=1}^{T} h_i. \quad (2)$$

B. Performance Prediction

The performance of each architecture is estimated by predicting its ranking among multiple architectures via RankNet. The RankNet adopted here is a feedforward neural network with two hidden layers, where the input includes the fixed-length sequences s_i, s_j of two architectures and the output p_{ij} indicates which one of the two architectures has a better performance. Then, the RankNet can be trained by minimizing the following cross-entropy cost function:

$$C_{ij} = -\bar{p}_{ij} \log p_{ij} - \left(1 - \bar{p}_{ij}\right) \log \left(1 - p_{ij}\right) \quad (3)$$

where $\bar{p}_{ij} \in \{0, 1\}$ is the target label.

In contrast to the Seq2Seq model that can be trained in advance, the RankNet should be trained during the search process in a supervised manner, as the target label has to be obtained by comparing the validation accuracies of two architectures. Given the fixed-length sequences $s_1, \ldots, s_N$ of N architectures and their validation accuracies $r_1, \ldots, r_N$, the training set for RankNet is formed as $\{([s_1, s_2], \bar{p}_{12}), \ldots, ([s_i, s_j], \bar{p}_{ij}), \ldots, ([s_{N-1}, s_N], \bar{p}_{(N-1)N})\}$, where $i < j \leq N$ and there are $N(N-1)/2$ training samples in total. In practice, a number of architectures should be evaluated via training and validation at the beginning of the search process, and then they are used to train the RankNet. During the search process, the performance of most candidate architectures is evaluated via RankNet, while some promising ones can be evaluated via training and validation and used to update the RankNet to enhance its ranking accuracy in the desired search regions.

V. Experiments

The experiments in this section consist of four parts. First, the effectiveness of the proposed ACEncoding is verified by comparing it with three existing encoding schemes on an EA, RL, and GD. Second, the performance of the proposed Seq2Rank is verified by comparing it with two existing surrogate-assisted performance evaluators. Third, the performance of ACEncoding and Seq2Rank is quantified on the popular CIFAR-10 data set [18] and compared to the state-of-the-art performance. Finally, the performance of ACEncoding and Seq2Rank is verified on CIFAR-100 and Fashion-MNIST [61]. All our experiments are conducted in PyTorch [62] with a single NVIDIA Tesla V100 GPU. The source code of ACEncoding and Seq2Rank is available at https://github.com/anonymone/ACE-NAS.

A. Performance Verification of ACEncoding

The effectiveness of the proposed ACEncoding is verified by integrating it with NSGA-Net [20], MetaQNN [34], and NAO [16], where NSGA-Net is based on an EA and the adjacent matrix-based encoding, MetaQNN is based on RL and the hyperparameter-based encoding, and NAO is based on GD and the block-based encoding. Hence, it is reasonable to adopt these three methods to compare ACEncoding with other encoding schemes on different optimizers. The parameter settings mainly follow those in the original papers of the three methods, the details of which are given in the following.

Structure of CNN: The widely used cell structure is adopted in ACEncoding to decrease the search space. Specifically, a normal cell and a reduction cell are encoded by ACEncoding independently, and the whole CNN consists of three normal cells and two reduction cells stacked alternately. For the hyperparameters in the convolutional and pooling layers, the filter size is determined by the encoding, the number of filters is always set to 16, and the stride size is set to 2 for the layers directly connected to the reduction cell's inputs and 1 for all the other layers.

Evolutionary Algorithm + ACEncoding: NSGA-Net is a genetic algorithm aiming to minimize both the validation error and the complexity of the architecture, which generates candidate architectures by a homogeneous crossover operator and a bit-flipping mutation operator. When integrating with ACEncoding, the optimizer randomly initializes a population with 40 architectures having 10–20 action commands for each cell, and then evolves the population for 25 generations to generate 1000 architectures in total. Since the crossover and mutation operators in NSGA-Net are tailored for the binary adjacent matrix, new crossover and mutation operators are designed for ACEncoding. To be specific, the crossover operator randomly selects an anchor point from the encodings of two parent architectures, and exchanges the action commands after the anchor point of the two encodings to generate two offspring architectures. The offspring architectures are then mutated by one of three operations, including adding a random action command, randomly deleting an action command, and randomly changing the variables of an action command. The crossover probability is 1 and the mutation probability is 0.1.

Reinforcement Learning + ACEncoding: MetaQNN iteratively generates architectures according to a Q-table and updates the Q-table according to the performance of the generated architectures via Q-learning. When integrating with ACEncoding, each action command is regarded as a state and the Q-table stores the value $Q(s_i, s_j)$ denoting the reward of selecting action command s_j after action command s_i. To generate an architecture, the agent sequentially selects action commands via the ϵ-greedy strategy according to the Q-table, until the termination state is selected or 20 action commands have been selected. This procedure repeats twice to generate the normal cell and reduction cell of an architecture, and the generated architecture is evaluated and stored in a replay dictionary for periodically updating the Q-values between action commands in the Q-table. The Q-learning rate is set to 0.1, the discount factor is set to 1 (i.e., there is no overprioritize short-term rewards), and the parameter ϵ is set to 1, 0.9, 0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, and 0.1 when 200, 400, 500, 600, 700, 800, 850, 900, 950, and 1000 architectures have been evaluated.

