Neural Architecture Search with Progressive Evaluation and Subpopulation Preservation

Yu Xue, *Senior Member, IEEE*, Jiajie Zha, Danilo Pelusi, *Member, IEEE*, Peng Chen, Tao Luo, *Member, IEEE*, Liangli Zhen, Yan Wang, and Mohamed Wahib

***Abstract*—Neural architecture search (NAS) is an effective approach for automating the design of deep neural networks. Evolutionary computation (EC) is commonly used in NAS due to its global optimization capability. However, the evaluation phase of architecture candidates in EC-based NAS is compute-intensive, limiting its application for many real-world problems. To overcome this challenge, we propose a novel progressive evaluation strategy for the evaluation phase in convolutional neural network architecture search, in which the number of training epochs of network individuals is progressively increased. In addition, a subpopulation preservation strategy is proposed to preserve medium-size and large-size architectures to avoid prematurely discarding networks that may not perform well in the early stages but have the potential to excel with further optimization. Our proposed algorithm reduces the computational cost of the evaluation phase and promotes population diversity and fairness by preserving promising networks based on their distribution. We evaluate the proposed progressive evaluation and subpopulation preservation of NAS (PEPNAS) algorithm on the CIFAR10, CIFAR100, and ImageNet benchmark datasets, and compare it with 36 state-of-the-art algorithms, including manually designed networks, reinforcement learning (RL) algorithms, gradient-based algorithms, and other EC-based ones. The experimental results demonstrate that PEPNAS effectively identifies networks with competitive accuracy while also markedly improving the efficiency of the search process. For instance, PEPNAS discovers the architecture on CIFAR10 with a low-error rate of 2.38% using only 0.7 GPU days. We directly adopt the searched architecture for the image classification on the CIFAR100 and ImageNet datasets, which achieves the top 1 error rates of 16.46% and 26.25%, respectively. The code is available at https://github.com/chajiajie/PEPNAS.**

***Index Terms*—Convolutional neural network (CNN), evolutionary computation (EC), genetic algorithm (GA), neural architecture search (NAS), progressive evaluation.**

Manuscript received 18 August 2023; revised 15 November 2023, 13 January 2024, and 14 March 2024; accepted 15 April 2024. Date of publication 25 April 2024; date of current version 8 October 2025. This work was supported in part by the National Natural Science Foundation of China under Grant 62376127, Grant 61876089, Grant 61876185, Grant 61902281, and Grant 61403206; in part by the Natural Science Foundation of Jiangsu Province under Grant BK20141005; and in part by the Natural Science Foundation of the Jiangsu Higher Education Institutions of China under Grant 14KJB520025. This article was approved by Associate Editor Y. Sun. *(Corresponding author: Yu Xue.)*

Yu Xue and Jiajie Zha are with the School of Software, Nanjing University of Information Science and Technology, Nanjing 210044, China (e-mail: 002493@nuist.edu.cn; 20211249386@nuist.edu.cn).

Danilo Pelusi is with the Faculty of Communication Science, University of Teramo, 64100 Teramo, Italy (e-mail: dpelusi@unite.it).

Peng Chen and Mohamed Wahib are with RIKEN-CCS, Kobe 650-0047, Japan (e-mail: peng.chen@a.riken.jp; mohamed.attia@riken.jp).

Tao Luo, Liangli Zhen, and Yan Wang are with the Institute of High Performance Computing, Agency for Science, Technology, and Research (A*STAR), Singapore (e-mail: luo_tao@ihpc.a-star.edu.sg; zhenll@ihpc.a-star.edu.sg; wangyan@ihpc.a-star.edu.sg).

Digital Object Identifier 10.1109/TEVC.2024.3393304

I. Introduction

DEEP learning [1] has been widely used in various fields, including speech recognition [2], image recognition [3], semantic segmentation [4], [5], and natural language processing [6], [7]. Convolutional neural networks (CNNs), a widely utilized type of network in deep learning, typically comprises convolutional, pooling, and fully connected layers. A deep learning model's performance is determined by its architecture and weight parameters. As a result, many researchers in the field of deep learning have dedicated themselves to designing new architectures. To date, they have designed excellent neural network architectures, such as AlexNet [8], VGG [9], GoogleNet [10], ResNet [11], [12], and DenseNet [13]. However, designing such architectures often requires expertise in the field of CNNs and manual adaptation by a large number of experts, making the process time-consuming and labor-intensive. Furthermore, the design of neural network architectures must be tailored to a specific task, and manually designed architectures that perform well on one task may not generalize well to other tasks. Therefore, the technology of neural architecture search (NAS) [14], which automates the design of neural architectures, has gained significant attention in deep learning, effectively addressing the limitations of manually designed architectures.

NAS algorithms could be mainly divided into three categories: 1) reinforcement learning (RL)-based algorithms, which sample subnetworks using RNN and obtain the reward by training on a validation set [15], [16]; 2) gradient-based algorithms, such as DARTS and DARTS+ [17], [18], which use gradients to search for architectures to improve search efficiency; and 3) evolutionary computation (EC)-based algorithms, such as AE-CNN, large-scale Evo, CARS, and CNN-GA [19], [20], [21], [22]. Although RL-based algorithms have achieved promising results, they are computationally expensive, requiring hundreds of GPUs to complete the search process [15]. Gradient-based search algorithms are more efficient. However, most gradient-based algorithms rely on a supernet and require human involvement. For example, Liu et al. [17] addressed the NAS problem in a differentiable way by successively relaxing the architecture representation and using gradients for the search process. Notably,

Xue and Qin [23] used an attention mechanism to select partial channels with higher weights to be sent to the operation for processing, and channels that are not selected are concatenated with the processed channels, which reduces redundant features and improves search efficiency. Despite these innovations, the gradient-based algorithms suffer from memory overhead and improper relaxation.

Recently, numerous researchers in the field of EC have contributed significantly to the development of NAS algorithms. For example, Stanley and Miikkulainen introduced the evolutionary neural network with enhanced topology (NEAT) [24], which not only optimizes the network weights but also modifies the network topology. In addition, Real et al. [20] proposed the Large-Scale Evo, which is able to find high-performing architectures on CIFAR10 and CIFAR100 using mutation operators only. Although this algorithm is effective, it requires huge computing resources for the evaluation of individuals' performance as the number of evolutionary generations increases. Sun et al. [19] designed an efficient encoding strategy based on ResNet blocks and DenseNet blocks, which enables the discovery of high-performing architectures with limited computing resources. Xie and Yuille [25] performed the search process on small datasets and transferred high-performance architectures to large-scale visual recognition.

Search strategies affect the performance of the searched architecture significantly, so many scholars have made outstanding contributions to the evolutionary search strategies. For example, Real et al. [26] improved the tournament selection strategy by eliminating older individuals and increasing the probability of selecting younger ones, thus ensuring the diversity of the population during the evolutionary process. Xue et al. [27] proposed a self-adaptive mutation mechanism to guide the direction of population evolution, which makes the direction of population evolution purposeful. Sun et al. [28] proposed polynomial mutation to reduce randomness in the mutation process. To sum up, these contributions have advanced the state-of-the-art (SOTA) in NAS and demonstrated the effectiveness of EC in finding optimal network architectures.

The evaluation phase is a major bottleneck in EC, particularly for NAS where the training of deep neural networks is computationally expensive. In most NAS algorithms, each architecture is trained on a training set, and then its performance is evaluated on a validation set, which is computationally expensive and inefficient for searching. To address this challenge, various acceleration methods have been proposed. One class of methods includes surrogate models [29], [30] and memory methods [28], which propose to evaluate fewer individuals. Another class of methods includes weight sharing [21], [31], early stopping mechanisms [18], [32], and weight inheritance [33], which propose to reduce the computational cost of individual evaluations. These methods have demonstrated a reduction in the computational overhead associated with NAS, making them more practical for many real-world applications.

Searching for high-precision architectures is still a challenge for existing methods. In the evolutionary process, most accelerated evaluation methods lead to the problem of individual evaluation bias, thus resulting in the premature discarding of some promising individuals and reduced diversity of the population. This often leads to finding only local high-performance solutions. In this article, we propose an efficient NAS algorithm that considers the correlation between the ranking of individuals in a population in terms of both the precision of the architecture at each training epoch and the final true precision of the architecture. To achieve this, we calculate the Kendall's Tau Rank Correlation Coefficient between the individuals' ranking in one epoch and the final individual's ranking in the last epoch, and our preliminary experimental results demonstrate a minimum Kendall's Tau Rank Correlation Coefficient of 0.45 on the CIFAR10, indicating a highly correlated ranking relevance of architectures in this search space. Building upon this finding, we introduce a progressive evaluation strategy that allows one to estimate an individual's performance after only a few training epochs. Furthermore, we gradually enhance the correlation of individual rankings during the evolutionary process by increasing the number of training iterations. Since the small architecture converges more quickly than the medium-size and large-size architectures under a few training epochs, the small architecture outperforms the medium-size and large-size architectures. However, the medium-size and large-size architectures perform better than the small architecture with more training epochs. The progressive evaluation strategy trains individuals with a few training epochs in the early stages, so some medium-size and large-size architectures are discarded prematurely. Thus, we propose a subpopulation preservation strategy to preserve some of the discarded individuals and allow them to compete with the individuals in the population during the evolutionary process. The contributions of this work are summarized as follows.

