AS-NAS: Adaptive Scalable Neural Architecture Search With Reinforced Evolutionary Algorithm for Deep Learning

Tong Zhang, *Member, IEEE*, Chunyu Lei, Zongyan Zhang, Xian-Bing Meng, and C. L. Philip Chen, *Fellow, IEEE*

***Abstract*—Neural architecture search (NAS) is a challenging problem in the design of deep learning due to its nonconvexity. To address this problem, an adaptive scalable NAS method (AS-NAS) is proposed based on the reinforced I-Ching divination evolutionary algorithm (IDEA) and variable-architecture encoding strategy. First, unlike the typical reinforcement learning (RL)-based and evolutionary algorithm (EA)-based NAS methods, a simplified RL algorithm is developed and used as the reinforced operator controller to adaptively select the efficient operators of IDEA. Without the complex actor–critic parts, the reinforced IDEA based on simplified RL can enhance the search efficiency of the original EA with lower computational cost. Second, a variable-architecture encoding strategy is proposed to encode neural architecture as a fixed-length binary string. By simultaneously considering variable layers, channels, and connections between different convolution layers, the deep neural architecture can be scalable. Through the integration with the reinforced IDEA and variable-architecture encoding strategy, the design of the deep neural architecture can be adaptively scalable. Finally, the proposed AS-NAS are integrated with the $L_{1/2}$ regularization to increase the sparsity of the optimized neural architecture. Experiments and comparisons demonstrate the effectiveness and superiority of the proposed method.**

Manuscript received August 27, 2020; revised November 26, 2020 and February 3, 2021; accepted February 17, 2021. Date of publication February 23, 2021; date of current version October 1, 2021. This work was supported in part by the National Key Research and Development Program of China under Grant 2019YFB1703600 and Grant 2019YFA0706200; in part by the National Natural Science Foundation of China under Grant 62076102, Grant U1813203, Grant U1801262, and Grant 62006081; in part by the National Natural Science Foundation of Guangdong for Distinguished Young Scholar under Grant 2020B1515020041; in part by the Science and Technology Major Project of Guangzhou under Grant 202007030006; in part by the Science and Technology Program of Guangzhou under Grant 202002030250; in part by the Guangdong–Hong Kong–Macao Greater Bay Area Center for Brain Science and Brain-Inspired Intelligence Fund under Grant 2019016; and in part by the China Postdoctoral Science Foundation under Grant 2020M672630. *(Corresponding authors: Xian-Bing Meng; C. L. Philip Chen.)*

Tong Zhang is with the School of Computer Science and Engineering, South China University of Technology, Guangzhou 510006, China, and also with Pazhou Lab, Guangzhou 510335, China (e-mail: tony@scut.edu.cn).

Chunyu Lei, Zongyan Zhang, and Xian-Bing Meng are with the School of Computer Science and Engineering, South China University of Technology, Guangzhou 510006, China (e-mail: axbmeng@gmail.com).

C. L. Philip Chen is with the School of Computer Science and Engineering, South China University of Technology, Guangzhou 510006, China, also with Pazhou Lab, Guangzhou 510335, China, and also with the Department of Computer and Information Science, Faculty of Science and Technology, University of Macau, Macau, China (e-mail: philipchen@scut.edu.cn).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TEVC.2021.3061466.

Digital Object Identifier 10.1109/TEVC.2021.3061466

***Index Terms*—Deep learning, I-Ching divination evolutionary algorithm (IDEA), neural architecture search (NAS), reinforced operator controller, variable-architecture encoding.**

I. Introduction

In RECENT years, the neural architecture search (NAS) has attracted great research interests [1]–[3]. Many deep learning methods are designed by NAS methods and have been successfully demonstrated their superiorities [4]–[7]. The design of the NAS method is of significance for realizing automated machine learning (AutoML) [1], [8]. As suggested in [9], the neural architecture with strong inductive biases can perform certain task without learning any weight parameters. Hence, the research on NAS methods has practical implication and theoretical significance [1], [2].

The existing optimization methods for NAS can be mainly classified as six categories, including: 1) the evolutionary algorithm (EA); 2) reinforcement learning (RL); 3) gradient-based method; 4) surrogated model; 5) grid and random search; and 6) hybrid method [1], [2], [8]. For EA-based NAS methods, the neural architectures are first encoded as individuals, which can then be initialized and evolved based on the fitness values related to the validation accuracies of the corresponding neural architectures [5], [10]. In typical RL-based NAS methods, an additional controller network is used to determine a sequence of operators and connection tokens and construct the neural architecture sequentially [11], [12]. Though grid and random search are two simple algorithms, they are also useful NAS methods [1]. Different from the above three methods, gradient-based NAS methods provide solutions by relaxing the discrete search space of neural architecture to continuous one [13]. Since the search space of neural architecture is usually very large, and evaluating the effectiveness of the optimized architecture is very computationally expensive, surrogate models are proposed and used as alternative NAS approaches [14]. Typical surrogate models are usually based on the Bayesian optimization and neural networks [1], [15]. Other NAS methods are mainly the integration of above five methods, such as the integration of RL and EA [6] and surrogate model-based EA [16]. Although all of these NAS methods can be successfully applied to design deep learning methods, there still exists improvement for them [1], [6]. It is an ongoing work to

design efficient NAS methods for balancing its convergency performance, generalization performance, and computational complexity.

The motivation of this article is an attempt to synthesize the merits of EA and RL and design an efficient and scalable NAS method. The performances of EA-based NAS methods depend on the specific operators of EA [5], [10]. Different operators may have different performances. Without considering these differences, EA-based methods may not have high search efficiency. As for RL-based NAS methods, it should learn the additional controller to search the architecture layer by layer [11]. The controller should try various actions to obtain the reward, which can then be used as a supervisory way to learn the controller and evaluate the corresponding architecture [6]. Obviously, the learning process is not efficient. In fact, there are various successful cases combining RL and EA [17]–[19]. However, it is still an ongoing work to integrate EA and RL into the NAS method.

Furthermore, another goal is to design a variable-architecture encoding strategy for improving the generalization performance and scalability of NAS method. The specific neural architecture varies from different tasks. The architecture parameters include the number of layers, the relationships between different layers (U-Net [20], fully connected layer, etc.), and the structure-related parameters (kernel size, channels, etc). It would be hard to design a fixed-architecture-encoding-strategy-based NAS method that can be applied to learn different architectures for solving different tasks. Hence, it is necessary to investigate the variable-architecture encoding strategy.

In this article, an adaptive scalable neural architecture search method (AS-NAS) is proposed based on the reinforced I-Ching divination EA (IDEA) and variable-architecture encoding strategy for designing deep learning models. Without complex actor–critic parts, a simplified RL algorithm is developed and used as a reinforced operator controller to adaptively select the operators of IDEA, thus improving the search efficiency of IDEA with lower computational cost. Moreover, a variable-architecture encoding strategy is developed to encode the neural architecture. Besides encoding the neural architecture with variable layers and channels, the encoding strategy can also design variable connections between different convolution layers. Additionally, $L_{1/2}$ regularization [21] is applied to further increase the sparsity of the AS-NAS method. Finally, experiments are conducted to investigate the effectiveness of AS-NAS.

The contributions of this article can be mainly summarized as three aspects.

1) AS-NAS is proposed to adaptively design the neural architecture, enhancing the generalization performance and scalability of the optimized neural architecture.
2) A simplified reinforced algorithm is designed as a reinforced operator controller to adaptively select the different operators of IDEA to enhance its search efficiency.
3) A variable-architecture encoding strategy is developed to encode the neural architecture with variable layers, channels, and connections between different layers.

The remainder of this article is organized as follows. Related works are briefly reviewed in Section II. The technical details about the proposed method is illustrated in Section III. Section IV presents the experiments and comparison studies. Finally, several conclusions and future works are discussed.

II. Related Works

The main topic of this article is about using the improved IDEA to design the NAS method. Hence, we mainly review the related works from the view of the existing NAS methods and IDEA.