Gradient Descent + ACEncoding: NAO adopts an encoder LSTM to embed the architecture into a continuous space, and takes the continuous representation as input to predict its performance by a feedforward neural network. Thus, better continuous representation can be obtained by moving the input along the gradient direction induced by the performance, and the updated continuous representation is fed into a decoder LSTM to generate a new architecture. When integrating with ACEncoding, the optimizer randomly initializes 600 architectures having 20 action commands for each cell, as the length of the decoder is fixed in NAO. Then, these architectures are evaluated and used to train the two LSTMs and the feedforward neural network jointly. Afterward, 200 new architectures are generated by performing GD on the best 200 architectures. The last step repeats twice to form 1000 architectures generated in total. The sizes of the embedding layer, the hidden layer of the LSTMs, and the hidden layer of the feedforward neural network are set to 64, 128, and 200, respectively, and the models are trained using Adam [63] for 1000 epochs with a learning rate of 0.001.

Performance Evaluation Strategy: All the above three methods generate 1000 architectures during the whole search process. Each architecture is evaluated via training and validation without using parameter sharing or surrogate model, where the 50 000 images in CIFAR-10 are randomly split into a training set of 45 000 samples and a validation set of 5000 samples. Each architecture is trained by SGD with a batch size of 128, a momentum of 0.9, and a learning rate ranging from 0.1 to 0.001 following a single period cosine schedule. The training samples are normalized and randomly augmented by several data preprocessing techniques, including random crop (i.e., padding each image by 4 pixels and cropping a random 32×32 patch), horizontal flipping [54], and cutout (with a mask length of 16) [64]. Besides, the scheduled path dropout (with a probability of 0.2) [5] and an auxiliary classifier (appended to the architecture at 2/3 depth

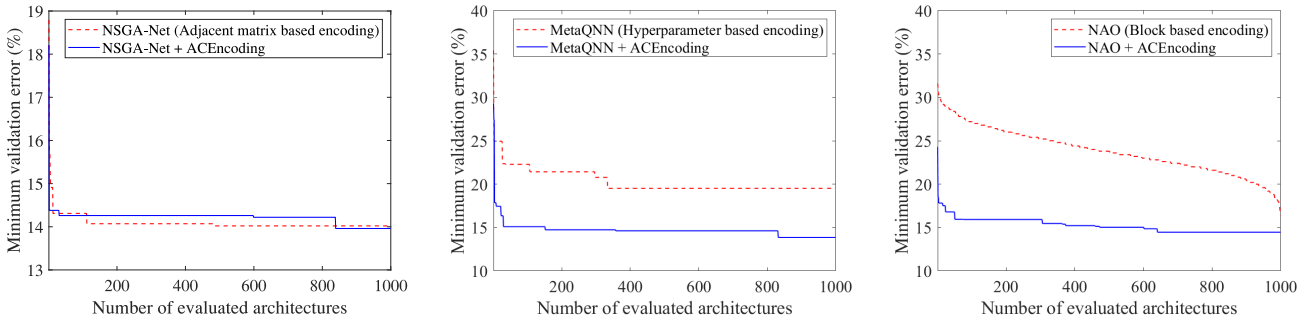


Fig. 5. Minimum validation error during the search process of the original NSGA-Net (left), MetaQNN (middle), and NAO (right) and those integrated with the proposed ACEncoding on CIFAR-10.

with a constant factor of 0.4) [65] are adopted in the training process.

Fig. 5 plots the minimum validation error during the search process of the original NSGA-Net, MetaQNN, and NAO and those integrated with ACEncoding. It can be found that the proposed ACEncoding is competitive to the adjacent matrix-based encoding on NSGA-Net, while ACEncoding converges obviously faster than the hyperparameter-based encoding on MetaQNN and the block-based encoding on NAO. Therefore, it can be concluded that ACEncoding is suitable for different optimizers, including EAs, RL, and GD, and it is competitive to or more effective than existing encoding schemes.

B. Performance Verification of Seq2Rank

The proposed performance evaluator Seq2Rank is compared to two existing surrogate models for predicting the performance of architectures, namely, E2EPP [24] and Peephole [60]. E2EPP is an end-to-end performance predictor based on random forest [66], which is effective to a limited number of training samples and can directly handle the discrete variables without embedding. Peephole holds a similar pipeline to the proposed Seq2Rank, which embeds the architecture by an LSTM with embedding layer and predicts the performance by a feedforward neural network. However, Peephole concatenates all the hidden states but not using mean pooling or attention mechanism, and it predicts the validation accuracy rather than the ranking of each architecture. More importantly, Seq2Rank uses the Seq2Seq model to learn a continuous representation in an unsupervised manner, while Peephole directly trains the whole model in a supervised manner. For the models in Peephole and Seq2Rank, the sizes of the embedding layer, the hidden layer of the LSTM, and the two hidden layers of the feedforward neural network are set to 62, 120, 60, and 30, respectively, and the models are trained using SGD for 25 epochs with a learning rate of 0.1.

