Complex & Intelligent Systems (2024) 10:1667–1687
https://doi.org/10.1007/s40747-023-01230-0

ORIGINAL ARTICLE

EGFA-NAS: a neural architecture search method based on explosion gravitation field algorithm

Xuemei Hu[1] · Lan Huang[1,2] · Jia Zeng[1] · Kangping Wang[1] · Yan Wang[1,3]

Received: 3 March 2023 / Accepted: 3 September 2023 / Published online: 30 September 2023

Abstract

Neural architecture search (NAS) is an extremely complex optimization task. Recently, population-based optimization algorithms, such as evolutionary algorithm, have been adopted as search strategies for designing neural networks automatically. Various population-based NAS methods are promising in searching for high-performance neural architectures. The explosion gravitation field algorithm (EGFA) inspired by the formation process of planets is a novel population-based optimization algorithm with excellent global optimization capability and remarkable efficiency, compared with the classical population-based algorithms, such as GA and PSO. Thus, this paper attempts to develop a more efficient NAS method, called EGFA-NAS, by utilizing the work mechanisms of EGFA, which relaxes the search discrete space to a continuous one and then utilizes EGFA and gradient descent to optimize the weights of the candidate architectures in conjunction. To reduce the computational cost, a training strategy by utilizing the population mechanism of EGFA-NAS is proposed. In addition, a weight inheritance strategy for the new generated dust individuals is proposed during the explosion operation to improve performance and efficiency. The performance of EGFA-NAS is investigated in two typical micro search spaces: NAS-Bench-201 and DARTS, and compared with various kinds of state-of-the-art NAS competitors. The experimental results demonstrate that EGFA-NAS is able to match or outperform the state-of-the-art NAS methods on image classification tasks with remarkable efficiency improvement.

Keywords Neural architecture search · Explosion gravitation field algorithm · Complex optimization task · Deep neural networks

Introduction

Deep neural networks (DNNs) have made significant progress in various challenging tasks, including image classification [1–4], object detection [5–7], and segmentation [8, 9]. One of the key factors behind the progress lies in the innovation of neural architectures. For example, VGGNet [1] suggested the use of smaller convolutional filters and stacked a series of convolution layers to achieve better performance. ResNet [10] introduced the residual blocks to benefit the training of deeper neural networks. DenseNet [11] designed the densely connected blocks to stack features from different depths. Generally, manually designing a powerful and efficient neural network architecture requires a lot of expert experiments and domain knowledge. It is not recently until a series of neural architecture search (NAS) methods have been proposed, bringing great convenience to ordinary users and learners, and allowing them to benefit from the success of deep neural networks.

✉ Lan Huang
huanglan@jlu.edu.cn

✉ Yan Wang
wy6868@jlu.edu.cn

Xuemei Hu
huxm18@mails.jlu.edu.cn

Jia Zeng
zengjia22@mails.jlu.edu.cn

Kangping Wang
wangkp@jlu.edu.cn

[1] College of Computer Science and Technology, Jilin University, Changchun 130012, China

[2] Key Laboratory of Symbol Computation and Knowledge Engineering of Ministry of Education, Jilin University, Changchun 130012, China

[3] School of Artificial Intelligence, Jilin University, Changchun 130012, China

Generally, a NAS task can be regarded as a complex optimization problem. In machine learning and computational intelligence, population-based intelligent optimization algorithms, such as genetic algorithm (GA) and particle swarm optimization (PSO), have been widely adopted as the concept of neuroevolutionary, to optimize the topology structure and hyperparameters of neural networks in the late 1990 [12–14]. Recently, lots of NAS methods employing population-based intelligent optimization algorithms as search strategies have attracted increasing attention. Although intelligent optimization algorithms, such as GA, have a competitive search performance on various complex optimization tasks, they still suffer from high computational costs. This shortcoming is particularly true in NAS tasks since the NAS process involves a large number of architecture evaluations. More specifically, for the NAS task, each network architecture evaluation involves the completed training of a deep neural network on a large amount of data from scratch. For example, Hierarchical EA [15] consumes 300 GPU days, and AmoebaNet-A [16] consumes 3150 GPU days to search architectures on the CIFAR-10.

In addition, reinforcement learning (RL) is also adopted to design neural architectures automatically, such as [7, 17, 18]. A significant limitation of RL-based NAS methods is also computationally expensive despite their remarkable performance. For example, it takes 2000 GPU days for the typical RL-based method NASNet-A to obtain an optimized CNN architecture on CIFAR-10. These methods require a large number of computational resources, which is unaffordable for most researchers and learners. To reduce the computational cost, ENAS [18] proposed a parameter-sharing strategy, which shares weights among the architectures through the use of superset and is adopted in various gradient descent (GD) NAS methods, such as [19–21]. Compared with EA-based and RL-based NAS methods, GD-based NAS methods are usually more efficient, which apply gradient descent to optimize the weights of candidate architectures. However, GD-based NAS methods still have some limitations, such as requiring excessive GPU memory during the search, and resulting in premature convergence to the local optimum [22, 23].

Recently, some population-based methods, such as the various EA-based methods [15, 16, 24–28], have been utilized for NAS tasks and have achieved some progress. The explosion gravitation field algorithm (EGFA) [29] inspired by the formation process of planets is a novel intelligent optimization algorithm with excellent global optimization capability and remarkable efficiency, compared with the classical population-based optimization algorithms, such as GA and PSO. Nowadays, computational time and resource limitations remain the major bottleneck in using and developing NAS methods. Thus, this paper attempts to develop a more efficient NAS method, by utilizing the work mechanisms of EGFA, to allow for discovering an optimal neural architecture with competitive learning accuracy, but only consuming a little computational time and resources. Specifically, the proposed EGFA-NAS utilizes EGFA and gradient descent to optimize the weights of the candidate architectures in conjunction. To reduce the computational cost, EGFA-NAS proposes a training strategy by utilizing the population mechanism of EGFA-NAS. To improve the efficiency and performance, EGFA-NAS proposes the weight inheritance strategy for the new generated dust individuals during the explosion operation. The main contributions of this paper are summarized as follows.

1. A novel population-based NAS method is proposed, called EGFA-NAS, which utilizes EGFA and gradient descent to optimize the weights of candidate architecture jointly, and is applicable to any universal micro search space with a fixed number of edges and a determined candidate operations set, such as NAS-Bench-201 and DARTS search space.
2. A training strategy is proposed to reduce the computational cost by utilizing the population mechanism. Specifically, all dust individuals cooperate to complete the training of the dataset at each epoch. Although each dust individual is only trained on part of batches at each epoch, it will be trained on all batches over a large number of epochs.
3. A weight inheritance is proposed to improve performance and efficiency. Specifically, during the explosion operation, the weights w of each new generated dust individual are inherited from the center dust. By utilizing this strategy, the new generated can be evaluated directly at the current epoch without retraining.
4. The experimental results show that the optimal neural network architectures searched by EGFA-NAS have competitive learning accuracy and require the least computational cost, compared with four kinds of state-of-the-art NAS methods.

The remainder of the paper is organized as follows. "Related work" introduces the related work of the work. "Proposed NAS method" describes the details of this proposed NAS method. The experimental design and results are presented in "Experimental design" and "Experimental results", respectively. The final part is the conclusion placed in "Conclusion".

Related work

General formulation of NAS task

NAS is an extremely complex optimization task, the primary objective of which is to transform the process of manually designing neural networks into automatically searching for optimal architectures. The process of the NAS can be depicted in Fig. 1. During the search, the search strategy samples a candidate architecture from the search space. Then we train the architecture to converge and evaluate the architecture's performance. Next, the search strategy picks up another candidate architecture for training and evaluation according to the evaluation result of the last architecture.

In NAS tasks, denote a neural network architecture as A and the weights of all functions of the neural network as w_A. Then the goal of NAS is to find an architecture A, which can achieve the minimum validation loss L_{V} after being trained by minimizing the training loss L_{T}, as shown in Eq. (1).

$$\begin{aligned} &\min_{A} \; L_{\mathrm{V}}\left(w_A^*,\, A\right) \\ &\text{s.t. } w_A^* = \arg\min_{w} \; L_{\mathrm{T}}\left(w^A,\, A\right), \end{aligned} \tag{1}$$

where w_A^* is the best weight of A and achieves the minimum loss of training dataset. L_{T} and L_{V} are the losses on training dataset and validation dataset, respectively. Both losses are determined not only by the architecture A, but also the weights w. This is a bi-level optimization problem [30] with A as the upper-level variable and w as the lower-level variable.

NAS methods

Search strategy determines how to sample the neural network architectures. According to the different kinds of search strategy, NAS methods can be roughly divided into three categories: EA-based NAS methods, RL-based NAS methods, GD-based NAS methods.

EA-based NAS methods

EA-based NAS methods use evolutionary algorithms (EAs) to sample neural architectures. Early EA-based research for the optimization of networks was proposed as the concept of neuroevolutionary [12–14], which not only optimizes the network's topology but also optimizes the hyperparameters and connection weights associated with the network. Over the past years, EA-based NAS methods have attracted increasing attention. For example, Xie et al. published the first EA-based NAS work GeNet [31] in 2017, which encodes the candidate architectures using fixed-length binary strings. Real et al. searched network architectures by EA, and started searching from trivial initial conditions [27]. Subsequently, Real et al. evolved an image classifier: AmoebaNet-A [16], which modifies the tournament selection by introducing a concept of age and surpasses hand designs for the first time. Liu et al. proposed Hierarchical EA [15], which combines a novel hierarchical genetic representation scheme that imitates the modularized design pattern and expressive search space. Elsken et al. proposed the LEMONADE [24], which is an evolutionary algorithm for multi-objective architecture search. Suganuma et al. constructed CNN architectures based on Cartesian genetic programming (CGP) [25]. Sun et al. proposed CNN-GA [26] and AE-CNN [32], which evolves CNN architectures using GA, based on ResNet and DenseNet blocks. To accelerate the fitness evaluation in evolutionary deep learning, Sun and Wang et al. proposed an end-to-end offline performance predictor based on the random forest [33].

Although the neural network architectures searched by above EA-based NAS methods have achieved competitive performance compared with the state-of-the-art handle-designed CNNs, however, as the population-based methods, they still suffer from huge resource costs because of involving a large number of fitness evaluations. During the search phase, each new generated candidate architecture needs to be trained on a training dataset and evaluated on a validation dataset. Then most EA-based NAS methods are time-consuming. For example, to search architectures on the CIFAR-10 dataset, Hierarchical EA [15] needs 300 GPU days, AmoebaNet-A [16] needs 3150 GPU days, CNN-GA [26] needs 35 GPU days, and AE-CNN [32] needs 27 GPU days. Then it is essential to accelerate the evaluation process for EA-based NAS methods, especially under the condition of limited computational resources.

RL-based NAS methods

The agent, environment, and reward are the three factors of reinforcement learning (RL). In the context of NAS, sampling the network architectures from the search space by the controller is defined as the action of the agent, the performance of network is regarded as the reward, and the controller is updated based on the reward in the next iteration. The earliest RL-based NAS method was proposed by Zoph et al. in 2017, which used RNNs as controllers to sample the network architecture and generate actions via policy gradients [7]. Subsequently, Zoph et al. used a proximal optimization strategy to optimize the RNN controller [17]. Cai et al. presented a RL-based algorithm: ProxylessNAS [34], which is an alternative strategy to handle hardware metrics. BlockQNN [35] automatically builds high-performance networks using the Q-Learning paradigm with epsilon-greedy exploration strategy.