1) We propose a progressive evaluation strategy that takes advantage of the correlation between the accuracy of individuals trained with a small number of epochs and their final true accuracy. This strategy improves search efficiency by avoiding the training of many individuals that yield low performance and cannot achieve acceptable performance even with extended training. As generations progress, more promising individuals survive during evolution, and they are trained for more epochs to improve their performance.
2) We propose a subpopulation preservation strategy to maintain potentially promising individuals and ensure the population diversity, improving the evolutionary process and promoting fair competition for medium-size and large-size architectures.
3) We generate a large neural network by stacking the cell architecture searched on CIFAR10 for eight times, then transfer it to conduct image classification tasks on CIFAR100 and ImageNet, which exhibits competitive performance. The architecture searched by our proposed algorithm achieves an error rate of 2.38% on CIFAR10 using only 0.7 GPU days.

The remainder of this article is organized as follows. Section II presents a literature review of related work in the field of NAS. Section III illustrates the details of our proposed

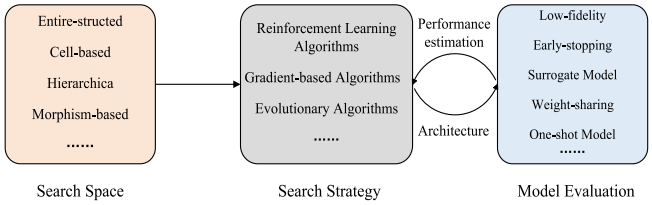


Fig. 1. Three main components of NAS: The NAS space, search strategy, and model evaluation methods.

algorithm. Section IV reports the experimental parameters and results. Finally, Section V draws conclusions and provides future research directions.

II. Literature Review and Related Work

In this section, we provide an overview of multiobjective NAS and review the strategies used to reduce the computational costs in evolutionary NAS (ENAS).

A. Evolutionary Neural Architecture Search

NAS is a popular research field in AutoML [34] that aims to automate the process of searching for neural network architectures. It consists of three main components, namely, search space, search strategy, and model evaluation, as shown in Fig. 1. There are numerous NAS algorithms dedicated to searching for the optimal CNN architecture.

Generally speaking, NAS aims to optimize two objectives: 1) the accuracy and 2) the complexity of the model [35], [36]. The number of parameters in the model is often used as a measure of complexity. Therefore, the search process can be considered as a bi-objective optimization problem

$$\begin{cases} \text{minimize}, \quad F(x) = (f_1(x), f_2(x)) \\ f_1(x) = f_{\text{valerror}}(x, D_{\text{val}}) \\ f_2(x) = f_{\text{complexity}}(x) \end{cases} \tag{1}$$

where $f_{\text{valerror}}(\cdot)$ denotes the error rate of architecture x on a validation set D_{val} and $f_{\text{complexity}}(\cdot)$ measures the parameter size of architecture x. X represents the population in the evolutionary process.

Nondominated sorting genetic algorithm II (NSGA-II) [37] is a genetic algorithm (GA) that utilizes genetic operators, nondominated sorting, crowding distance, and an elitist strategy to achieve an effective multiobjective search. One of the first evolutionary multiobjective algorithms used in NAS is NEMO [38], which uses NSGA-II to maximize the accuracy and minimize the inference time of the network. Another method proposed by Lu et al. [36] considered model performance and the number of floating-point operations as optimization objectives. In this article, our proposed algorithm is designed with the NSGA-II framework. NSGA-II mainly consists of the following steps: 1) population initialization; 2) individual fitness evaluation; 3) crossover and mutation operations; 4) offspring fitness evaluation (measuring error rate and model parameters); 5) nondominated sorting and crowding distance computation; 6) environment selection (including elitism strategy); and 7) repetition of steps 3)–6) until the termination condition is satisfied. However, the individual fitness evaluation process is computationally expensive, requiring complete training on a training set and evaluation on a validation set for each individual. For instance, AE-CNN [19] takes 27 GPU days and 36 GPU days on the CIFAR10 and CIFAR100 benchmarks, respectively, while genetic CNN consumes 17 GPU days [25] on the CIFAR10 dataset. Therefore, it is necessary to speed up the fitness evaluation process to reduce computational resource consumption.

B. Accelerated Methods for Model Evaluation

To reduce the computational cost of NAS, several methods have been proposed and can be categorized as follows.

1) Surrogate Model: The surrogate model-based ENAS methods aim to reduce the number of costly fitness evaluations by encoding the architecture and performance of individuals through encoding strategies, and then training a surrogate model to predict the performance of architectures. For instance, Sun et al. [29] used a random forest as a surrogate model to predict the fitness of newly generated individuals, effectively saving 68% of computational costs compared to AE-CNN without a surrogate model, while achieving an accuracy of 94.70% on the CIFAR10 dataset. To address the issue of insufficient training data, HAAP [30] encoded an architecture by using an adjacency matrix and a layer type list, thus generating more homogeneous architectures by exchanging layers in the layer list type.

2) Early Stopping: The early stop mechanism aims to save computational resources during the search process by terminating the training of inferior networks. For example, DARTS may produce architectures with many skip-connects when the number of search stages is too large. DARTS+ [18] uses an early stopping criterion that halts the search process when two or more skip-connections are presented in a cell. This approach has significantly improved the search efficiency. In ENAS, Suganuma et al. [32] introduced an early termination strategy based on a reference curve, which stops the training of a network if its accuracy curve is below the reference curve of excellent networks.

3) Reduced Training Set: To reduce training time, one approach is to use a training subset with a similar distribution to the original training set. For example, Liu et al. [39] explored this approach by training on a subset and using transfer learning on the original dataset. Since there are numerous benchmark datasets in the image classification domain, architectures can be evaluated on smaller datasets (e.g., CIFAR10) first and then applied to larger datasets (e.g., CIFAR100 and ImageNet [40]). This means that small datasets can be used as proxies for large datasets.

4) Population Reduction and Memory: Population reduction and population memory are two techniques used to improve the efficiency of evolutionary algorithms for ENAS. Population reduction involves reducing the size of the population during evolution, while population memory aims to prevent the training of duplicate individuals in the population. For instance, Fan et al. [41] proposed a $(\mu+\lambda)$ evolution strategy that divides evolution into three stages to balance computational efficiency and global search. The first stage uses

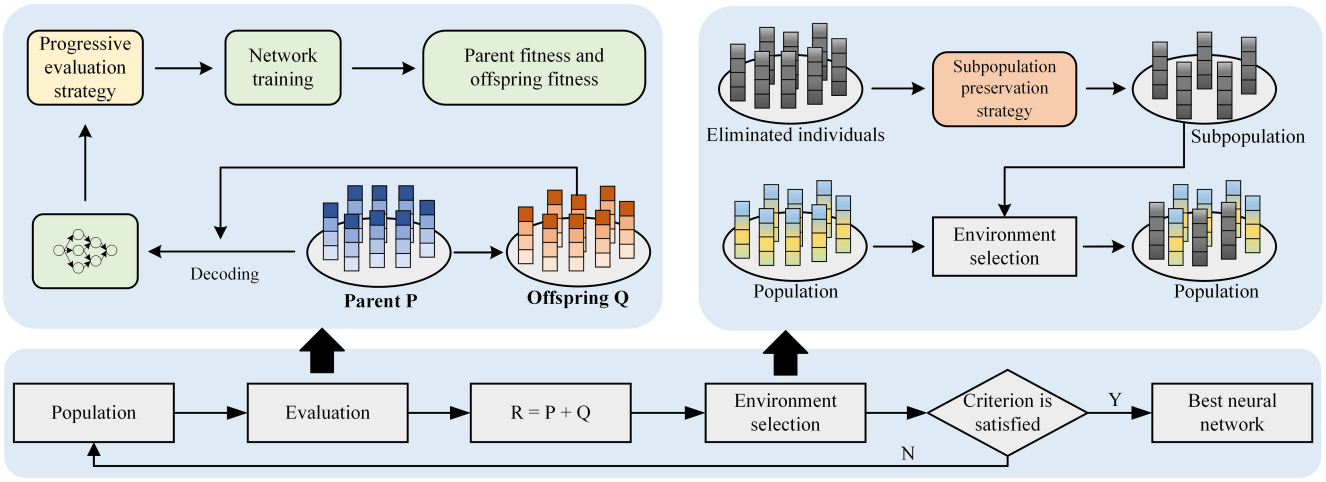


Fig. 2. Overview of the proposed framework (PEPNAS).

a large population to ensure thorough exploration, while the later stages employ smaller populations to reduce the number of evaluated individuals. Sun et al. [22] used hash coding to record each individual's network structure and fitness value, allowing them to avoid redundant training by directly querying the fitness value of duplicate individuals.