A. NAS Methods

The key of NAS methods is to design search space, architecture optimizer, and model estimation method [2]. The existing works about NAS can be mainly summarized from these three aspects. Among these three topics, the model estimation method emphasizes the research on evaluating the performance of the optimized neural architecture based on the given search space. Since it is computationally expensive to directly train the model with optimized architecture and then estimate its corresponding performance on unseen data, various methods have been proposed. The model estimation methods mainly consist of five categories [1]. This first category refers to the methods that use lower fidelities of the actual performance to accelerate the model evaluation process [12], [22], [23]. The second one is based on surrogate models that learn the curve extrapolation to estimate a specific architecture's performance without directly evaluating the architecture [15]. Weight sharing and early stop criteria are also useful methods [24], [25]. Another category emphasizes the balance between the performance and corresponding resource budget [2], [26]. In this article, we focus on the search space and architecture optimizer. Hence, we emphasize reviewing these two aspects.

Search space defines the representation format of the neural architecture. By treating the neural architecture as the direct acyclic graph (DAG), the design of search space can be transformed to design DAG. Since the search space determines candidate neural architecture that an architecture optimizer can search, it is of significance to design an efficient search space. The existing works about search space can be mainly classified as four aspects [1], [2]. The intuitive method is to directly define the whole architecture of the search space [27]. However, it is computationally expensive to directly search the whole architecture. Moreover, the whole architecture may lack flexibility and transferability. The learned architecture for a specific task may not be further used as a prior for another similar task [1], [11]. In fact, the other three methods are proposed to address these problems. Cell-based search space focuses on learning efficient cells and constructing a cell-based architecture, which may be further used a prior for another task or used to directly solve different tasks [7], [12], [22]. However, the cell-based search space also has limitations. The learned cells would be used for all layers, thus lacking of diversity. To address this problem, a hierarchical search space is designed. For example, different cells would be further learned, and the

learned different cells would then be used to construct different layers [14], [26]. Morphism-based search space is another promising method. By inheriting the knowledge prior from well-trained networks, it cannot only handle arbitrary nonlinear activation functions but also accelerate the training process for a new task [28], [29].

Each search space has its own feature and has been demonstrated its effectiveness. However, there may still exist room for the further improvement. For example, the distinctive feature of genetic convolutional neural network (CNN) is that it can formulate variable connection relationships between different convolution-layer-based cells. However, the convolution kernel sizes and the number of channels are fixed and cannot be optimized [30]. Moreover, the probability of existence of each cell does not equal to 0.5 in the designed neural architecture, and thus whether there is a cell would not vary with specific problem. Inspired from this work, we will study how to address the aforementioned problems and investigate to design variable-architecture-based search space.

As for the architecture optimizer, there are mainly six categories as aforementioned.

1) *Evolutionary Algorithm:* EAs are stochastic algorithms that mimic the mechanisms inspired by nature to iteratively optimize the problem of interest. The essence of designing EA is to balance its exploration and exploitation performance [31]. There are various EAs [32], including the genetic algorithm (GA) [10], particle swarm optimization (PSO) [5], differential evolution [33], IDEA [34], chicken swarm optimization [35], bird swarm algorithm [36], and so on. For EA-based NAS methods, the key is to design efficient EA and appropriate encoding strategy [2], [8], [10]. The essence of NAS is to solve the combination optimization problem. All EA variants can be used to solve the combination problem. Hence, all of EA variants can be used as NAS methods in theory. For example, a multiobjective GA is proposed to solve NAS and have been successfully applied to solve the CIFAR-10 dataset [10]. Based on decomposition, PSO has also been applied to solve NAS [5]. How to enhance the search efficiency of EA is the main topic of EA-based NAS methods. In this article, we would investigate how to use RL to enhance the search efficiency of the EA-based NAS method.
2) *Reinforcement Learning:* The difficulties of RL-based NAS methods lie in how to learn the additional controller to design the neural architecture layer by layer. Many RL-based NAS methods have been designed [1], [11], [27]. A typical learning procedure can be summarized as follows. First, a controller is used to generate specific neural architectures. Second, the generated architecture-based deep learning model is trained by a certain parameter optimizer. RL can then be used to learn and update the controller by maximizing the expected accuracy of the learned model on a specific validation set [27]. For example, Pham *et al.* [11] proposed a long short-term memory-based NAS method with parameter sharing. Zoph *et al.* [12] proposed a recurrent neural network-based transferable NAS method. Given the fact that the additional controller should be learned before learning the neural architecture, it is necessary to investigate how to reduce the computational complexity of RL-based NAS methods.
3) *Gradient-Based Method:* By relaxing the discrete space to a continuous one, the gradient-based method can then be used as NAS methods. The key of this method is how to formulate the search space and the corresponding architecture [1]. The pioneering work, DARTS, is proposed by using a softmax function to relax the discrete space to a continuous one [7]. There are many works to further investigate how to use the gradient-based method to solve NAS problems [3], [4]. For example, a revised DARTS is proposed to regularize the search space by controlling the proportion of skip-connect operations [14]. Another useful way to regularize the search space can be considered based on memory efficiency [3].
4) *Surrogate Model:* The main goal of this method is to learn efficient model to accelerate the search process for finding a promising architecture. The traditional surrogate models, including neural networks, random forests, the Bayesian optimization methods, and the tree-structured parzen estimator, can be used as NAS methods in theory [1], [2]. For example, Kandasamy *et al.* [15] used a Bayesian optimization method as the surrogate model to solve the NAS problem. Camero *et al.* [37] proposed a random forests-based Bayesian NAS method. To predict architecture progressively, LSTM is also used as a surrogate model [14].
5) *Grid and Random Search:* Different from EA-based NAS methods, grid-based NAS methods first divide the search space into regular intervals and then select the architecture with best performance. As for random search-based NAS methods, they randomly explore the given search space to find the architecture with best performance [38]. Without explicit mechanisms to learn and guide the search process, the two methods would be computational expensive [1].
6) *Hybrid Method:* Different methods have their own merits. It would be a good choice to synthesize different methods. For example, random forests are used as a surrogate model and integrated into EA to serve as the NAS method [16]. The EA and RL-based NAS methods are to synthesize the two methods' merits [1], [6]. The RL-based mutation controller is integrated into EA to automatically optimize the neural architecture. With the help of a reinforced mutation controller, the operators in EA can be more efficient than the original ones [6]. Though the hybrid method can improve the search efficiency of EA, it also needs to train the RL-based mutation controller at expensive cost. To further address this problem, a simplified reinforced algorithm is designed and integrated into EA to be used as an efficient NAS method in this article.

B. IDEA

IDEA is inspired from I-Ching transformations. By mimicking the intricate, synthesis, and mutual transformations in I-Ching divination, three operators, namely, intrication, turnover, and mutual operators, are designed. In IDEA, the search space is called the hexagram space $\boldsymbol{H}$. The individual in the hexagram space is named a hexagram $\boldsymbol{h_{il}}$, $i = 1, 2, \ldots, N$. l is the length of bit strings and N denotes the size of hexagram space. $\boldsymbol{h_{il}}$ is encoded as bit strings. A hexagram state $\boldsymbol{\xi} = (\boldsymbol{h_1}, \boldsymbol{h_2}, \ldots, \boldsymbol{h_n})^T$ is a subset of $\boldsymbol{H}$ with state size n. Here, n can be regarded as the population size [34], [39].

For M-ary strings, there would exist M^l elements in $\boldsymbol{H}$. The intrication operator $\boldsymbol{I}$ can be formulated as follows:

$$\boldsymbol{I}(\boldsymbol{h_i}) = \{\boldsymbol{I}(h_{i1}), \boldsymbol{I}(h_{i2}), \ldots, \boldsymbol{I}(h_{ij}), \ldots, \boldsymbol{I}(h_{il})\} \tag{1}$$

$$\boldsymbol{I}(h_{ij}) = \begin{cases} \text{mod}_M(h_{ij} + x_{ij}), & \textit{if rand} < p_m \\ h_{ij}, & \text{else} \end{cases} \tag{2}$$

where $j \in [1, l]$, rand denotes a random value ranged from 0 to 1. $x_{ij}(x_{ij} \in \{0, 1, 2, \ldots, M-1\})$ is an independent and identically distributed random variable. $p_m \in [0, 1]$ is a parameter.

The turnover operator $\boldsymbol{\tau}$ is a generalization of the synthesis transformation in I-Ching, and can be formulated as follows:

$$\boldsymbol{\tau}(h_{ij}h_{ij+1}\cdots h_{iq}) = \begin{cases} h_{iq}\ h_{iq-1}\cdots h_{ij}, & \text{rand} > 0.5 \\ h_{ij}\ h_{ij+1}\cdots h_{iq}, & \text{else} \end{cases} \tag{3}$$

where $j < q, 1 \le j \le n, 1 < q \le n$. j and q are randomly selected.