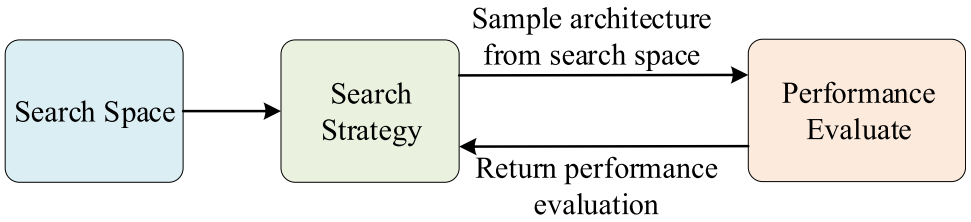

Fig. 1 Process of neural architecture search

Earlier RL-based NAS methods are usually computationally expensive. To reduce the computational cost, work [17] proposed the well-known NASNet search space, which allows us to search the best cell on the CIFAR-10 dataset and then apply this cell to the ImageNet dataset by stacking together more copies of this cell. ENAS [18] proposed a parameter-sharing strategy and the one-shot estimator (OSE), which regards all candidate architectures as the subgraphs of the super-network. Then all candidate architectures can share the parameters.

GD-based NAS methods

Recently, there is an increasing interest in adopting gradient descent (GD) methods for NAS tasks. A typical GD-based NAS method is DARTS [19], which optimizes the network architecture parameters by GD methods after converting the discrete search space into a continuous search space through a relaxation strategy. Subsequently, Dong et al. proposed the GDAS [20], which develops a learnable differentiable sampler to accelerate the search procedure. Xie et al. proposed the SNAS [21], which trains neural operation parameters and architecture distribution parameters by proposing a novel search gradient. Above-mentioned ProxylessNAS [34] proposed a gradient-based approach to handle non-differentiable hardware objectives.

Compared with EA-based and RL-based NAS methods, GD-based NAS methods are every efficient, because they represent the structures of the candidate networks as directed acyclic diagrams (DAGs) and use the parameter-sharing strategy. However, GD-based NAS methods have some drawbacks. For example, references [22, 23] point out that the DARTS tends to select skip-connection operations, which leads to performance degradation of searched architectures. To overcome the shortcoming of DARTS [19], several variants of DARTS methods have been proposed, such as DARTS- [36], DARTS + [37], RC-DARTS [38], and β-DARTS [39].

Besides the above three kinds of NAS methods, there are also other NAS methods that are not mentioned or do not fully fall into the above categories. For example, Liu et al. proposed the PNAS [40], which uses a sequential model-based optimization (SMBO) strategy.

Explosion gravitation field algorithm

Explosion gravitation field algorithm (EGFA) [29] is a novel optimization algorithm based on the original GFA [40–43], which stimulates the formation process of planets based on SNDM [44]. It was proposed by our research team in 2019 and has achieved good performance when solving optimization problems and tasks, such as benchmark functions [29] and feature selection tasks [45]. Compared with the classical population-based intelligent algorithm, such as genetic algorithm (GA) and particle swarm optimization (PSO), EGFA has better global optimization capability and remarkable efficiency. In addition, the fact that EGFA converges to the global best solution with probability 1 under some conditions has been proven [29].

In EGFA, all individuals can be mimicked as dust particles with mass, and each of them belong to a certain group. In every group, the one with the maximum mass value is regarded as the center dust and the others are surrounding dust particles. Based on the idea of SNDM [44], each center dust attracts its surrounding dust by the gravitation field, and the gravitation field makes all surrounding dust particles move toward their centers. In EGFA, each dust particle can be represented by a four-tuple (location, mass, group, flag), where flag is a Boolean value indicating whether it is a center, location corresponds to a solution for the problem, group indicates the group number, mass is the value of objective function. When the value of mass is bigger, the solution is better. There are six basic operations for EGFA as Fig. 2: (1) dust sampling (DS), (2) initialize, (3) group, (4) move and rotate, (5) absorb, and (6) explode. The detailed processes of EGFA are summarized as follows:

Step 1: Subspace location by dust sampling (DS). The task of DS is to efficiently locate a small enough search space which more likely contains the optimal solution.

Step 2: Initialize the dust population randomly based on the subspace located by Step 1.

Step 3: Divide the dust population into several subgroups randomly, and calculate the mass value of all individuals. In each group, set the dust particle with maximum mass value

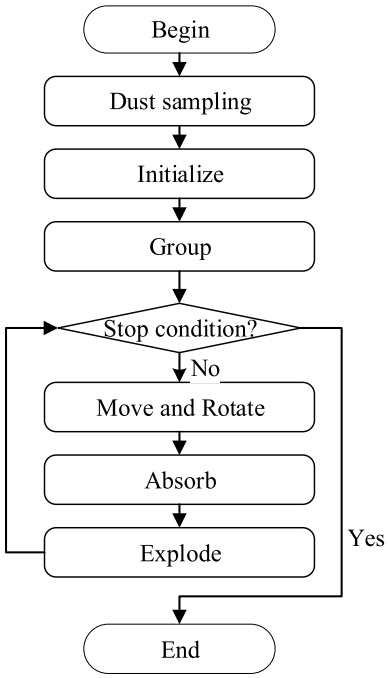

Fig. 2 Flow chart of EGFA

as the center, and set its flag as 1; set the other individuals as the surrounding dust particles, set their flag as 0.

Step 4: Check the stop condition. If the stop condition is met, return the best solution and the algorithm terminates, otherwise goes to Step 5.

Step 5: Perform the movement and rotation operation. In each group, each center attracts its surrounding dust particles by the gravitation field, and the gravitation field makes all surrounding dust particles move toward their centers.

Step 6: Perform the absorbing operation. Some surrounding dust particles which are close to their centers enough are absorbed by the centers. The size of the dust population will decrease in this process.

Step 7: Perform the explosion operation, and some new dust particles are generated around the centers. When explosion operation is accomplished, algorithm goes to Step 4.

In addition, DS in Step 1 avoids a long iterative process because the algorithm only searches in the subspace which is small enough compared with the original search space. The explosion operation maintains the size of the population and can stop the algorithm from being in stagnation behavior because of falling into local optima.

In this work, we proposed a NAS method based on explosion gravitation field algorithm, EGFA-NAS for short. In EGFA-NAS, an individual (a dust particle) represents a candidate network architecture. EGFA-NAS aims to discover a network architecture with the best performance, such as accuracy on the testing dataset. For the NAS task, the subspace small enough that contains the best architecture is hard to locate and computationally intensive. Therefore, EGFA-NAS abandons the first operation DS. As a population-based method for the NAS task, there are several key issues to be addressed. Namely, (1) which type of search space to search, (2) how to represent and code a CNN network, (3) how to accelerate the network architecture evaluation process, (4) how to use heuristic information to guide the search process.

Proposed NAS method

Micro search spaces, such as NASNet [17], DARTS [19], and NAS-Bench-201 [23] search spaces, are popularly utilized for NAS tasks recently, which search for the neural cells to form the blocks and construct the macro skeleton of network by stacking multiple blocks multiple times as [16–20, 23, 46]. In this work, we propose an efficient NAS method for micro search space. To investigate the performance of our proposed method sufficiently, we choose two classical micro search spaces: i.e., NAS-Bench-201 and DARTS search space to test.

Representation of search space

In this work, we search for a computation cell as the building block of the final architecture and represent a cell as a directed acyclic diagram (DAG). Specifically, a node represents the information flow, e.g., a feature map in CNNs, and an edge between two nodes donates the candidate operation, which is known as successful modules designed by human experts. We denoted O as the candidate operations set. To process the intermediate nodes more efficiently in the forward propagation, two kinds of cells need to be searched: normal cell with stride of 1 and reduction cell (block) with stride of 2. Once the two kinds of cells are identified, we can stack multiple copies of the searched cell to make up a whole neural network. In the rest of this section, we introduce the two search spaces: NAS-Bench-201 and DARTS search space, respectively.

NAS-Bench-201

NAS-Bench-201 was proposed by Dong et al. [23], which is an algorithm-agnostic micro search space. Specifically, a cell from NAS-Bench-201 includes one input node, three computational nodes, the last computational node is also the output node for next cell. Every edge in a cell has five candidate options. Then a cell in NAS-Bench-201 can be represented as a DAG, the nodes of which are connected fully, and there is $5^{C_4^2} = 15,625$ cell candidates in total. In NAS-Bench-201, the candidate operations set O contains the following FIVE operations: (1) zeroize, (2) skip-connection, (3) 1×1

convolution, (4) 3×3 convolution, and (5) 3×3 average pooling.

As shown in Fig. 3, the macro skeleton of NAS-Bench-201 is mainly stacked by three normal blocks and connected by two reduction blocks. Each normal block consists of B normal cells. The reduction block is the basic reduction block [10], which serves to down-sample the spatial size and double the channels of an input feature map. The skeleton is initiated with one 3×3 convolution, and ends up with a global average pooling layer to flatten the feature map into a feature vector.

Additionally, work [23] evaluates each candidate architecture for NAS-Bench-201 on three different datasets: CIFAR-10, CIFAR-100 [47], and ImageNet-16-120 [48]. Then once the final architecture is found, the retraining process is not essential, and we can directly obtain the network final performance by the API provided by [23].

DARTS search space

DARTS [19] search space is a popular micro search space, proposed by Liu et al. in 2019, which is similar to NASNet [17] search space but removes some unused operations and adds some powerful operations. Specifically, a cell from the DARTS search space contains two input nodes, four computational nodes, and one output node. The output node is the concatenation of four computational nodes. As the depiction in Fig. 4, there are 14 edges in a cell for search, and each edge has 8 options. Unlike the NAS-Bench-201, the nodes in a cell are not connected fully during the search phase. Moreover, during the evaluation phase, each node only connects with two previous nodes. In DARTS search space, the candidate operations set O contains the following eight operations: (1) identify, (2) zeroize, (3) 3×3 depth-wise separate convolution, (4) 3×3 dilated depth-wise separate convolution, (5) 5×5 depth-wise separate convolution, (6) 5×5 dilated depth-wise separate convolution, (7) 3×3 average pooling, (8) 3×3 max pooling.

As shown in Fig. 4, B normal cells are stacked as one normal block. For a given image, it forwards thought a 3×3 convolution and then forwards thought three normal blocks with two reduction cells in between. In this paper, we follow the work [19] to set up the overall network architecture of DARTS search space.

Overall of search process

Figure 5 shows the overall of search process in EGFA-NAS. (a) Operations on edges are initialized unknown. (b) Continuous relaxation of search space and sampling the candidate operations for the edges with the mix probabilities. (c) Optimize the mix probabilities and the weights of cells simultaneously. (d). Inferring the final structure of cell from the learned mixing probabilities

Representation and encoding of cell

As discussed in "Representation of search space", the cells to search in this work can be represented by the DAGs. Specifically, each computational node represents one feature map, which is transformed from the previous feature map. Each edge in this DAG is associated with an operation transforming the feature map from one node to another node. All possible operations are selected from a candidate operation set O. Then the output of any node j can be formulated as Eq. (2).

$$I_j = \sum_{i<j} o_{i,j}(I_i), \tag{2}$$

where I_i and I_j represent the output of the node i and node j, respectively. $o_{i,j}$ represents the operation transforming the feature map from node i to node j, which is selected from the candidate operation set O.