5) Weight Sharing and Weight Inheritance: Weight sharing techniques [31], [42], [43] are commonly used in NAS to reduce computational cost. These techniques are based on a supernet, which is a large neural network that contains all possible subnetworks. During the search process, subnetworks are sampled from the supernet and evaluated without further training. Weight inheritance is another technique that is used to reduce the number of evaluations required during the search process. In weight inheritance, the children individuals inherit some of the weights of the parent individuals. For instance, Zhang et al. [33] proposed to directly inherit some of the weights of the parent when using crossover operators, while mutation operators use exchange mutation to reduce the computational resources needed.

III. Proposed Algorithm

This section presents the details of our proposed progressive evaluation and subpopulation preservation of NAS (PEPNAS) for automatic CNN design. The proposed algorithm uses GA as the search strategy. The effectiveness of using GA as a search strategy in NAS has been demonstrated in [19] and [22]. To illustrate the search process, we provide an overview of the algorithm in Fig. 2. The initial population is first created, followed by training and evaluation of the CNN architectures. The evolutionary process then iteratively executes with the inclusion of progressive evaluation and subpopulation preservation strategies, until the stopping criterion is satisfied. Finally, the best-CNN architecture is output as the result of the algorithm.

The proposed algorithm for automatic CNN design differs from existing GA-based NAS algorithms in two main aspects: 1) architecture performance evaluation and 2) search strategy. Unlike conventional GA [19], [22], the proposed algorithm adopts a progressive evaluation strategy for CNN architecture performance evaluation, which progressively increases the number of training epochs of the architecture rather than performing a complete training. Besides, a subpopulation preservation strategy is proposed in the search strategy. It prevents the search process from converging onto small models and preserves medium-size and large-size CNN architectures.

The following sections will describe the search space, the encoding strategy, the proposed progressive evaluation strategy, and the subpopulation preservation strategy.

A. Search Space and Encoding Strategy

1) Search Space: The cell-based search space has been widely adopted for the automatic design of CNNs and has achieved great success [44], [45]. Designing an appropriate search space is essential for NAS to achieve good performance of NAS. The search space has a decisive effect on the quality of the architecture. In this article, a cell-based search space is used, where each node of the cell processes information flows using various operators. The final architecture is constructed by stacking the cells evolved by the algorithm [45].

In Fig. 3, each cell is represented by a directed acyclic graph (DAG) [25]. The nodes in the DAG represent operations, and the edges represent the flow of information, such as feature maps. Each node only chooses the input from the previous nodes, and nodes without a successor in the cell are concatenated to provide the final output. The proposed algorithm constructs two types of cells: 1) normal cells and 2) reduction cells. Different from normal cells, in the reduction cells, we follow the principle of halving the size of the feature map and doubling the number of channels [17], [45], [46], [47]. N in Fig. 3(b) represents the number of stacking normal cells, and the number next to the normal cell is the size of the feature map as it passes through the normal cell. During the search process, PEPNAS builds an architecture with 5 cells by taking the value of N as 1 to perform the search. When PEPNAS searches for the best normal and reduction cells, a large network architecture with

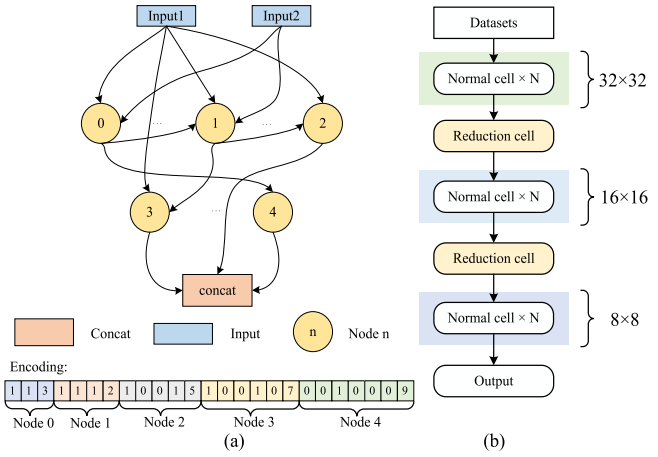


Fig. 3. Search space of PEPNAS. (a) Example cell represented by a DAG with five nodes. (b) Backbone network.

TABLE I
NODES OPERATION CODING SPACE. THE SPACE INCLUDES TWELVE TYPES OF COMPUTATIONAL OPERATIONS

Type of operation	Code
1×1 convolution	0
3×3 convolution	1
1×3 then 3×1 convolution	2
1×7 then 7×1 convolution	3
2×2 max pooling	4
3×3 max pooling	5
5×5 max pooling	6
2×2 average pooling	7
3×3 average pooling	8
5×5 average pooling	9
Identity	10
SELayer	11

$3N+2$ cells is constructed by stacking N normal cells, and the performance of the architecture is evaluated on the dataset.

2) Encoding strategy: In Fig. 3(a), a cell structure with five nodes is illustrated, where the numbers in the nodes represent their serial numbers, such as node 0 and node 1. The number of nodes is not fixed and can vary during the evolutionary process. We present only the cell with five nodes in Fig. 3. The details of the number of nodes are described in Section IV-C. Thus, each individual may have a different number of nodes, allowing our proposed algorithm to adaptively search for the best-CNN architecture with optimal depth.

The proposed encoding scheme encodes each node's connection in the cell structure with a binary sequence, where 1 indicates the selection of a certain information flow and 0 indicates nonselection. Table I lists the operations that a node can represent. In Table I, the "Type of operation" column represents the type of operation, and "Code" denotes the code corresponding to each type of operation. Since each node can only select the information flow from previous nodes, node 0 in Fig. 3(a) can only select the information flow from input 1 and input 2, and node 2 can only select the information flow from input 1, input 2, node 0 and node 1. As illustrated in Fig. 3(a), node 0 selects information flow from input 1 and input 2, and operation "3" is used in node 0, resulting in the binary sequence (1, 1, 3). Node 2, on the other hand, selects information flow from input 1 and node 1, and without selecting information flow from input 2 and node 0, with operation "5" being used, yielding the binary sequence (1, 0, 0, 1, 5). The encoding of the cell is obtained by concatenating the binary sequences of all nodes in the cell. Therefore, the total length of the cell encoding is $[M(M+5)/2]$, where M represents the number of nodes in the cell. In Fig. 3(a), the length of the cell encoding with five nodes is twenty-five.

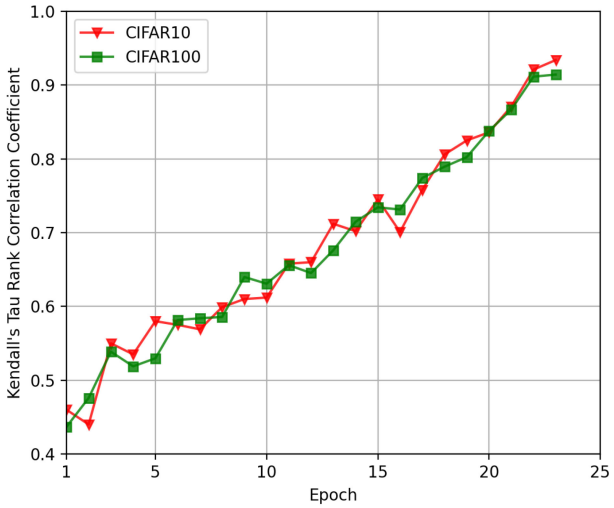


Fig. 4. Kendall's Tau Rank Correlation Coefficient calculation for 500 architectures sampled from the proposed search space. It proves that the ranking of each pair of individuals in the early stage is consistent with the final ranking.

B. Progressive Evaluation Strategy

Training a neural network completely on a dataset is computationally expensive and limits the number of possible architecture evaluations. Zheng et al. [47] studied the performance ranking of ResNet18, DenseNet, AlexNet, and LeNet networks at different epochs and found that the performance ranking is generally consistent.

We sample 500 architectures from our search space and calculate the Kendall's Tau Rank Correlation Coefficient (τ for short) to measure the correlation between their rankings under each training epoch and their final performance rankings [48]. First, we divide the training sets of CIFAR10 and CIFAR100 into training sets and validation sets at a ratio of 9 to 1. Then we use the training set to train the sampled architectures and record their validation accuracy ranking at each epoch. Finally, we compare the ranking of each epoch's architectures with the final ranking of the architectures to obtain the τ. The index $\tau \in [-1, 1]$, where $\tau = 1$ indicates a positive correlation, while $\tau = -1$ indicates a negative correlation. In Fig. 4, we can observe that the τ on the CIFAR10 and CIFAR100 datasets have correlations above 0.45 and 0.43 in the first epoch, respectively, which means that the ranking of individuals is moderately correlated. Based on this observation, we propose a progressive evaluation strategy.

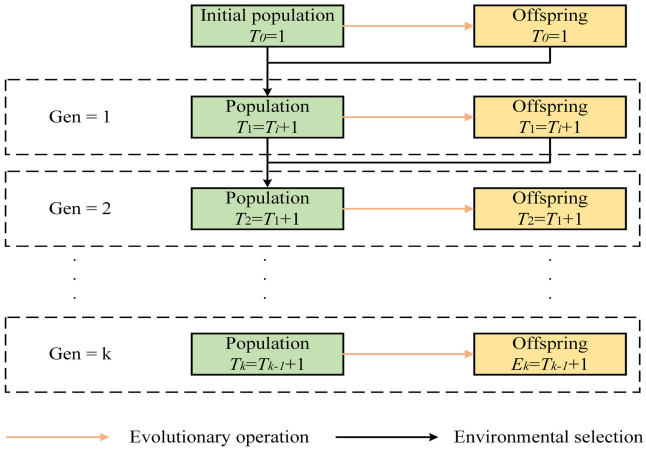


Fig. 5. Demonstration of the proposed progressive evaluation strategy. T_k represents the number of training epochs for each individual in the population.