The influence of middle bits in I-Ching is simply formulated by the mutual operator $\boldsymbol{\mathcal{M}}$. Its mathematical expression can be described as follows:

$$\boldsymbol{\mathcal{M}}(\boldsymbol{h_i}) = \{\boldsymbol{\mathcal{M}}(h_{i1}), \boldsymbol{\mathcal{M}}(h_{i2}), \ldots, \boldsymbol{\mathcal{M}}(h_{ij}), \ldots, \boldsymbol{\mathcal{M}}(h_{il})\} \tag{4}$$

$$\boldsymbol{\mathcal{M}}(h_{ij}) = \begin{cases} h_{ij+r},\ 1 \le j \le \left\lceil \frac{l}{2} \right\rceil \\ h_{ij-r},\ \left\lceil \frac{l}{2} \right\rceil \le j \le l \end{cases} \tag{5}$$

where $r = \text{round}((1/6) \times l)$.

Just like other EAs, a typical procedure of IDEA can be described as follows. First, an initial hexagram state is generated from the hexagram space randomly. Second, the fitness value of each hexagram is evaluated by a fitness function related to a specific problem. The elitism strategy is then used to select the parent generation to generate the next hexagram state, which is based on the aforementioned three operators. The above steps are iteratively implemented before a certain criterion meets.

III. Proposed Method

The key to automatically designing an efficient deep learning model is about the search space and architecture optimizer. Hence, we attempt to design the NAS method from these two aspects. From the view of search space, a variable-architecture-based encoding strategy is proposed to learn variable layers, channels, and connections between different layers simultaneously. The learned neural architecture-based model can then be trained by a gradient-based optimizer. With this variable-architecture encoding strategy, the learned neural architecture can be easily scalable. From the perspective of the architecture optimizer, a simplified RL-based EA, reinforced IDEA, is developed to adaptively select the different operators of IDEA based on the performance feedback of the learned model, thus improving the search efficiency of IDEA. With the help of this reinforced operator controller, the learned neural architecture can also be adaptively optimized based on the reward feedback. Through the integration with the two strategies both from the view of search space and architecture optimizer, the design of the neural architecture can be adaptively scalable. Additionally, $L_{1/2}$ regularization would be further integrated into AS-NAS to increase its sparsity.

A. Variable-Architecture-Based Search Space

The neural architecture can be formulated as DAG. Hence, the key to NAS is to design the number of layers, the connection relationships between different layers and other structure-related parameters, such as kernel size and channels, and so on. To address these problems, a framework of the variable-architecture-based encoding strategy is proposed based on the encoding strategy in [30]. In the proposed encoding strategy, the designed neural architecture consists of a series of cells based on some ordered blocks, and is encoded into fixed-length binary strings composed of three parts, whereas the corresponding binary strings just have one part in [30]. Here, the first part denotes whether the specific cell exists or not. As aforementioned, the probability of the existence of each cell does not equal to 0.5 in the designed neural architecture [30]. However, this new additional part can efficiently resolve this problem, make the learned neural architecture scalable and vary with specific problem. The second one indicates the output channels of specific convolutional layers. Four specific channels are selected and encoded into 00, 01, 10, and 11, respectively. The last part represents the connection relationship between different blocks in each cell. This part is the same as the encoding strategy in [30]. As a result, the goal is to solve a combination optimization problem to find an optimal neural architecture formulated by the three parts.

For simplicity, the convolutional and pooling layers are used as basic blocks. Another reason of choosing the convolutional and pooling layers as blocks is that they are widely used and have been successfully proven its broad practicality [30], [40], [41]. It should be noted that different kernel sizes and pooling strategies can be chosen. In this article, however, the kernel size in each convolutional layer is simply set to 3×3. For the pooling layer, the max pooling with 2×2 window is used.

The key to the proposed encoding strategy is to encode the connection relationship between difference cells and blocks in the third part of binary strings. In the proposed encoding strategy, there exist two connection relationships. The first one is the connection relationship between different cells. 1 denotes the corresponding cell exists, whereas 0 indicates the corresponding one does not exist. If two specific cells both exist, there are connections between them. To ensure the specific binary valid, three default blocks are integrated into this neural architecture. On the one hand, a predefined convolutional layer is placed

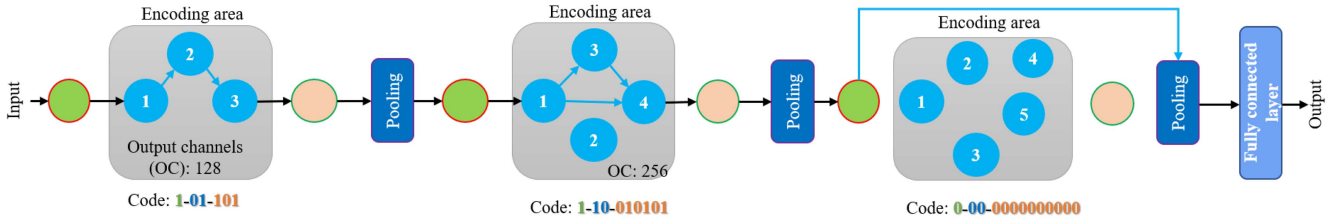

Fig. 1. Illustrated example for the encoding strategy.

before each cell. On the other hand, the predefined convolutional layer and pooling layer are successively placed after the cell. If a specific cell does not exist, the predefined convolutional layer before the cell and the pooling layer after the cell would be kept. However, the convolutional layer after the cell would be deleted along with the cell. Moreover, in each convolutional layer, there also exists ReLU after the convolutional operation. A predefined fully connected layer is placed at the end of the designed neural architecture. Moreover, the validity of the optimized neural architecture can also be ensured from the optimization strategy, which would be illustrated later.

As for the second connection relationship between different blocks in each cell, it can be summarized as follows. Consider a candidate network with m cells, the number of blocks in the ith cell is $n_i, i \in [1, m]$. In each cell, there exist several ordered blocks, which are convolutional layers. Only the lower numbered blocks can connect to the higher numbered one. In the i cell, $0.5 \times n_i(n_i - 1)$ bits should be used to encode the connection relationship between blocks, which is encoded sequentially according to the order of blocks. For the encoded binary strings, there are $n_i - 1$ parts. For the first part, the bit denotes the connection between $G_{i,1}$ and $G_{i,2}$ in the ith cell. As for the second part, there are two bits that indicate the connections between $G_{i,1}$, $G_{i,2}$, $G_{i,3}$, and so on. For the last part, there exist $n_i - 1$ bits that represent the connections between $G_{i,1}$, $G_{i,2}$, ..., G_{i,n_i-1}, and G_{i,n_i}. Given two blocks in the ith cell G_{i,j_1} and $G_{i,j_2}(j_1 < j_2)$, if the corresponding bit equals 1, G_{i,j_1} connects to G_{i,j_2}, namely, there exists an edge between these two blocks. Moreover, block G_{i,j_2} takes the output of G_{i,j_1} as a part of the elementwise summation.

In summary, the whole encoded binary strings consist of the corresponding strings of all cells, which are concatenated successively according to the sequence number of each cell. In each cell, the binary strings consist of the three parts of fixed-length binary strings. The whole neural architecture can be encoded into binary strings with length $L = 0.5 \sum_{i=1}^{m} (n_i(n_i - 1))$. It should be noted that the probability of the existence of a cell is 0.5 and, thus, whether there is a specific cell varies with specific problems. Hence, the design of the neural architecture can be scalable.

An illustrated example of the proposed encoding strategy is presented in Fig. 1. There exist three cells that would be used to construct the candidate neural architecture. The code of each cell consists of three parts. The first part indicates whether the corresponding cell exists or not. It can be obviously seen that only the first two cells exist. The second part denotes the output channels. Because "01" denotes 128, the number of the first cell equals 128. The last part encodes the connection between different convolution layers in each cell. Here, the first cell is used as an example. There is an edge between the first and second layers and, thus, the first bit equals 1. Because there is no connection between the first and third layers, the second bit is set to 0. Since there exists an edge between the second and third layers, the third bit is assigned to 1.