In NAS-Bench-201 [23], a normal cell contains four nodes, i.e., $\{I_i | 0 \le i \le 3\}$. I_0 is the output tensor of the previous layer, I_1, I_2, I_3 are the output tensors of node 1, 2, and 3, calculated by Eq. (2). According to work [23], a normal cell contains six edges and each edge has five candidate operations.

In DARTS search space, a cell contains seven nodes, i.e., $\{I_i | 0 \le i \le 6\}$. I_0 and I_1 are the input tensors, I_2, I_3, I_4 and I_5 are the output tensor of node 2, 3, 4, and 5. I_6 indicates the output of this cell, which is the concatenation of the four computational nodes, i.e., $I_6 = I_2 \cap I_3 \cap I_4 \cap I_5$.

Define e as the number of edges for a cell, $|O|$ represents the size of the candidate operations set O. According to the above description of NAS-Bench-201 and DARTS search space, a cell can be encoded as A with size of $e \times |O|$. In NAS-Bench-201, $e = 6$, $|O| = 5$, A is a tensor with size of 6×5. In DARTS search space, $e = 14$, $|O| = 8$, A is a tensor with size of 14×8. A general representation for a cell is formulated as Eq. (3).

$$A = \begin{bmatrix} \vec{a}_0 \\ \vec{a}_1 \\ \vdots \\ \vec{a}_p \\ \vdots \\ \vec{a}_{e-1} \end{bmatrix} = \begin{bmatrix} a_0^0, & a_0^1, & \cdots, & a_0^q, & \cdots, & a_0^{|O|-1} \\ a_1^0, & a_1^1, & \cdots, & a_1^q, & \cdots, & a_1^{|O|-1} \\ \vdots & \vdots & \cdots & \vdots & \cdots & \vdots \\ a_p^0, & a_p^1, & \cdots, & a_p^q, & \cdots, & a_p^{|O|-1} \\ \vdots & \vdots & \cdots & \vdots & \cdots & \vdots \\ a_{e-1}^0, & a_{e-1}^1, & \cdots, & a_{e-1}^q, & \cdots, & a_{e-1}^{|O|-1} \end{bmatrix}, \tag{3}$$

where $\vec{a}_p$ represents the probabilities of sampling the $|O|$ candidate operations for edge p, a_p^q is the qth element of $\vec{a}_p$ and represents the probability of sampling the qth candidate operation for edge p. In fact, the way of encoding for cell as

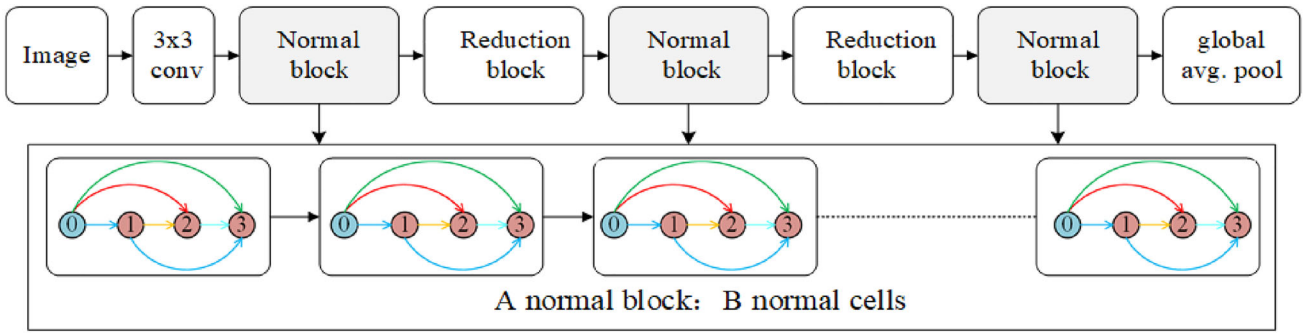

Fig. 3 Macro skeleton of NAS-Bench-201

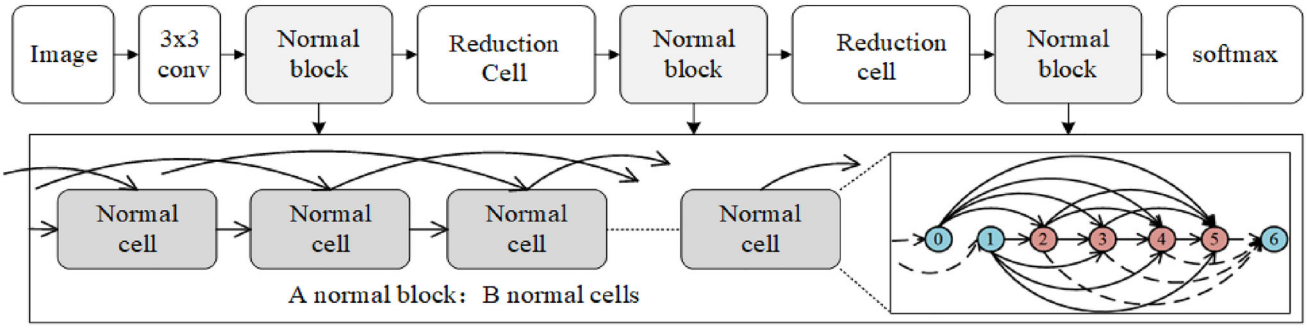

Fig. 4 Macro skeleton of DARTS search space

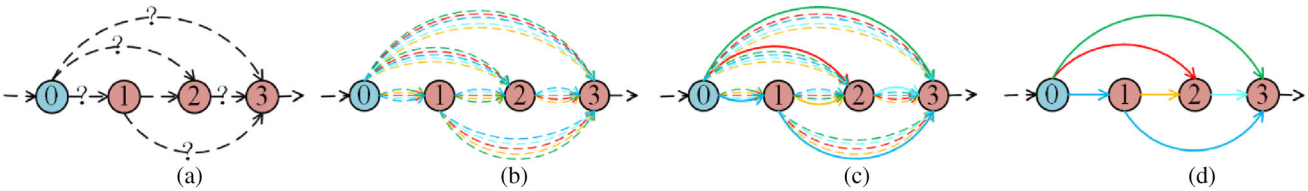

Fig. 5 Overall of search process

Eq. (3) can be used for any micro search space, the searched cell of which has a fixed number of edges e and a defined candidate operations set O.

Continuous relaxation of the search space

As described in "Representation of search space", a neural network architecture consists of many copies of the cell. These cells are sampled from the NAS-Bench-201 and DARTS search space. Specifically, from node j to node i, we sample the transformation function from the candidate operation set O with a discrete probability $\alpha_{(i\leftarrow j)}$. During the search, we calculated each node in a cell by Eq. (4).

$$I_i = \sum_{j=0}^{i-1}\sum_{k=0}^{|O|}\alpha_{(i\leftarrow j)}^{k} o^k\left(I_j, w_{(i\leftarrow j)}^{k}\right), \tag{4}$$

where $|O|$ is the number of candidate operation of the set O, $\alpha_{(i\leftarrow j)}^{k}$ represents the probability for the edge$_{(i\leftarrow j)}$ (from node j to node i) that selects the kth candidate operation as the transformation function, o^k represents the kth candidate operation, I_j is the output of node j, $w_{(i\leftarrow j)}^{k}$ is the weight for the function of o^k on edge$_{(i\leftarrow j)}$. To make the search space continuous, we relax the probability of a particular operation $\alpha_{(i\leftarrow j)}^{k}$ to a softmax over all possible operations by Eq. (5).

$$\alpha_{(i\leftarrow j)}^{k} = \frac{\exp\left(\left(\alpha_{(i\leftarrow j)}^{k} + c_k\right) + \tau\right)}{\sum_{k'=0}^{k'<|O|}\exp\left(\left(\alpha_{(i\leftarrow j)}^{k'} + c_{k'}\right) + \tau\right)}, \tag{5}$$

where c_k are i.i.d that samples from Gumble(0, 1), $c_k = -log(-log(u))$ with u $\sim$ Unif[0, 1]. τ is a softmax temperature; in this work, τ is set 10 as same as study [23].

Training strategy

In this work, we aim to reduce the computational cost by utilizing the population mechanism of EGFA-NAS. The main

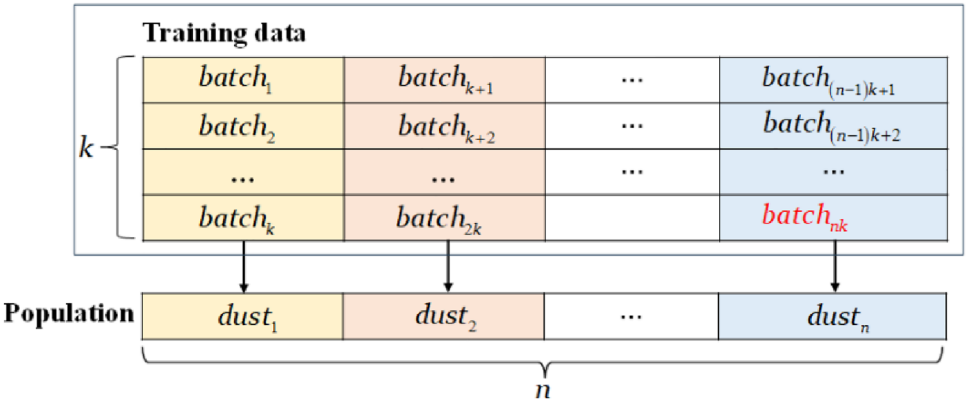

Fig. 6 Training strategy for EGFA-NAS

idea of the training strategy is illustrated as Fig. 6. Specifically, define D_T as the training dataset, batch_num as the number of batches of D_T, n as the population size. At each epoch, each dust individual is training on k batches, where $k = \lceil \text{batch\_num}/n \rceil$. All dust individuals cooperate to complete the training of the dataset at each epoch. This training process repeats until the maximum number of epochs is reached. Each dust individual (architecture network) will be trained on many different batches since the number of batches batch_num is usually larger than the population size n and the training process is repeated for a large number of epochs. In this work, set batch_num $= 98$, $n = 20$, $k = 5$ for the CIFAR-10, and set the maximum number of epochs as 80 and 200 for NAS-Bench-201 and DARTS search space, respectively. Although each dust individual (architecture network) is trained only on a subset ($1/n$ training data) at each epoch, it will be trained on all training data over a large number of epochs by this training strategy.

In addition, due to the facts that each dust individual is responsible for part of the training work, and the complete training of each epoch is done with the participation of all individuals, therefore the efficiency of EGFA-NAS is not sensitive to the setting of population size n, which will be experimentally confirmed in "Parameter settings for NAS-Bench-201".