As evolutionary generations progress, the population's individuals tend to perform relatively better, so we gradually increase the number of epochs to evaluate them more accurately. Fig. 5 illustrates our proposed progressive evaluation strategy. Initially, we evaluate each individual in the initial population after training one epoch. We then generate offspring using crossover and mutation operators and evaluate each offspring after training one epoch. The population of $Gen = 1$ is selected from the population and offspring by the nondominated ranking and crowding distance of NSGA-II. As evolutionary generations increase, we simultaneously increase the number of epochs used for evaluation. For example, our algorithm evaluates the offspring in the $Gen = 1$ after training for two epochs. Notably, individuals in the population of $Gen = 1$ are trained for one epoch. Since the population and offspring need to maintain the same training epochs to compete fairly, we preserve the weights of individuals in the population that were trained in the previous generation. Individuals in the population in $Gen = 1$ only need to be trained for one epoch under inheriting the previous weights, avoiding training the individuals in the population in $Gen = 1$ from scratch. In the evolutionary process, individuals in the early stages are not performing very well. This progressive training method avoids the unnecessary training of some inferior individuals in the early stages compared to the complete training method [19], [22], thus effectively reducing computational resource consumption.

C. Subpopulation Preservation Strategy

In progressive evaluation, individuals in the current generation have fewer training epochs than those in the next generation, which may lead to evaluation bias and the premature discarding of some excellent individuals in the current generation, especially some potentially high-performing medium-size and large-size architectures. This phenomenon is discovered by the pNSGA-III algorithm proposed in CARS [21], which recognizes that it is hard for these medium-size and large-size architectures to reach convergence under a small number of epochs and show weak performance in fewer training epochs. In contrast to small architectures, medium-size and large-size architectures tend to perform better in later stages. To preserve potentially good individuals, we propose the subpopulation preservation strategy that mitigates the problem of the small model trap phenomenon as presented in [21] to some extent.

Algorithm 1 Subpopulation Preservation Strategy

Input: Population size: S, Population: P, Eliminated population: E, Offspring: Q, Epoch (Initial): 1, Current generation: g, Training dataset D_{train}, Validation dataset D_{valid}.
Output: Population: P, Eliminated individuals: E.
1: $P, E_g \leftarrow$ Select S individuals from $P \bigcup Q$ by environment selection of NSGA-II;
2: **if** $g >= 1$ **then**
3: Train individuals in E_{g-1} for 1 epoch on D_{train} and evaluate them on D_{valid};
4: $Pop \leftarrow$ Select S individuals from $E_g \bigcup E_{g-1}$ by environment selection of NSGA-II;
5: $Avg_{pop} \leftarrow$ Calculate the average length of individuals in Pop;
6: Subpopulation $\leftarrow$ Select individuals from Pop with the length of an individual greater than Avg_{pop};
7: $P, E \leftarrow$ Select S individuals from $P \bigcup$ Subpopulation by the environment selection of NSGA-II;
8: **end if**
9: Return P,E.

The main goal of the subpopulation preservation strategy which is shown in Algorithm 1 is to ensure that potentially good individuals are not prematurely discarded during the evolutionary process. In Fig. 6, the population in $Gen = g - 1$ represents the population in GA, and the eliminated individuals (E_{g-1} for short) represent the individuals that are not selected in the environment selection in $Gen = g - 1$. In traditional GA, environment selection is performed in the population and offspring, selected individuals are readded to the population, and unselected individuals are directly eliminated by the GA. In the progressive evaluation strategy, some potentially high-performing medium-size and large-size architectures cannot be evaluated accurately with a few training epochs, and thus are eliminated by the GA's environment selection. The subpopulation preservation strategy preserves the eliminated individuals, i.e., E_{g-1} and E_g in Fig. 6. Meanwhile, in order to select some individuals with good performance from the eliminated individuals, the subpopulation preservation strategy provides the opportunity for E_{g-1} and E_g to compete. Note that in the progressive evaluation strategy, the individuals in E_g are trained for one more epoch than the individuals in E_{g-1}. First, we need to train one more epoch for the individual in E_{g-1} by inheriting the weights trained by the individual in E_{g-1} under $Gen = g - 1$ to ensure a fair competition between E_{g-1} and E_g. Second, we use the NSGA-II [37] based on error rate and number of parameters to select S (S represents the size of the population) individuals from E_{g-1} and E_g, which are stored in Pop. Pop is represented as follows:

$$Pop = \left(Ind^1, Ind^2, \ldots, Ind^{S-1}, Ind^S\right) \tag{2}$$

where Ind^i represents the individuals in Pop. Third, we calculate the average length of the individuals in Pop, which is named Avg_{pop}. Then, we compare the length of each individual in Pop with Avg_{pop}. If it is greater than Avg_{pop}, we add this individual to the subpopulation. This process is repeated until

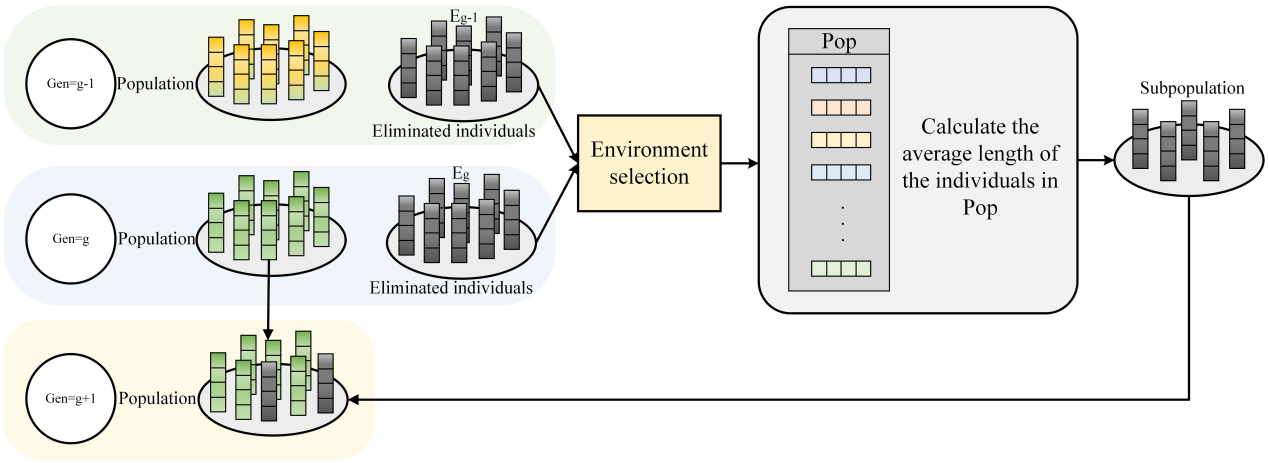


Fig. 6. Algorithmic framework for subpopulation preservation strategy.

all individuals are compared, and the final subpopulation is constructed. This approach enables the proposed algorithm to preserve as many medium-size and large-size architectures as possible that may be discarded in the early stage.

Since medium-size and large-size architectures are evaluated with a few training epochs, thus resulting in evaluation bias and their performance lagging behind some small architectures. The subpopulation preservation strategy preserves medium-size and large-size architectures in the subpopulation. In the progressive evaluation strategy, the accuracy of evaluating individuals in the subpopulation is improved by increasing the training epochs of the individuals in the subpopulation. Thus, the performance of the individuals in the subpopulation with a large number of training epochs will exceed the performance of the individuals in the population. Finally, in order to preserve these medium-size and large-size architectures to rejoin the population and evolve, we used NSGA-II environment selection in the subpopulation and population. This increases the diversity of the population in the next iteration and preserves the medium-size and large-size architectures with excellent performance as much as possible. Algorithm 1 outlines the main steps of our approach, which allows these individuals to showcase their excellent performance in later stages.

D. Overall Framework

Algorithm 2 presents the pseudocode for PEPNAS. To optimize both accuracy and number of parameters, NSGA-II is used, which is highly effective for bi-objective problems. Fig. 2 illustrates the two main features of our approach: 1) the progressive evaluation strategy and 2) the subpopulation preservation strategy. Together, these strategies enable efficient resource usage during the evolution phase.

In Algorithm 2, PEPNAS starts by initializing a population of individuals randomly (line 3), then decodes and trains these individuals for one epoch. To efficiently evaluate the individuals, a progressive evaluation strategy is used that increases the number of epochs with the increase of generations (line 3). Next, a roulette wheel selection is used to select parent individuals, and the parent individuals generate offspring using crossover and mutation operators (lines 5 and 6). Then, train and evaluate individuals in offspring Q (line 7). The parents and offspring then undergo environment selection to select the population for the next generation while preserving the eliminated individuals by using the subpopulation preservation strategy (line 8). This process is repeated until the termination condition is satisfied (lines 9 and 10), and it finally outputs the best cell. Finally, we stack the normal cell eight times to achieve the final CNN architecture, as detailed in Section IV-F.

