Electronic Research Archive

http://www.aimspress.com/journal/era

ERA, 32(2): 1017–1043.
DOI: 10.3934/era.2024050
Received: 01 December 2023
Revised: 04 January 2024
Accepted: 16 January 2024
Published: 22 January 2024

Research article

Similarity surrogate-assisted evolutionary neural architecture search with dual encoding strategy

Yu Xue[1,*], **Zhenman Zhang**[1] **and Ferrante Neri**[1,2]

[1] School of Computer Science, Nanjing University of Information Science and Technology, Jiangsu, China

[2] School of Computer Science and Electronic Engineering, University of Surrey, United Kingdom

* **Correspondence:** Email: xueyu@nuist.edu.cn.

Abstract: Neural architecture search (NAS), a promising method for automated neural architecture design, is often hampered by its overwhelming computational burden, especially the architecture evaluation process in evolutionary neural architecture search (ENAS). Although there are surrogate models based on regression or ranking to assist or replace the neural architecture evaluation process in ENAS to reduce the computational cost, these surrogate models are still affected by poor architectures and are not able to accurately find good architectures in a search space. To solve the above problems, we propose a novel surrogate-assisted NAS approach, which we call the similarity surrogate-assisted ENAS with dual encoding strategy (SSENAS). We propose a surrogate model based on similarity measurement to select excellent neural architectures from a large number of candidate architectures in a search space. Furthermore, we propose a dual encoding strategy for architecture generation and surrogate evaluation in ENAS to improve the exploration of well-performing neural architectures in a search space and realize sufficiently informative representations of neural architectures, respectively. We have performed experiments on NAS benchmarks to verify the effectiveness of the proposed algorithm. The experimental results show that SSENAS can accurately find the best neural architecture in the NAS-Bench-201 search space after only 400 queries of the tabular benchmark. In the NAS-Bench-101 search space, it can also get results that are comparable to other algorithms. In addition, we conducted a large number of experiments and analyses on the proposed algorithm, showing that the surrogate model measured via similarity can gradually search for excellent neural architectures in a search space.

Keywords: evolutionary algorithm; neural architecture search; surrogate-assisted; encoding strategy

1. Introduction

The rapid advancement of deep neural networks has immensely facilitated the daily life of the general public, yet the burden on neural architecture designers is perennial. Typically, designers must craft a precise neural architecture that is tailored for a specific task, involving professionals in an arduous manual trial-and-error process that is both tedious and time-consuming [1]. In order to design efficient neural architectures automatically, neural architecture search (NAS), as a pivotal branch of automated machine learning, has gradually emerged as a focal point of research [2].

NAS is implemented gradually by selecting neural architectures from an artificially defined search space and evaluating the selected architectures according to a performance evaluation strategy [3]. The current NAS process usually relies on reinforcement learning, evolutionary algorithms, and gradient-based methods [4]. The reinforcement learning-based NAS approaches use a controller to select an architecture within a search space [5]. NAS methods based on an evolutionary algorithm can generate and retain the neural architecture by mimicking the evolution process of a population [6]. The gradient-based NAS methods perform architecture searches by jointly optimizing network parameters and network structure weights [7].

Although some experiments have shown that the neural architecture searched by NAS methods based on an evolutionary algorithm is superior to those searched by NAS methods based on reinforcement learning in terms of performance; NAS methods based on an evolutionary algorithm usually consume excessively large amounts of computing resources due to the characteristics of the evolutionary algorithm itself [8]. This difficulty reduces the possibility of practical application of NAS methods based on evolutionary algorithms.

NAS methods based on evolutionary algorithms are well known for their powerful ability to explore a search space [9]. Neuroevolution, proposed earlier by Floreano et al., uses evolutionary algorithms to search for neural architectures and their corresponding weights simultaneously [10]. Evolutionary neural architecture search (ENAS), on the other hand, has only been proposed in recent years and focuses solely on the process of searching neural architectures by using evolutionary algorithms, and its corresponding weights are computed by using gradient-based optimization algorithms after the architecture is determined. ENAS algorithms encode neural structures in the search space into vectors, and then they rely on evolutionary strategies such as crossover, mutation, evaluation, and selection to gradually explore neural architectures and realize excellent performance in the search space. The cost of computing resources and time required to evaluate one neural architecture in a search space is often acceptable, but evaluating thousands of neural architectures, as is required the ENAS approach is, often prohibitively expensive [11].

To retain the powerful search ability of ENAS algorithms and reduce the cost of computing resources, there are two kinds of methods to accelerate the efficiency of ENAS. One is proxy metrics-based ENAS, which uses other methods such as low fidelity estimates, learning curve extrapolation, or one-shot models to evaluate the architecture, but this still requires updating of the parameters of the searched neural architectures based on gradients, which adds additional computational overhead [12]. The other is surrogate-based ENAS, which uses the surrogate models (or performance predictors) to evaluate the performance of the searched neural architectures that only depend on the information of the architecture itself, without considering the parameters of the searched architectures [13]. Reduction of the overhead of computing resources is achieved by building surrogate models to directly predict the performance of candidate architectures, rather than actually evaluating the performance of architectures using concrete datasets.

At present, most ENAS methods based on surrogate models use a regression model as the surrogate model. The surrogate model is trained on the existing historical data (architecture, performance) pairs so that it can learn the mapping of architecture features to its performance and directly predict the performance of candidate architectures [13]. For example, graph neural networks (GNNs) [14] are used as surrogate models to directly predict the performance of architectures [15]. Wei et al. proposed NPENAS, a regression performance predictor-guided evolutionary algorithm method, to enhance the exploration ability of evolutionary algorithms for NAS [16]. In addition to directly predicting performance, Wang et al. chose a support vector machine (SVM) as a surrogate model to predict which convolutional neural network (CNN) would achieve better performance [17]. In recent years, some researchers have also used the ranking model as a surrogate model to perform performance ranking of candidate neural architectures and realize the selection of neural architectures with high levels of performance [18]. However, these methods are often affected by poor architectures of the historical data and their own distribution, which reduce the reliability of the surrogate model and degrades its functionality when exploring the architectures in a search space.

In this paper, to solve the ENAS problems mentioned above, we propose a novel approach, i.e., similarity surrogate-assisted ENAS with dual encoding strategy (SSENAS), to realize a computationally efficient ENAS by simultaneously accelerating the performance evaluation process and improving the search strategy efficiency. The contributions of this paper are summarized as follows:

1) To prevent the surrogate model from being affected by inferior architecture information, we propose a basic online learning framework for SSENAS. We propose the use of a surrogate model based on similarity to reliably select excellent neural architectures from a large number of candidate architectures. The neural architectures with excellent performance can be found step by step from a search space by using the proposed method.

2) In order to represent the information of one architecture more comprehensively, we propose a novel encoding strategy based on inter-node information transmission to encode an architecture. It ensures that the scale of variables within each dimension of architecture encoding is consistent, and it achieves a more efficient representation of architecture information.

3) To generate neural architectures with excellent performance during evolution, we have designed a novel encoding strategy for the generation of candidate architectures. By decoupling the connection between the architecture nodes and the operation types of the architecture nodes, it explores architectures that are similar to the current excellent architecture in the search space through evolutionary operations, so as to improve the efficiency of the algorithm and reduce the cost of computing resources.

2. Related work

In this section, we briefly introduce the related work, including ENAS and surrogate model methods used to accelerate the NAS process. Based on the above, the limitations of the traditional surrogates, which are based on regression or ranking models, are explained in Section 2.2.

2.1. ENAS

NAS can be mathematically described as an optimization problem as follows:

$$\begin{aligned} \min_{\alpha} \quad & Loss(\alpha, D_{train}, D_{val}) \\ & s.t., \quad \alpha \in \Omega \end{aligned} \tag{2.1}$$

where Ω represents the search space containing all candidate architectures, and α represents one architecture in the search space. High-performance neural architectures are obtained by minimizing the loss ($Loss$) of architectures on the validation set D_{val} after being trained on the training set D_{train}. NAS usually consists of three components: a search space, search strategy, and performance estimation strategy [12]. The process of NAS is as follows. First, an architecture α is selected by the search strategy from a predefined search space Ω. Then, the architecture is passed to a performance estimation strategy, which returns the estimated performance of α to the search strategy. NAS methods that use evolutionary algorithms instead of ordinary search strategies are often called ENAS methods. ENAS searches for excellent neural architectures in the search space by implementing evolutionary algorithms with balanced global and local exploration capabilities.

The process of ENAS is as follows. First, some neural architectures are initialized from the defined search space, and these architectures (individuals) are evaluated to obtain their fitness values. Then, the parent individuals are selected via environmental selection, and the offspring individuals are obtained via genetic operations that depend on the parent individuals. The offspring individuals are then evaluated and their fitness values are obtained. Finally, parent and offspring individuals are merged, and new parent individuals are selected to enter the next generation through environmental selection. Relying on the powerful exploration capability of evolutionary algorithms, ENAS is able to search for excellent neural architectures in a search space. For example, Xie and Yuille. earlier represented the architecture in a search space as a fixed-length binary string and applied genetic algorithms to automate the process of learning the architecture of deep CNNs [19]. Sun et al. proposed EvoCNN, which uses a variable-length encoding strategy to encode architectures, and they realized the automatic evolution of the architectures and the weights of the CNN for image classification problems [20]. Huang et al. used the particle swarm optimization (PSO) algorithm to evolve compact CNN architectures in a modified parameter-efficient MBConv blocks search space [6]. Xue et al. proposed SaMuNet, which relies on an adaptive mutation strategy and semi-complete binary competition strategy to automatically construct CNN architectures in the search space that has been constructed based on ResNet and DenseNet blocks [21]. By encoding architectures through encoding strategies and relying on evolutionary algorithms to explore a predefined search space step by step, excellent neural architectures can be generated and selected from the search space.

2.2. Surrogate model for NAS

Full evaluation of a neural architecture in a search space can be time-consuming and take anywhere from a few minutes to a few hours, depending on many factors, such as the number of neural architecture parameters, the size of the dataset, and the required hardware. In order to accelerate the process of performance estimation for NAS, a surrogate model has been developed proposed to assist or replace the process of evaluating neural architectures. Surrogate models are also known as performance predictors, and earlier surrogate models focused on designing regression models to directly predict the performance of candidate neural architectures. For example, Deng et al. proposed Peephole, which uses an end-to-end approach to jointly optimize long short-term memory (LSTM) and multilayer perceptron (MLP) to make direct predictions about neural architecture performance [22]. Tang et al. proposed a semi-supervised assessor to evaluate the neural architectures by predicting their performance directly [23]. Wen et al. used the graph convolutional network (GCN) regression model as a surrogate model to directly predict the performance of neural architectures [15]. However, the reliability of surrogate models based on regression models is often reduced due to inconsistent data distribution. Especially in ENAS, the performance distribution of the first sampled architectures is inconsistent with that of the later generated architectures, which makes the regression model-based surrogate model unable to accurately predict the performance of the neural architecture.