Explosion operation and weights inheritance

In the context of neural architectural search, a dust in EGFA-NAS represents a candidate architecture and not only maintains the original four attributes: location, mass, group number, and a Boolean flag indicating whether it is a center as description in 2.3, but also maintains an attribute "w" to record the weights of functions in cells. Each dust particle can be represented by a five-tuple (location, w, mass, group, flag). In EGFA-NAS, the location is denoted as the operations mixing probability A, then a neural network architecture can be represented as (A, w, mass, group, flag).

As a population-based NAS method, the main computational bottleneck of EGFA-NAS is involving a lot of evaluation of architectures. In this work, we attempt to reduce the computational cost by taking advantage of the working mechanism of EGFA. At each epoch, additional computational cost is caused because a number of new generated dust particles (architectures) need to be trained during the explosion operation. On the other hand, the new dust particles are generated based on the center dust, and there are close relationships between the new generated dust particles and their center. Based on the above two observations, we proposed a weight inheritance strategy during explosion operation. The detail of explosion operation in EGFA-NAS is described in Algorithm 1.

Algorithm 1 Explosion operation

Input: the size of dust population n, the absorptivity abs, the number of epochs $epoch_{\max}$, the maximum radius $r_{\max}$ and minimum radius $r_{\min}$, current epoch $epoch_{cur}$, dust population $Dust_{absorb}$, the center dust $center$, new generated dust population $Dust_{new} \leftarrow \varnothing$.

Output: the dust population $Dust_{explode}$

1. $r = r_{\max} - (r_{\max} - r_{\min}) / epoch_{\max} * epoch_{cur}$
2. **for** each group **do**
3. **for** each new generated individual $dust_i$ **do**
4. $dust_i.A = center.A * (1 - r) + A_{random} * r$
5. $dust_i.w = center.w$
6. $Dust_{new} = Dust_{new} \cup dust_i$
7. **end for**
8. **For**
9. **for** each individual $dust_i$ in $Dust_{new}$ **do**
10. Construct architecture based on parameters A, w of $dust_i$
11. Compute the loss L_T and L_V
12. $dust_i.mass = L_T^2 + L_V^2$
13. $dust_i.w = dust_i.w - \xi_1 \nabla_{dust_i.w} L_T(dust_i.w, dust_i.A)$
14. **end for**
15. $Dust_{explode} = Dust_{absorb} \cup Dust_{new}$
16. **for** each group **do**
17. Set the dust with maximum mass value as the $center$
18. **end for**
19. Return $Dust_{explode}$

As shown in Algorithm 1, the first part (lines 1–8) is the process of generating new individuals based on the center dust. The mix probabilities A of candidate operations of $dust_i$ are computed as line 4, the weights w of functions in cells are inherited from the center dust as line 5. The second part (lines 9–14) calculates the mass value for new generated dust particle, and update the parameter w. Line 15 combines the dust population $Dust_{absorb}$(output of previous process) and the new generated dust population $Dust_{new}$. The last part (lines 16–18) updates the center dust for each group. By utilizing the weight inheritance, the new generated dust can be evaluated directly at the current epoch without retraining.

Figure 7 illustrates the process of generating new dust particles by means of weight inheritance during the explosion operation. $\vec{a}_i$ represents the probabilities of sampling the $|O|$ candidate operations for edge i, w_i records the weights of functions for edge i. The right partition in Fig. 7 shows the new generated dust population with size of m, the mixing probability A of new dust particles is based on their center as line 4 in Algorithm 1, and parameters w are inherited from their center dust particle as line 5 in Algorithm 1.

Process of EGFA-NAS

As described above, during the process of NAS, the two parameters: architecture A and weight w need to be optimized meanwhile. To solve the bi-level optimization problem, we divide the original training dataset into two parts: the new training dataset D_{T} and the validation dataset D_{V}, then use the new training dataset D_{T} to optimize the parameter w, use the validation dataset D_{V} to optimize the parameter A. In EGFA-NAS, we apply the EGFA and gradient descent

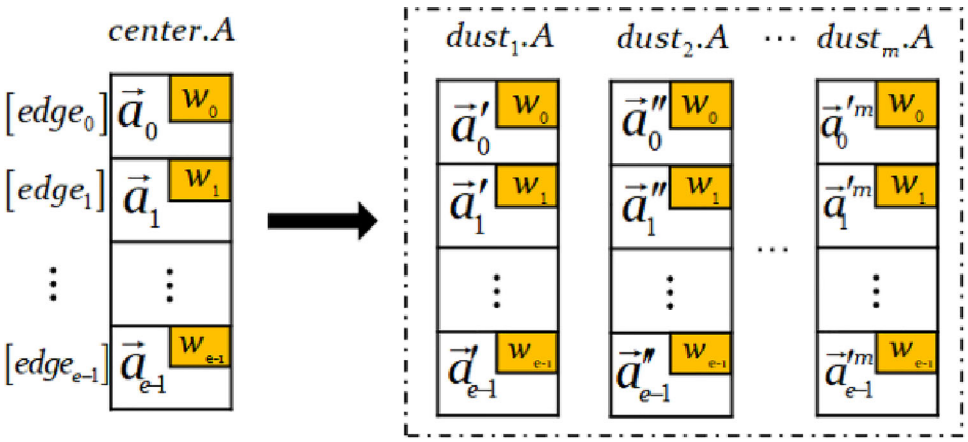

Fig. 7 Process of generating new dust particles by weight inheritance during explosion operation

jointly to optimize the parameter w and architecture A meanwhile in an iterative way. The processes of EGFA-NAS are described in detail as follows:

Step 1: Initialize all parameters, including the size of dust population n, the number of group g, the absorptivity abs for absorb operation, the number of epochs $epoch_{\max}$, the maximum radius $r_{\max}$ and minimum radius $r_{\min}$ for explosion strategy; initialize the dust population $Dust = \{dust_0, dust_1, \cdots, dust_{n-1}\}$ randomly. For each $dust_i$, the location (the ith cell architecture $dust_i.A$) is initialized randomly, which is a $e \times |O|$ tensor as Eq. (3). After initialization, each cell can be stacked into a neural network. Then the loss on training dataset L_{T} and the loss on validation dataset L_{V} can be calculated. To optimize the two parameters w and A meanwhile, we use Eq. (6) to evaluate the performance of network architecture, and denote Eq. (6) as the mass value of $dust_i$. It is noted that the L_{T} and L_{V} are not the loss of network architecture after full training, but the loss on the training dataset and the validation dataset at the current epoch, respectively.

$$dust_i.mass = L_{\mathrm{T}}^2 + L_{\mathrm{V}}^2, \tag{6}$$

where the losses L_{T} and L_{V} are calculated by Eq. (7), which are the cross-entropy loss functions [49].

$$L = -\frac{1}{s}\sum_{x}\left(y \ln \hat{y} + (1-y)\ln\left(1-\hat{y}\right)\right), \tag{7}$$

where x represents the data sample, y is the true label, $\hat{y}$ represents the predicted label, and s is the size of data.

Step 2: Divide the dust population into g subgroups. In EGFA-NAS, the value of g is set as 2; set the dust particle with maximum mass as the center dust, and the others are the surrounding dust particles. For $dust_i$, the attribute $flag$ is set as Eq. (8), where $best\_mass_j$ is the maximum mass value in group j.

$$dust_i.flag = \begin{cases} 1, & dust_i.mass = best\_mass_j,\ j < g \\ 0 \end{cases}, \tag{8}$$

Step 3: Check the termination conditions. There are two termination conditions in EGFA-NAS, one is the maximum epochs, the other one is the average change of mass value of dust population. Once one condition is met, the main loop of EGFA-NAS ends. Then return the optimal network architecture A and deduce the structure of the neural network; otherwise, go to Step 4.

Step 4: Perform the movement and rotation operation. The surrounding dust particles move toward the center dust. For each dust particle $dust_i$, the pace of movement is calculated by Eq. (9).

$$\begin{aligned}\Delta A_1 = {} & p * \left(\exp\left(center.A + 3\right) - \exp\left(dust_i.A + 3\right)\right) \\ & + q * A_{\mathrm{random}},\end{aligned} \tag{9}$$

where $center.A$ presents the cell structure of the center dust; $dust_i.A$ represents the ith cell structure; A_{random} is a 6×5 tensor generated randomly. p is the pace of movement, q is a value close to zero. In this work, we set $p = 0.1$, $q = 0.001$, respectively. We denote the pace of the movement and rotation operation on the location of $dust_i$ as ΔA_1. In addition, in EGFA-NAS, we also apply the gradient descent to optimize the parameters: A and w. We denote the pace of gradient descent on the location of $dust_i$ as ΔA_2, which is calculated by Eq. (10).

$$\Delta A_2 = -\xi_2 \nabla_{dust_i.A} L_V\left(dust_i.w, dust_i.A\right), \tag{10}$$

where ξ_2 is the learning rate, $\nabla_{dust_i.A} L_V$ represents the architecture gradient on validation dataset.

As shown in Fig. 8, considering the impacts of the above two factors on the cell structure A, the location of $dust_i$ is updated as Eq. (11)

$$dust_i.A = dust_i.A + \Delta A_1 + \Delta A_2. \tag{11}$$

During this process, for each dust particle $dust_i$, we not only need to optimize the parameter $dust_i.A$, but also need

to optimize the parameter $dust_i.w$, which is updated by Eq. (12).

$$dust_i.w = dust_i.w - \xi_1 \nabla_{dust_i.w} L_T(dust_i.w, dust_i.A), \tag{12}$$

where ξ_1 is the learning rate, $\nabla_{dust_i.w} L_T$ represents the architecture gradient on training dataset.

Step 5: Perform the absorption operation. Some surrounding dust particles with small mass value will be absorbed by their center dust. During this process, the size of dust population will change, the new size is determined by the absorptivity abs as Eq. (13).

$$n = n * (1 - \text{abs}), \tag{13}$$

where n is the size of the initial population, abs represents the absorptivity. In this work, we set abs as 0.5.

Step 6: Perform the explosion operation. During the process of Step 5, some dust particles with small mass value are absorbed by their center dust particles. To maintain the size of dust population, some new dust particles will be generated around the center dust particles during this process. This part is descripted in "Explosion operation and weights inheritance" in detail.

Once Step 6 finishes, go to Step 3.

According to the above detailed description of EGFA-NAS, the pseudo-code of EGFA-NAS is shown in Algorithm 1. Step 1 (lines 1–3) is the initialization. Step 2 (lines 4–5) is the operation of grouping. Step 3 (line 6) checks the termination conditions. Step 4 (lines 7–12) is the process of movement and rotation. Step 5 (line 13) is the absorption operation. Step 6 (line 14) is the explosion operation.

Algorithm 1: EGFA-NAS

Input: the training dataset D_T, the validation dataset D_V, the population size n, the number of group g, the absorptivity abs, the maximum radius $r_{\max}$ and minimum radius $r_{\min}$ for explosion strategy, the maximum and current number of epochs $epoch_{\max}$, $epoch\_cur = 0$, dust population $Dust \leftarrow \varnothing$, best dust particle $best \leftarrow \varnothing$.