A number of 10 000 architectures represented by ACEncoding are randomly generated to form the data set, where different numbers of samples are used as training samples and the rest are used as test samples. Table III lists the performance of E2EPP, Peephole, and Seq2Rank in terms of Kendall's Tau metric, which is calculated by [68]

$$KTau = 2 \times \frac{M}{C_N^2} - 1 \tag{4}$$

TABLE III
KENDALL'S TAU VALUES OF E2EPP, PEEPHOLE, AND THE PROPOSED SEQ2RANK FOR PREDICTING THE RANKINGS OF THE ARCHITECTURES REPRESENTED BY ACENCODING

# training samples	E2EPP	Peephole	Seq2Rank
1,000	**0.2595**	0.1636	0.1996
3,000	0.1168	0.1586	**0.5157**
5,000	0.0573	0.0632	**0.6260**
7,000	0.0536	0.2253	**0.5366**
9,000	0.0304	0.3402	**0.6754**

where N is the total number of test samples and M is the concordant pair that the predicted ranking is correct. Obviously, the value of Kendall's Tau ranges from -1 to 1 and a higher value corresponds to a higher accuracy. It can be found from the table that Peephole always has a worse performance than Seq2Rank, and E2EPP outperforms Seq2Rank only with 1000 training samples due to the superiority of random forest on small training sets. To summarize, it can be verified that Seq2Rank is more suitable for predicting the performance of architectures represented by ACEncoding than some existing surrogate-assisted performance evaluators.

C. Performance Comparison on CIFAR-10

To compare the performance of the proposed ACEncoding and Seq2Rank to existing neural architecture search methods, the architecture having the lowest validation error among the results obtained by each optimizer equipped with ACEncoding in Fig. 5 is selected. These architectures are used to construct larger CNNs, where the number of cells is 20 (i.e., six normal cells stacked between each two reduction cells) and the number of filters is set to 36. These CNNs are trained for 600 epochs with the same training strategy and data preprocessing techniques, and the classification error on the 10 000 test samples of CIFAR-10 is recorded.

Moreover, it can be found from Fig. 5 that the NSGA-Net obtains lower validation error than MetaQNN and NAO at last; hence, a neural architecture search method is established by integrating NSGA-Net with both ACEncoding and Seq2Rank. This search method trains and validates 40 architectures at the beginning and uses them to train the Seq2Rank, and then evolves the population for 25 000 generations. The newly generated architectures are evaluated via training and validation at

TABLE IV
COMPARISON OF PERFORMANCE BETWEEN THE CNNS FOUND BY HUMAN EXPERTS AND NEURAL ARCHITECTURE SEARCH METHODS ON CIFAR-10

Architecture	Test error (%)	Number of parameters (M)	Search cost (GPU days)	Optimizer
Wide ResNet ($k = 2$) [67]	5.33	2.2	-	manual
Wide ResNet ($k = 4$)	4.97	8.9	-	manual
Wide ResNet ($k = 10$)	4.17	36.5	-	manual
DenseNet ($k = 12$) [55]	4.10	7.0	-	manual
DenseNet ($k = 24$)	3.74	27.2	-	manual
DenseNet-BC ($k = 40$)	3.46	25.6	-	manual
AmoeabaNet-A [25]	3.34	3.2	3150	EA
AmoeabaNet-B	3.37	2.8	3150	EA
Hier-EA [29]	3.75	15.7	300	EA
Hier-EA (more filters)	3.63	-	300	EA
Large-Scale Evolution [4]	5.40	5.4	2666	EA
NSGA-Net [20]	2.75	3.3	4	EA
NSGA-Net (more filters)	2.50	26.8	4	EA
BlockQNN-S [31]	4.38	6.1	96	RL
BlockQNN-S (more filters)	3.54	39.8	96	RL
ENAS [33]	2.89	4.6	0.45	RL
MetaQNN [34]	6.92	11.2	80	RL
NAS v3 [7]	4.47	7.1	3150	RL
NAS v3 (more filters)	3.65	37.4	3150	RL
NASNet-A [5]	2.65	3.3	2000	RL
NASNet-A (more filters)	2.40	27.6	2000	RL
NASNet-B	3.73	2.6	2000	RL
NASNet-C	3.59	3.1	2000	RL
DARTS (1st order) [8]	3.00	3.3	1.5	GD
DARTS (2nd order)	2.76	3.3	4	GD
NAONet [16]	3.18	10.6	200	GD
NAONet (more filters)	2.98	28.6	200	GD
NAONet + WS	3.53	2.5	0.3	GD
BOHB [37]	3.18	27.6	32	Bayesian optimization
NASBOT [41]	12.09	-	40	Bayesian optimization
PNASNet [46]	3.41	3.2	225	Heuristic search
EA + ACEncoding	2.80	3.0	7	EA
RL + ACEncoding	3.16	2.9	8.3	RL
GD + ACEncoding	3.03	3.2	12.4	GD
EA + ACEncoding + Seq2Rank	2.90	2.4	1.5	EA
EA + ACEncoding + Seq2Rank (more filters)	2.32	9.8	1.5	EA

the 5000th, 10 000th, 15 000th, 20 000th, and 25 000th generation, and they are used to fine-tune the Seq2Rank; besides, the newly generated architectures are evaluated via the Seq2Rank at all the other generations. Although this method evolves the population for more generations than the method without Seq2Rank, the former is in fact much faster due to the use of the surrogate model. The procedure of NSGA-Net with ACEncoding and Seq2Rank is summarized in Algorithm 2.