As a result, the corresponding neural architecture of the specific model can be illustrated as follows. The data of interest are first input as a convolutional layer (green block), the output of which is then input in the first cell with three ordered convolutional layers. The output of the first cell, namely, the output of the third convolutional layer in the first cell, is input the following convolutional layer (light orange block) and polling layer, the output of which is then input in the following convolutional layer (green block), the second cell. Because the third cell does not exist, the output of the third convolutional layer (green block) would directly input in the third polling layer, the output of which is finally input in a fully connected layer. The final result can be obtained from the fully connected layer.

B. Reinforced IDEA Based on Simplified RL and $L_{1/2}$ Regularization

Here, IDEA is used as the architecture optimizer to optimize the neural architecture. Given the fact there are 2^L numbers of candidate neural architectures, the search space is very large and, thus, it would be very hard to solve the combination optimization problem. To enhance the search efficiency and generalization performance of IDEA, an improved IDEA is designed based on the simplified RL and $L_{1/2}$ regularization.

1) Reinforced IDEA Based on Simplified RL: The key to NAS is to design an efficient architecture optimizer. For EA, however, its evolution process relies heavily on random operators [6]. In IDEA, its three operators are implemented sequentially, and there are no efficient feedback to guide the evolution process by adaptively selecting efficient operators. Moreover, the mutual operator is not suitable for the designed NAS. The reasons can be summarized as follows. On the one hand, the encoded bit strings have definite meaning. Hence, it is not suitable for the mutual operator to use only part of bits to generate a valid neural architecture. On the other hand, there are at least 4 bits for using mutual operator. However, there are 2 and 3 bits in several parts of the encoded bit strings. To address this problem, a crossover operator is used to replace the mutual operator. Just like the crossover operator in GA, there is a probability p_c that determine whether the bits of

one individual would be replaced by the corresponding bits of another individual with a better fitness value. If there is no individual that has better fitness value than a specific one, the corresponding individual can randomly select a bit to change.

To further improve the search efficiency of IDEA, a reinforced operator controller is proposed and integrated into IDEA. Just like the original IDEA, the turnover and crossover operators are also implemented based on the intrication operator. However, these two operators would not be implemented simultaneously. Their search performances would be used as feedback for reinforced operator controller to adaptively select which operator would be implemented. The better the optimization performance of the operator, the more likely it is to be chosen. If the turnover operator outperforms the crossover one, the turnover operator would be assigned to a positive reinforcement reward and, thus, the probability of choosing the turnover operator would be increased, and vice versa. To design this reinforced operator controller, a simplified version of the reinforced algorithm [42], [43] is developed as follows.

Without loss of generality, let $y_i = 1$ denote the ith individual selecting a turnover operator and $y_i = 0$ represent the ith individual selecting the crossover operator. Here, y_i is a Bernoulli random variable, namely, y_i only chooses 0 and 1. Given a specific y_i, $g_i(y_i, p_i)$ is the probability mass function related to p_i, it can be formulated as follows:

$$g_i(y_i,\ p_i) = \begin{cases} p_i,\ \ y_i = 1 \\ 1 - p_i,\ \ y_i = 0. \end{cases} \tag{6}$$

Here, a simplified reinforced algorithm is developed to learn the probability g_i. Consider an associative immediate-RL task with respect to parameters w. w are adjusted following receipt of the reinforcement value at each trial. The reinforced algorithm is such an algorithm that describes how to update w. The typical reinforced algorithm is an acronym for "Reward increment = Nonnegative factor * Offset reinforcement * characteristic eligibility" [42], which can be formulated by (7). In this article, we simply formulate the reinforced algorithm as follows:

$$\begin{aligned} \Delta w_i &= \eta_i \cdot (r_i(t) - b_i)\frac{\partial \ln g_i}{\partial w_i(t)} \\ w_i(t) &= w_i(t-1) + \Delta w_i \end{aligned} \tag{7}$$

$$\begin{aligned} \Delta p_i(t) &= f(w_i(t)) = \frac{1}{1+\exp(-w_i(t))} - 0.5 \\ p_i(t) &= p_i(t-1) + \Delta p_i(t) \end{aligned} \tag{8}$$

where the reinforcement baseline b is set to 0 and $\eta_i > 0$ denotes a learning rate. $r_i(t)$ is the reward value obtained by the ith individual at time step t.

Given the fact that there is no operator preference in the beginning, the probability p_i is initially set to 0.5 and w_i is set to 0. As the iterative process progresses, the preferences of these two operators will gradually form. Obviously, the bigger the reward value, the bigger the probability of selecting the corresponding operator, and vice verse. Based on the proposed reinforced operator controller formulated by (6), (7), and (8), the probability of selecting the specific operator is proportional to the corresponding reward value. This can be demonstrated by the following theorem.

Theorem: Consider the i individual, $\eta_i > 0$, p_i computed by 8 is directly proportional to r_i.

Proof: According to (7), (8), and the chain rule, we have

$$\begin{aligned} \frac{\partial \ln g_i}{\partial w_i} &= \frac{\partial \ln g_i}{\partial p_i}\frac{\partial p_i}{\partial w_i} = \frac{y_i - p_i}{p_i(1-p_i)} \times p_i(1-p_i) \\ &= y_i - p_i. \end{aligned} \tag{9}$$

Substituting (9) into (7), $w_i(t)$ is formulated as

$$\begin{aligned} \Delta w_i &= \eta_i \cdot r_i(y_i - p_i) \\ w_i(t) &= w_i(t-1) + \Delta w_i. \end{aligned} \tag{10}$$

For a specific y_i, the corresponding p_i is determined and, thus, $y_i - p_i$ would be constant. Combining $\eta_i > 0$ and (10), we can conclude that w_i is proportional to r_i. According to (8), w_i is also proportional to p_i.

Hence, p_i is proportional to r_i. ■

How to determine the reward value $r_i(t)$ is problem-dependent. Generally, the improvement in the fitness value at the beginning will be greater than the improvement in the later stage. To fairly treat these two stages, $r_i(t)$ is simply formulated as a function related to the difference between the current and previous fitness values, and the difference between the current and previous best fitness values. Just like [43], four different values, including 0, 0.5, 0.75, and 1, are used to distinguish the different effects of the two operators' performance feedback. The probabilities would be updated every k iterations.

The aforementioned operators would execute on the whole encoded binary strings. Meanwhile, each operator would be implemented on the three parts of the encoded binary strings separately. Hence, the separate optimizer cannot destroy the specific parts of the neural architecture encoded by the binary strings. Hence, the optimized neural architecture can be valid.

2) $L_{1/2}$ Regularization: Although the learned neural architecture can be adaptive scalable based on the reinforced IDEA and variable encoding strategy, this superiority of the proposed method would be obtained at the cost of a large number of parameters. To address this problem, $L_{1/2}$ regularization is integrated with the reinforced IDEA to increase the sparsity of the learned neural network. Generally, $L_{1/2}$ regularization is easier to be solved than L_0 regularization. Moreover, it is more stable than L_1 regularization, and can also generate more sparse results [21]. However, the $L_{1/2}$ regularization term is not differentiable at the origin, and may lead to error oscillation. Here, a smoothing function is used to approximate the nonsmooth function $|w|$ [44] and it can be formulated as follows:

$$f(w) = \begin{cases} |w|,\ \ |w| \geq \varepsilon > 0 \\ -\frac{1}{8\varepsilon^3}w^4 + \frac{1}{4\varepsilon}w^2 + \frac{3}{8}\varepsilon,\ \ |w| < \varepsilon \end{cases} \tag{11}$$

where ε is a positive constant close to 0.