Output: $center$, $best$

1. initialize all parameters
2. $Dust \leftarrow$ initialize the dust population with size of n randomly
3. Compute mass value for each dust particle by Eq. (6)
4. Divide the dust population into g groups
5. Set the dust with max mass value as the center for each group
6. **while** termination conditions are not met do
7. **for** each individual $dust_i$ ($flag = 0$) **do**
8. Update $dust_i.A$ by Eq. (9)-(11)
9. Update $dust_i.w$ by Eq. (12)
10. Update $dust_i.mass$ by Eq. (6)
11. **end for**
12. Update the $center$, $best$
13. Perform the absorption operation
14. Perform the explosion operation as Algorithm 1
15. Update $center$, $best$
16. **end while**
17. **return** $center$, $best$

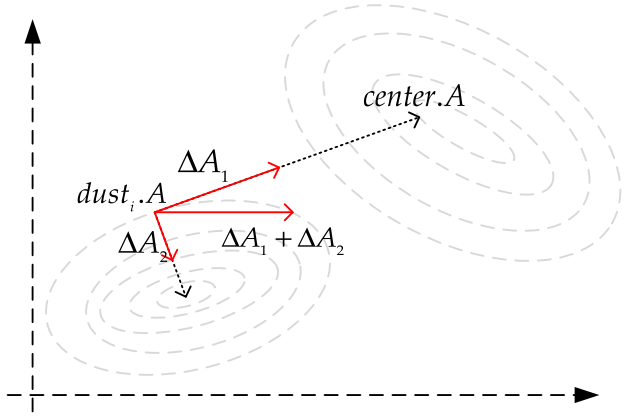

Fig. 8 Change of the *i* th cell structure *A* during the process of movement and rotation operation

Experimental design

The goal of EGFA-NAS is to search the optimal neural network architecture automatically which can achieve satisfying performance on a complex task, such as image classification. For this purpose, a series of experiments is designed to demonstrate the advantages of the proposed EGFA-NAS compared with the state-of-the-art NAS methods. First, we utilize the proposed EGFA-NAS to search neural network architectures in the benchmark search space: NAS-Bench-201, and evaluate the performance of proposed EGFA-NAS by investigating the classification accuracy and computational cost of the searched architecture on CIFAR-10, CIFAR-100, and ImageNet-16-120. Second, we investigate the consistency of relative evaluation with absolute evaluation, in terms of the accuracy and loss. Third, we investigate the effectiveness of the weight inheritance strategy. Finally, we examine the proposed EGFA-NAS in the larger and more practical search space: DARTS search space, and investigate the performance and universality of EGFA-NAS.

We first perform the proposed EGFA-NAS in the benchmark search space: NAS-Bench-201. When the search process terminates, the absolute performance evaluation of the optimal architecture can be obtained directly by the NAS-Bench-201's API with negligible computational cost. By utilizing NAS-Bench-201, we verify the consistency of relatively performance evaluation and absolute performance evaluation for the searched network architectures without retraining from scratch. In addition, we verify the effectiveness of weight inheritance in NAS-Bench-201 search space. But when the search process in DARTS search space terminates, the optimal network architecture needs to be retrained from scratch and be test on the test datasets. The test classification accuracy is reported as the results of our experiments.

In the rest of this section, we introduce the peer competitors to compare with this proposed EGFA-NAS, the benchmark datasets, and finally the parameter setting for the two typical search spaces: NAS-Bench-201 and DARTS search space.

Peer competitors

To demonstrate the advantage of the proposed EGFA-NAS, a series of competitors are chosen for comparison. "Competitors of NAS-Bench-201" introduces the competitors compared with the performance of the optimal architecture searched by EGFA-NAS in NAS-Bench-201 search space, and "Competitors of DARTS search space" introduces the competitors compared with the performance of optimal architecture searched by EGFA-NAS in DARTS search space.

Competitors of NAS-Bench-201

Due to the facts that NAS-Bench-201 (only has five candidate operations) is a smaller search space, and the best architecture has lower classification accuracy compared with the best one searched in other search space, the performance of optimal architecture searched by EGFA-NAS in NAS-Bench-201 are only compared with the competitors which have reported the results in NAS-Bench-201 search space.

The selected competitors are mainly the efficient GD-based NAS methods, including DARTS-V1 [19], DARTS-V2 [19], SETN [50], iDARTS [51], and GDAS [20]. The other three selected NAS competitors, namely ENAS [18], RSPS [22], and EvNAS [52], utilize RL, random search, and EA as the search strategies for NAS tasks, respectively.

Competitors of DARTS search space

DARTS search space is a functional search space for NAS tasks, in which the optimal network architecture has promising performance compared with the state-of-the-art manually designed CNN architectures. To compare the performance of the optimal network architecture searched by EGFA-NAS in the DARTS search space, we select four different kinds of competitors for comparison.

1. The first kind of competitors are the state-of-the-art CNN architectures, manually designed by domain experts, including ResNet-101 [10], DenseNet-BC [11], SENet [53], IGCV3 [54], ShuffleNet [55], VGG [1], and Wide ResNet [56].
2. The second kind of competitors are the state-of-the-art EA-based NAS methods, including Hierarchical EA [15], AmoebaNet-A [16], LEMONADE [24], CGP-CNN [25], CNN-GA [26], AE-CNN [32], and AE-CNN + E2EPP [33], LargeEvo [27], GeNet [31], SI-EvoNet [57], NSGA-Net [28], and MOEA-PS [58].

3. The third kind of competitors utilize RL to search for CNN architectures, such as NASNet-A [17], NASNet-A + CutOut [17], Proxyless NAS [34], BlockQNN [35], DPP-Net [59], MetaQNN [60], and ENAS [18].
4. The fourth kind of competitors are mainly the GD-based NAS methods, such as DARTS-V1 + CutOut [19], DARTS-V2 + CutOut [19], RC-DARTS [38], and SNAS [21]. In addition, PNAS [40] is also selected for comparison, which use a sequential model-based optimization (SMBO) strategy.

Benchmark datasets

To investigate the performance of EGFA-NAS on NAS tasks, we test EGFA-NAS in two different search space, including NAS-Bench-201 and DARTS search space. All experiments involve three benchmark datasets: CIFAR-10, CIFAR-100 [47], and ImageNet-16-120 [48], which are widely adopted in experimental studies of state-of-the-art CNNs and NAS methods. In this work, each architecture searched in NAS-Bench-201 is trained and evaluated on CIFAR-10, CIFAR-100 [47], and ImageNet-16-120 [48]. Each architecture searched in DARTS search space is trained and evaluated on CIFAR-10, CIFAR-100. Each dataset is split into three subsets: training set, validation set, and test set.

CIFAR-10: It is an image classification dataset consisting of 60K images with with classes. The original set contains 50K training images and 10K test images. Due to the need for a validation set, the original training set is randomly split into two subsets with equal size, each subset contains 25K images with ten classes. In this work, we regard one subset as the new training set and another as the validation set.

CIFAR-100: It has the same images as CIFAR-10, but it categorizes the images into 100 fine-grained classes. The CIFAR-100 original contains 50K images in the training set and 10K images in the test set. In this work, the original training set is randomly split into two subsets with equal size. One is regarded as the training set and another as the new validation set.

ImageNet-16-120: ImageNet is a large-scale and well-known dataset for image classification. Image-16-120 was built with 16×16 pixels from the down-sampling variant of ImageNet [61] (i.e., ImageNet 16×16). ImageNet-16-120 contains all images with labels $\in [0, 119]$. In sum, ImageNet-16-120 consists of 151.7K images for training, 3K images for validation, and 3K images for testing with 120 classes.

Parameter settings

This section introduces the parameter setting for EGFA-NAS in detailed.

Table 1 Hyperparameter settings of searching process

Parameter	Value
Initial channels	16
B	5
Optimizer	SGD
Nesterov	1
Momentum	0.9
Batch size	256
LR scheduler	Cosine
Initial LR	2.5×10^{-2}
min_LR	1×10^{-3}
Weight decay	5×10^{-4}
Random flip	0.5

Parameter settings for NAS-Bench-201

For the NAS-Bench-201 search space, the parameter settings are only involved in the search process, because NAS-Bench-201 provides the absolute (final) performance evaluation for each architecture, and we can obtain the evaluation of the optimal architecture directly without retraining from scratch. We adopt the same skeleton network following [23] as Fig. 3. Specifically, we set the number of initial channels for the first convolution layer as 16; set the number of cells in one normal block B as 5. During the search, almost parameter settings follows [23], as shown in Table 1. Specifically, we train each architecture via Nesterov momentum SGD, using the cross-entropy loss as the loss function with batch size 256. We set the weight decay as 5×10^{-4} and decay the learning rate from 2.5×10^{-2} to 1×10^{-3} with a cosine annealing scheduler.

In NAS-Bench-201 search space, we set up the same hyperparameters on three different datasets: CIFAR-10, CIFAR-100 [47], and ImageNet-16-120 [48], except for the part of data augmentation due to the slightly difference of images' resolution. For CIFAR-10 and CIFAR-100, we use the random flip with probability of 0.5, the random crop 32 $\times$ 32 patch with 4 pixels padding, and the normalization over RGB channels. For ImageNet-16-120, we use the same strategies, except for random crop 16×16 patch with 2 pixels padding.

The parameters listed in Table 1 are related to neural network architecture. As a population-based method, EGFA-NAS has its own parameters. Specifically, we set the number of groups g as 2, set the absorptivity abs as 0.5 for absorb operation, set the maximum radius $r_{\max}$ as 0.1, and set the minimum radius $r_{\min}$ as 0.001 for the explosion operation.

As a population-based NAS method, a larger number of epochs may lead to better performance, but the computational cost will also increase. We investigate the impact of

Table 2 Relative and absolute performance (accuracy) of best architectures searched by EGFA-NAS on CIFAR-10 with different number of epochs

Dataset	Number of epochs	Relative performance	Absolute performance	Search cost (GPU days)
CIFAR-10	40	38.12	91.71	0.025
	60	43.91	92.16	0.037
	80	48.27	93.67	0.048
	100	53.05	93.67	0.062
	120	57.58	93.67	0.076

Table 3 Relative and absolute performance (accuracy) of best architecture searched by EGFA-NAS on CIFAR-10 with different population size

Dataset	Population size	Relative performance	Absolute performance	Search cost (GPU days)
CIFAR-10	10	50.08	93.28	0.0481
	15	49.00	93.36	0.0482
	20	51.02	93.67	0.0482
	25	48.83	93.67	0.0481
	30	49.61	93.67	0.0482

Note that all experimental settings are constrained by the computational resources available to us. All experiments are implemented via PyTorch 1.7 on one NVIDIA GeForce RTX 3090 GPU card. The computational cost is evaluated in terms of "GPU days", calculated by multiplying the number of GPU cards by the search time in the days, following [19, 20, 62].

the maximum number of epochs on the performance and computational cost on the CIFAR-10 dataset. The relative and absolute performance (accuracy) of the best architecture searched by EGFA-NAS on CIFAR-10 with different numbers of epochs are shown in Table 2. The relative performance of the searched architectures is evaluated at the last epoch in the search phase without retraining. The absolute performance of the searched architecture is inquired by the API provided by NAS-Bench-201. From the results in Table 2, we can observe that the best performance (93.67% accuracy on CIFAR-10) is achieved when the number of epochs is set as 80. When the number of epochs is increased to 100, no improvement of the absolute performance is achieved, although the computational cost becomes more. Hence, we set the number of epochs as 80 in the experiments for NAS-Bench-201.