However, rather than understanding the exact performance of a set of neural architectures, we are actually more concerned about which neural architecture will lead to better performance. In recent years, more and more ranking-based surrogate models have been proposed to predict the ranking of neural architectures. For example, Xu et al. proposed ReNAS to rank multiple neural architectures before training [18]. Chen et al. proposed CTNAS, which uses the results of comparisons between architectures as a reward to search for promising architectures [24]. Huang et al. proposed a task-transferable NAS method by training a pairwise relation predictor to solve graph-ordering problems [25]. However, the performance of these ranking-based surrogate models is still affected by poor architectures, and the performance of the surrogate models is also affected by the imbalanced distribution of the samples used to train the surrogate models. To solve these problems, in this paper, we propose a novel surrogate model based on similarity measurement, which is different from the surrogate models based on regression or ranking. The surrogate model based on similarity pays more attention to the architecture with good performance, and the performance of the model is not affected by the existing poor architectures.

3. Methods

In this section, we go into the details of the proposed algorithm. First, the overall framework of the proposed algorithm is introduced through the pseudo-code and the figure in Section 3.1. Then, a similarity-based surrogate model is described in detail in Section 3.2. Finally, an encoding strategy for surrogate evaluation is described in Section 3.3, and an encoding strategy for generating candidate neural architectures is described in Section 3.4.

3.1. Overall framework

The overall framework of the proposed surrogate-assisted ENAS is shown in Figure 1. In this framework, since there are additional data that can be actively collected during the evolutionary process to update the surrogate and guide the search, we refer to this type of framework as an online learning framework. In addition, Algorithm 1 gives the pseudo-code of the whole process to explain the proposed algorithm in more detail. Given a search space Ω, this algorithm can obtain an excellent neural architecture in the search space.

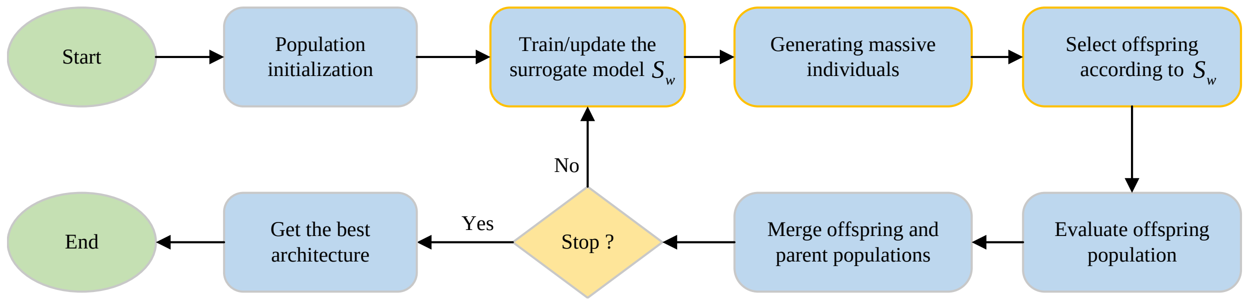

Start
Population initialization
Train/update the surrogate model S_w
Generating massive individuals
Select offspring according to S_w
Evaluate offspring population
Merge offspring and parent populations
Stop ?
No
Yes
Get the best architecture
End

Figure 1. The overall framework of the proposed method.

Algorithm 1 The framework of the proposed method

```
Input: $\Omega$: Search space; $N$: Population size; $M$: New individuals' size; maximum iterations.
Output: $\alpha^*$: A high-performance architecture from $\Omega$.
1: $t \leftarrow 0$: Initialize an iteration counter.
2: $Arcs^0 \leftarrow \{\alpha_1^0, \alpha_2^0, \ldots, \alpha_N^0\}$: Randomly initialize a set of architectures from $\Omega$ as the initial population.
3: $Y^0 \leftarrow \{y_1^0, y_2^0, \ldots, y_N^0\}$: Evaluate $Arcs^0$ to get their ground-truth performance on a specific task.
4: $S_w^0$: A surrogate model designed for coarse-grained screening of architectures in $\Omega$ and trained by $(Arcs^0, Y^0)$ pairs.
5: while $t <$ maximum iterations do
6:     $Inds^{t+1} \leftarrow \{\alpha_1^{t+1}, \alpha_2^{t+1}, \ldots, \alpha_M^{t+1}\}$: Generate massive individuals (new architectures) through crossover and mutation operations by $(Arcs^t, Y^t)$ pairs.
7:     $Offs^{t+1} \leftarrow \{\alpha_{s1}^{t+1}, \alpha_{s2}^{t+1}, \ldots, \alpha_{sN}^{t+1}\}$: Select $N$ individuals from $Inds^{t+1}$ as the offspring population according to $S_w^t$, where $N < M$.
8:     $Y_{offs}^{t+1} \leftarrow \{y_1^{t+1}, y_2^{t+1}, \ldots, y_N^{t+1}\}$: Evaluate $Offs^{t+1}$ to get their ground-truth performance.
9:     $(Arcs^{t+1}, Y^{t+1}) \leftarrow (Arcs^t, Y^t) \bigcup (Offs^{t+1}, Y_{offs}^{t+1})$: Update the current information for all authentically evaluated architectures.
10:    $S_w^{t+1}$: Update the weights of $S_w^t$ by $(Arcs^{t+1}, Y^{t+1})$ pairs.
11:    $t \leftarrow t + 1$.
12: end while
13: Select the best architecture $\alpha^*$ from all authentically evaluated architectures.
14: Return: $\alpha^*$
```

First, at the initialization stage, an iteration counter is initialized to count the current number of generations of the evolutionary algorithm. Then, N architectures are randomly initialized according to the search space to act as the initial population of the evolutionary algorithm. Then, the ground-truth performance of these architectures is measured in terms of specific tasks, which is a time-consuming step in ENAS. To speed up the process of architecture evaluation, a surrogate model has been designed to perform coarse-grained screening of the architectures in the search space. The surrogate model is designed, and then its parameters are updated according to the architecture–performance pairs in the initialization population; this endows the surrogate model with the ability to distinguish between excellent architectures and poor architectures in the search space. Second, the main loop proceeds as follows: 1) Evolutionary operations are used to generate a large number, M, of new neural architectures in the search space, relying on the information of the existing evaluated architectures; 2) Then, the surrogate model is used to roughly evaluate the network architectures, and the N architectures with the best performance are selected as the offspring population according to the evaluation metric of the surrogate model; 3) Then, these network architectures are evaluated to obtain their ground-truth performance; 4) All architecture-related information that has been truly evaluated is updated, and followed by the parameters of the surrogate model based on this information. Based on the local and global search ability of the evolutionary algorithm, it can achieve a balance between the exploitation and exploration of the search space. With the increase of evolution generation, the reliability of the surrogate model will gradually increase, which ensures that the surrogate model can select some excellent neural architectures from a large number of neural architectures. To efficiently identify outstanding architectures in the search space while minimizing computational resource usage, we have developed an innovative approach. It introduces a surrogate model that leverages similarity for the screening of architectures at a coarse-grained level, as well as a novel dual encoding strategy for surrogate evaluation and architecture generation.

3.2. *Similarity-based surrogate model*

Unlike the traditional surrogate models based on regression models, a surrogate model based on similarity measurement has been developed to replace the performance evaluation process for neural architectures, and this serves to screen out the excellent neural architectures from the massive neural architectures in a short period of time. We have constructed a triplet neural network as the surrogate model, $TripletNet$, and use a triplet loss function, $Loss$, to train the neural network. The structure of the surrogate model is shown in Figure 2.

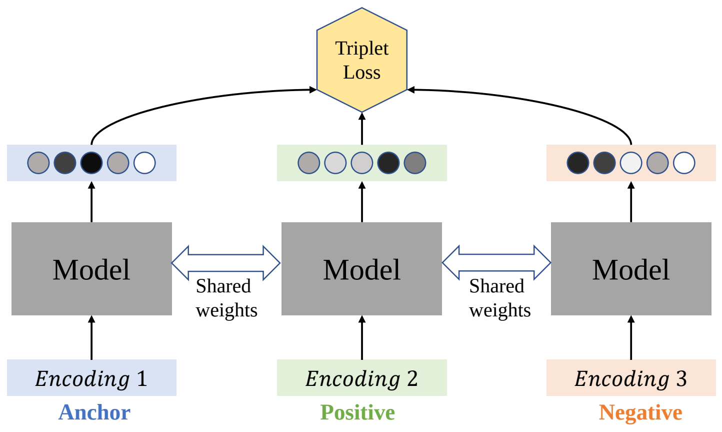


Figure 2. The structure of the surrogate model.

It takes three encoding vectors, which are introduced in Section 3.3, and maps the encoding vectors to three latent vectors in a latent space, respectively, through a weight-sharing model. Then, $Loss$ is calculated through the latent vectors, and the optimization of the model parameters is achieved by relying on the backpropagation algorithm. Finally, architectures with similar performance are mapped to similar distances in the latent space. Greater closeness of two latent vectors in the latent space means greater similarity of the two latent vectors, and vice versa.

Suppose that the surrogate model based on similarity measurement is represented as S_w. The trainable parameters of S_w are trained based on set $[\boldsymbol{X}, \boldsymbol{y}]$, where $\boldsymbol{X} = \{X_1, X_2, \ldots, X_N\}$ represents the encoding of the neural architecture and $\boldsymbol{y} = \{y_1, y_2, \ldots, y_N\}$ is the relevant performance metric of the corresponding neural architecture. In order to train S_w, the first step is to build a triplet (X_a, X_p, X_q) based on $[\boldsymbol{X}, \boldsymbol{y}]$, where X_a and X_p are of the same class and X_a and X_q are of different classes; additionally, each tuple in the triple needs to be determined depending on (y_a, y_p, y_q). How to define the similarities between architectures and divide them into different categories is crucial to the training of S_w. The algorithm implemented to generate the triplet that is used as a similarity dataset to train S_w is shown in Algorithm 2. First, the anchor, (X_a, y_a), is initialized by random sampling and the similarity threshold, i.e., H, is set. Then, (X_t, y_t) is obtained via the random sampling method, and (X_t, y_t) is determined as (X_p, y_p) or (X_q, y_q) by calculating the similarity between y_t and y_a. Until the condition is satisfied, the loop ends and the generated triplet is returned.