Table IV presents the test error of 31 CNNs found by human experts and 17 neural architecture search methods. It can be observed that the CNNs obtained by ACEncoding have 2.80%–3.16% test errors and 2.4M to 3.2M parameters, which are quite competitive to the other CNNs listed in the table. As for the EA with both ACEncoding and Seq2Rank, the test error of the obtained CNN is slightly higher than the CNN obtained by the EA with only ACEncoding; in contrast, the search cost can decrease to 1.5 GPU days from 7 GPU days. Furthermore, the CNN obtained by the EA with ACEncoding and Seq2Rank is enlarged by setting the number of filters to 96, which can lead to a test error of 2.32%, better than all the other CNNs in the table.

Fig. 6 depicts the test error and number of parameters of the CNNs listed in Table IV. It can be found that there are four CNNs having the best tradeoffs between test error and complexity, where two of them are obtained by the EA with ACEncoding and Seq2Rank, one of them is obtained by NASNet, and the remaining one is wide ResNet. It should be noted that NASNet obtains the CNN by searching for 2000 GPU days, whereas the EA with ACEncoding and Seq2Rank consumes only 1.5 GPU days. Although the wide ResNet is manually designed with no search cost, its test error is much higher than the other three CNNs. Besides, Fig. 7 plots the test error and search cost of the CNNs, where the best tradeoff

Algorithm 2: Procedure of EA With ACEncoding and Seq2Rank

Input: *pop* (population size), *Gen* (maximum number of generations), *GenList* (indexes of generations for real evaluation)
Output: P (Set of optimal neural architectures)

```
Seq2Seq ← Train a Seq2Seq model in advance;
P ← Randomly initialize pop solutions; //Each solution is an architecture
P ← Decode and evaluate the solutions in P via training and validation;
Seq2Rank ← Train a Seq2Rank model based on Seq2Seq and P;
for g = 1 to Gen do
    P' ← Select parents from P via the mating selection in [?];
    P_new ← Generate an offspring population based on P' via the proposed crossover and mutation;
    if g ∈ GenList then
        P_new ← Decode and evaluate the solutions in P_new via training and validation;
        Seq2Rank ← Fine-tune Seq2Rank based on P_new;
    else
        P_new ← Evaluate the solutions in P_new via Seq2Rank;
    P ← P ∪ P_new;
    P ← Select half the solutions from P via the environmental selection in [?];
return P;
```

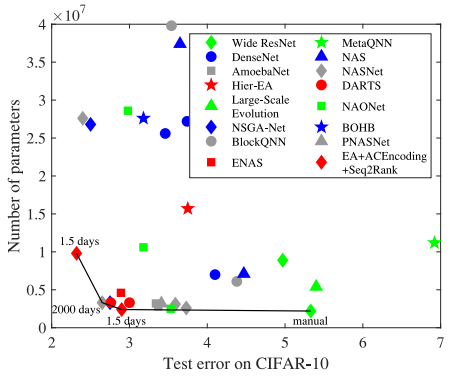


Fig. 6. Test error and number of parameters of the CNNs found by human experts and neural architecture search methods on CIFAR-10, where the CNNs having the best tradeoffs between test error and complexity are connected.

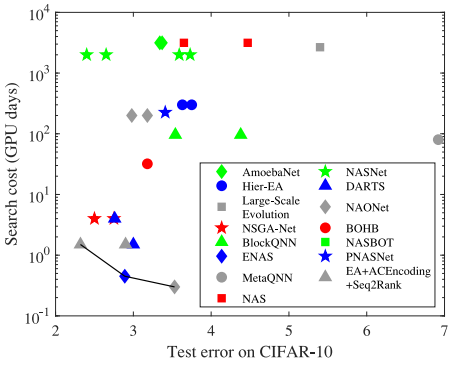


Fig. 7. Test error and search cost of the CNNs found by neural architecture search methods on CIFAR-10, where the CNNs having the best tradeoffs between test error and search cost are connected.

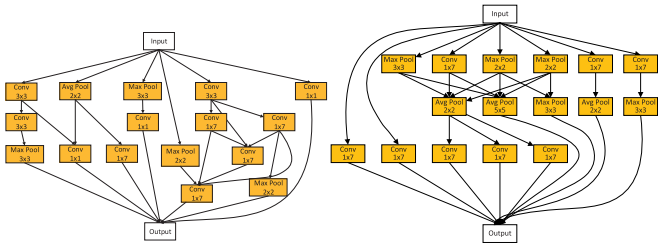


Fig. 8. Architectures of the normal cell (left) and reduction cell (right) found by the EA with ACEncoding and Seq2Rank.