Generally, population size is a vital factor for the performance and efficiency of the population-based method, a larger population size usually leads to better performance, but also leads to an increment in search cost. But, in EGFA-NAS, we proposed a training strategy, which utilizes all dust individuals to complete the data training at each epoch. This training strategy reduce the sensitivity of performance to the population size, which can be verified by the results in Table 3. Specifically, EGFA-NAS not only has a similar performance, but also has similar search cost (GPU days) with different population size. In addition, the architectures searched by EGFA-NAS achieve the best absolute performance when population sizes $n \geq 20$. In view of above observation, we set the population size n as 20 in this work. In a word, absolute performance (accuracy) and search cost (GPU days) of EGFA-NAS are closely related to the maximum number of epochs, but are not much related to the population size.

Parameter settings for DARTS search space

The neural cells for CNNs are searched in DARTS search space on CIFAR-10/100 following [7, 17]. The macro skeleton of DARTS search space is shown as Fig. 4. The parameter setting for DARTS search space can be divided into two parts: (1) searching phase and (2) evaluation phase.

During searching phase, we set the number of initial channels for the first convolutional layer as 16, set the number of cells in a normal block B as 2, set the number of epochs as 200. For training parameter w, we optimize each architecture via Nesterov momentum SGD with batch size of 256, set the initial learning rate as 2.5×10^{-2}, and anneal it down to 1×10^{-3} with a cosine annealing scheduler. We set the momentum as 0.9 and decay weight as 5×10^{-4}. To optimize parameter A, we use the Adam optimizer with default settings.

During evaluation phase, we train the searched network by 600 epochs in total. We set the initial channels as 33, and set the number of cells in a normal block B as 6 or 8. We start the learning rate of 2.5×10^{-2} and reduce it to 0 with the cosine scheduler. We set the probability of path drop as 0.2 and the auxiliary tower with the weight of 0.4. Other

Table 4 Hyperparameter settings for DARTS search space

Parameter	Searching	Evaluation
Epochs	200	600
Initial channels	16	33
B	2	6/8
Optimizer	SGD/Adam	SGD
Batch size	256	256
Nesterov	1	1
Momentum	0.9	0.9
Scheduler	Cosine	Cosine
Initial LR	2.5×10^{-2}	2.5×10^{-2}
Min_LR	1×10^{-3}	0
Decay weight	5×10^{-4}	5×10^{-4}

parameter settings are set as same as in the searching phase (Table 4).

Compared with NAS-Bench-201 ($e = 6$, $|O| = 5$), DARTS search space ($e = 14$, $|O| = 8$) is a larger search space. Then we set the number of epochs as 200 to explore DARTS search space. The other parameters about EGFA-NAS, such as population size n, the number of groups g, the absorptivity abs, the maximum radius $r_{\max}$, and the maximum radius $r_{\min}$, are set as same as "Parameter settings for NAS-Bench-201".

Experimental results

Overall results in NAS-Bench-201 search space

The experimental results of the optimal network discovered by EGFA-NAS and other competitors in NAS-Bench-201, in terms of classification accuracy and computational cost (GPU days), are presented in Table 5. The symbol "–" means that the corresponding result was not reported. The results of iDARTS [51] and EvNAS [52] are sourced from the original published paper, and the consequences of the other competitors are extracted from [23]. The results highlighted in bold are the results of optimal best architectures and the results of the architectures searched by EGFA-NAS.

From the results in Table 5, we can observe that EGFA-NAS can achieve better performance than the peer competitors: DARTS-V1 [19], DARTS-V2 [19], SETN [50], iDARTS [51], GDAS [20], ENAS [18], RSPS [22], and EvNAS [52]. Specifically, in the NAS-Bench-201 search space, EGFA-NAS discovers a network architecture with only 1.29M parameters, which consumes 0.048 GPU days and achieves 93.67% accuracy on CIFAR-10. For the CIFAR-100 dataset, EGFA-NAS achieves 71.29% accuracy with 1.23M parameters, and consumes 0.094 GPU day. For ImageNet-16-120, the architecture searched by EGFA-NAS obtains 42.33% accuracy with 1.32M parameters and 0.236 GPU days cost. Limited by the small search space: NAS-Bench-201, the performance of the network architecture searched is not comparable with the state-of-the-art designed CNN networks. But the performance of network architecture searched by EGFA-NAS has the smallest difference (0.7% worse on CIFAR-10, 2.22% worse on CIFAR-100, and 4.95% worse on ImageNet-16–120) with the performance of the optimal theoretical architecture, compared with the other competitors in the NAS-Bench-201 search space. In addition, the proposed EGFA-NAS has the best efficiency compared with all selected peer competitors.

Note that the search cost (GPU Days) of the competitors listed in Table 5 is extracted from [23]. But reference [23] does not indicate to which dataset the result belongs. The number of parameters (Params) for the peer competitors is obtained by running the code provided by [23] on the CIFAR-10 dataset. The search cost (GPU Days) of EGFA-NAS is the computational consumption counted for the three datasets, respectively, on the computational platform with one NVIDIA GeForce RTX 3090 GPU card.

Effectiveness of the relative performance evaluation

Due to the fact that NAS-Bench-201 [23] provides the evaluation information for each candidate architecture, in this section, we utilize the API provided by NAS-Bench-201 to obtain the absolute (final) performance evaluation (loss and accuracy) for the searched architectures without retraining, and verify the effectiveness of the evaluation strategy adopted by EGFA-NAS. Figure 9 shows the comparison of relative performance evaluation with absolute performance evaluation, in terms of loss (Fig. 9a) and accuracy (Fig. 9b) on CIFAR-10. In Fig. 9, the label "rel" represents the relative performance, and the label "abs" represents the absolute performance. The relative performance of the searched architectures is obtained on the validation dataset at the current epoch during the architecture search phase. From the results in Fig. 9, we can observe that the relative performance of searched architectures cannot be comparable with their absolute performance, this is because the architectures searched during the search phase are not trained sufficiently. Figure 9 illustrates that the trend of the relative performance is consistent with the absolute performance of the searched architectures. In addition, we can observe that EGFA-NAS is only not stable enough for the first several epochs and can achieve architectures with stable performance when the number of epochs is larger than 30. The observation above verifies the effectiveness of the evaluation strategy adopted by EGFA-NAS.

Table 5 Comparison of EGFA-NAS with the peer competitors in terms of the classification accuracy (%) and the computational cost (GPU days) on CIFAR-10, CIFAR-100, and ImageNet-16-120 datasets

Method	Search strategy	GPU days	Params(M)	CIFAR-10	CIFAR-100	ImageNet-16-120
DARTS-V1 [19]	GD	0.13	0.07[a]	54.30	15.61	16.32
DARTS-V2 [19]	GD	0.41	0.07[a]	54.30	15.61	16.32
iDARTS [51]	GD	–	–	93.58	70.83	40.89
SETN [50]	GD	0.35	0.41[a]	86.19	56.87	31.90
GDAS [20]	GD	0.33	1.2[a]	93.51	70.61	41.71
ENAS [18]	RL	0.15	0.07[a]	54.30	15.61	16.32
RSPS [22]	Random	0.10	0.43[a]	87.66	58.33	31.44
EvNAS [52]	EA	0.26	–	92.18	66.74	39.00
Optimal	–	–	–	**94.37**	**73.51**	**47.31**
EGFA-NAS	**EGFA**	**0.048**	**1.29**	**93.67**	–	–
EGFA-NAS	**EGFA**	**0.094**	**1.23**	–	**71.29**	–
EGFA-NAS	**EGFA**	**0.246**	**1.32**	–	–	**42.33**

[a]Calculated by running the code publicly released by [23]

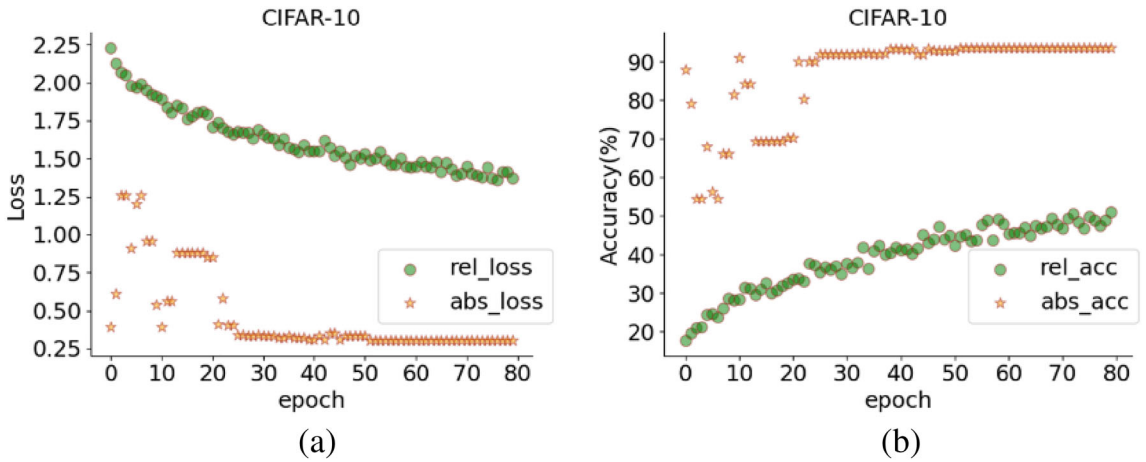

Fig. 9 Comparison of relative evaluation and absolute evaluation of the architecture searched by EGFA-NAS

Effectiveness of weight inheritance strategy

To improve the efficiency of EGFA-NAS and reduce the computational cost, we propose a weight inheritance strategy during the explosion operation as described in "Explosion operation and weights inheritance". Specifically, the parameters w of new generated dust individuals are inherited from their centers. In this section, we attempt to verify the effectiveness of the weight inheritance strategy by replacing this proposed strategy with generating the parameter w randomly on CIFAR-10, and other settings are kept unchanged. To observe the difference between our proposed strategy and the way of generating parameter w randomly more clearly, we set the number of epochs as 300 in this experiment. The estimated (relative) performance of searched network architectures using weight inheritance and the way of generating the parameter w randomly are shown in Fig. 10a and c, in terms of accuracy and loss, respectively. The final (absolute) performance of the network architectures searched by the two strategies is shown in Fig. 10b and d, in terms of accuracy and loss, respectively. The results in Fig. 10 show a big difference between the estimated (relative) performance of the two strategies. Although the final (absolute) performance of the architectures searched by the two strategies is similar on CIFAR-10, EGFA-NAS using the proposed weight inheritance can achieve the best network architecture earlier than utilizing the way of generating the parameter w randomly. In addition, the final performance of the architecture searched by inheritance weight is slightly better (93.67% accuracy)