Algorithm 2 Triplet generation

```
Input: [X, y]: Architecture metrics; $H$: similarity threshold.
Output: $(X_a, X_p, X_q)$: A triplet.
1: $(X_a, y_a)$: randomly sample $(X, y) \in [\boldsymbol{X}, \boldsymbol{y}]$ as an anchor.
2: $(X_p, y_p) \leftarrow None$, $(X_q, y_q) \leftarrow None$: Initialization.
3: while $(X_p, y_p)$ is None or $(X_q, y_q)$ is None do
4:     $(X_t, y_t)$: randomly sample $(X, y) \in [\boldsymbol{X}, \boldsymbol{y}]$.
5:     $\nabla y \leftarrow y_t - y_a$: Calculate the error of $y_t$ and $y_a$.
6:     if $\nabla y \leq 0$ then
7:         $similarity \leftarrow (2/(1 + e^{\nabla y})) - 1$.
8:     else
9:         $similarity \leftarrow 1 - (2/(1 + e^{\nabla y}))$.
10:    end if
11:    if $similarity > H$ and $(X_p, y_p)$ is None then
12:        $(X_p, y_p) \leftarrow (X_t, y_t)$.
13:    end if
14:    if $similarity \leq H$ and $(X_q, y_q)$ is None then
15:        $(X_q, y_q) \leftarrow (X_t, y_t)$.
16:    end if
17: end while
18: Return: $(X_a, X_p, X_q)$
```

The input to the surrogate model is a triplet, i.e., (X_a, X_p, X_q), consisting of an anchor example, a positive example, and a negative example. By optimizing the trainable parameters of the surrogate model via a backpropagation algorithm, the similarity measurement between samples is realized. The ultimate optimization goal is to narrow the distance between X_a and X_p, and to stretch the distance between X_a and X_q. The triplet loss function is defined as

$$Loss = Max(Distance_S - Distance_D + m, 0) \tag{3.1}$$

The positive number m in Eq (3.1) is included to ensure the existence of a margin between distances of negative pairs and distances of positive pairs. $Distance_S$ and $Distance_D$ in the above equation represent the Euclidean distance of similar pairs (X_a, X_p) in latent space and the Euclidean distance of dissimilar pairs (X_a, X_q) in latent space, respectively. Their exact equations are as follows:

$$Distance_S = \|Model(X_a) - Model(X_p)\|_2 \tag{3.2}$$

$$Distance_D = \|Model(X_a) - Model(X_q)\|_2 \tag{3.3}$$

Regarding the loss function, $Loss$, it contains three cases, as shown in Figure 3:

1) *Case 1*: When $Distance_S + m < Distance_D$, it means that the distance between X_a and X_p plus the margin is less than the distance between X_a and X_q, that is, the distance between the anchor and the positive sample is closer, while the distance between the anchor and the negative sample is farther. In this case, no optimization is required, and the positive and negative examples are naturally separated.

2) *Case 2-1*: When $Distance_S + m > Distance_D$ and $Distance_S < Distance_D$, it indicates that the distance between X_a and X_p plus the margin is greater than the distance between X_a and X_q, whereas the distance between X_a and X_p is less than the distance between X_a and X_q. In this case, there is a loss ($Loss < m$), and the loss is relatively small; additionally, and the surrogate model's trainable parameters are optimized slightly.

3) *Case 2-2*: When $Distance_S + m > Distance_D$ and $Distance_S > Distance_D$, it indicates that the distance between X_a and X_p plus the margin is greater than the distance between X_a and X_q, and the distance between X_a and X_p is greater than the distance between X_a and X_q. In this case, there is a loss ($Loss > m$), and the loss is relatively large; also, and the surrogate model's trainable parameters are optimized sharply.

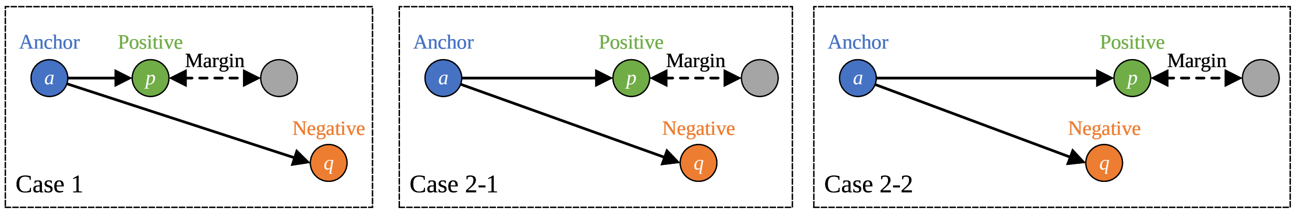


Figure 3. There are three possible scenarios for loss function optimization.

With this loss, we can appropriately adjust the trainable parameters of the surrogate model to map similar neural architectures to locations close to each other in latent space. The surrogate model is used with the evolutionary algorithm. The evolutionary algorithm affords the possibility of jumping out of local minima, while the surrogate model implements the actual process of jumping out of local minima. As for how to detect the fall into local minima, it is simply a matter of observing whether there are architectures in each generation that have been evaluated realistically that are better than the previous generation. If one of the architectures in each generation outperforms all of the architectures in the previous generation, then the algorithm is not trapped in local minima. Conversely, if no architecture in a generation performs better than the optimal architecture in the previous generation, then the algorithm is trapped in local minima.

3.3. Encoding strategy for surrogate evaluation

Proper design of the encoding strategy for neural architecture representation in a search space is crucial for surrogate models. Generally, a neural architecture G can be represented by using a directed acyclic graph (DAG), $G = (V, E)$, which uses either nodes V to represent operations and edges E to represent information flows, such as NAS-Bench-101 [26], or nodes V to represent information and edges E to represent operations, such as NAS-Bench-201 [27]. We propose the encoding of neural architectures that can be represented by DAGs through the use of an approach based on information transfer between nodes. It relies on the transmission of information between nodes to obtain more comprehensive information about the neural architectures represented by the DAG. First, the topological ordering of nodes is obtained according to the DAG, and each node is encoded by a one-hot strategy to get representation vector h. Then, the information transmission between nodes is carried out according to the connections between nodes. Specifically, the formula for the $k + 1$th information transmission of node v is shown as follows:

$$h_v^{k+1} = h_v^k + \sum_{u \in N(v)} \frac{h_u^k}{d_u} \tag{3.4}$$

where h_v^k represents the representation vector after the transmission of the kth information round of node $v, v \in V$, and $N(v)$ represents all neighbors of node v. d_u represents the degree (out and in) of node u, and h_u^k represents the representation vector after the transmission of the kth information round of node u. When updating the representation vector, the current node should not only consider the representation vector information of the current neighbor node, it should also consider its own representation vector information. Through multiple rounds of information transmission between nodes, a richer representation vector of each node can be obtained, instead of a simple one-hot vector. Since the representation vector corresponding to each node is one-hot encoded at the beginning, after node transmission and aggregation, they still have the characteristics of the same scale of variables in each dimension. It means that there is no need for the re-scaling of a vector of the final architecture encoding. Ultimately, the proposed encoding strategy for surrogate evaluation can be represented by the following equation:

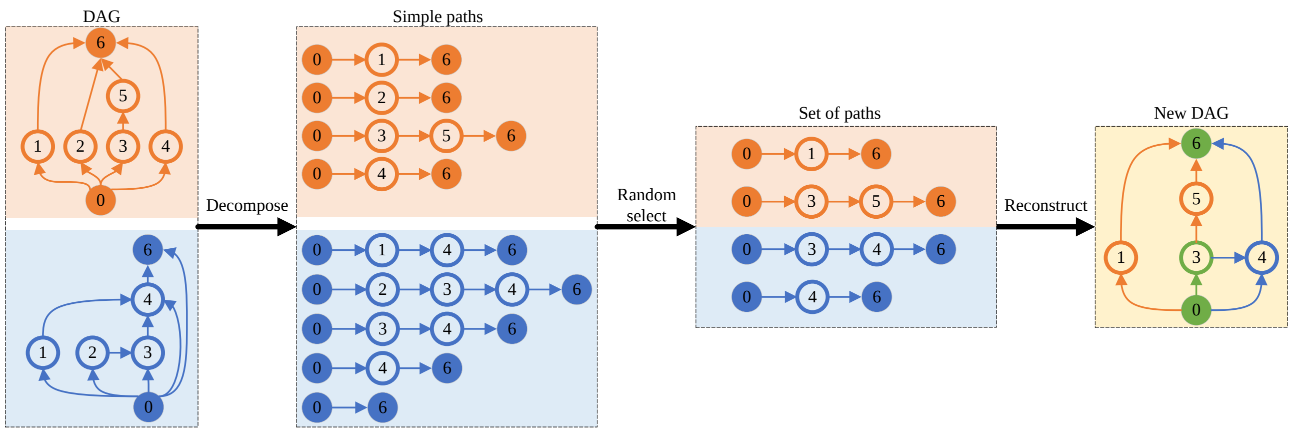
$$X_i = flattened(T_{sort}(I_{transmit}(one\text{-}hot(V), E))) \tag{3.5}$$

where $one\text{-}hot(V)$ indicates that the nodes are one-hot encoded, $I_{transmit}(\cdot)$ denotes information transmission of the nodes according to Eq (3.4), and $T_{sort}(\cdot)$ denotes the topological sorting of nodes. Finally, the vector is flattened into a one-dimensional vector with $flattened(\cdot)$.

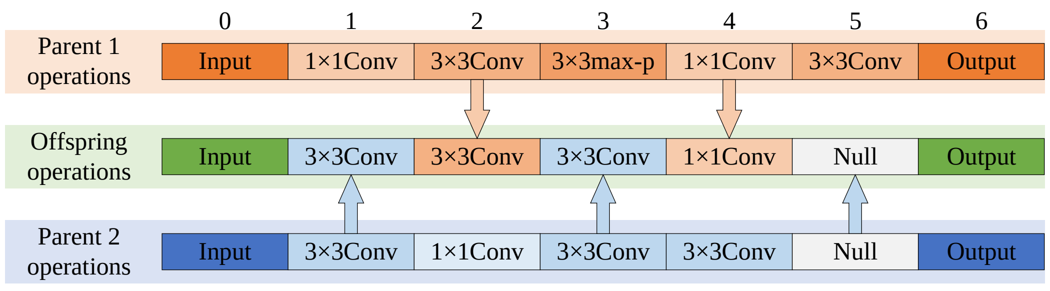
Since the topological ordering of a DAG is often not unique, this can result in multiple different encodings representing the same DAG. This is clearly not what we want. Inspired by an existing paper [28], we apply multiple different encodings of a DAG as a data enhancement technique in the hope of achieving adequate training of the surrogate model.

3.4. Encoding strategy for architecture generation

We propose a novel encoding strategy specifically for new architecture generation. The types of operations represented by DAG nodes and the connections between them are decoupled. The new architecture is generated in two steps. The first step only considers the connection mode between nodes, without considering the operation type of each node, and the second step only considers the operation type of each node, without considering the connection mode between nodes. Figure 4 shows an example of new architecture generation in the search space of NAS-Bench-101. The first step of the proposed encoding strategy is shown in Figure 4(a). Given two parent individuals randomly selected from the parent population and represented by respective DAGs, the two DAGs are first decomposed to obtain all of the simple paths constituting each of the two DAGs. Then, using a random selection method, half of the paths from all of the simple paths generated by the two parents are chosen to form a set of paths. Nodes with the same number are merged into one node (shown in green), while nodes with different numbers remain separate (shown in orange and blue). Finally, a new DAG is constructed from this set of paths.



(a) First step: Decomposition and reconstruction of DAG.



(b) Second step: Generating the offspring operations.

Figure 4. The encoding strategy for architecture generation. Architectures in the NAS-Bench-101 search space are used here as an example: (a) shows the process of constructing a new DAG, and (b) shows the way to generate new offspring operations.

This approach is novel, as it is based on an encoding strategy that reconstructs how the internal nodes of an architecture are connected based on simple paths. After the first step is complete, a new

DAG is obtained without considering node types. The second step of the proposed encoding strategy is illustrated in Figure 4(b). Given the operation type of the corresponding node of the parent individual, the input node and output node of the offspring individual are fixed, while the intermediate nodes are randomly inherited from the two parents with an inheritance probability of 50% each. For example, taking the operation type of child node 2 as an example, it has a 50% probability of being $3 \times 3 Conv$, as inherited from parent 1, and a 50% probability of being $1 \times 1 Conv$, as inherited from parent 2. In summary, the first step determines how inner nodes of the architecture are connected, and the second step determines the type of architecture node. By generating a new architecture in this way, a completely new architecture is obtained that is similar to, but different from, the two parent architectures. Finally, it is judged whether the generated architecture is within the defined search space. If it is, the current generated architecture is retained; if not, the current architecture is discarded.