CNN found by the EA with ACEncoding and Seq2Rank has larger search cost but lower test error than the other tradeoff CNNs. In short, the proposed ACEncoding and Seq2Rank can strike a balance among the search performance, the complexity of architectures, and the computational cost of search process.

For a visual observation, Fig. 8 depicts the architectures of the normal cell and reduction cell found by the EA with ACEncoding and Seq2Rank. It can be seen that the architectures have more than ten layers with many connections. For adjacent matrix-based encoding and graph-based encoding, they require a long sequence of variables to represent such complex architectures. Besides, the block-based encoding is almost impossible to represent these architectures by the block-based topology.

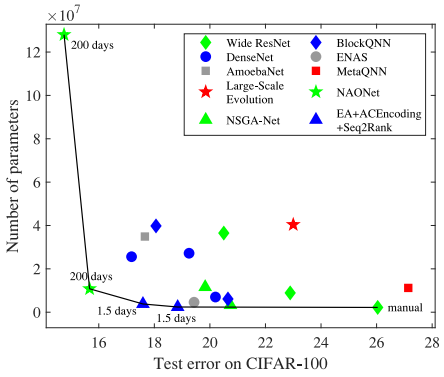


Fig. 9. Test error and number of parameters of the CNNs found by human experts and neural architecture search methods on CIFAR-100, where the CNNs having the best tradeoffs between test error and complexity are connected.

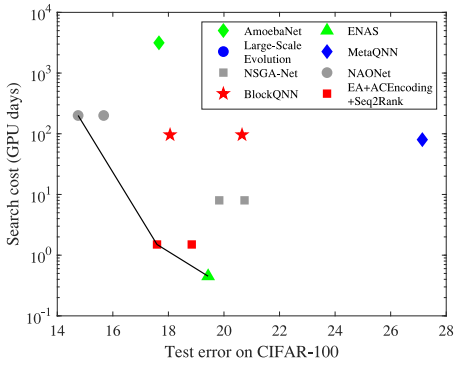


Fig. 10. Test error and search cost of the CNNs found by neural architecture search methods on CIFAR-100, where the CNNs having the best tradeoffs between test error and search cost are connected.

D. Performance Comparison on CIFAR-100

The transferability of the architectures found by ACEncoding and Seq2Rank is then verified on CIFAR-100 [18], where the same CNNs are trained by the 50 000 training samples of CIFAR-100 and tested on the 10 000 test samples. According to the results listed in Table V, the CNNs obtained by ACEncoding and Seq2Rank also have competitive test errors and complexities to the others. Besides, the CNN generated based on the second best architecture found by the EA with ACEncoding and Seq2Rank is evaluated, which has higher complexity but lower test error than the other CNNs obtained by ACEncoding and Seq2Rank.

Fig. 9 depicts the test error and number of parameters of the CNNs listed in Table IV. Obviously, the EA with ACEncoding and Seq2Rank can also obtain two tradeoff CNNs, and NAO can obtain one tradeoff CNN. Although the CNN obtained by NAO has the lowest test error, it is found by searching for 200 GPU days, which is much higher than the 1.5 GPU days consumed by ACEncoding and Seq2Rank. As can be further observed from Fig. 10, the CNN found by the EA with ACEncoding and Seq2Rank strikes good balance between test error and search cost. In short, the architectures found by ACEncoding and Seq2Rank are transferable to more complex data sets.

TABLE V
COMPARISON OF PERFORMANCE BETWEEN THE ARCHITECTURES FOUND BY HUMAN EXPERTS AND NEURAL ARCHITECTURE SEARCH METHODS ON CIFAR-100

Architecture	Test error (%)	Number of parameters (M)	Search cost (GPU days)	Optimizer
Wide ResNet ($k = 2$) [67]	26.04	2.2	-	manual
Wide ResNet ($k = 4$)	22.89	8.9	-	manual
Wide ResNet ($k = 10$)	20.50	36.5	-	manual
DenseNet ($k = 12$) [55]	20.20	7.0	-	manual
DenseNet ($k = 24$)	19.25	27.2	-	manual
DenseNet-BC ($k = 40$)	17.18	25.6	-	manual
AmoeabaNet-B [25]	17.66	34.9	3150	EA
Large-Scale Evolution [4]	23.00	40.4	2666	EA
NSGA-Net [20]	20.74	3.3	8	EA
NSGA-Net (more filters)	19.83	11.6	8	EA
BlockQNN-S [31]	20.65	6.1	96	RL
BlockQNN-S (more filters)	18.06	39.8	96	RL
ENAS [33]	19.43	4.6	0.45	RL
MetaQNN [34]	27.14	11.2	80	RL
NAONet [16]	15.67	10.8	200	GD
NAONet (more filters)	14.75	128.0	200	GD
EA + ACEncoding	18.65	3.0	7	EA
RL + ACEncoding	19.65	2.9	8.3	RL
GD + ACEncoding	18.69	3.2	12.4	GD
EA + ACEncoding + Seq2Rank	18.84	2.4	1.5	EA
EA + ACEncoding + Seq2Rank (second best in search)	17.59	9.8	1.5	EA