Algorithm 2 Pseudocode of the Proposed PEPNAS

Input: Population size: S, Population: P, Eliminated population: E, Epoch (Initial): 1, Maximum number of generations: Gen, Training dataset: D_{train}, Validation dataset: D_{valid}
Output: Population: P.
1: $P \leftarrow \emptyset, E \leftarrow \emptyset, g \leftarrow 0$;
2: $P \leftarrow$Initialize a population with a size of S;
3: Train and evaluate the individuals in P by our progressive evaluation on D_{train} and D_{valid};
4: **while** $g < Gen$ **do**
5: $PS \leftarrow$ Select two parent individuals using roulette wheel selection from P;
6: $Q \leftarrow$ Generate two new offspring by crossover and mutation operators using PS;
7: Train and evaluate the individuals in Q for g epochs by progressive evaluation strategy on D_{train} and D_{valid}, respectively;
8: $P, E \leftarrow$ Select individuals using subpopulation preservation strategy(P, Q, E, g) in Algorithm 1;
9: $Epoch \leftarrow Epoch + 1$;
10: $g \leftarrow g + 1$;
11: **end while**
12: Return the best individual from P and decode it into the corresponding neural network.

IV. Experiment

This section evaluates the efficiency of PEPNAS in discovering the optimal neural network structure for the benchmark datasets. First, we describe the experimental setup, which includes the benchmark datasets, the other SOTA comparison algorithms, and parameter settings. Second, we present and analyze the effectiveness of the progressive evaluation strategy and the subpopulation preservation strategy. Third, we compare the algorithms based on test error, search cost (Section IV-C), and the number of parameters, followed by a

discussion of the findings. Finally, we compare the stacking times of cells in terms of test error rates on the CIFAR10 dataset.

A. Benchmark Datasets

We utilize three widely used datasets: 1) CIFAR10 [49]; 2) CIFAR100 [49]; and 3) ImageNet [40], to verify the effectiveness and efficiency of our proposed algorithm. CIFAR10 consists of 50 000 training images and 10 000 test images. This dataset is divided into ten classes, and each has 6,000 images with dimensions of 32×32. We evaluate the performance of PEPNAS on the CIFAR10 dataset and compare it with the other SOTA algorithms. CIFAR100 is a more challenging dataset comprising 100 classes, with 500 training images and 100 test images per class. ImageNet is a massive dataset comprising over 1.3 million images belonging to various categories. The images have high resolution and are standardized to a size of 224×224. Furthermore, we search the architecture on the CIFAR10 dataset and evaluate its performance on the CIFAR100 and ImageNet datasets by transferring only the architecture without the weight parameters of the architecture.

B. Baseline Algorithms

To demonstrate the superiority of PEPNAS, we compare it with other SOTA algorithms. We group the compared algorithms into three different categories as follows.

1) The first group comprises commonly used CNNs that are manually designed by human experts, such as ResNet [11], wide ResNet [50], DenseNet [13], VGG [9], SENet [3], and ShuffleNet [51].
2) The second group consists of various non-ENAS algorithms, including gradient-based and RL-based methods. The compared algorithms include NAS [15], NASNet [45], PNASNet [52], MetaQNN [53], ENAS [31], ProxylessNAS [54], DARTS [17], P-DARTS [55], and Firefly [56].
3) The third group includes SOTA ENAS algorithms for CNN architecture design, such as large-scale evolution [20], hierarchical evolution [57], AmoebaNet [26], LEMONADE evolution [35], NSGANet [36], AE-CNN [19], CARS [21], SI-EvoNet-S [33], and MSNAS [58].

C. Experimental Configuration

This section provides detailed information of the parameter settings for PEPNAS, which is divided into two parts: 1) evolutionary search phase and 2) evaluation phase. The evolutionary parameter settings follow standard practices in EC. However, since the progressive evaluation method may discard some good medium-size and large-size architectures, we employ a subpopulation preservation strategy to preserve some of these individuals, whose effect is discussed in Section IV-D. The summary of evolutionary parameter settings for PEPNAS is presented in Table II.

For the search process, we adopted the practice suggested in [19] and used 90% of the CIFAR10's training set as the training set and the remaining 10% as the validation set. The number of neural nodes in the cell is variable (ranging from 5 to 12) following [59], and we set the number of initial convolutional channels to 16, as is common in NAS settings [17]. We optimize the network weights using the stochastic gradient descent with momentum (SGDM) [46] with an initial learning rate of 0.1 and a single period cosine decay learning rate schedule. The corresponding configurations, including population size and batch size, are provided in Table II.

TABLE II
HYPERPARAMETER SETTINGS IN PEPNAS

Parameters	Evolution	Evaluation
Population size	20	-
Generations	25	-
Number of nodes	[5,12]	-
Training set rate	0.9	1
Initial channels	16	44
Train batch size	128	80
Evaluation batch size	500	500
Epoch	1(initial)	900
Optimizer	SGDM	SGDM
Momentum	0.9	0.9
Weight decay	3e-4	5e-4
Initial learning rate	0.1	0.025
Learning rate schedule	Cosine Annealing	Cosine Annealing

We conduct two experimental studies. The first study involves comparing our approach with the other SOTA methods, including many NAS algorithms for automatically designing CNN architectures. The second study demonstrates the effectiveness of our approach in preventing the small model trap problem by subpopulation preservation strategy.

The experiments are conducted on a single NVIDIA GeForce RTX 3090. To compare the search cost, we use "GPU days" as a metric [19], which is calculated by multiplying the number of GPU cards by the execution time in days.

D. Effectiveness of Progressive Evaluation and Subpopulation

We conduct an experiment to examine the impact of the proposed subpopulation preservation strategy on population distribution. We perform a 25-generation evolutionary search on the CIFAR10 dataset under both the subpopulation preservation strategy and a variant without it, both of which use a progressive evaluation strategy. The aim is to measure any changes in population distribution resulting from the inclusion of the subpopulation preservation strategy.

Fig. 7 illustrates the distribution of the population under different strategies. Our study reveals that the progressive evaluation strategy without the subpopulation preservation strategy only generates individuals of size below 0.06 MB. In contrast, the progressive evaluation strategy with subpopulation preservation strategy preserves medium-size and large-size architectures, which is evident from the black points in Fig. 7. The largest architecture in the population, with a size of 0.0879 MB, performs the best in terms of error rate. Moreover, several other larger architectures demonstrate good performance. When an individual's size is around 0.07 MB, the subpopulation preservation strategy also preserves more medium-size architectures with outstanding

TABLE III
COMPARISON OF PEPNAS WITH SOTA ALGORITHMS ON CIFAR10 DATASETS BASED ON TEST ERROR RATE, NUMBER OF WEIGHTS, AND COMPUTATIONAL COST

Architectures	Test Error Rate (%)	# GPU days	# Params	Search method
VGG [9]	6.66	-	28.05M	Manual
Pre-ResNet [12]	4.64	-	10.3M	Manual
ResNet-110 [11]	6.43	-	1.7M	Manual
Wide-ResNet [50]	4.17	-	36.5M	Manual
DenseNet [13]	3.46	-	25.6M	Manual
SENet [3]	4.05	-	11.2M	Manual
MobileNetV2 [61]	5.44	-	2.1M	Manual
ShuffleNet [51]	9.13	-	1.06M	Manual
NAS V3 [15]	4.47	22400	7.1M	RL
NASNet-A [45]	3.41	22400	3.3M	RL
Block-QNN-S [16]	3.3	90	6.1M	RL
MetaQNN [53]	6.92	90	11.2M	RL
ENAS [31]	2.89	0.45	4.6M	RL
ProxylessNAS [54]	2.08	1500	5.7M	RL
PNASNet [52]	3.32	150	3.2M	SMBO
MdeNAS [62]	2.55	0.16	3.61M	MDL
DARTS [17]	2.72	1	3.4M	GD
P-DARTS [55]	2.50	0.3	3.4M	GD
PC-DARTS [46]	2.57	0.1	3.6M	GD
SNAS [42]	2.83	1.5	2.8M	GD
Firefly [56]	2.73	1.5	3.3M	GD
AdaptNAS-S [61]	2.50	2	5.3M	GD
GibbsNAS+cutout [62]	2.53	0.5	4.1M	GD
BPC-DARTS [46]	2.57	0.1	3.6M	GD
SGAS+PT [63]	2.46	0.29	3.9M	GD
Large-Scale Evo [20]	5.4	2750	40.4M	EA
Hierarchical Evo [57]	3.63	300	15.7M	EA
AmoebaNet-B [26]	2.55	3150	2.8M	EA
LEMONADE [35]	2.58	90	13.1M	EA
NSGANet [36]	2.75	4	3.3M	EA
AE-CNN [19]	4.3	27	2.0M	EA
CARS [21]	2.62	0.4	3.6M	EA
EcoNAS [64]	2.60	8	2.9M	EA
SI-EvoNet-S [33]	2.69	0.458	1.84M	EA
MSNAS [58]	2.53	0.23	3.25M	EA
MOEA-PS [65]	2.77	2.6	3.0M	EA
PEPNAS (ours)	**2.38**	**0.7**	**4.23M**	**EA**