4. Experiments

In this section, the experimental details are presented and the results are analyzed. First, Section 4.1 introduces the NAS benchmarks and evaluation metrics. Then, in Section 4.2, the experimental details on training the similarity-based surrogate model are presented. Next, in Sections 4.3 and 4.4, the experimental results of the proposed method are introduced and compared with those of the current methods. Finally, the ablation studies of the proposed method are presented in Section 4.5.

4.1. NAS benchmarks and evaluation metrics

NAS benchmarks. Experiments in this paper were conducted using NAS benchmarks, which have been specifically designed for the purpose of evaluating and comparing the performance of NAS algorithms, including NAS-Bench-101 [26] and NAS-Bench-201 [27]. We introduce these NAS benchmarks briefly as follows:

- NAS-Bench-101 is the first NAS benchmark. It provides tabular benchmarks, with a total of 423,624 neural architectures covering the entire proposed search space. Each architecture represented by a DAG, usually called a cell, is trained and evaluated three times on CIFAR-10. A complete neural architecture in the search space consists of a cell stacked multiple times. The node of a cell indicates the candidate operation, and the edge indicates the data flow.
- NAS-Bench-201 has 15,625 neural architectures in a predefined search space. Similar to NAS-Bench-101, a complete neural architecture in NAS-Bench-201 consists of a single cell stacked multiple times. The difference is that cells, in this search space, nodes are used to represent data flow and edges are used to represent candidate operations. Each architecture in the search space has the same topology, but the operations to connect the nodes are different. It provides tabular benchmarks with more diagnostic information on multiple datasets, including CIFAR-10, CIFAR-100, and ImageNet-16-120.

Evaluation metrics. Since the ultimate goal of NAS is to find neural architectures that perform well in a search space, we chose to use the classification accuracy (mean ± std) of the searched architectures on the validation and test sets as the evaluation metric. In addition, the effectiveness of the proposed method is measured by the number of queries. The lower the number of queries, the more efficient is the method and the less time it consumes. Therefore, the number of queries is also applied as an evaluation metric.

4.2. Training surrogate model

The detailed hyperparameter settings in the experiments are listed in Table 1. The number of generations of the evolutionary algorithm was set as 50 in the experiments, and the population size was set to 20. In the initialization phase, 20 architectures are randomly sampled in a search space and their architecture information is obtained by querying the corresponding tabular benchmark. These architectures are encoded as one-dimensional vectors by the encoding strategy proposed in Section 3.3. The encoding length of the architecture in NAS-Bench-101 was set as 35 (five operation types, seven nodes representing operations), and the encoding length of the architecture in NAS-Bench-201 was set as 30 (five operation types, six edges representing operations). The backbone network of $TripNet$, which is used as a surrogate model, is an MLP, which only has two hidden layers, and the number of neurons in each layer was set as 256 and 128, respectively, for NAS-Bench-101 and NAS-Bench-201. We chose to use the Adam optimizer, with a learning rate of 0.001, to train the surrogate model. As the number of sampled architectures increased, we gradually increased the batch size and epochs. The batch size and epoch were revised for each additional set of 250 samples. All experiments were run on a single GPU card of NVIDIA GeForce RTX 2080 Ti.

Table 1. Hyperparameter settings.

Hyperparameter	Value
Number of generations	50
Population size	20
Number of generated individuals	1000 for NAS-Bench-101 and 5000 for NAS-Bench-201
Encoding length	35 for NAS-Bench-101 and 30 for NAS-Bench-201
Latent vector length	16 for NAS-Bench-101 and 6 for NAS-Bench-201
Number of information transmissions	7 for NAS-Bench-101 and 6 for NAS-Bench-201
Number of hidden-layer neurons	[256, 128]
Optimizer	Adam
Learning rate	0.001
Similarity threshold	0.5
Batch size	[32, 64, 128, 256]
Epoch	[100, 150, 200, 250]

4.3. Experimental results on NAS benchmarks

The experimental results for NAS-Bench-101 are shown in Table 2. Each experiment was run 10 times. We show the experimental results under specific generations during the evolution. The bolded values in the table are the optimal values for the column. The architecture in NAS-Bench-101 has three metrics with different random initializations. We report the accuracy of the architecture searched on the validation set of the first group of metrics and the accuracy of this architecture on the test set of the first group of metrics, as shown in the columns "Validation (trial 0)" and "Test (trial 0)". In addition, we list the average accuracy of the searched architecture on the test set for three metrics, as shown in the last column of Table 2. As can be seen from the table, with the increase of the number of generations, the classification accuracy of the validation set for the searched optimal architecture steadily increases, but its corresponding test set classification accuracy has some fluctuations. This is due to the rank-correlation gap between the validation set and the test set for the tabular benchmark. In order to better show the performance of the algorithm, we also present the boxplots to illustrate the performance of the searched architectures as the number of evolution generations increases, as shown in Figure 5.

Table 2. Experimental results on NAS-Bench-101.

	Validation (trial 0)		Test (trial 0)		Test (average)	
Oracle	95.15		94.66		94.32	
Generation	Mean ± std	Best	Mean ± std	Best	Mean ± std	Best
1	93.71 ± 0.42	94.39	92.99 ± 0.47	93.72	92.91 ± 0.33	93.59
3	94.28 ± 0.26	94.73	93.62 ± 0.32	94.21	93.33 ± 0.43	**94.19**
5	94.40 ± 0.24	94.73	93.71 ± 0.36	94.38	93.51 ± 0.45	**94.19**
7	94.55 ± 0.16	94.73	93.95 ± 0.26	94.38	93.67 ± 0.29	94.18
10	94.66 ± 0.12	94.91	93.93 ± 0.22	94.23	93.57 ± 0.25	93.89
15	94.75 ± 0.12	94.93	94.00 ± 0.18	94.23	93.70 ± 0.14	93.89
20	94.80 ± 0.10	94.93	93.87 ± 0.21	94.13	93.69 ± 0.11	93.90
30	94.87 ± 0.11	95.02	94.05 ± 0.27	**94.42**	93.73 ± 0.19	94.06
40	94.92 ± 0.11	**95.11**	**94.17 ± 0.16**	94.34	**93.89 ± 0.13**	94.09
50	**94.96 ± 0.09**	**95.11**	94.10 ± 0.20	94.34	93.83 ± 0.14	94.09

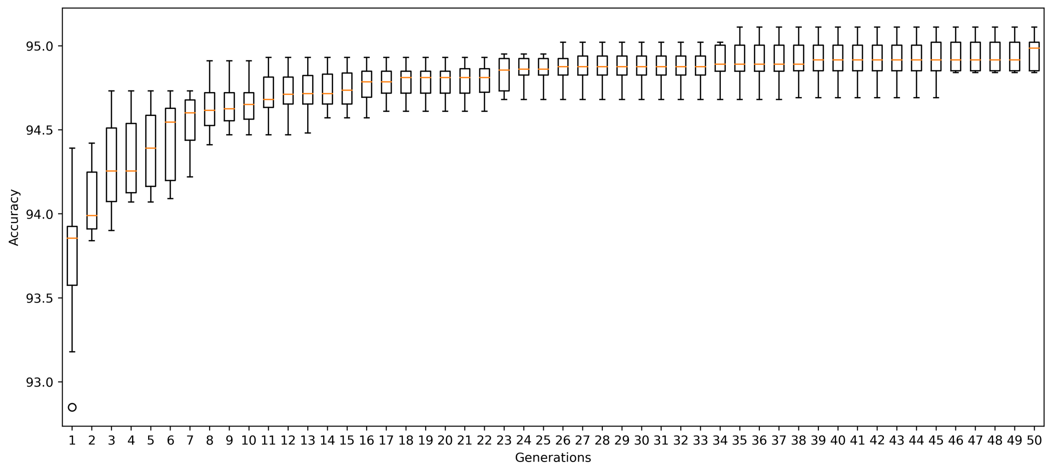


Figure 5. Boxplot of 10 experiments on NAS-Bench-101. The abscissa represents the evolution generations, and the ordinate represents the accuracy of the searched optimal architecture on the validation set.

Table 3 shows the results of the experiments on NAS-Bench-201. Each experiment was run 10 times. We show the experimental results under specific generations. The bolded values in the table are the optimal values for the column. Each experiment was designed to rely only on the architecture-related information on the validation set for the search. We report the performance of the optimal architectures searched on three datasets, i.e., CIFAR-10, CIFAR-100, and ImageNet-16-120. As can be seen in the table, the proposed algorithm can always search out the architecture with the best performance in the search space within 20 generations for CIFAR-10 and CIFAR-100. This means that, with a population size of 20 per generation, we only need 400 queries to search for architectures that perform well in the search space. With the increase of the number of generations, the accuracy of the searched architectures gradually increases and the standard deviation gradually decreases, which indicates that the proposed algorithm is not only effective, it is also relatively stable. We also show the boxplot (CIFAR-10) for the results of 10 experiments, as shown in Figure 6.

Table 3. Experimental results on NAS-Bench-201.

	CIFAR-10				CIFAR-100				ImageNet-16-120			
	Validation accuracy		Test accuracy		Validation accuracy		Test accuracy		Validation accuracy		Test accuracy	
Oracle	91.61		94.37		73.49		73.51		46.73		47.31	
Generation	Mean ± std	Best	Mean ± std	Best	Mean ± std	Best	Mean ± std	Best	Mean ± std	Best	Mean ± std	Best
1	89.84 ± 0.44	90.90	93.06 ± 0.50	93.84	69.75 ± 0.68	70.78	69.85 ± 0.69	70.79	43.27 ± 1.14	44.82	43.32 ± 1.19	44.76
3	91.20 ± 0.36	**91.61**	93.94 ± 0.33	**94.37**	71.61 ± 0.57	72.51	71.60 ± 0.90	73.14	46.20 ± 0.37	46.56	45.95 ± 0.59	**46.67**
5	91.39 ± 0.21	**91.61**	94.15 ± 0.24	**94.37**	72.59 ± 0.66	**73.49**	72.44 ± 0.93	**73.51**	46.41 ± 0.22	**46.73**	43.38 ± 0.26	46.59
7	91.54 ± 0.09	**91.61**	94.18 ± 0.22	**94.37**	73.03 ± 0.53	**73.49**	72.90 ± 0.63	**73.51**	46.52 ± 0.12	**46.73**	**46.47 ± 0.14**	46.59
10	91.57 ± 0.04	**91.61**	94.32 ± 0.17	**94.37**	73.28 ± 0.45	**73.49**	73.20 ± 0.55	**73.51**	46.58 ± 0.10	**46.73**	46.43 ± 0.16	46.59
15	91.60 ± 0.02	**91.61**	94.32 ± 0.17	**94.37**	73.44 ± 0.12	**73.49**	73.44 ± 0.15	**73.51**	46.68 ± 0.08	**46.73**	46.32 ± 0.18	46.59
20	**91.60 ± 0.00**	**91.61**	**94.37 ± 0.00**	**94.37**	**73.49 ± 0.00**	**73.49**	**73.51 ± 0.00**	**73.51**	46.70 ± 0.07	**46.73**	46.24 ± 0.12	46.59
30	**91.60 ± 0.00**	**91.61**	**94.37 ± 0.00**	**94.37**	**73.49 ± 0.00**	**73.49**	**73.51 ± 0.00**	**73.51**	**46.73 ± 0.00**	**46.73**	46.20 ± 0.00	46.20
40	**91.60 ± 0.00**	**91.61**	**94.37 ± 0.00**	**94.37**	**73.49 ± 0.00**	**73.49**	**73.51 ± 0.00**	**73.51**	**46.73 ± 0.00**	**46.73**	46.20 ± 0.00	46.20
50	**91.60 ± 0.00**	**91.61**	**94.37 ± 0.00**	**94.37**	**73.49 ± 0.00**	**73.49**	**73.51 ± 0.00**	**73.51**	**46.73 ± 0.00**	**46.73**	46.20 ± 0.00	46.20