TABLE VI
COMPARISON OF PERFORMANCE BETWEEN THE ARCHITECTURES FOUND BY HUMAN EXPERTS AND NEURAL ARCHITECTURE SEARCH METHODS ON FASHION-MNIST

Architecture	Test error (%)	Number of parameters (M)	Optimizer
VGG16 [69]	6.50	26.0	manual
GoogLeNet [53]	6.30	101.0	manual
ResNet ($k = 18$) [54]	5.10	33.2	manual
Wide ResNet ($k = 10$) [67]	4.01	36.5	manual
DENSER [70]	4.89	-	EA
EvoCNN [28]	7.28	6.5	EA
NSGANet [20]	4.07	3.4	EA
MetaQNN [34]	10.02	12.5	RL
NAONet [16]	3.50	11.5	GD
DARTS (2nd order) [8]	3.94	3.4	GD
EA + ACEncoding +Seq2Rank	3.91	2.2	EA

E. Performance Comparison on Fashion-MNIST

Finally, Table VI shows the test error and number of parameters of the architectures found by human experts and neural architecture search methods on Fashion-MNIST, which is an image classification data set containing 60 000 training samples and 10 000 test samples [61]. It is obvious that the test error of the CNN obtained by the EA with ACEncoding and Seq2Rank is lower than all the other CNNs besides the one found by NAONet. Besides, the CNN found by NAONet has a significantly larger complexity than the CNN found by the proposed method. The experimental results further demonstrate that ACEncoding and Seq2Rank can hold a good balance between the performance and complexity of architectures.

VI. CONCLUSION AND FUTURE WORK

We have presented a novel encoding scheme and a performance evaluator for neural architecture search. The proposed encoding scheme ACEncoding defines a set of action commands and encodes each architecture in a sequence consisting of the codons of multiple action commands. The proposed encoding scheme inherits the advantages of existing encoding schemes that can represent complex architectures as a relatively short sequence, and has shown better effectiveness than some popular encoding schemes in EAs, RL, and GD. The proposed performance evaluator Seq2Rank adopts a Seq2Seq model to convert the variable-length encoding of ACEncoding into a fixed-length sequence, and then predicts the ranking of the architecture determined by the encoding via RankNet. The proposed performance evaluator has been empirically verified to be better than some existing surrogate models in predicting the performance of architectures represented by ACEncoding.

In the experiments, the EA equipped with ACEncoding and Seq2Rank has exhibited competitive performance and efficiency to some state-of-the-art neural architecture search methods on popular image classification data sets. In the future, the effectiveness of the proposed ACEncoding and Seq2Rank should be verified on larger data sets such as ImageNet [71] and other computer vision problems such as object detection [72]. Due to the flexibility of ACEncoding, it is also suitable to customize new action commands to apply it to language modeling tasks [73].

REFERENCES

[1] L. Liu, C. Shen, and A. van den Hengel, "Cross-convolutional-layer pooling for image recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 11, pp. 2305–2313, Nov. 2017.

[2] V. Badrinarayanan, A. Kendall, and R. Cipolla, "SegNet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 12, pp. 2481–2495, Dec. 2017.

[3] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 1137–1149, Jun. 2017.

[4] E. Real *et al.*, "Large-scale evolution of image classifiers," in *Proc. 34th Int. Conf. Mach. Learn.*, vol. 6, 2017, pp. 4429–4446.

[5] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 8697–8710.

[6] L. Xie and A. Yuille, "Genetic CNN," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, pp. 1388–1397.

[7] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *Proc. 5th Int. Conf. Learn. Represent.*, 2017, pp. 1–16.

[8] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," in *Proc. 7th Int. Conf. Learn. Represent.*, 2019, pp. 1–13.

[9] X. Yao, "A review of evolutionary artificial neural networks," *Int. J. Intell. Syst.*, vol. 8, no. 4, pp. 539–567, 1993.

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

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

[12] R. Miikkulainen *et al.*, "Evolving deep neural networks," in *Artificial Intelligence in the Age of Neural Networks and Brain Computing*. Oxford, U.K.: Academic, 2019, pp. 293–312.

[13] J. Bergstra, D. Yamins, and D. D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in *Proc. 30th Int. Conf. Mach. Learn.*, 2013, pp. 115–123.

[14] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Proc. 2011 Adv. Neural Inf. Process. Syst.*, 2011, pp. 2546–2554.

[15] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 229–256, 1992.

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

[17] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

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

[19] F. H. C. Crick, L. Barnett, S. Brenner, and R. J. Watts-Tobin, "General nature of the genetic code for proteins," *Nature*, vol. 192, no. 4809, pp. 1227–1232, 1961.

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

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

[22] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 3104–3112.