Hence, the loss function can be formulated as follows:

$$\ell = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} c_{i,j}\log z_{i,j} + \lambda f(w)^{1/2} \tag{12}$$

where the first term of this loss function is the cross-entropy error. N denotes the number of samples in a batch, C indicates

Algorithm 1: AS-NAS

Input: The hexagram state size n, maximal number of iterations N_{max}, intrication probability p_m, crossover probability p_c, η, k, λ, training and validatation sets D_{train}, D_{valid}, batch size B_s, training epoch number E_p

Output: Optimized neural architecture associated with its corresponding parameters

1 Initialize the hexagrams and the probabilities p of choosing different operators.
2 For each epoch, decode the learned neural architectures represented by the hexagrams, use $|D_{train}|/B_s$ batches of samples as input, and use Adam to optimize the weights and biases of the learned neural architecture based on the loss function.
3 After executing E_p epochs, use $|D_{valid}|/B_s$ batches of samples as input, and compute the validation errors of the optimized neural architecture. The validation errors are used as the fitness values of the hexagrams.
4 For each hexagram, use intrication-based turnover operator to generate new hexagram when p is bigger than a random value ranged from 0 and 1. Otherwise, use intrication-based crossover operator to generate new hexagram.
5 Execute step 2 and step 3 to compute the fitness values of all hexagrams, and update the hexagrams.
6 Update the reward and p of each hexagram every k iterations.
7 Execute step 4-6 until N_{max} iterations are reached.

the number of classes, $c_{i,j}$ is the label in one-hot fashion, $z_{i,j}$ represents the output of specific network after softmax, and $\lambda(\lambda > 0)$ is a parameter.

Given the fact that the optimization of NAS is a bilevel optimization problem, the two-stage learning scheme is developed to optimize the neural architecture and the corresponding parameters. During the training stage, the loss function is used to evaluate the effectiveness of the optimized encoded neural architecture. As for the validation stage, the test errors are used as the criterion to learn the parameters of the selected neural architecture with the best validation performance. Here, Adam [45] is used to train the parameters of the optimized neural architecture. In Adam, beta1 = 0.9, beta2 = 0.999, and epsilon = 1e-8. Based on the two-stage learning, the neural architecture and the corresponding parameters can be finally optimized. The whole process of the proposed method is summarized in Algorithm 1.

IV. Experiments and Comparison Validations

To investigate the effectiveness of the proposed method AS-NAS, experiments and comparisons would be conducted from two aspects. On the one hand, ablation experiments would be conducted to verify the necessities and effectiveness of the main parts of the proposed methods. On the other hand, several NAS method-based deep learning approaches and well-designed machine learning methods are used as comparison methods to further investigate the superiority of AS-NAS. For a convinced comparison, the experimental results of the comparison methods either directly use the results in the corresponding references or use the codes obtained from the corresponding references to calculate.

In the experiments, the benchmark and datasets include NAS-Bench-201 [46], New York University object recognition benchmark (NORB) [47], and CIFAR-10. NORB consists of 48 600 images of 50 different 3-D generic toys belonging to five categories: 1) humans; 2) animals; 3) cars; 4) airplanes; and 5) trucks. Each image has 2*32*32 pixels. Here, 24 300 images of 25 objects are used for training and the others are used for testing. CIFAR-10 is composed of 60 000 32×32 RGB images belonging to ten categories [48]. 50 000 images are used for training and the other 10 000 images is selected for testing. As a new NAS benchmark, NAS-Bench-201 is an extension of NAS-Bench-101. Its search space consists of four nodes and five operations (zeroize, skip connection, 1-by-1 convolution, 3-by-3 convolution, and 3-by-3 average pooling layer), resulting in a total of 15 625 architectures. The training log using same setup and the performance for each architecture candidate are provided for three datasets, including CIFAR-10, CIFAR-100, and ImageNet16-120 [49]. All experiments are performed on a server equipped with Intel Xeon Silver 4214 CPU, 8 GeForce RTX 2080 Ti, and 11-GB main memory.

A. Ablation Experiments

To investigate the necessities and effectiveness of the variable-architecture encoding strategy, reinforced IDEA, and $L_{1/2}$ regularization, ablation experiments would be divided into two parts. The first one is to use NAS-Bench-201 to verify the effectiveness of the reinforced IDEA. Since the NAS benchmark has a fixed search space and focuses solely on the search algorithm itself, we select another dataset NORB to conduct the second experiment to investigate the effectiveness of the other parts of the proposed method.

In the first part of ablation experiments, seven methods, including several EAs and basic IDEA, are used as comparison methods. For IDEA and reinforced IDEA, the state space size, p_c, p_m, k, and η are 20, 0.4, 0.6, 4, and 0.1, respectively. 500 independent runs are conducted. We then report the mean performance of the immediate validation and test regret on the CIFAR-10, CIFAR-100, and ImageNet16-120 datasets.

As shown in Table I, it can be clearly seen that the reinforced IDEA outperforms the basic IDEA on all of the three datasets. Compared with the basic IDEA, the reinforced IDEA can increase the accuracy by 1%–2% on CIFAR-10 and CIFAR-100 datasets. Thus, we can safely draw a conclusion that the reinforced operator controller can indeed provide an efficient feedback to guide the evolution process by adaptively selecting efficient operators. With the merit of the reinforced operator controller, the reinforced IDEA can also show its superiority over the other six comparison methods.

To further investigate the effectiveness of the variable-architecture encoding strategy and $L_{1/2}$ regularization, the second part of ablation experiments is conducted. Because the

TABLE I
COMPARISON RESULTS OF THE ABLATION EXPERIMENT ON NAS-BENCH-201

Method	CIFAR-10		CIFAR-100		ImageNet16-120	
	valid	test	valid	test	valid	test
ENAS [11]	39.77±0.00	54.30±0.00	15.03±0.00	15.61±0.00	16.43±0.00	16.32±0.00
REA [22]	91.19±0.31	93.92±0.30	71.81±1.12	71.84±0.99	45.15±0.89	45.54±1.03
REINFORCE [42]	91.09±0.37	93.85±0.37	71.61±1.12	71.71±1.09	45.05±1.02	45.24±1.18
RS [50]	90.93±0.36	93.70±0.36	70.93±1.09	71.04±1.07	44.45±1.10	44.57±1.25
RSPS [51]	80.42±3.58	84.07±3.61	52.12±5.55	52.31±5.77	27.22±3.24	26.28±3.09
DE [52]	91.11±0.05	-	-	73.13±0.13	-	46.38±0.35
IDEA	91.22±0.16	93.84±0.21	71.64±0.46	71.92±0.33	45.71±0.18	44.76±0.59
Reinforced IDEA	91.71±0.01	94.01±0.26	73.61±0.07	73.99±0.29	46.99±0.37	46.90±0.38

TABLE II
COMPARISON RESULTS OF THE ABLATION EXPERIMENT ON NORB

Method	Parameters(M)	Test Error(%)
NAES[30] + reinforced IDEA	1.75±0.0180	7.05±0.6475
NAES[30] + reinforced IDEA + $L_{1/2}$	1.75±0.0109	10.87±0.5994
Ours without $L_{1/2}$	7.62±1.6831	4.93±0.3801
Ours	11.52±0.5279	6.14±0.3837

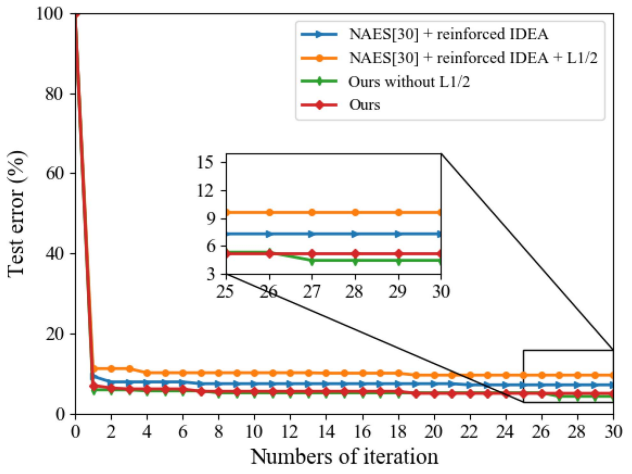

Fig. 2. Convergence curves of the four comparison methods for the NORB dataset.

proposed encoding strategy is based on the neural architecture encoding strategy (NAES) in [30], NAES is chosen as a comparison method. Based on the reinforced IDEA, comparison experiments are conducted to investigate the impact of the presence or absence of the variable-architecture encoding strategy and $L_{1/2}$ regularization on the performance of the proposed algorithm. Statistic results based on ten independent trials are used to analyze the comparison results. For fair comparisons, the corresponding parameters of all comparison methods are the same. Here, the candidate neural architecture consists of two cells, and the number of blocks in which is 3 and 5, respectively. The four output channels are 32, 64, 128, and 256.