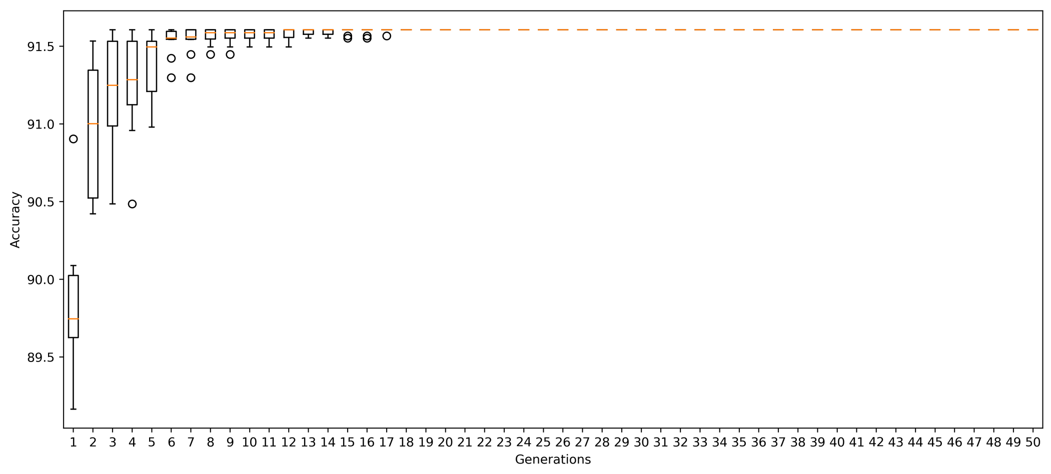


Figure 6. Boxplot of 10 experiments on NAS-Bench-201 CIFAR-10. The abscissa represents the evolution generations, and the ordinate represents the accuracy of the searched optimal architecture on the validation set.

4.4. *Comparison of different methods*

We compare the proposed algorithm with several state-of-the-art algorithms. The results for NAS-Bench-101 are shown in Table 4. The first column is the name of the algorithm, and the second column, "#Queries", represents the number of neural architectures queried from NAS-Bench-101.

The third column of the table represents the average accuracy (mean ± std) for the three trials corresponding to the neural architecture with the highest validation dataset obtained by the algorithm under a specific number of queries. The fourth column of the table shows how the mean accuracy in the third column ranks over the entire tabular benchmark. It can be ascertained from the table that the proposed algorithm outperforms most algorithms in the case of the same number of queries. Even after only one generation of the evolutionary algorithm (#Queries = 40), the architectures searched still rank relatively high in the overall search space.

Table 4. Comparisons with state-of-the-art algorithms in NAS-Bench-101 search space. "-" means unavailable results.

Algorithms	#Queries	Accuracy (average)	Rank (%)
NAR (statistics) [29]	4236	94.07 ± 0.09	0.0057
NAR (random) [29]	4236	94.06 ± 0.04	0.0064
Random [26,30]	1000	93.54*	0.8179
RL [26,31]	1000	93.58*	0.6310
BO [26,31]	1000	93.72*	0.2226
RE [26,31]	1000	93.72*	0.2226
Peephole [22,28]	1000	93.41 ± 0.34	1.6390
E2EPP [28,32]	1000	93.77 ± 0.13	0.1447
SSANA [23,28]	1000	94.01 ± 0.12	0.0113
HAAP [28]	1000	94.09 ± 0.11	0.0040
NAO [31,33]	1000	93.74*	0.1903
RFGIAug [34]	1000	94.20**	0.004
SSENAS (ours)	1000	93.83 ± 0.14	0.0857
MbML-NAS(GB) [35]	860	93.26*	3.0468
GenNAS-N [35,36]	500	93.92*	0.0323
BANANAS [30,37]	500	94.08*	0.0050
ReNAS [18,24]	423	93.90 ± 0.21	0.0399
CTNAS [24]	423	93.92 ± 0.18	0.0323
Random [24]	423	89.31 ± 3.92	71.276
SSENAS (ours)	400	93.69 ± 0.11	0.2811
FBNet [24,38]	-	92.29 ± 1.25	18.204
SPOS [24,39]	-	89.85 ± 3.80	63.674
FairNAS [24,40]	-	91.10 ± 1.84	41.115
SSENAS (ours)	40	93.18 ± 0.26	4.0038

* represents the mean result of multiple experiments.
** represents the best result of multiple experiments.

The results in Table 5 for NAS-Bench-201 show that the proposed method is more competitive than other algorithms. In the case of only 400 queries of the tabular benchmark, the proposed algorithm is able to find the best neural architecture in the search space when applied to the CIFAR-10 and CIFAR-100 datasets. Similarly, on ImageNet-16-120, the proposed method is still superior to other algorithms.

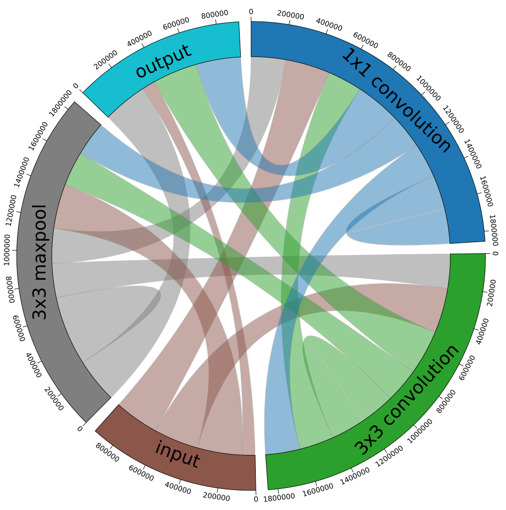
Table 5. Comparisons with state-of-the-art algorithms in NAS-Bench-201 search space. "-" means unavailable results.

Algorithm	#Queries	CIFAR-10		CIFAR-100		ImageNet-16-120	
		Validation	Test	Validation	Test	Validation	Test
Oracle	N/A	91.61	94.37	73.49	73.51	46.73	47.31
NAR [29]	1000	91.44 ± 0.10	94.33 ± 0.05	72.54 ± 0.44	72.89 ± 0.37	46.16 ± 0.37	**46.66 ± 0.23**
GenNAS-N [35,36]	1000	-	94.18 ± 0.10	-	72.56 ± 0.74	-	45.59 ± 0.54
NASWOT [41]	1000	89.69 ± 0.73	92.96 ± 0.81	69.86 ± 1.21	69.98 ± 1.22	43.95 ± 2.05	44.44 ± 2.10
MbML-NAS (RF) [35]	860	-	93.36 ± 0.20	-	70.33 ± 0.85	-	42.99 ± 4.21
MbML-NAS (GB) [35]	860	-	93.03 ± 0.52	-	70.02 ± 1.17	-	44.28 ± 1.42
RFGIAug [34]	424	91.43	94.25*	-	-	-	-
SSENAS (ours)	400	**91.61 ± 0.00**	**94.37 ± 0.00**	**73.49 ± 0.00**	**73.51 ± 0.00**	**46.72 ± 0.05**	46.24 ± 0.12
NASWOT [41]	100	89.55 ± 0.89	92.81 ± 0.99	69.35 ± 1.70	69.48 ± 1.70	42.81 ± 3.05	43.10 ± 3.16
Random [42]	100	91.07 ± 0.27	93.82 ± 0.24	71.44 ± 0.83	71.40 0.83	45.37 ± 0.63	45.36 ± 0.67
REA [8,42]	100	91.37 ± 0.25	94.06 ± 0.29	72.79 ± 0.69	72.72 ± 0.72	46.15 ± 0.45	45.99 ± 0.51
BANANAS [37,42]	100	91.50 ± 0.15	94.23 ± 0.30	73.27 ± 0.57	73.25 ± 0.63	46.45 ± 0.25	46.31 ± 0.31
GP_bayesopt [37,42]	100	91.45 ± 0.23	94.16 ± 0.31	73.11 ± 0.65	73.05 ± 0.75	46.51 ± 0.25	46.25 ± 0.34
DNGO [42,43]	100	91.41 ± 0.17	94.08 ± 0.26	72.71 ± 0.66	72.66 ± 0.67	46.11 ± 0.44	46.00 ± 0.48
Bohamiann [37,42]	100	91.41 ± 0.18	94.09 ± 0.26	72.73 ± 0.64	72.65 ± 0.66	46.15 ± 0.45	46.03 ± 0.48
GCN_Predictor [37,42]	100	91.02 ± 0.32	93.74 ± 0.33	71.43 ± 0.72	71.43 ± 0.77	45.47 ± 0.72	45.36 ± 0.78
BONAS [42]	100	91.56 ± 0.10	94.32 ± 0.15	73.32 ± 0.40	73.30 ± 0.46	46.56 ± 0.19	46.31 ± 0.30
Arch2vec_RL [31,42]	100	91.43 ± 0.28	94.23 ± 0.26	73.12 ± 0.66	73.06 ± 0.87	46.32 ± 0.20	46.35 ± 0.35
Arch2vec_BO [31,42]	100	91.51 ± 0.13	94.31 ± 0.14	73.38 ± 0.34	73.44 ± 0.22	46.33 ± 0.21	46.35 ± 0.27
NPENAS-SSRL [44], [42]	100	91.56 ± 0.14	94.32 ± 0.19	73.47 ± 0.22	73.47 ± 0.30	46.53 ± 0.33	45.83 ± 0.60
NPENAS-CCL [42,44]	100	91.57 ± 0.13	94.32 ± 0.19	73.48 ± 0.15	73.49 ± 0.23	46.62 ± 0.34	45.61 ± 0.41
SAENAS-NE [42]	100	91.58 ± 0.09	94.34 ± 0.12	73.46 ± 0.18	73.46 ± 0.20	46.59 ± 0.14	46.36 ± 0.26
ReNAS [18]	90	90.90 ± 0.31	93.99 ± 0.25	71.96 ± 0.99	72.12 ± 0.79	45.85 ± 0.47	45.97 ± 0.49

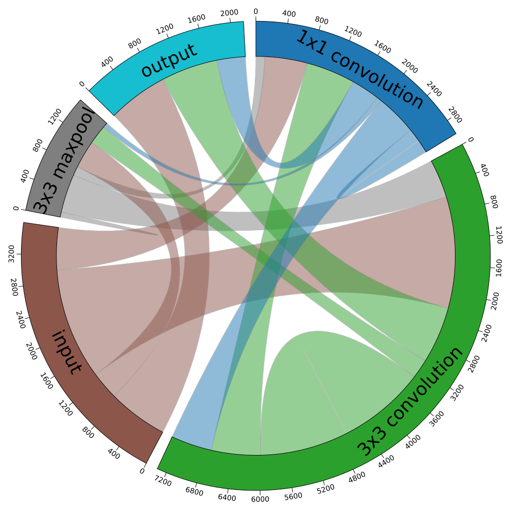
* represents the best result of multiple experiments.