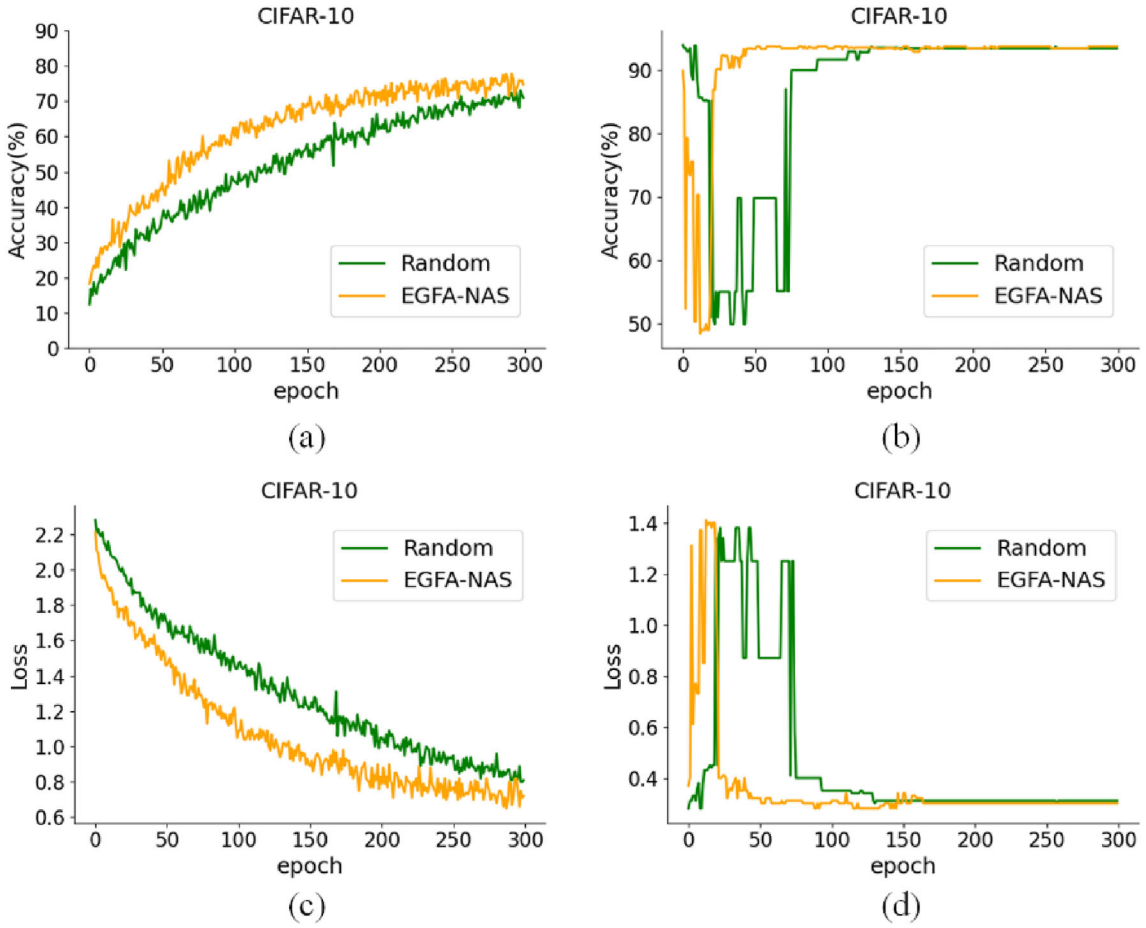

Fig. 10 Comparison of the performance of EGFA-NAS using weight inheritance strategy and by way of generation parameter w randomly on CIFAR-10

than utilizing the way of generating parameter w randomly (96.36% accuracy).

Overall results in DARTS search space

The experimental results of the optimal network discovered by EGFA-NAS in DARTS search space, in terms of classification accuracy and computational cost (GPU days), are presented in Table 6. The symbol "–" means that the corresponding results were not reported. The symbol "*" means that the results are extracted from [19]. The mode "a/b" in Table 5.4 means that "a" is the result for CIFAR-10 and "b" is the result for CIFAR-100. The results of most competitors are extracted from the original published papers. $B = 6$ or 8 represents the number of normal cells in a normal block in the retraining phase. The results highlighted in bold are the result of the architectures searched by EGFA-NAS.

The results in Table 6 show that EGFA-NAS ($B = 8$) can achieve better performance than most state-of-the-art manual-designed CNN networks, including ResNet-101, ResNet + CutOut, SENet, IGCV3, ShuffleNet, VGG, and Wide ResNet, but a little worse than DenseNet-BC (1.05% on CIFAR-100). The performance improvement of optimal network architecture searched by EGFA-NAS ($B = 8$) is 13.9% on CIFAR-100, and 3.89% on CIFAR-10, compared with VGG.

Compared with the 12 EA-based NAS methods, EGFA-NAS ($B = 8$) achieves better performance than Hierarchical EA, AmoebaNet-A, CGP-CNN, CNN-GA, AE-CNN, AE-CNN + E2EPP, LargeEvo, GeNet, SI-EvoNet, and MOEA-PS, but slightly worse than LEMONADE (0.19%) and NSGA-Net (0.02%) on CIFAR-10. EGFA-NAS ($B = 8$) achieves the best classification accuracy (81.85%) on the CIFAR-100, and consumes the least search cost (0.21 GPU days) than all selected EA-based NAS methods.

Compared with the six RL-based NAS methods, EGFA-NAS ($B = 8$) achieves better performance than NASNet-A, NASNet-A + CutOut, BlockQNN, DPP-Net, MetaQNN, and ENAS, but a little worse than Proxyless NAS (0.86%) on the CIFAR-10. The performance improvement of the optimal network architecture searched by EGFA-NAS ($B = 8$) is 4.15% on the CIFAR-10, and 8.99% on the CIFAR-100,

Table 6 Comparison of EGFA-NAS with the peer competitors in terms of the classification accuracy (%) and the computational cost (GPU days) on CIFAR-10, CIFAR-100

Method	Search strategy	GPU days	Params (M)	CIFAR-10	CIFAR-100
ResNet-101 [10]	Manual	–	1.7	93.57	74.84
ResNet + CutOut [10]	Manual	–	1.7	95.39	77.90
DenseNet-BC [11]	Manual	–	25.6	96.54	82.82
SENet [53]	Manual	–	11.2	95.95	–
IGCV3 [54]	Manual	–	2.2	94.96	77.95
ShuffleNet [55]	Manual	–	1.06	90.87	77.14
VGG [1]	Manual	–	28.05	93.34	67.95
Wide ResNet [56]	Manual	–	36.48	95.83	79.50
Hierarchical EA [15]	EA	300	61.3	96.37	–
AmoebaNet-A [16]	EA	3150	3.2	96.66	81.07
LEMONADE [24]	EA	90	13.1	97.42	–
CGP-CNN [25]	EA	27	1.7	94.02	–
CNN-GA [26]	EA	35/40	2.9/4.1	96.78	79.47
AE-CNN [32]	EA	27/36	2.0/5.4	95.3	77.6
AE-CNN + E2EPP [33]	EA	7/10	4.3/20.9	94.7	77.98
LargeEvo [27]	EA	2750/2750	5.4/40.4	94.6	77.00
GeNet [31]	EA	–	–	94.61	74.88
SI-EvoNet [57]	EA	0.46/0.81	0.51/0.99	96.02	79.16
NSGA-Net [28]	EA	4/8	3.3/3.3	97.25	79.26
MOEA-PS [58]	EA	2.6/5.2	3.0/5.8	97.23	81.03
NASNet-A [17]	RL	2000	3.3	96.59	–
NASNet-A + CutOut [17]	RL	2000	3.1	97.17	–
Proxyless NAS [34]	RL	1500	5.7	97.92	–
BlockQNN [35]	RL	96	39.8	96.46	–
DPP-Net [59]	RL	8	0.45	94.16	
MetaQNN [60]	RL	90	11.2	93.08	72.86
ENAS [18]	RL	0.5	4.6	97.06	–
ENAS [18]*	RL	4	4.2	97.09	–
DARTS-V1 + CutOut [19]	GD	1.5	3.3	97.00	
DARTS-V2 + CutOut [19]	GD	4	3.4	97.18	82.46
RC-DARTS [38]	GD	1	0.43	95.83	
SNAS [21]	GD	1.5	2.8	97.15	–
PNAS [40]	SMBO	225	3.2	96.37	80.47
EGFA-NAS ($B = 6$)	**EGFA**	**0.21/0.4**	**2.56/2.15**	**96.57**	**80.08**
EGFA-NAS ($B = 8$)	**EGFA**	**0.21/0.4**	**3.47/2.88**	**97.23**	**81.85**

*Extracted from the reference [19]

compared with MetaQNN. The proposed EGFA-NAS ($B = 8$) has the best efficiency and consumes the least GPU days even compared with the ENAS, which only consumes 0.5 GPU days on the CIFAR-10 in the published paper.

Compared with four GD-based NAS methods and PNAS, EGFA-NAS ($B = 8$) achieves better performance than DARTS-V1 + CutOut, RC-DARTS, and SNAS, but a little worse than DARTS-V2 + CutOut (0.61%) on the CIFAR-100. Although GD-based NAS methods usually have better efficiency than EA-based and RL-based methods, our proposed EGFA-NAS ($B = 8$) has the best efficiency compared to all selected GD-based NAS methods.

In addition, EGFA-NAS can obtain better final learning accuracy when setting larger number of cells in a normal block during the retraining phase, but will lead to larger number of parameters. The overall results in Table 6 show that this proposed EGFA-NAS not only has competitive learning accuracy but also has the best efficiency compared with the four kinds of competitors.

Conclusion

This paper proposes an efficient population-based NAS method based on the EGFA, called EGFA-NAS, which can achieve an optimal neural architecture with competitive learning accuracy but consumes a little computational cost. Specifically, EGFA-NAS relaxes the discrete search space to a continuous one and then utilizes EGFA and gradient descent to optimize the weights of the candidate architectures in conjunction. The proposed training and weight inheritance strategies for EGFA-NAS reduce the computational cost dramatically. The experimental results in two typical micro search spaces: NAS-Bench-201 and DARTS, demonstrate that EGFA-NAS is able to match or outperform the state-of-the-art NAS methods on image classification tasks with remarkable efficiency improvement. Specifically, to search the CIFAR-10 on the computational platform with one NVIDIA GeForce RTX 3090 GPU card, EGFA-NAS obtains the optimal neural architectures in NAS-Bench-201 search space with 93.67% accuracy but only consumes 0.048 GPU days, discovers the optimal neural architectures in DARTS search space with 97.23% accuracy and a cost of 0.21 GPU day.

Although EGFA-NAS is promising for designing high-performance neural networks automatically, it still has one limitation. Similar to the other NAS methods using the low-fidelity evaluation strategy, the relative evaluation adopted in EGFA-NAS during the search phase may lead to missing some promising architectures. In future work, we will attempt to design a better evaluation strategy with better rank consistency for lightweight NAS.

Acknowledgements This work was supported by the National Natural Science Foundation of China (no. 62072212), the Development Project of Jilin Province of China (no. 20220508125RC, 20230201065GX), and the Jilin Provincial Key Laboratory of Big Data Intelligent Cognition (no. 20210504003GH).

Data availability Data will be made available on request.