Table II presents the comparison results on the NORB dataset. It can be clearly seen that our AS-NAS methods with and without $L_{1/2}$ regularization can both outperform the NAES-based methods. As shown the convergence curve of one trial in Fig. 2, our methods can also converge faster than the NAES-based methods. These results clearly demonstrate the superiority of the proposed variable-architecture encoding strategy. The reasons of these superiorities are summarized as follows. The number of candidate neural architectures in the proposed encoding strategy is 2^{19}, whereas the corresponding one is 2^{13} in NAES. In theory, it is more likely to find the optimal neural architecture from a bigger search space than a smaller one. Moreover, NAES focuses solely on optimizing the connections between different blocks in each cell. For the proposed encoding strategy, the output channels in each block can also be optimized, whereas they are fixed in NAES. With these merits, our methods can achieve better accuracies than NAES-based methods. As for the superiority of the convergence performance of our method, it can be illustrated as follows. Once the cells and their corresponding blocks for candidate neural architectures are predefined, the probability of the existence of a specific cell is close to 1 in NAES. However, the corresponding probability is equal to 0.5 in the proposed encoding strategy. As a result, the diversity of candidate neural architectures in the proposed encoding strategy is better than that in NAES. Moreover, if the cells and blocks are not well designed, the initialized neural architecture generated by NAES would be more likely to perform worse than that by the proposed encoding strategy, and the candidate neural architectures generated by NAES may be more likely to fall into the local optimum. However, this is not the case for the proposed encoding strategy. What is more, the reinforced IDEA can use the performance of the generated neural architectures as feedback to adaptively guide the search process. Hence, our method can be more likely to converge faster than NAES-based methods.

As for the $L_{1/2}$ regularization, it clearly shows in Table II that $L_{1/2}$ regularization would result in a decrease in our method performance. Moreover, the neural architecture learned by our AS-NAS method with $L_{1/2}$ regularization would have more parameters than that without $L_{1/2}$ regularization. The reason is that the regularized method not only focuses on minimizing the training errors but also emphasizes improving sparsity. However, the $L_{1/2}$ regularization can efficiently increase the sparsity of the learned neural architecture. As shown in Table III, the average number of parameters of the neural architecture learned by AS-NAS with $L_{1/2}$ regularization is $11.52M$, among which only $6.30M$ parameters' absolute values are bigger than 0.001, and only $4.31M$ parameters' absolute values are bigger than 0.01. For AS-NAS without the $L_{1/2}$ regularization, however, the corresponding values are $7.62M$, $7.56M$, and $6.99M$, respectively. To investigate the

TABLE III
COMPARISON RESULTS OF SPARSE NEURAL NETWORKS ON NORB

Method	Parameters(M)	Test Error(%)	Parameters(>0.001)	Test Error(parameters>0.001)	Parameters(>0.01)	Test Error(parameters>0.01)
Ours without $L_{1/2}$	7.62±1.6831	4.93±0.3801	7.56±1.6691	4.93±0.3849	6.99±1.5437	4.97±0.3622
Ours	11.52±0.5279	6.14±0.3837	6.30±2.6354	6.14±0.3854	4.31±2.4314	6.35±0.4486

TABLE IV
COMPARISON RESULTS ON NORB

Method	Parameters(M)	Test error (%)
DARTS [7]	3.34	7.8
NSGA [10]	0.08	7.4
Genetic CNN [30]	1.76	11.9
Stacked auto-encoders [53]	-	13.7
Deep belief nets [53]	-	11.5
Deep boltzmann machines [53]	-	10.3
Hierarchical ELM [53]	-	8.7
Broad learning [53]	-	10.7
Ours	3.75	5.97

effect of sparsity on the accuracy of the learned neural architecture, the corresponding parameters smaller than 0.001 and 0.01 are entirely deleted. As shown in Table III, the accuracy of the compressed neural network with parameters bigger than 0.01 is decreased, whereas the corresponding one with parameters bigger than 0.001 hardly decreases. Hence, we only consider the parameters bigger than 0.001 in this article. It can be clearly seen that the compressed neural network based on the $L_{1/2}$ regularization has a smaller number of parameters than that without $L_{1/2}$ regularization.

In summary, the variable-architecture encoding strategy and reinforced IDEA play important roles in optimizing neural architecture. Although $L_{1/2}$ regularization may result in a decrease in the proposed method performance, it can indeed increase the sparsity of the learned neural networks. If we focus on the sparsity of the neural networks, and the accuracy of the compressed neural networks being slightly decreased, the $L_{1/2}$ regularization-based method would be a good alternative method to the one without $L_{1/2}$ regularization.

B. Comparison Results

Several NAS-based methods and other well-designed machine learning methods are used to further investigate the superiority of our AS-NAS. The hyperparameters here are presented as follows. For Adam, its epsilon parameter is 10^{-8}, beta parameters are 0.9 and 0.999, and the initial learning rate is 10^{-4}. The batch size and epoch are 50 and 30, respectively. ε, λ, and η are 0.05, 0.08, and 0.1, respectively. The state space size, p_c, p_m, k, and maximal iteration number are 20, 0.4, 0.6, 4, and 30, respectively.

As shown in Table IV, the proposed AS-NAS can outperform all the comparison methods for solving NORB. The optimal neural architecture found by AS-NAS is presented in Fig. 3.

To further investigate the effectiveness and superiority of AS-NAS, a widely used dataset, CIFAR-10, is used. The hyperparameters are illustrated below. The epoch and λ are 240 and 0.001, respectively. The initial learning rate is 0.001,

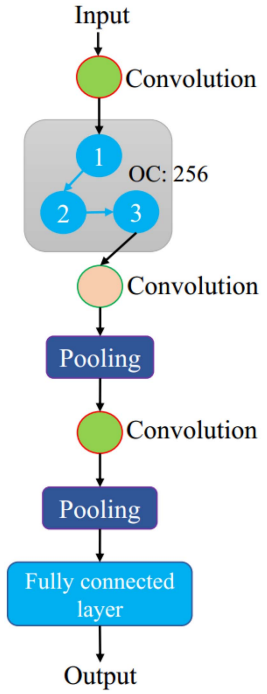

Fig. 3. Optimal neural architecture obtained by AS-NAS for solving the NORB dataset.

and it would be reduced by a factor of 0.8 before the first 200 epochs. Other hyperparameters are the same to the corresponding ones in the experiment above. The candidate neural architecture is composed of three cells, the number of blocks in which is 3, 4, and 5, respectively. The four output channels are 64, 128, 256, and 512. The number of candidate neural architectures is 2^{28}. However, the corresponding value is 2^{19} in [30]. In theory, it may be more likely for the proposed method to find the optimal neural architecture than the method in [30]. As shown in Table V, it can also be clearly seen AS-NAS can obtain the best result than other methods. The optimal neural architecture achieved by AS-NAS is presented in Fig. 4.

Because the total number of CIFAR-10 is bigger than that of NORB, and it is also more complex than NORB, the total number of parameters in the learned neural architecture on CIFAR-10 are also bigger than that on NORB. The $L_{1/2}$ regularization can also efficiently increase the sparsity of the learned neural architecture. As shown in Fig. 5, the parameters bigger than 0.001 in our AS-NAS with $L_{1/2}$ regularization only account for 4.8% of the total number of parameters. For our method without $L_{1/2}$ regularization, the corresponding value is 98.5%. Although $L_{1/2}$ regularization would result in a decrease in our method performance, our method with $L_{1/2}$ regularization can still obtain 2.23% test error and perform better than other comparison methods.