4.5. Ablation studies

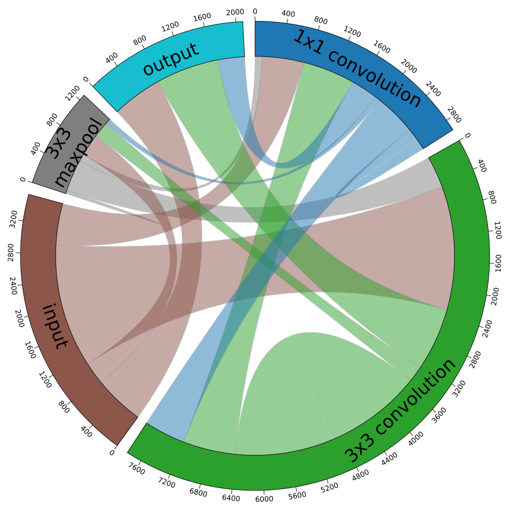
Exploring similarities between architectures with similar performance: Take the NAS-Bench-101 search space as an example; to better understand the correlation between good architectures in the search space and the architectures found, we drew a chord diagram, as shown in Figure 7. It contains a total of four chord diagrams. The operation types of the nodes have been arranged radially around a circle, and the connection relationships between the operations have been drawn as arcs connecting operations. With the exception of Figure 7(a), the remaining three plots present the labels on the outside of the circle at scale intervals of 400 to indicate the number of operations. Figure 7(a) shows the chord diagram drawn by all architectures in the entire search space. As can be seen in the figure, the distribution of the operations that make up the architectures in the search space NAS-Bench-101 is quite uniform. Not only do the *input* and *output* operations appear an equal number of times, but, also, the three operations of 1×1 *convolution*, 3×3 *convolution*, and 3×3 *maxpool* appear an equal number of times. The uniformity of the distribution is also reflected in the number of connections between the operations. In the case of *input*, for example, the operation has an equal number of connections to the operations of 1×1 *convolution*, 3×3 *convolution*, and 3×3 *maxpool*. Figure 7(b) represents the inter-relationships among the operations of the best 1000 architectures on the validation set. Figure 7(c) represents the inter-relationships among the operations of the best 1000 architectures on the test set. Figure 7(d) shows the mean of the inter-relationships among the operations of 5000 architectures obtained in five experiments. It is clear that the architectures found by our algorithm are very similar to the top 1000 architectures on both the validation and test sets, but they are far from the chord diagram of the entire search space shown in Figure 7(a). Comparing Figure 7(d) with Figure 7(b),(c) respectively, we can see that the number of operations occurring is roughly equal and the number of connections between operations is also roughly equal. In the case of 1×1 *convolution*, the number of times it appears in the three chords is very close, at a little over 2800. From the above analysis, it can be stated that there is a certain similarity between good neural architectures in the search space.



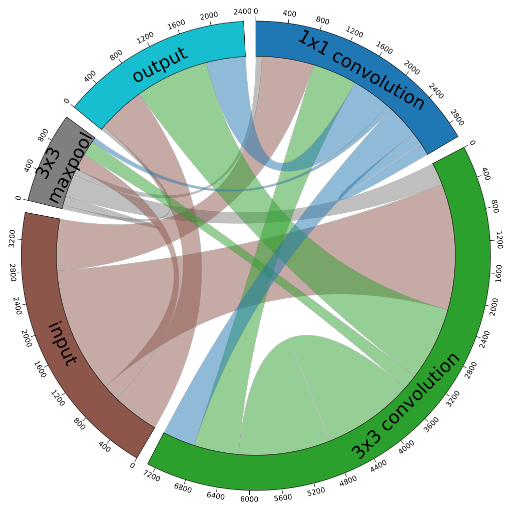
(a) All architectures



(b) Top 1000 architectures (validation)



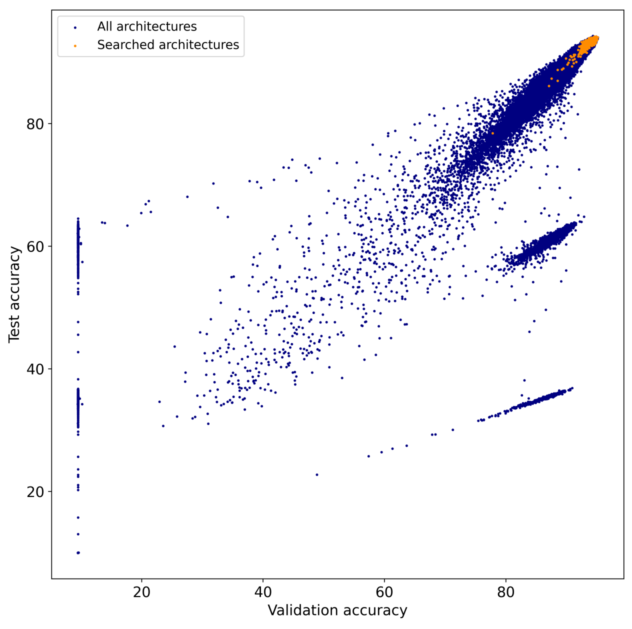
(c) Top 1000 architectures (test)



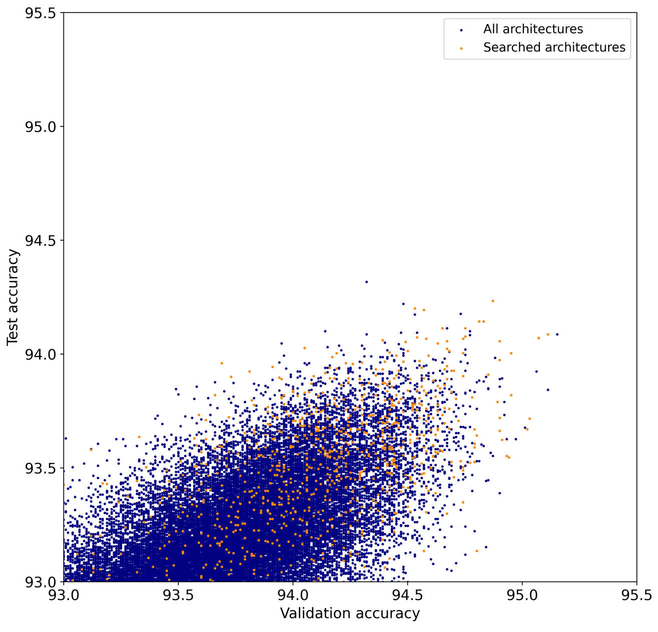
(d) Searched architectures

Figure 7. Chord diagram showing the inter-relationships among the node types of the architectures. The thicker the chord, the greater the number of operations from the current type of operation to another type of operation.

Efficiency of the proposed method: In order to more intuitively show the performance of the proposed algorithm, we chose to plot the validation and test accuracy of all models. As shown in Figure 8, the right figure is a zoomed version of the left figure, with an accuracy of 93.0% to 95.5%. The blue dots represent all of the neural architectures (423,634) in the NAS-Bench-101 search space, and the orange dots represent the 1000 neural architectures searched by the proposed algorithm. The horizontal coordinate is the validation accuracy of trial 0, and the vertical coordinate is the average test accuracy of three trials. It can be intuitively understood from the figure that the proposed algorithm can search the neural architecture while exhibiting excellent performance in the search space. However, due to the inconsistency between the validation accuracy and test accuracy of the neural architectures, the algorithm cannot accurately find the neural architecture with the highest test accuracy.



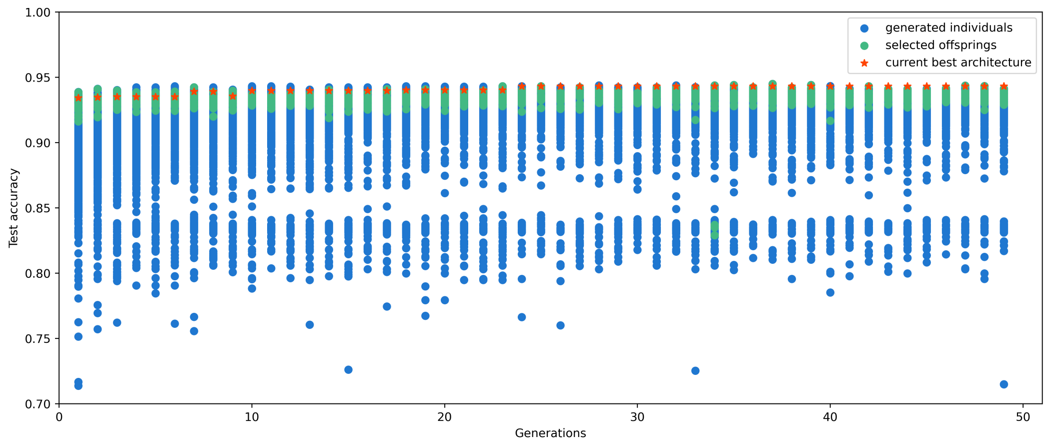
(a) Validation and test accuracy from 0 to 100



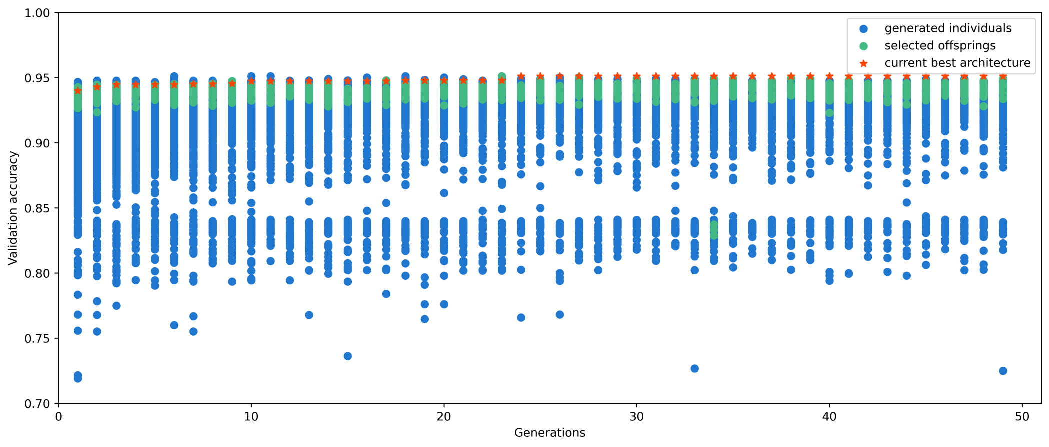
(b) Validation and test accuracy from 93 to 95.5

Figure 8. Validation and test accuracy in NAS-Bench-101. The blue dots represent all neural architectures in the search space, and the orange dots represent the 1000 neural architectures searched by SSENAS.