[23] C. Burges *et al.*, "Learning to rank using gradient descent," in *Proc. 22nd Int. Conf. Mach. Learn.*, 2005, pp. 89–96.

[24] Y. Sun, H. Wang, B. Xue, Y. Jin, G. G. Yen, and M. Zhang, "Surrogate-assisted evolutionary deep learning using an end-to-end random forest-based performance predictor," *IEEE Trans. Evol. Comput.*, vol. 24, no. 2, pp. 350–364, Apr. 2020.

[25] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proc. AAAI Conf. Artif. Intell.*, 2019, pp. 4780–4789.

[26] M. Suganuma, M. Ozay, and T. Okatani, "Exploiting the potential of standard convolutional autoencoders for image restoration by evolutionary search," in *Proc. Mach. Learn. Res.*, vol. 80, 2018, pp. 4771–4780.

[27] Z. Yang *et al.*, "CARS: Continuous evolution for efficient neural architecture search," in *Proc. IEEE Comput. Society Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 1826–1835.

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

[29] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical representations for efficient architecture search," in *Proc. 6th Int. Conf. Learn. Represent.*, 2018, pp. 1–13.

[30] C. Zhang, P. Lim, A. K. Qin, and K. C. Tan, "Multiobjective deep belief networks ensemble for remaining useful life estimation in prognostics," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 10, pp. 2306–2318, Oct. 2017.

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

[32] 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.

[33] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," in *Proc. 35th Int. Conf. Mach. Learn.*, 2018, pp. 6522–6531.

[34] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," in *Proc. 5th Int. Conf. Learn. Represent.*, 2019, pp. 1–18.

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

[36] Y. Fan, X. Tang, G. Zhou, and J. Shen, "EfficientAutoGAN: Predicting the rewards in reinforcement-based neural architecture search for generative adversarial networks," *IEEE Trans. Cogn. Devel. Syst.*, early access, Nov. 26, 2020, doi: 10.1109/TCDS.2020.3040796.

[37] A. Zela, A. Klein, S. Falkner, and F. Hutter, "Towards automated deep learning: Efficient joint neural architecture and hyperparameter search," 2018. [Online]. Available: arXiv:1807.06906.

[38] R. Shin, C. Packer, and D. Song, "Differentiable neural network architecture search," in *Proc. 6th Int. Conf. Learn. Represent.*, 2018, pp. 1–4.

[39] X. Dong and Y. Yang, "One-shot neural architecture search via self-evaluated template network," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2019, pp. 3681–3690.

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

[41] 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.

[42] C. Zhang, M. Ren, and R. Urtasun, "Graph hypernetworks for neural architecture search," 2018. [Online]. Available: arXiv:1810.05749.

[43] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "SMASH: One-shot model architecture search through hypernetworks," in *Proc. Int. Conf. Learn. Represent.*, 2018, pp. 1–21.

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

[45] M. Wistuba, "Finding competitive network architectures within a day using UCT," 2017. [Online]. Available: arXiv:1712.07420.

[46] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L. J. Li, and K. Murphy, "Progressive neural architecture search," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 19–34.

[47] G. F. Miller, S. U. Todd, and P. M. Hegde, "Designing neural networks using genetic algorithms," in *Proc. 3rd Int. Conf. Genet. Algorithms Their Appl.*, 1989, pp. 379–384.

[48] S. A. Harp, T. Samad, and A. Guha, "Toward the genetic synthesis of neural networks," in *Proc. 3rd Int. Conf. Genet. Algorithms Appl.*, 1989, pp. 360–369.

[49] 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.

[50] Y. Jin and B. Sendhoff, "Pareto-based multiobjective machine learning: An overview and case studies," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 38, no. 3, pp. 397–415, May 2008.

[51] T. Domhan, J. T. Springenberg, and F. Hutter, "Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves," in *Proc. 24th Int. Joint Conf. Artif. Intell.*, 2015, pp. 3460–3468.

[52] H. Mendoza, M. Klein, A. Feurer, J. Springenberg, and F. Hutter, "Towards automatically-tuned neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 58–65.

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

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

[55] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. 30th IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 2261–2269.

[56] J. W. L. Merrill and R. F. Port, "Fractally configured neural networks," *Neural Netw.*, vol. 4, no. 1, pp. 53–60, 1991.

[57] A. Klein, S. Falkner, S. Bartels, P. Hennig, and F. Hutter, "Fast Bayesian optimization of machine learning hyperparameters on large datasets," in *Proc. 20th Int. Conf. Artif. Intell. Stat.*, vol. 54, 2017, pp. 528–536.

[58] S. Saxena and J. Verbeek, "Convolutional neural fabrics," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 4053–4061.

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

[60] B. Deng, J. Yan, and D. Lin, "Peephole: Predicting network performance before training," 2017. [Online]. Available: arXiv:1712.03351.