Declarations

Conflict of interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

References

1. Simonyan K, Zisserman A (2015) Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv preprint, arXiv:1409.1556
2. Huang G, Sun Y, Liu Z et al (2016) Deep networks with stochastic depth. In: Leibe B, Matas J, Sebe N, Welling M (eds) Computer Vision—ECCV 2016. Springer International Publishing, Cham, pp 646–661
3. Ciresan D, Meier U, Schmidhuber J (2012) Multi-column deep neural networks for image classification. In: Proceedings of the IEEE international conference on computer vision. CVPR, Providence, pp 3642–3649
4. Krizhevsky A, Sutskever I, Hinton GE (2017) ImageNet classification with deep convolutional neural networks. Commun ACM 60:84–90. https://doi.org/10.1145/3065386
5. Girshick R (2015) Fast r-cnn. In: Proceedings of the IEEE international conference on computer vision. ICCV, pp 1440–1448
6. Zhao Z, Zheng P, Xu S (2019) Object detection with deep learning: a review. IEEE Trans Neural Netw Learning Syst 30:3212–3232. https://doi.org/10.1109/TNNLS.2018.2876865
7. Zoph B, Le QV (2017) Neural Architecture Search with Reinforcement Learning. arXiv preprint, arXiv:1611.01578
8. Hesamian MH, Jia W, He X, Kennedy P (2019) Deep learning techniques for medical image segmentation: achievements and challenges. J Digit Imaging 32:582–596. https://doi.org/10.1007/s10278-019-00227-x
9. Ghosh S, Das N, Das I, Maulik U (2020) Understanding deep learning techniques for image segmentation. ACM Comput Surv 52:1–35. https://doi.org/10.1145/3329784
10. He K, Zhang X, Ren S, et al (2016) Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. CVPR, pp 770–778
11. Huang G, Liu Z, Van Der Maaten L et al (2017) Densely connected convolutional networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp 4700–4708
12. Praczyk T (2016) Cooperative co-evolutionary neural networks. IFS 30:2843–2858. https://doi.org/10.3233/IFS-162095
13. Garcia-Pedrajas N, Hervas-Martinez C, Munoz-Perez J (2003) COVNET: a cooperative coevolutionary model for evolving artificial neural networks. IEEE Trans Neural Netw 14:575–596. https://doi.org/10.1109/TNN.2003.810618
14. Yao X (1999) Evolving artificial neural networks. Proc IEEE 87:1423–1447. https://doi.org/10.1109/5.784219
15. Liu H, Simonyan K, Vinyals O, et al (2018) Hierarchical Representations for Efficient Architecture Search. arXiv preprint, arXiv:1711.00436
16. Real E, Aggarwal A, Huang Y, Le QV (2019) Regularized evolution for image classifier architecture search. AAAI 33:4780–4789. https://doi.org/10.1609/aaai.v33i01.33014780
17. Zoph B, Vasudevan V, Shlens J, Le QV (2018) Learning transferable architectures for scalable image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. CVPR, pp 8697–8710
18. Pham H, Guan M, Zoph B, et al (2018) Efficient neural architecture search via parameters sharing. In: Proceedings of the 35th International Conference on Machine Learning. PMLR, pp 4095–4104
19. Liu H, Simonyan K, Yang Y (2019) DARTS: Differentiable Architecture Search. arXiv preprint, arXiv:1806.09055

20. Dong X, Yang Y (2019) Searching for a robust neural architecture in four gpu hours. In: Proceedings of the IEEE international conference on computer vision. CVPR, pp 1761–1770
21. Xie S, Zheng H, Liu C, Lin L (2020) SNAS: Stochastic Neural Architecture Search. arXiv preprint, arXiv:1812.09926
22. Li L, Talwalkar A (2020) Random search and reproducibility for neural architecture search. In: Adams RP, Gogate V (eds) Proceedings of the 35th uncertainty in artificial intelligence conference. PMLR, pp 367–377
23. Dong X, Yang Y (2020) NAS-Bench-201: Extending the Scope of Reproducible Neural Architecture Search. arXiv preprint, arXiv:2001.00326
24. Elsken T, Metzen JH, Hutter F (2019) Efficient Multi-objective Neural Architecture Search via Lamarckian Evolution. arXiv preprint, arXiv:1804.09081
25. Suganuma M, Shirakawa S, Nagao T (2017) A genetic programming approach to designing convolutional neural network architectures. In: Proceedings of the genetic and evolutionary computation conference. ACM, Berlin, pp 497–504
26. Sun Y, Xue B, Zhang M et al (2020) Automatically designing CNN architectures using the genetic algorithm for image classification. IEEE Trans Cybern 50:3840–3854. https://doi.org/10.1109/TCYB.2020.2983860
27. Real E, Moore S, Selle A et al (2017) Large-scale evolution of image classifiers. In: Proceedings of the 34th international conference on machine learning. PMLR, pp 2902–2911
28. Lu Z, Whalen I, Boddeti V, et al (2019) NSGA-Net: neural architecture search using multi-objective genetic algorithm. In: Proceedings of the genetic and evolutionary computation conference. ACM, Prague, pp 419–427
29. Hu X, Huang L, Wang Y, Pang W (2019) Explosion gravitation field algorithm with dust sampling for unconstrained optimization. Appl Soft Comput 81:105500. https://doi.org/10.1016/j.asoc.2019.105500
30. Gould S, Fernando B, Cherian A, et al (2016) On differentiating parameterized argmin and argmax problems with application to bi-level optimization. arXiv:1607.05447
31. Xie L, Yuille A (2017) Genetic CNN. In: Proceedings of the IEEE international conference on computer vision. ICCV, pp 1379–1388
32. Sun Y, Xue B, Zhang M, Yen GG (2020) Completely automated CNN architecture design based on blocks. IEEE Trans Neural Netw Learn Syst 31:1242–1254. https://doi.org/10.1109/TNNLS.2019.2919608
33. Sun Y, Wang H, Xue B et al (2020) Surrogate-assisted evolutionary deep learning using an end-to-end random forest-based performance predictor. IEEE Trans Evol Comput 24:350–364. https://doi.org/10.1109/TEVC.2019.2924461
34. Cai H, Zhu L, Han S (2019) ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. arXiv preprint, arXiv:1812.00332
35. Zhong Z, Yang Z, Deng B et al (2021) BlockQNN: efficient block-wise neural network architecture generation. IEEE Trans Pattern Anal Mach Intell 43:2314–2328. https://doi.org/10.1109/TPAMI.2020.2969193
36. Chu X, Wang X, Zhang B, et al (2021) DARTS-: Robustly Stepping out of Performance Collapse Without Indicators. arXiv preprint, arXiv:2009.01027
37. Liang H, Zhang S, Sun J, et al (2020) DARTS+: Improved Differentiable Architecture Search with Early Stopping. arXiv preprint, arXiv:1909.06035
38. Jin X, Wang J, Slocum J, et al (2019) RC-DARTS: Resource Constrained Differentiable Architecture Search. arXiv preprint, arXiv:1912.12814
39. Ye P, Li B, Li Y, et al (2022) β-DARTS: Beta-Decay Regularization for Differentiable Architecture Search. In: Proceedings of the IEEE conference on computer vision and pattern recognition. CVPR, New Orleans, LA, USA, pp 10864–10873. https://doi.org/10.1109/CVPR52688.2022.01060
40. Liu C, Zoph B, Neumann M et al (2018) Progressive neural architecture search. In: Proceedings of the European conference on computer vision. ECCV, pp 19–34
41. Zheng M, Liu G, Zhou C et al (2010) Gravitation field algorithm and its application in gene cluster. Algorithms Mol Biol 5:32. https://doi.org/10.1186/1748-7188-5-32
42. Zheng M, Sun Y, Liu G et al (2012) Improved gravitation field algorithm and its application in hierarchical clustering. PLoS One 7:e49039. https://doi.org/10.1371/journal.pone.0049039
43. Zheng M, Wu J, Huang Y et al (2012) Inferring gene regulatory networks by singular value decomposition and gravitation field algorithm. PLoS One 7:e51141. https://doi.org/10.1371/journal.pone.0051141
44. Safronov VS (1972) Evolution of the protoplanetary cloud and formation of the earth and the planets. Israel Program for Scientific Translations, Jerusalem
45. Huang L, Hu X, Wang Y, Fu Y (2022) EGFAFS: a novel feature selection algorithm based on explosion gravitation field algorithm. Entropy 24:873. https://doi.org/10.3390/e24070873
46. Real E, Moore S, Selle A, et al (2017) Large-scale evolution of image classifiers. In: International conference on machine learning. PMLR, pp 2902–2911
47. Krizhevsky A, Hinton G (2009) Learning multiple layers of features from tiny images. 7.
48. Chrabaszcz P, Loshchilov I, Hutter F (2017) A Downsampled Variant of ImageNet as an Alternative to the CIFAR datasets. arXiv preprint, arXiv:1707.08819
49. Zhang Z, Sabuncu M (2018) Generalized cross entropy loss for training deep neural networks with noisy labels. Adv Neural Inf Process Syst. 31.
50. Dong X, Yang Y (2019) One-shot neural architecture search via self-evaluated template network. In: Proceedings of the IEEE international conference on computer vision. ICCV, pp 3681–3690
51. Zhang M, Su SW, Shirui P et al (2021) iDARTS: Differentiable architecture search with stochastic implicit gradients. In: Proceedings of the 38th international conference on machine learning. PMLR, pp 12557–12566
52. Sinha N, Chen K-W (2021) Evolving neural architecture using one shot model. In: Proceedings of the genetic and evolutionary computation conference. ACM, Lille France, pp 910–918
53. Jie H, Li S, Gang S (2018) Squeeze-and-excitation networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. CVPR, pp 7132–7141
54. Sun K, Li M, Liu D, Wang J (2018) IGCV3: Interleaved Low-Rank Group Convolutions for Efficient Deep Neural Networks. arXiv preprint, arXiv:1806.00178
55. Zhang X, Zhou X, Lin M, Sun J (2018) ShuffleNet: an extremely efficient convolutional neural network for mobile devices. In: Proceedings of the IEEE conference on computer vision and pattern recognition. CVPR, pp 6848–6856
56. Zagoruyko S, Komodakis N (2017) Wide Residual Networks. arXiv preprint, arXiv:1605.07146
57. Zhang H, Jin Y, Cheng R, Hao K (2021) Efficient evolutionary search of attention convolutional networks via sampled training and node inheritance. IEEE Trans Evol Comput 25:371–385. https://doi.org/10.1109/TEVC.2020.3040272
58. Xue Y, Chen C, Słowik A (2023) Neural architecture search based on a multi-objective evolutionary algorithm with probability stack. IEEE Trans Evol Comput 27:778–786. https://doi.org/10.1109/TEVC.2023.3252612
59. Dong J, Cheng AC, Juan D, et al (2018) DPP-Net: device-aware progressive search for pareto-optimal neural architectures. In: Proceedings of the European conference on computer vision. ECCV, pp 517–531

60. Baker B, Gupta O, Naik N, Raskar R (2017) Designing Neural Network Architectures using Reinforcement Learning. arXiv preprint, arXiv:1611.02167
61. Deng J, Dong W, Socher R, et al (2009) ImageNet: a large-scale hierarchical image database. In: Proceedings of the IEEE international conference on computer vision. CVPR, Miami, pp 248–255
62. Fan L, Wang H (2022) Surrogate-assisted evolutionary neural architecture search with network embedding. Complex Intell Syst. https://doi.org/10.1007/s40747-022-00929-w