TABLE V
COMPARISON RESULTS ON CIFAR-10

Method	Test error (%)	Parameters (M)	Search method
DenseNet-BC [54]	3.46	25.6	manual
ENAS + cutout [11]	2.89	4.6	RL
NASNet-A+cutout [12]	2.65	3.3	RL
NAS + more filters [27]	3.65	37.4	RL
NSGA-Net [10]	2.50	26.8	evolution
AmoebaNet-B+cutout [22]	2.55	2.8	evolution
Genetic CNN [30]	7.10	-	evolution
Hierarchical evolution [55]	3.75	15.7	evolution
PNAS [14]	3.41	3.2	surrogate model
Sparse CNN [41]	7.47	-	gradient-based+regularization
PC-DARTS [3]	2.57	3.6	gradient-based
RDARTS [4]	2.95	-	gradient-based
DARTS(second order)+cutout [7]	2.76	3.3	gradient-based
diffGrad [13]	5.63	-	gradient-based
ResNet50 (LReLU) [13]	5.69	-	gradient-based
SNAS (single-level)+moderate constraint+cutout [56]	2.85	2.8	gradient-based
Ours	2.23	16.6	evolution+RL

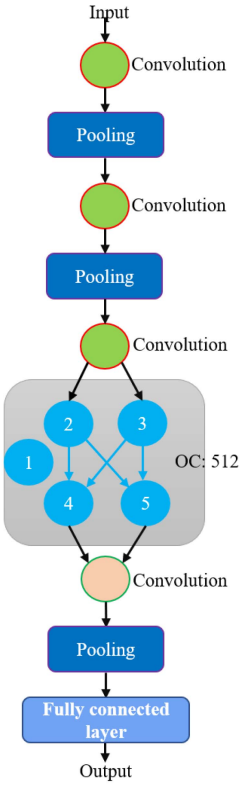

Fig. 4. Optimal neural architecture obtained by AS-NAS for solving the CIFAR-10 dataset.

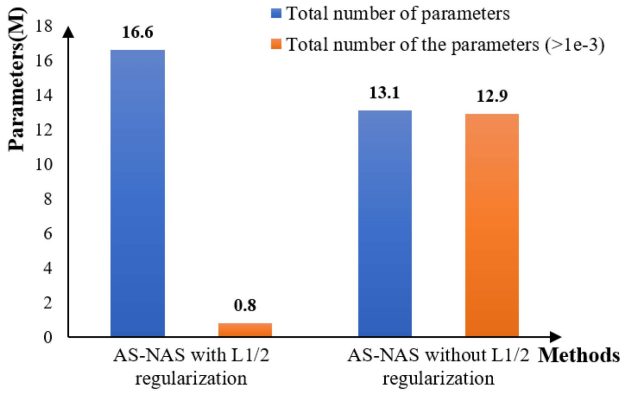

Fig. 5. Total numbers of parameters in AS-NAS with and without $L_{1/2}$ regularization on the CIFAR-10 dataset.

According to these experiments and extensive comparisons, it can be safely concluded that AS-NAS can be successfully used to design efficient deep learning methods.

V. CONCLUSION

In this article, an adaptive scalable NAS method was proposed based on the reinforced EA and variable-architecture encoding strategy. To design an efficient NAS method and improve its generalization performance and scalability, we provided a solution from the perspective of the search space and architecture optimizer simultaneously. On the one hand, a variable-architecture encoding strategy was developed to formulate the search space with variable layers, variable output channels, and variable connections between different convolution layers. The design of neural architecture can then be scalable. On the other hand, a reinforced operator controller was developed and integrated into IDEA to enhance the search efficiency of IDEA. Through the integration with the reinforced IDEA, $L_{1/2}$ regularization, and variable-architecture encoding strategy, the designed NAS method can efficiently balance its efficiency and generalization performance and have good sparsity. Experiments on NORB and CIFAR-10 datasets clearly demonstrate the effectiveness and superiority of AS-NAS.

Although the proposed work has achieved good results and shown its good scalability, there are still some works worthy of in-depth study.

First, other EAs can be further investigated to use as NAS methods. Moreover, other types of deep learning methods can also be further used to be automatically designed by AS-NAS. In fact, the proposed reinforced EA is a framework about using the reinforced operator controller to adpatively select the efficient operators. The reinforced IDEA is just used as an

illustration. Similarly, except for the convolution layer, the traditional feedforward network and LSTM may be also used as blocks. The proposed method can then be further used for the automated design of deep neural networks and deep LSTM.

Second, it is necessary to further investigate how to tune the hyperparameters and other complex architecture-based parameters. In AS-NAS, the kernel size of the convolution layer, different activation functions, and hyperparameters in the reinforced operator controller would influence the performance of the proposed method. However, they are all not optimized. The key to optimizing these parameters is how to encode them and improve the search efficiency of the operators.

Last but not least, the transferability of AS-NAS can be further investigated. With the variable architecture and $L_{1/2}$ regularization, the optimized neural architecture obtained by AS-NAS can be further applied to solve various datasets and to verify the generalization and transferability of AS-NAS.

REFERENCES

[1] X. He, K. Zhao, and X. Chu, "AutoML: A survey of the state-of-the-art," 2019. [Online]. Available: arXiv:1908.00709.

[2] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," 2018. [Online]. Available: arXiv:1808.05377.

[3] Y. Xu *et al.*, "PC-DARTS: Partial channel connections for memory-efficient architecture search," in *Proc. Int. Conf. Learn. Represent.*, 2020, pp. 1–13.

[4] A. Zela, T. Elsken, T. Saikia, Y. Marrakchi, T. Brox, and F. Hutter, "Understanding and robustifying differentiable architecture search," in *Proc. Int. Conf. Learn. Represent.*, 2020, pp. 1–28.

[5] J. Jiang, F. Han, Q. Ling, J. Wang, T. Li, and H. Han, "Efficient network architecture search via multiobjective particle swarm optimization based on decomposition," *Neural Netw.*, vol. 123, pp. 305–316, Mar. 2020.

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

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

[8] F. Hutter, L. Kotthoff, and J. Vanschoren, *Automated Machine Learning: Methods, Systems, Challenges*. Cham, Switzerland: Springer, 2019.

[9] A. Gaier and D. Ha, "Weight agnostic neural networks," in *Advances in Neural Information Proces. Syst.*, 2019, pp. 5364–5378, Vancouver, BC, Canada: Curran Associates, Inc.

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

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

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

[13] S. R. Dubey, S. Chakraborty, S. K. Roy, S. Mukherjee, S. K. Singh, and B. B. Chaudhuri, "diffGrad: An optimization method for convolutional neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 11, pp. 4500–4511, Nov. 2020.

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

[15] K. Kandasamy, W. Neiswanger, J. Schneider, B. Poczos, and E. P. Xing, "Neural architecture search with Bayesian optimisation and optimal transport," in *Advances in Neural Information Processing Systems*. Red Hook, NY, USA: Curran, 2018, pp. 2016–2025.

[16] 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.

[17] M. M. Drugan, "Reinforcement learning versus evolutionary computation: A survey on hybrid algorithms," *Swarm Evol. Comput.*, vol. 44, pp. 228–246, Feb. 2019.

[18] Y. Huang, W. Li, F. Tian, and X. Meng, "A fitness landscape ruggedness multiobjective differential evolution algorithm with a reinforcement learning strategy," *Appl. Soft Comput.*, vol. 96, pp. 1–13, Nov. 2020.

[19] F. Zou, G. G. Yen, L. Tang, and C. Wang, "A reinforcement learning approach for dynamic multi-objective optimization," *Inf. Sci.*, vol. 546, pp. 815–834, Feb. 2021.

[20] T. Falk *et al.*, "U-Net: Deep learning for cell counting, detection, and morphometry," *Nat. Methods*, vol. 16, no. 1, pp. 67–70, 2019.

[21] Z. Xu, H. Zhang, Y. Wang, X. Chang, and Y. Liang, "$L_{1/2}$ regularization," *Sci. China Inf. Sci.*, vol. 53, no. 6, pp. 1159–1169, 2010.

[22] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, 2019, pp. 4780–4789.

[23] Y.-Q. Hu, Y. Yu, W.-W. Tu, Q. Yang, Y. Chen, and W. Dai, "Multi-fidelity automatic hyper-parameter tuning via transfer series expansion," in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, 2019, pp. 3846–3853.

[24] C. Wong, N. Houlsby, Y. Lu, and A. Gesmundo, "Transfer learning with neural AutoML," in *Advances in Neural Information Processing Systems*. Red Hook, NY, USA: Curran, 2018, pp. 8356–8365.

[25] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Aging evolution for image classifier architecture search," in *Proc. AAAI Conf. Artif. Intell.*, 2019, pp. 5048–5056.

[26] B. Wu *et al.*, "FBNet: Hardware-aware efficient convnet design via differentiable neural architecture search," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Long Beach, CA, USA, 2019, pp. 10734–10742.

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

[28] T. Wei, C. Wang, Y. Rui, and C. W. Chen, "Network morphism," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 564–572.