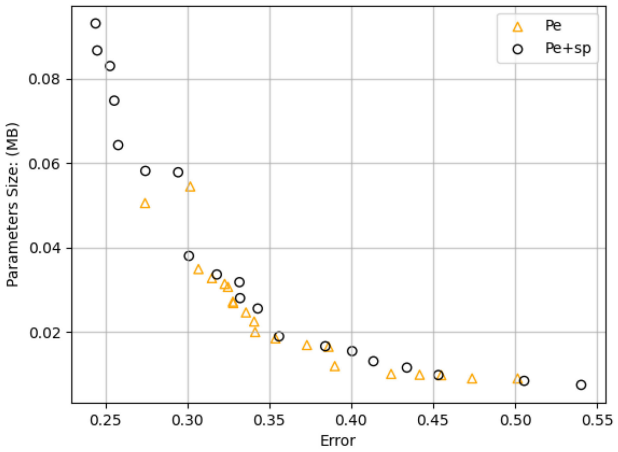


Fig. 7. Comparison of our proposed algorithm is conducted on the CIFAR10 dataset using two strategies: the progressive evaluation strategy (Pe) and the progressive evaluation strategy with the addition of the subpopulation preservation strategy (Pe+Sp).

performance. The addition of the subpopulation preservation strategy enhances the overall performance of PEPNAS and facilitates the discovery of potentially superior individuals during the evolutionary process, compared to using only the progressive evaluation strategy.

E. Results and Discussion

In this section, we provide the results of our algorithm and compare them with those of other SOTA algorithms. Tables III–V summarize the best-test error, number of parameters, and search cost results of the algorithms on the CIFAR10, CIFAR100, and ImageNet datasets. The "Search Method" column in these tables denotes the method used, with "MANUAL" referring to a manually designed method, "SMBO" referring to the sequential model-based global optimization, and "MDL" referring to the multinomial distribution learning. The results obtained by our proposed algorithm are presented in bold in the last row of Tables III–V.

The results obtained by the proposed algorithm in three runs are introduced and analyzed. Table III summarizes the results obtained by the PEPNAS algorithm on the CIFAR10 dataset, which achieves an impressive test error rate of 2.38%. PEPNAS outperforms most of the peer competitors, including manually designed architectures, non-ENAS approaches, and ENAS approaches. Specifically, the test error

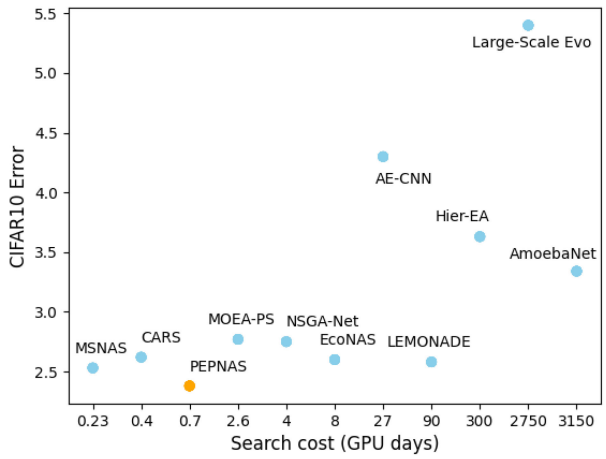


Fig. 8. Comparison of search efficiency between PEPNAS and other algorithms in terms of computational cost required on the CIFAR10 dataset.

obtained by the PEPNAS algorithm is between 1.08%-6.75% lower than that of the manually designed CNN architectures on the CIFAR10 dataset. Among the 17 non-ENAS methods, PEPNAS achieves the best error over 16 methods, including NAS V3, NASNet-A, Block-QNN-S, MetaQNN, ENAS, PNASNet, MdeNAS, DARTS, P-DARTS, PC-DARTS, SNAS, Firefly, AdaptNAS-S, GibbsNAS, BPC-DARTS, and SGAS+PT. While PEPNAS obtains slightly worse-test error compared to ProxylessNAS (−0.3), it uses a lower-cost search strategy and fewer parameters of the final architecture. Compared to ProxylessNAS, our PEPNAS algorithm requires less search cost and results in fewer architectural parameters.

In terms of search cost, the proposed PEPNAS requires only 0.7 GPU days to find neural architectures with a 2.38% error rate on the CIFAR10 dataset, which is faster than most other algorithms, including ten methods of NAS V3, NASNet, Block-QNN-S, MetaQNN, ProxylessNAS, PNASNet, DARTS, SNAS, Firefly, and AdaptNAS-S. Finally, compared to the EA-based NAS methods, we obtain the best-test error using fewer GPU days. The results obtained by our proposed algorithm on the CIFAR100 and ImageNet datasets are summarized in Tables IV and V, respectively, and the search method used in each comparison algorithm is listed in the "Search Method" column of the tables. The results obtained by our algorithm are presented in the last row of each table.

The final error rate and computational cost are shown in Fig. 8, from which we can see that the proposed progressive evaluation significantly reduces search time by up to 4500 times in terms of search cost. Besides, the proposed subpopulation strategy ensures the performance of PEPNAS in finding the architecture by preserving potential excellent individuals. Moreover, PEPNAS achieves the best-error rate of 2.38% using lower-search costs. Compared with these 10 algorithms, the proposed algorithm searches for better-CNN architectures with lower-search cost.

Fig. 9 shows the best-CNN architecture found by our proposed algorithm on the CIFAR10 dataset. We transferred the architecture directly from CIFAR10 to CIFAR100 for training, named PEPNAS (transfer). The proposed algorithm

TABLE IV
COMPARISON OF PEPNAS WITH SOTA ALGORITHMS ON CIFAR100 BASED ON TEST ERROR RATE, NUMBER OF WEIGHTS, AND COMPUTATIONAL COST

Architectures	Test Error (%)	# GPU Days	# Params	Search method
VGG [9]	32.05	-	28.05M	Manual
Pre-ResNet [12]	22.71	-	10.3M	Manual
ResNet-110 [11]	25.16	-	1.7M	Manual
Wide-ResNet [50]	20.5	-	36.5M	Manual
DenseNet [13]	17.18	-	25.6M	Manual
MobileNetV2 [60]	22.91	-	2.1M	Manual
ShuffleNet [51]	22.86	-	1.06M	Manual
Block-QNN-S [16]	17.05	90	6.1M	RL
MetaQNN [53]	17.14	100	11.2M	RL
PNASNet [52]	17.20	150	3.2M	SMBO
DARTS [17]	17.54	1	3.4M	GD
P-DARTS [55]	17.20	0.3	3.4M	GD
Large-Scale Evo [20]	23	2750	40.4M	EA
AmoebaNet-A [26]	18.93	3150	3.3M	EA
NSGANet [36]	20.74	8	11.6M	EA
AE-CNN [19]	20.85	36	5.4M	EA
CNN-GA [22]	20.53	40	4.1M	EA
AE-CNN+E2EPP [29]	22.02	10	20.9M	EA
SI-EvoNet-S [33]	15.7	0.813	3.32M	EA
MSNAS [58]	17.20	0.23	3,25M	EA
PEPNAS (transfer)	**16.46**	**0.7**	**4.24M**	**EA**
PEPNAS (search)	**16.05**	**0.85**	**4.34M**	**EA**

performs competitively on the CIFAR100 dataset as well, finding a robust and scalable CNN architecture. We note that hand-designed methods and non-ENAS approaches also transfer their architectures from CIFAR10 to CIFAR100 for training, as reported in their published articles. Table IV reports that the PEPNAS algorithm achieves excellent test error using the architecture found on CIFAR10, which is 0.72% to 15.59% lower than manually designed CNN architectures. CNN-GA and SI-EvoNet-S searched directly on the CIFAR100 dataset, and their architectures are better fit to the CIFAR100 dataset. We transfer the architecture found on CIFAR10 to CIFAR100 and obtain a classification error rate of 16.46%, which is 4.07% lower than CNN-GA and 6.54% lower than large-scale Evo, with a lower number of parameters. Although PEPNAS's best-test error is slightly lower than that of SI-EvoNet-S (-0.76%), the search time of PEPNAS is slightly faster. We also conducted a search on CIFAR100 and named it PEPNAS (search). PEPNAS (search) achieved an error rate of 16.05%, which is 4.8% lower than AE-CNN and 1.15% lower than MSNAS. These results demonstrate that PEPNAS's search results are competitive and illustrate the robustness and extensibility of the architectures.

In addition, we also evaluate the transferability of the PEPNAS algorithm by applying the architecture found on CIFAR10 to the ImageNet dataset. We train the network from scratch and report the top 1 and top 5 accuracies on the test set in Table V. PEPNAS achieves a top 1 accuracy of 73.75% and a top 5 accuracy of 91.78%, which is competitive with the SOTA methods, such as MobileNet, MnasNet, DARTS, and AmoebaNet-C. These results demonstrate the potential of our proposed algorithm in real-world applications. In summary, the proposed algorithm represents a promising direction for automatic CNN design.