Efficiency of the surrogate model: To demonstrate the effectiveness of the similarity-based surrogate model, we have plotted the architectures selected from a large number of architectures during evolution, as shown in Figure 9. The red dots in the figure are the best architectures that have been evaluated so far, the blue dots are the neural architectures generated by the proposed evolutionary algorithm, and the green dots represent the architectures selected as offspring by calculating the similarity between the architectures generated by the proposed method and the current optimal architectures. As can be ascertained from the figure, the proposed surrogate model can efficiently select the architectures with excellent performance from the massive architectures. For the NAS-Bench-101 experiment, we employed an evolutionary algorithm to produce 1000 architectures per generation. Leveraging the proposed similarity surrogate model, we strategically selected 20 candidate architectures as offspring individuals. This approach obviates the necessity to query the performance of all 1000 architectures, as only the top 20 are considered. Consequently, this results in a significant computational reduction, equivalent to a 98% decrease in the amount of computation required for the thorough evaluation of network architectures.



(a) Test accuracy vs. Generation



(b) Validation accuracy vs. Generation

Figure 9. Example of accuracy on NAS-Bench-101. In the evolution process, according to the currently known optimal architecture (red star), the proposed surrogate model selects the offspring architectures (green dot) from the massive generated architectures (blue dot).

Efficiency of the architecture generation: The validity of the proposed method is verified by comparing the Gaussian kernel density function of the initial randomly sampled architectures with those of the new architectures generated via genetic operation. As shown in Figure 10, the blue part represents the accuracy on the validation set of randomly sampled 1000 architectures and the corresponding Gaussian kernel density estimation curve, while the orange part represents the accuracy of 1000 architectures generated by the proposed architecture generation strategy and the corresponding Gaussian kernel density estimation curve. The green part represents the accuracy on the validation set of 20 architectures selected by the surrogate model. It is clear from the figure that, with this architecture generation strategy, we can get better architectures than those obtained by randomly sampling architectures. Of course, we will also get some architectures that are inferior to the current architecture. We chose to use the surrogate model based on similarity measurement to filter out the good architectures to gradually search the search space for the good neural architectures. It can be ascertained from the figure that the evolutionary algorithm can generate architectures that exhibit excellent performance, and the proposed surrogate model can find the architectures with excellent performance. It is worth noting that this figure presents the first round of generation information of the evolutionary algorithm.

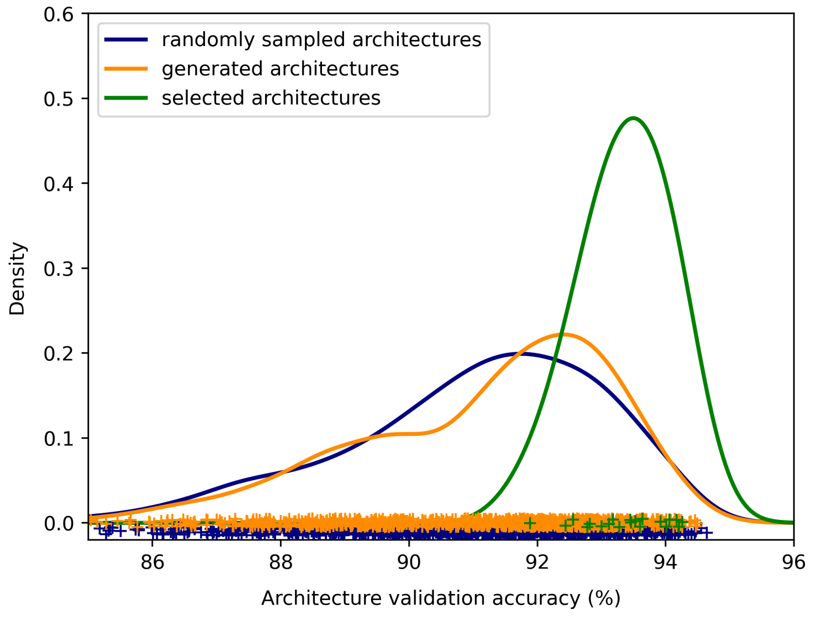
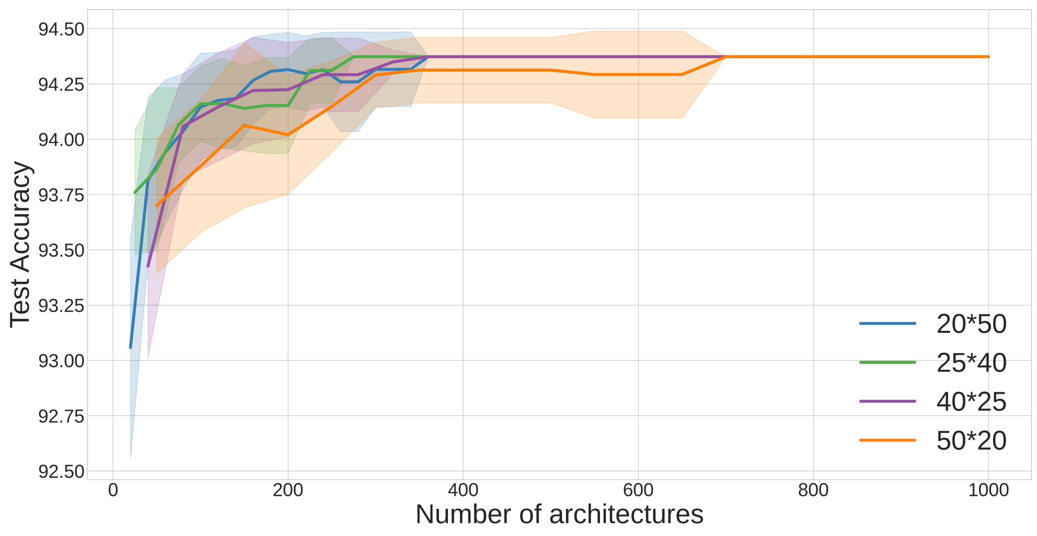
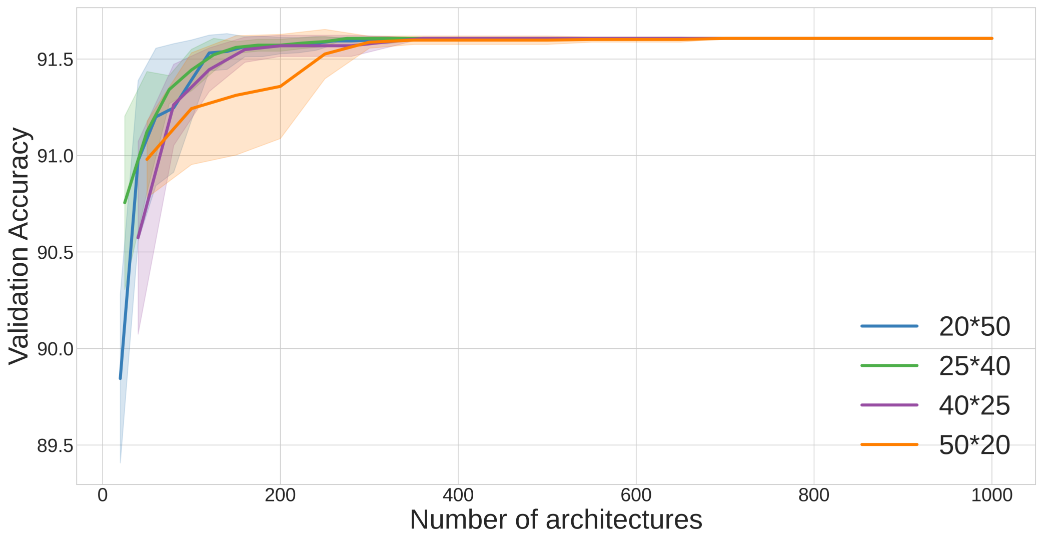


Figure 10. Gaussian kernel density estimation curves illustrating the architectures' performance (NAS-Bench-101).

Sensitivity analysis for evolutionary parameters: The sensitivity analysis for the parameters of the evolutionary algorithm can help us to better understand the performance of the algorithm and possible directions for improvement. We conducted parameter sensitivity analysis experiments to evaluate the population size and the number of generations. These two parameters were set to widely used values according to the community convention [17]: 20*50, 25*40, 40*25, and 50*20, respectively. For example, 20*50 means that the population size is 20 and the number of generations is 50 during the evolutionary process. The reason for this setup is to ensure that the total number of query architectures per experiment is 1000. We conducted experiments on NAS-Bench-201, with each set of parameter experiments performed seven or more times. In Figure 11(a),(b), we respectively show the mean and variance of the performance of the searched architectures for different parameter combinations on the validation set and the test set on CIFAR-10 during the search. The information shown in these two figures indicates that the performance of the proposed algorithm will gradually degrade as the population size gradually exceeds the generations of evolution. As can be ascertained from the figure, from the performance trend of the searched architectures, our method can quickly find the architectures with the optimal performance for NAS-Bench-201 (CIFAR-10) on the validation set. It is worth noting that, except for the 50*20 parameter combination, all parameter combinations can find the architecture with the best performance on the validation set under the condition of less than 400 architectures. This fully demonstrates that our proposed algorithm is not only effective, it is also relatively stable.



(a) Performance curves for test accuracy



(b) Performance curves for validation accuracy

Figure 11. Sensitivity analysis for evolutionary parameters on NAS-Bench-201, showing the performance curves for architectures searched by the algorithm during evolution under different parameter combinations.

5. Conclusions

In this paper, we have proposed a novel approach called SSENAS for the purpose of efficiently searching for excellent neural architectures by using an evolutionary algorithm. A similarity surrogate model has been designed to prevent the surrogate model from being affected by poor-quality architecture information. It is used to select excellent neural architectures as offspring architectures from the massive candidate architectures generated by the evolutionary algorithm. In addition, we have incorporated the dual encoding strategy for evaluation of the surrogate model and generation of excellent neural architectures, respectively. Among them, the encoding strategy for surrogate evaluation focuses on a more comprehensive representation of neural architecture information, and it is achieved via information transmission and the aggregation of architecture nodes, while the encoding strategy for neural architecture generation realizes the decoupling of connections between architecture nodes and the operation types of architecture nodes and focuses more on exploring the architectures that are similar to the current optimal architecture in a search space. Experimental results on NAS benchmarks show that the proposed algorithm is efficient as a tool to progressively find well-performing neural architectures in the search space. In addition, we also show that neural architectures with similar performance have some similarity in their structures. In the search space of NAS-Bench-101, due to the large search space and low performance correlation between the validation set and the test set, this method can still yield promising architectures, although it cannot yield the optimal architecture. However, in the NAS-Bench-201 search space with a small search space and high performance correlation between the validation set and test set, this method only needs 400 queries to find the best performance architecture. In the future work, we will further study the surrogate model for ENAS based on similarity measurement by incorporating the method of weight inheritance, so as to search for the neural architecture with excellent performance under the condition of low resource overhead. In addition, how to generate neural architectures with excellent performance by using evolutionary algorithms is still the focus of our attention.