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

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

[31] X.-B. Meng, X. Z. Gao, Y. Liu, and H. Zhang, "A novel bat algorithm with habitat selection and doppler effect in echoes for optimization," *Expert Syst. Appl.*, vol. 42, nos. 17–18, pp. 6350–6364, 2015.

[32] A. N. Sloss and S. Gustafson, "2019 evolutionary algorithms review," in *Genetic Programming Theory and Practice XVII*. Cham, Switzerland: Springer, 2020, pp. 307–344.

[33] M. Kaur and D. Singh, "Multi-modality medical image fusion technique using multi-objective differential evolution based deep neural networks," *J. Ambient Intell. Humanized Comput.*, pp. 1–11, Aug. 2020. doi: 10.1007/s12652-020-02386-0.

[34] C. L. P. Chen, T. Zhang, L. Chen, and S. C. Tam, "I-ching divination evolutionary algorithm and its convergence analysis," *IEEE Trans. Cybern.*, vol. 47, no. 1, pp. 2–13, Jan. 2017.

[35] X. Meng, Y. Liu, X. Gao, and H. Zhang, "A new bio-inspired algorithm: Chicken swarm optimization," in *Proc. Int. Conf. Swarm Intell.*, 2014, pp. 86–94.

[36] X.-B. Meng, X. Z. Gao, L. Lu, Y. Liu, and H. Zhang, "A new bio-inspired optimisation algorithm: Bird swarm algorithm," *J. Exp. Theor. Artif. Intell.*, vol. 28, no. 4, pp. 673–687, 2016.

[37] A. Camero, H. Wang, E. Alba, and T. Bäck, "Bayesian neural architecture search using a training-free performance metric," 2020. [Online]. Available: arXiv:2001.10726.

[38] P. Liashchynskyi and P. Liashchynskyi, "Grid search, random search, genetic algorithm: A big comparison for NAS," 2019. [Online]. Available: arXiv:1912.06059.

[39] T. Zhang, C. L. P. Chen, L. Chen, X. Xu, and B. Hu, "Design of highly nonlinear substitution boxes based on I-Ching operators," *IEEE Trans. Cybern.*, vol. 48, no. 12, pp. 3349–3358, Dec. 2018.

[40] T. Zhang, X. Wang, X. Xu, and C. L. P. Chen, "GCB-Net: Graph convolutional broad network and its application in emotion recognition," *IEEE Trans. Affect. Comput.*, early access, Aug. 27, 2019, doi: 10.1109/TAFFC.2019.2937768.

[41] Q. Xu, M. Zhang, Z. Gu, and G. Pan, "Overfitting remedy by sparsifying regularization on fully-connected layers of CNNs," *Neurocomputing*, vol. 328, pp. 69–74, Feb. 2019.

[42] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, no. 3, pp. 229–256, 1992.

[43] X.-B. Meng, H.-X. Li, and X.-Z. Gao, "An adaptive reinforcement learning-based bat algorithm for structural design problems," *Int. J. Bio-Inspired Comput.*, vol. 14, no. 2, pp. 114–124, 2019.

[44] Q. Fan, J. M. Zurada, and W. Wu, "Convergence of online gradient method for feedforward neural networks with smoothing $L_{1/2}$ regularization penalty," *Neurocomputing*, vol. 131, pp. 208–216, May 2014.
[45] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014. [Online]. Available: arXiv:1412.6980.
[46] X. Dong and Y. Yang, "NAS-Bench-201: Extending the scope of reproducible neural architecture search," 2020. [Online]. Available: arXiv:2001.00326.
[47] Y. Lecun, F. J. Huang, and L. Bottou, "Learning methods for generic object recognition with invariance to pose and lighting," in *Proc. IEEE Comput. Soci. Conf. Comput. Vis. Pattern Recognit.*, vol. 2, 2004, pp. 97–104.
[48] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Dept. Comput. Sci., Univ. Toronto, Toronto, ON, Canada, Rep. TR-2009, 2009.
[49] P. Chrabaszcz, I. Loshchilov, and F. Hutter, "A downsampled variant of imagenet as an alternative to the cifar datasets," 2017. [Online]. Available: arXiv:1707.08819.
[50] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, no. 2, pp. 281–305, 2012.
[51] L. Li and A. Talwalkar, "Random search and reproducibility for neural architecture search," in *Proc. 35th Uncertainty Artif. Intell. Conf.*, 2020, pp. 367–377.
[52] N. Awad, N. Mallik, and F. Hutter, "Differential evolution for neural architecture search," 2020. [Online]. Available: arXiv:2012.06400.
[53] C. L. P. Chen and Z. Liu, "Broad learning system: An effective and efficient incremental learning system without the need for deep architecture," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 1, pp. 10–24, Jan. 2018.
[54] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Honolulu, HI, USA, 2017, pp. 4700–4708.
[55] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical representations for efficient architecture search," 2017. [Online]. Available: arXiv:1711.00436.
[56] S. Xie, H. Zheng, C. Liu, and L. Lin, "SNAS: Stochastic neural architecture search," 2018. [Online]. Available: arXiv:1812.09926.

Tong Zhang (Member, IEEE) received the B.S. degree in software engineering from Sun Yat-sen University, Guangzhou, China, in 2009, and the M.S. degree in applied mathematics and the Ph.D. degree in software engineering from the University of Macau, Macau, China, in 2011 and 2016, respectively.

He is currently an Associate Professor with the School of Computer Science and Engineering, South China University of Technology, Guangzhou, also with Pazhou Lab, Guangzhou, China. He has been working in publication matters for many IEEE conferences. His research interests include affective computing, evolutionary computation, neural network, and other machine learning techniques and their applications.

Chunyu Lei received the B.S. degree in computer science and technology from Zhengzhou University, Zhengzhou, China, in 2020. He is currently pursuing the M.E. degree in computer science and technology with the South China University of Technology, Guangzhou, China.

His research interests mainly include neural architecture search, computational intelligence, and transfer learning.

Zongyan Zhang received the B.S. degree in computer science and technology from the South China University of Technology, Guangzhou, China, in 2020, where he is currently pursuing the Ph.D. degree in computer science and technology.

His research interests mainly include multimodal optimization, neural architecture search, and computational intelligence.

Xian-Bing Meng received the Ph.D. degree from Central South University, Changsha, China, in 2019.

He is currently a Postdoctoral Researcher with the School of Computer Science and Engineering, South China University of Technology, Guangzhou, China. His research interests include computational intelligence, machine learning, and intelligent modeling.

C. L. Philip Chen (Fellow, IEEE) received the M.S. degree in electrical and computer science from the University of Michigan at Ann Arbor, Ann Arbor, MI, USA, in 1985, and the Ph.D. degree in electrical and computer science from Purdue University, West Lafayette, IN, USA, in 1988.

He is the Chair Professor and the Dean of the School of Computer Science and Engineering, South China University of Technology, Guangzhou, also with Pazhou Lab, Guangzhou, China. Being a Program Evaluator of the Accreditation Board of Engineering and Technology Education in the U.S., for computer engineering, electrical engineering, and software engineering programs, he successfully architects the University of Macau's Engineering and Computer Science programs receiving accreditations from Washington/Seoul Accord through Hong Kong Institute of Engineers, Hong Kong, of which is considered as his utmost contribution in engineering/computer science education for Macau as the former Dean of the Faculty of Science and Technology. His current research interests include cybernetics, systems, and computational intelligence.

Dr. Chen was a recipient of the 2016 Outstanding Electrical and Computer Engineers Award from his alma mater, Purdue University in 1988. He received the IEEE Norbert Wiener Award in 2018 for his contribution in systems and cybernetics, and machine learning. He is also a highly cited researcher by Clarivate Analytics in 2018, 2019, and 2020. He is currently the Editor-in-Chief of the IEEE TRANSACTIONS ON CYBERNETICS, an Associate Editor of the IEEE TRANSACTIONS ON ARTIFICIAL INTELLIGENCE and IEEE TRANSACTIONS ON FUZZY SYSTEMS. He was the IEEE Systems, Man, and Cybernetics Society President from 2012 to 2013, and the Editor-in-Chief of the IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS: SYSTEMS from 2014 to 2019. He was the Chair of TC 9.1 Economic and Business Systems of International Federation of Automatic Control from 2015 to 2017. He is a Fellow of AAAS, IAPR, CAA, and HKIE, and a member of Academia Europaea and European Academy of Sciences and Arts.