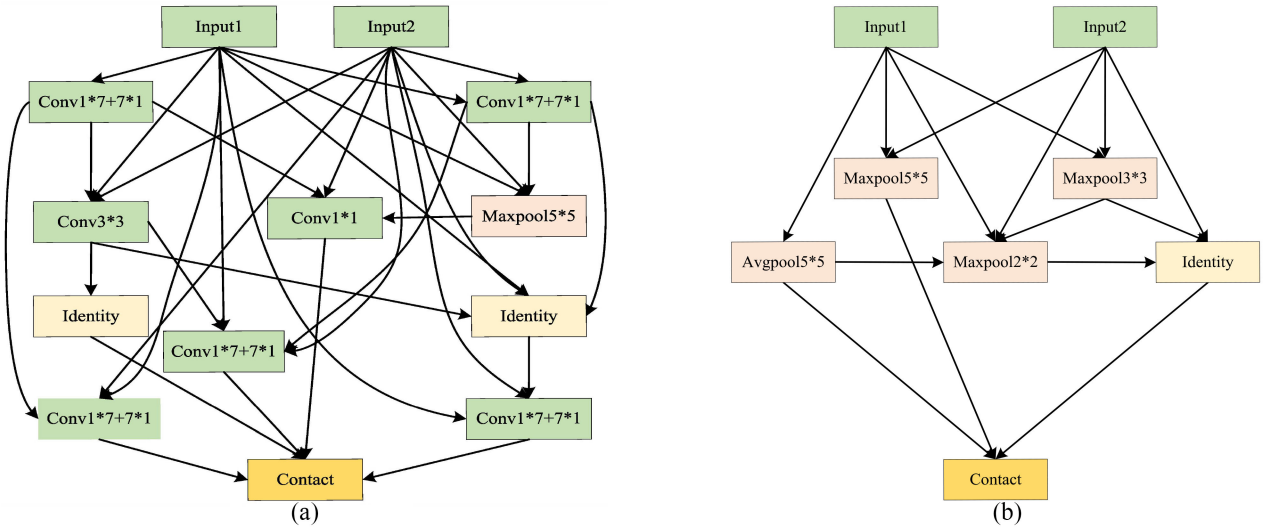


Fig. 9. Best normal and reduction cells found by our proposed PEPNAS on the CIFAR10 dataset. Regarding each cell, the solid black lines indicate information flow, and the nodes represent the operations. (a) Normal cell. (b) Reduction cell.

TABLE V
COMPARISON OF PEPNAS WITH SOTA ALGORITHMS ON IMAGENET DATASETS BASED ON TEST ERROR RATE, NUMBER OF WEIGHTS, AND COMPUTATIONAL COST

Architectures	Error Top1 (%)	Error Top5 (%)	# GPU Days	# Params	Search method
Inception-V1 [10]	30.2	10.1	-	6.6M	Manual
MobileNetV2 [59]	29.4	10.5	-	4.2M	Manual
ShuffleNet [51]	26.4	10.2	-	5M	Manual
NASNet-A [26]	26.0	8.4	2000	5.3M	RL
PNASNet [52]	25.8	8.7	225	4.7M	SMBO
DARTS [17]	26.7	8.7	4	4.7M	GD
ProxylessNAS [54]	24.9	7.5	8.3	7.1M	GD
SNAS [42]	27.3	9.2	1.5	4.3M	GD
AmoebaNet-C [26]	24.3	7.6	3150	7.6M	EA
CARS-E [21]	26.3	8.4	0.4	4.4M	EA
PEPNAS (ours)	**26.25**	**8.22**	**0.7**	**6.71M**	**EA**

TABLE VI
COMPARING THE EFFECT OF NORMAL CELL STACKING TIMES OF IN TERMS OF THE TEST ERROR RATES ON THE CIFAR10 DATASET

Number of Stacked Normal Cells	Test Error Rate (%)	# Params
2	2.99	1.07M
4	2.57	2.12M
6	2.47	3.17M
8	**2.38**	**4.23M**
10	3.06	5.28M

F. Analysis of Performance Versus Number of Stacked Cells

Based on the search space of cells, in the search phase, we construct the shallow neural network by stacking the normal cells sequentially for one time. In the evaluation phase, we build a deep neural network by stacking N normal cells sequentially. The design idea is inspired by ResNet [11], the same a stack of multiple ResNet blocks to build the deep neural network. The purpose of the reduction cell is to reduce the spatial resolution of the feature map and help the stacked architecture extract deeper features. The output layer of the architecture consists of a stack, including a global average pooling layer, a dropout layer, and a fully connected layer sequentially.

We also conduct ablation experiments on the number of stacking normal cells. First, the optimal normal cell is obtained by our algorithm. Next, we stack the optimal normal cell with different times. The "Number of Stacked Normal Cells" column in Table VI denotes the number of stacked normal cells. As shown in Table VI, we stack the normal cells 2, 4, 6, 8, and 10 times, respectively. Then, we train the network architectures with different stacking times on the CIFAR10 training set and obtain test error rates on the CIFAR10 test set. As can be seen from Table VI, when the normal cell is stacked eight times, we achieve the best-error rate of 2.38%. We also note that when the stacked normal cells are less than ten, the accuracy also increases as the number of stacked normal cells increases. However, stacking ten times the normal cell results in a decrease in accuracy. We can see that when the number of stacked normal cells exceeds ten, the more times the cell is stacked, the larger the number of model parameters, but it does not improve the performance or even achieve the worst performance. Therefore, we obtain the best accuracy by setting the number of stacked normal cells to eight.

V. CONCLUSION AND FUTURE WORK

In this article, we proposed a novel approach to automatic CNN architecture design using a progressive evaluation strategy and a subpopulation preservation strategy. The progressive evaluation strategy aims to reduce computational costs, while the subpopulation preservation strategy is intended to increase population diversity and the chances of preserving potentially good individuals during the evolution process. We provided a detailed analysis of the effectiveness of the progressive evaluation strategy and the subpopulation preservation strategy.

To evaluate the performance of the proposed algorithm, we conducted a search process on the CIFAR10 dataset, then

transferred the searched architecture to the CIFAR100 and ImageNet datasets. We compared the proposed algorithm with 36 SOTA algorithms. The results, which include test errors, GPU days, and the number of parameters, demonstrate the effectiveness of our proposed approach.

For future work, we plan to explore the development of high-performance architectures with lower-resource consumption. In addition, we will investigate how to accurately evaluate some promising network architectures without complete training, which is similar to the NAS method based on surrogate models. Finally, we consider introducing a surrogate model in our future research to further improve search efficiency.

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[2] L. Pedrelli and X. Hinaut, "Hierarchical-task reservoir for online semantic analysis from continuous speech," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 6, pp. 2654–2663, Jun. 2022.

[3] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 7132–7141.

[4] L. Li, T. Zhou, W. Wang, J. Li, and Y. Yang, "Deep hierarchical semantic segmentation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2022, pp. 1246–1257.

[5] J. Hwang, S. Kim, J. Son, and B. Han, "Weakly supervised instance segmentation by deep community learning," in *Proc. IEEE/CVF Winter Conf. Appl. Comput. Vis.*, 2021, pp. 1020–1029.

[6] G. Rotman and R. Reichart, "Multi-task active learning for pre-trained transformer-based models," *Trans. Assoc. Comput. Linguist.*, vol. 10, pp. 1209–1228, Nov. 2022.

[7] A. Feder et al., "Causal inference in natural language processing: Estimation, prediction, interpretation and beyond," *Trans. Assoc. Comput. Linguist.*, vol. 10, pp. 1138–1158, Oct. 2022.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, 2017.

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

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

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

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 630–645.

[13] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 4700–4708.

[14] X. He, K. Zhao, and X. Chu, "AutoML: A survey of the state-of-the-art," *Knowl. Based Syst.*, vol. 212, Jan. 2021, Art. no. 106622.

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

[16] Z. Zhong et al., "BlockQNN: Efficient block-wise neural network architecture generation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 7, pp. 2314–2328, Jul. 2020.

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

[18] H. Liang et al., "DARTS+: Improved differentiable architecture search with early stopping," 2019, *arXiv:1909.06035*.

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

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

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

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

[23] Y. Xue and J. Qin, "Partial connection based on channel attention for differentiable neural architecture search," *IEEE Trans. Ind. Informat.*, vol. 19, no. 5, pp. 6804–6813, May 2023.

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

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

[26] 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.

[27] Y. Xue, Y. Wang, J. Liang, and A. Slowik, "A self-adaptive mutation neural architecture search algorithm based on blocks," *IEEE Comput. Intell. Mag.*, vol. 16, no. 3, pp. 67–78, Aug. 2021.

[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] 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.

[30] Y. Liu, Y. Tang, and Y. Sun, "Homogeneous architecture augmentation for neural predictor," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2021, pp. 12249–12258.

[31] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 4095–4104.

[32] M. Suganuma, M. Kobayashi, S. Shirakawa, and T. Nagao, "Evolution of deep convolutional neural networks using cartesian genetic programming," *Evol. Comput.*, vol. 28, no. 1, pp. 141–163, 2020.