Use of AI tools declaration

The authors declare they have not used Artificial Intelligence (AI) tools in the creation of this article.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (62376127, 61876089, 61876185, and 61902281), the Natural Science Foundation of Jiangsu Province (BK20141005), and the Postgraduate Research & Practice Innovation Program of Jiangsu Province (KYCX23_1373).

Conflict of interest

Yu Xue and Ferrante Neri are guest editors for [Electronic Research Archive] and were not involved in the editorial review or the decision to publish this article. All authors declare that there are no competing interests.

References

1. C. Swarup, K. U. Singh, A. Kumar, S. K. Pandey, N. varshney, T. Singh, Brain tumor detection using CNN, AlexNet & GoogLeNet ensembling learning approaches, *Electron. Res. Arch.*, **31** (2023), 2900–2924. https://doi.org/10.3934/era.2023146

2. X. He, K. Zhao, X. Chu, AutoML: A survey of the state-of-the-art, *Knowledge-Based Syst.*, **212** (2021), 106622. https://doi.org/10.1016/j.knosys.2020.106622

3. B. Zoph, Q. V. Le, Neural architecture search with reinforcement learning, in *5th International Conference on Learning Representations*, (2017), 1–16.

4. P. Ren, Y. Xiao, X. Chang, P. Huang, Z. Li, X. Chen, et al., A comprehensive survey of neural architecture search: Challenges and solutions, *ACM Comput. Surv.*, **54** (2022), 1–34. https://doi.org/10.1145/3447582

5. B. Lyu, S. Wen, K. Shi, T. Huang, Multiobjective reinforcement learning-based neural architecture search for efficient portrait parsing, *IEEE Trans. Cybern.*, **53** (2023), 1158–1169. https://doi.org/10.1109/TCYB.2021.3104866

6. J. Huang, B. Xue, Y. Sun, M. Zhang, G. G. Yen, Particle swarm optimization for compact neural architecture search for image classification, *IEEE Trans. Evol. Comput.*, **27** (2023), 1298–1312. https://doi.org/10.1109/TEVC.2022.3217290

7. Y. Xue, J. Qin, Partial connection based on channel attention for differentiable neural architecture search, *IEEE Trans. Ind. Inf.*, **19** (2023), 6804–6813. https://doi.org/10.1109/TII.2022.3184700

8. E. Real, A. Aggarwal, Y. Huang, Q. V. Le, Regularized evolution for image classifier architecture search, in *Proceedings of the AAAI Conference on Artificial Intelligence*, AAAI Press, (2019), 4780–4789. https://doi.org/10.1609/aaai.v33i01.33014780

9. Y. Liu, Y. Sun, B. Xue, M. Zhang, G. G. Yen, K. C. Tan, A survey on evolutionary neural architecture search, *IEEE Trans. Neural Networks Learn. Syst.*, **34** (2023), 550–570. https://doi.org/10.1109/TNNLS.2021.3100554

10. D. Floreano, P. Dürr, C. Mattiussi, Neuroevolution: From architectures to learning, *Evol. Intell.*, **1** (2008), 47–62. https://doi.org/10.1007/s12065-007-0002-4

11. S. Liu, H. Zhang, Y. Jin, A survey on computationally efficient neural architecture search, *J. Autom. Intell.*, **1** (2022), 100002. https://doi.org/10.1016/j.jai.2022.100002

12. T. Elsken, J. H. Metzen, F. Hutter, Neural architecture search: A survey, *J. Mach. Learn. Res.*, **20** (2019), 1997–2017.

13. C. White, A. Zela, R. Ru, Y. Liu, F. Hutter, How powerful are performance predictors in neural architecture search, in *35th Conference on Neural Information Processing Systems*, (2021), 1–16.

14. X. Xu, X. Zhao, M. Wei, Z. Li, A comprehensive review of graph convolutional networks: Approaches and applications, *Electron. Res. Arch.*, **31** (2023), 4185–4215. https://doi.org/10.3934/era.2023213

15. W. Wen, H. Liu, Y. Chen, H. Li, G. Bender, P. J. Kindermans, Neural predictor for neural architecture search, *Computer Vision – ECCV 2020*, Springer, (2020), 660–676. https://doi.org/10.1007/978-3-030-58526-6_39

16. C. Wei, C. Niu, Y. Tang, Y. Wang, H. Hu, J. Liang, NPENAS: Neural predictor guided evolution for neural architecture search, *IEEE Trans. Neural Networks Learn. Syst.*, **34** (2022), 8441–8455. https://doi.org/10.1109/TNNLS.2022.3151160

17. B. Wang, B. Xue, M. Zhang, Surrogate-assisted particle swarm optimization for evolving variable-length transferable blocks for image classification, *IEEE Trans. Neural Networks Learn. Syst.*, **33** (2021), 3727–3740. https://doi.org/10.1109/TNNLS.2021.3054400

18. Y. Xu, Y. Wang, K. Han, Y. Tang, S. Jui, C. Xu, et al., ReNAS: Relativistic evaluation of neural architecture search, in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, (2021), 4409–4418. https://doi.org/10.1109/CVPR46437.2021.00439

19. L. Xie, A. Yuille, Genetic CNN, in *2017 IEEE International Conference on Computer Vision (ICCV)*, IEEE, (2017), 1388–1397. https://doi.org/10.1109/ICCV.2017.154

20. Y. Sun, B. Xue, M. Zhang, G. G. Yen, Evolving deep convolutional neural networks for image classification, *IEEE Trans. Evol. Comput.*, **24** (2020), 394–407. https://doi.org/10.1109/TEVC.2019.2916183

21. Y. Xue, Y. Wang, J. Liang, A. Slowik, A self-adaptive mutation neural architecture search algorithm based on blocks, *IEEE Comput. Intell. Mag.*, **16** (2021), 67–78. https://doi.org/10.1109/MCI.2021.3084435

22. B. Deng, J. Yan, D. Lin, Peephole: Predicting network performance before training, preprint, arXiv:1712.03351.

23. Y. Tang, Y. Wang, Y. Xu, H. Chen, B. Shi, C. Xu, et al., A semisupervised assessor of neural architectures, in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, IEEE, (2020), 1807–1816. https://doi.org/10.1109/CVPR42600.2020.00188

24. Y. Chen, Y. Guo, Q. Chen, M. Li, W. Zeng, Y. Wang, et al., Contrastive neural architecture search with neural architecture comparators, in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, (2021), 9497–9506. https://doi.org/10.1109/CVPR46437.2021.00938

25. M. Huang, Z. Huang, C. Li, X. Chen, H. Xu, Z. Li, et al., Arch-Graph: Acyclic architecture relation predictor for task-transferable neural architecture search, in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, (2020), 11871–11881. https://doi.org/10.1109/CVPR52688.2022.01158

26. C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, F. Hutter, NAS-Bench-101: Towards reproducible neural architecture search, in *Proceedings of the 36th International Conference on Machine Learning*, PMLR, (2019), 7105–7114.

27. X. Dong, Y. Yang, NAS-Bench-201: Extending the scope of reproducible neural architecture search, in *International Conference on Learning Representations*, (2020), 1–16.

28. Y. Liu, Y. Tang, Y. Sun, Homogeneous architecture augmentation for neural predictor, in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, IEEE, (2021), 12229–12238. https://doi.org/10.1109/ICCV48922.2021.01203

29. B. Guo, T. Chen, S. He, H. Liu, L. Xu, P. Ye, et al., Generalized global ranking-aware neural architecture ranker for efficient image classifier search, in *Proceedings of the 30th ACM Interna-tional Conference on Multimedia*, ACM, (2022), 3730–3741. https://doi.org/10.1145/3503161.3548149

30. X. Zhou, S. Liu, K. Wong, Q. Lin, K. Tan, A hybrid search method for accelerating convolutional neural architecture search, in *Proceedings of the 2023 15th International Conference on Machine Learning and Computing*, ACM, (2023), 177–182. https://doi.org/10.1145/3587716.3587745

31. S. Yan, Y. Zheng, W. Ao, X. Zeng, M. Zhang, Does unsupervised architecture representation learning help neural architecture search, in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., (2020), 12486–12498.

32. Y. Sun, H. Wang, B. Xue, Y. Jin, G. G. Yen, M. Zhang, Surrogate-assisted evolutionary deep learning using an end-to-end random forest-based performance predictor, *IEEE Trans. Evol. Comput.*, **24** (2020), 350–364. https://doi.org/10.1109/TEVC.2019.2924461

33. R. Luo, F. Tian, T. Qin, E. Chen, T. Y. Liu, Neural architecture optimization, in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., (2018), 7827–7838.

34. X. Xie, Y. Sun, Y. Liu, M. Zhang, K. C. Tan, Architecture augmentation for performance predictor via graph isomorphism, *IEEE Trans. Cybern.*, **2023** (2023), 1–13. https://doi.org/10.1109/TCYB.2023.3267109

35. G. T. Pereira, I. B. Santos, L. P. Garcia, T. Urruty, M. Visani, A. C. De Carvalho, Neural architecture search with interpretable meta-features and fast predictors, *Inf. Sci.*, **649** (2023), 119642. https://doi.org/10.1016/j.ins.2023.119642

36. Y. Li, C. Hao, P. Li, J. Xiong, D. Chen, Generic neural architecture search via regression, in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., (2021), 20476–20490.

37. C. White, W. Neiswanger, Y. Savani, BANANAS: Bayesian optimization with neural architectures for neural architecture search, in *Proceedings of the AAAI Conference on Artificial Intelligence*, AAAI Press, (2021), 10293–10301. https://doi.org/10.1609/aaai.v35i12.17233

38. B. Wu, K. Keutzer, X. Dai, P. Zhang, Y. Wang, F. Sun, et al., FBNet: Hardware-aware efficient ConvNet design via differentiable neural architecture search, in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, (2019), 10726–10734. https://doi.org/10.1109/CVPR.2019.01099

39. Z. Guo, X. Zhang, H. Mu, W. Heng, Z. Liu, Y. Wei, et al., Single path one-shot neural architecture search with uniform sampling, in *Computer Vision – ECCV 2020*, Springer, (2020), 544–560. https://doi.org/10.1007/978-3-030-58517-4_32

40. X. Chu, B. Zhang, R. Xu, FairNAS: Rethinking evaluation fairness of weight sharing neural architecture search, in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, IEEE, (2021), 12219–12228. https://doi.org/10.1109/ICCV48922.2021.01202

41. J. Mellor, J. Turner, A. Storkey, E. J. Crowley, Neural architecture search without training, in *Proceedings of the 38th International Conference on Machine Learning*, PMLR, (2021), 7588–7598.

42. L. Fan, H. Wang, Surrogate-assisted evolutionary neural architecture search with network embedding, *Complex Intell. Syst.*, **9** (2023), 3313–3331. https://doi.org/10.1007/s40747-022-00929-w

43. J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, et al., Scalable Bayesian optimization using deep neural networks, in *Proceedings of the 32nd International Conference on Machine Learning*, PMLR, (2015), 2171–2180.

44. C. Wei, Y. Tang, C. N. C. Niu, H. Hu, Y. Wang, J. Liang, Self-supervised representation learning for evolutionary neural architecture search, *IEEE Comput. Intell. Mag.*, **16** (2021), 33–49. https://doi.org/10.1109/MCI.2021.3084415

AIMS Press