[61] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms," 2017. [Online]. Available: arXiv:1708.07747.
[62] A. Paszke *et al.*, "Automatic differentiation in PyTorch," in *Proc. NIPS Workshop*, 2017, pp. 1–4.
[63] D. P. Kingma and L. J. Ba, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. Learn. Represent.*, 2015, pp. 1–15.
[64] T. DeVries and G. W. Taylor, "Improved regularization of convolutional neural networks with cutout," 2017. [Online]. Available: arXiv:1708.04552.
[65] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 2818–2826.
[66] T. K. Ho, "Random decision forests," in *Proc. 3rd Int. Conf. Document Anal. Recognit.*, 1995, pp. 278–282.
[67] S. Zagoruyko and N. Komodakis, "Wide residual networks," 2016. [Online]. Available: arXiv: 1605.07146.
[68] P. K. Sen, "Estimates of the regression coefficient based on Kendall's Tau," *J. Amer. Stat. Assoc.*, vol. 63, no. 324, pp. 1379–1389, 1968.
[69] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. Int. Conf. Learn. Represent.*, 2015, pp. 1–14.
[70] F. Assunção, N. Lourenço, P. Machado, and B. Ribeiro, "DENSER: Deep evolutionary network structured representation," *Genet. Program. Evol. Mach.*, vol. 20, no. 1, pp. 5–35, 2019.
[71] O. Russakovsky *et al.*, "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, 2015.
[72] J. Huang *et al.*, "Speed/accuracy trade-offs for modern convolutional object detectors," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 7310–7311.
[73] G. Melis, C. Dyer, and P. Blunsom, "On the state of the art of evaluation in neural language models," 2017. [Online]. Available: arXiv:1707.05589.

Ye Tian received the B.Sc., M.Sc., and Ph.D. degrees from Anhui University, Hefei, China, in 2012, 2015, and 2018, respectively.

He is currently an Associate Professor with the Institutes of Physical Science and Information Technology, Anhui University and also a Postdoctoral Research Fellow with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong. His current research interests include evolutionary computation and its applications.

Dr. Tian is the recipient of the 2018 and 2021 IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION Outstanding Paper Award and the 2020 *IEEE Computational Intelligence Magazine* Outstanding Paper Award.

Shichen Peng received the B.Sc. degree from Anhui University, Hefei, China, in 2017, where he is currently pursuing the M.Sc. degree with the School of Computer Science and Technology.

His current research interests include evolutionary optimization and deep learning.

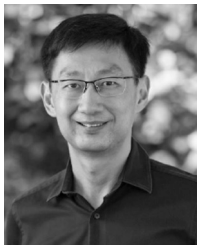
Shangshang Yang received the B.Sc. degree from Anhui University, Hefei, China, in 2017, where he is currently pursuing the Ph.D. degree with the School of Computer Science and Technology.

His current research interests include evolutionary multiobjective optimization and neural architecture search.

Xingyi Zhang (Senior Member, IEEE) received the B.Sc. degree from Fuyang Normal College, Fuyang, China, in 2003, and the M.Sc. and Ph.D. degrees from Huazhong University of Science and Technology, Wuhan, China, in 2006 and 2009, respectively.

He is currently a Professor with the School of Artificial Intelligence, Anhui University, Hefei, China. His current research interests include unconventional models and algorithms of computation, multiobjective optimization, and membrane computing.

Prof. Zhang is the recipient of the 2018 and 2021 IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION Outstanding Paper Award and the 2020 *IEEE Computational Intelligence Magazine* Outstanding Paper Award.



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

He is currently a Chair Professor with the Department of Computing, Hong Kong Polytechnic University, Hong Kong. He has published over 200 refereed articles and six books, and holds one U.S. patent on surface defect detection.

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

Yaochu Jin (Fellow, IEEE) received the B.Sc., M.Sc., and Ph.D. degrees from Zhejiang University, Hangzhou, China, in 1988, 1991, and 1996, respectively, and the Dr.-Ing. degree from Ruhr University Bochum, Bochum, Germany, in 2001.

He is currently a Distinguished Chair Professor of Computational Intelligence with the Department of Computer Science, University of Surrey, Guildford, U.K., where he heads the Nature Inspired Computing and Engineering Group. He was a Finland Distinguished Professor and a Changjiang Distinguished Visiting Professor.

Dr. Jin is the recipient of the 2014, 2016, and 2020 *IEEE Computational Intelligence Magazine* Outstanding Paper Award, the 2018 and 2021 IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION Outstanding Paper Award, and the Best Paper Award of the 2010 IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology. He has been named a Highly Cited Researcher for 2019 and 2020 by the Web of Science group. He was awarded the Alexander von Humboldt Professorship for Artificial Intelligence in 2021. He is the Editor-in-Chief of the IEEE TRANSACTIONS ON COGNITIVE AND DEVELOPMENTAL SYSTEMS and *Complex & Intelligent Systems*. He is also an Associate Editor or Editorial Board Member of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, IEEE TRANSACTIONS ON CYBERNETICS, *Evolutionary Computation*, and *Soft Computing*. He was an IEEE Distinguished Lecturer from 2013 to 2015 and from 2017 to 2019. He is a member of Academia Europaea.