[33] H. Zhang, Y. Jin, R. Cheng, and K. Hao, "Efficient evolutionary search of attention convolutional networks via sampled training and node inheritance," *IEEE Trans. Evol. Comput.*, vol. 25, no. 2, pp. 371–385, Apr. 2021.

[34] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *J. Mach. Learn. Res.*, vol. 20, no. 1, pp. 1997–2017, 2019.

[35] T. Elsken, J. H. Metzen, and F. Hutter, "Efficient multi-objective neural architecture search via lamarckian evolution," 2018, *arXiv:1804.09081*.

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

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

[38] Y.-H. Kim, B. Reddy, S. Yun, and C. Seo, "NEMO: Neuro-evolution with multiobjective optimization of deep neural network for speed and accuracy," in *Proc. ICML AutoML Workshop*, 2017, pp. 1–8.

[39] P. Liu, M. D. El Basha, Y. Li, Y. Xiao, P. C. Sanelli, and R. Fang, "Deep evolutionary networks with expedited genetic algorithms for medical image denoising," *Med. Image Anal.*, vol. 54, pp. 306–315, May 2019.

[40] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2009, pp. 248–255.

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

[42] S. Xie, H. Zheng, C. Liu, and L. Lin, "SNAS: Stochastic neural architecture search," 2018, *arXiv:1812.09926*.

[43] H. Zhang, S. Kiranyaz, and M. Gabbouj, "Finding better topologies for deep convolutional neural networks by evolution," 2018, *arXiv:1809.03242*.

[44] P. Ren, Y. Xiao, X. Chang, P.-Y. Huang, Z. Li, X. Chen, and X. Wang, "A comprehensive survey of neural architecture search: Challenges and solutions," *ACM Comput. Surveys*, vol. 54, no. 4, pp. 1–34, 2021.

[45] 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.*, 2018, pp. 8697–8710.

[46] Y. Xu et al., "PC-DARTS: Partial channel connections for memory-efficient architecture search," 2019, *arXiv:1907.05737*.

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

[48] P. K. Sen, "Estimates of the regression coefficient based on Kendall's tau," *J. Am. Statist. Assoc.*, vol. 63, no. 324, pp. 1379–1389, 1968.

[49] A. Krizhevsky, *Learning Multiple Layers of Features from Tiny Images*, Univ. Toronto, Toronto, ON, Canada, 2009.

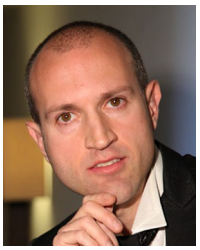
[50] S. Zagoruyko and N. Komodakis, "Wide residual networks," 2016, *arXiv:1605.07146*.

[51] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 6848–6856.
[52] C. Liu et al., "Progressive neural architecture search," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 19–34.
[53] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," 2016, *arXiv:1611.02167*.
[54] H. Cai, L. Zhu, and S. Han, "ProxylessNAS: Direct neural architecture search on target task and hardware," 2018, *arXiv:1812.00332*.
[55] X. Chen, L. Xie, J. Wu, and Q. Tian, "Progressive differentiable architecture search: Bridging the depth gap between search and evaluation," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2019, pp. 1294–1303.
[56] L. Wu, B. Liu, P. Stone, and Q. Liu, "Firefly neural architecture descent: A general approach for growing neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, pp. 22373–22383.
[57] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical representations for efficient architecture search," 2017, *arXiv:1711.00436*.
[58] J. Dong, B. Hou, L. Feng, H. Tang, K. C. Tan, and Y.-S. Ong, "A cell-based fast memetic algorithm for automated convolutional neural architecture design," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 34, no. 11, pp. 9040–9053, Nov. 2023.
[59] S. Yang, Y. Tian, X. Xiang, S. Peng, and X. Zhang, "Accelerating evolutionary neural architecture search via multifidelity evaluation," *IEEE Trans. Cogn. Develop. Syst.*, vol. 14, no. 4, pp. 1778–1792, Dec. 2022.
[60] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 4510–4520.
[61] Y. Li, Z. Yang, Y. Wang, and C. Xu, "Adapting neural architectures between domains," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, pp. 789–798.
[62] C. Xue, X. Wang, J. Yan, Y. Hu, X. Yang, and K. Sun, "Rethinking bi-level optimization in neural architecture search: A Gibbs sampling perspective," in *Proc. AAAI Conf. Artif. Intell.*, 2021, pp. 10551–10559.
[63] R. Wang, M. Cheng, X. Chen, X. Tang, and C.-J. Hsieh, "Rethinking architecture selection in differentiable NAS," 2021, *arXiv:2108.04392*.
[64] D. Zhou et al., "EcoNAS: Finding proxies for economical neural architecture search," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 11396–11404.
[65] Y. Xue, C. Chen, and A. Słowik, "Neural architecture search based on a multi-objective evolutionary algorithm with probability stack," *IEEE Trans. Evol. Comput.*, vol. 27, no. 4, pp. 778–786, Aug. 2023.

Yu Xue (Senior Member, IEEE) received the Ph.D. degree from the School of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, China, in 2013.

He is a Professor with the School of Software, Nanjing University of Information Science and Technology. He was a Visiting Scholar with the School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand, from August 2016 to August 2017. He was a Research Scholar with the Department of Computer Science and Engineering, Michigan State University, East Lansing, MI, USA, from October 2017 to November 2018. His research interests include deep learning, evolutionary computation, machine learning, and computer vision.

Jiajie Zha is currently pursuing the M.Sc. degree with a focus on deep learning, evolutionary computation, and neural architecture search, with the School of Software, Nanjing University of Information Science and Technology, Nanjing, China.



Danilo Pelusi (Member, IEEE) received the master degree in physics from the University of Bologna, Bologna, Italy, in 1998, and the Ph.D. degree in computational astrophysics from the University of Teramo, Teramo, Italy, in 2006.

He is currently an Associate Professor of Computer Science with the Department of Communication Sciences, University of Teramo, Teramo. His research interests include fuzzy logic, neural networks, information theory, machine learning, and evolutionary algorithms.

Dr. Pelusi is the World's 2% Top Scientist 2021 and 2022. He is an Editor of Springer, Elsevier, and CRS books, and an Associate Editor of IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTATIONAL INTELLIGENCE from 2017 to 2020, IEEE ACCESS since 2018, IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS since 2022, IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS since 2022, and IEEE TRANSACTIONS ON FUZZY SYSTEMS since 2024. He is a Guest Editor for Elsevier, Springer, and MDPI journals. He is a Keynote Speaker, a Guest of Honor, and the Chair of IEEE conferences, and he is an Inventor of International Patents on Artificial Intelligence.

Peng Chen received the B.E. degree in navigation from Dalian Maritime University, Dalian, China, in 2005, the M.E. degree in traffic information engineering and control from Shanghai Maritime University, Shanghai, China, in 2007, and the Ph.D. degree from Tokyo Institute of Technology, Tokyo, Japan, in 2020.

He is also a Visiting Scientist with RIKEN Center for Computational Science, Kobe, Japan. His research interests include parallel computing, image processing, and machine learning.

Dr. Chen is a Researcher with National Institute of Advanced Industrial Science and Technology.

Tao Luo (Member, IEEE) received the bachelor's degree from Harbin Institute of Technology, Harbin, China, in 2010, the master's degree from the University of Electronic Science and Technology of China, Chengdu, China, in 2013, and the Ph.D. degree from the School of Computer Science and Engineering, Nanyang Technological University, Singapore, in 2018.

He is currently a Senior Research Scientist with the Institute of High Performance Computing, Agency for Science, Technology and Research, Singapore (A*STAR), Singapore. His current research interests include high-performance computing, efficient and green AI, quantum computing, hardware–software co-exploration, and application of AI.

Liangli Zhen received the Ph.D. degree in computer science from Sichuan University, Chengdu, China, in 2018.

He is a Senior Scientist and a Group Manager with the Institute of High Performance Computing, Agency for Science, Technology and Research (A*STAR), Singapore. His current research interests include machine learning and computer vision. He has published more than 30 papers in top tier journals and conferences, including IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, IEEE TRANSACTIONS ON IMAGE PROCESSING, IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS, ICLR, CVPR, and ICCV.

Yan Wang received the Ph.D. degree in computer science from Sichuan University, Chengdu, China, in 2021.

He is a Scientist with the Institute of High Performance Computing, Agency for Science, Technology and Research (A*STAR), Singapore. His current research interests include deep neural networks, neural architecture search, transfer learning, and medical image analysis.

Mohamed Wahib received the Ph.D. degree in computer science from Hokkaido University, Sapporo, Japan, in 2012.

He is a Team Leader of the "High Performance Artificial Intelligence Systems Research Team" with RIKEN Center for Computational Science, Kobe, Japan. Prior to that, he is a Senior Scientist with AIST/TokyoTech Open Innovation Laboratory, Tokyo, Japan. His research interests revolve around the central topic of high-performance programming systems, in the context of HPC and AI. He is actively working on several projects, including high-level frameworks for programming traditional scientific applications, as well as high-performance AI.