



A social development algorithm for neural architecture search

Boqian Du¹ · Sheng Su^{1,2} · Haijie Yu³ · Yandong Wang¹ · Cuiying Feng¹ · Jing Tian¹ · Hongyu Huang¹ · Ssemwogerere Rajab¹ · Jingyi Wang³

Received: 14 October 2024 / Accepted: 16 April 2025 / Published online: 6 June 2025
© Springer-Verlag GmbH Germany, part of Springer Nature 2025

Abstract

The design of deep neural network architectures is crucial in deep learning, with superior architectures yielding exceptional performance. The evolutionary algorithms commonly used in neural architecture search often suffer from local optima and slow convergence. Based on an analysis of human societal development, this paper proposes a social development algorithm (SDA) to address NAS challenges. The novelty of the algorithm primarily lies in maintaining population diversity through the social collaborative operation and elimination and supplementation strategy, which helps avoid local optima. Additionally, the failure tolerance mechanism ensures the search efficiency of the algorithm. Experimental results indicate that the social development algorithm (SDA) improves architecture fitness by 4% within the same timeframe as the Zen-NAS algorithm. We conducted a comparative analysis of SDA against existing state-of-the-art methods on the CIFAR datasets and ImageNet-1K. On the CIFAR-10 dataset, SDA demonstrated superior performance, surpassing the classification accuracy of the Zen-NAS and LEMONADE algorithms by 0.76% and 0.65%, respectively. On the CIFAR-100 dataset, SDA achieved a significant improvement, outperforming Zen-NAS, EffPNet, and AmoebaNet-A by 1.17%, 1.53%, and 1.97%, respectively. On the ImageNet-1K dataset, although EfficientNet-B7 achieved the highest accuracy of 84.4% at a high computational cost of 3800 GPU days, SDA achieved a competitive accuracy of 82.6% with only 0.42 GPU days, demonstrating superior efficiency in neural architecture search. Moreover, SDA outperformed other state-of-the-art methods, showcasing exceptional efficiency and cost-effectiveness in both classification and search capabilities.

Keywords Evolutionary Algorithms · neural architecture search (NAS) · Image Classification · Deep Learning · Convolutional neural networks (CNNs)

1 Introduction

Deep Neural Networks (DNNs), the cornerstone of deep learning, have demonstrated immense success in various practical applications, including image classification, natural language processing, price forecasting [1–3] and speech recognition. Unlike traditional Gaussian process regression [4–6], the performance of DNNs depends on architecture and weights. Optimal performance is only achieved when

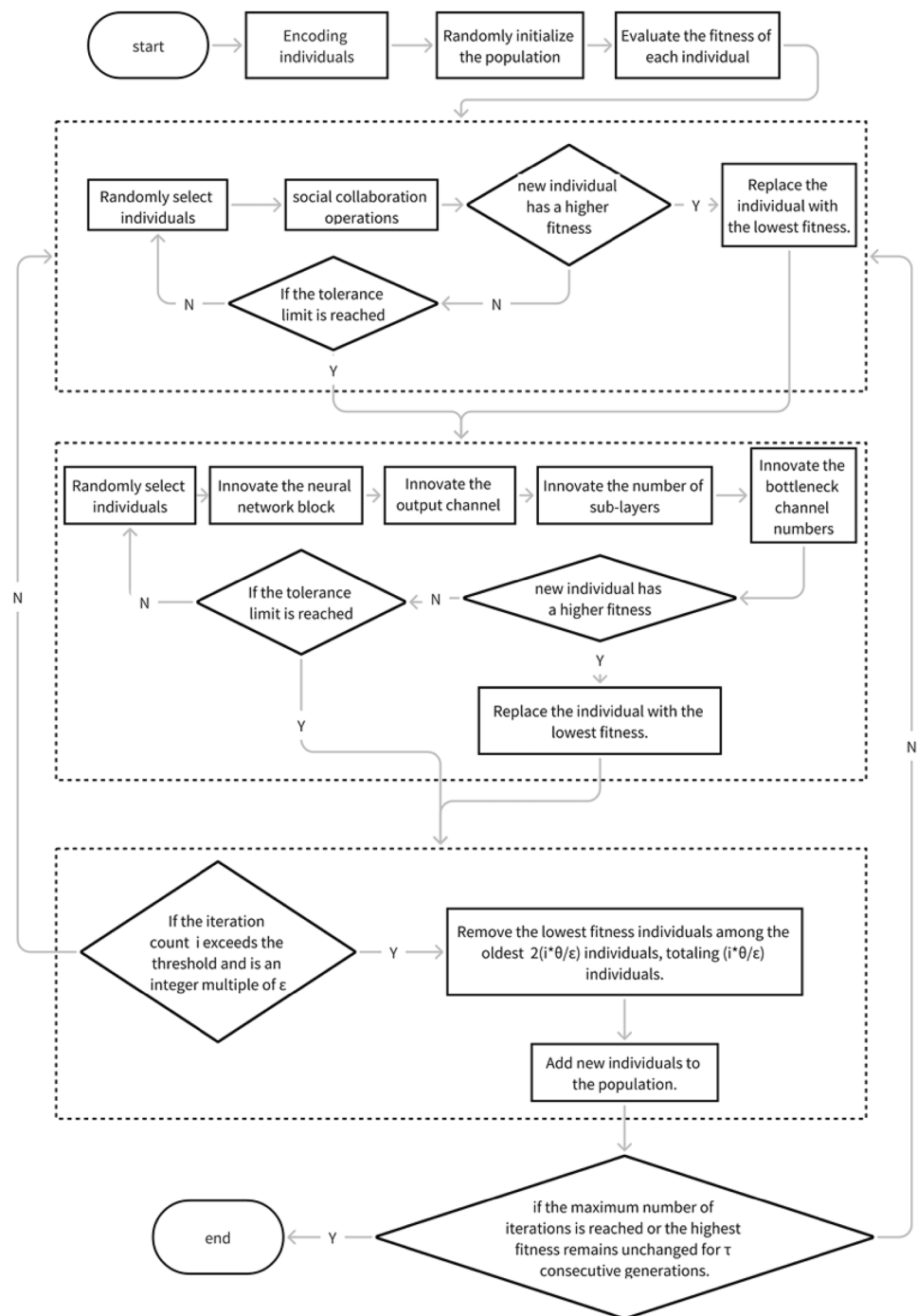
both are in their best state. Optimal weights are usually obtained through the training process: a continuous loss function measures the difference between the actual output and the expected output, and a gradient-based algorithm is typically used to minimize this loss. Historically, promising DNN architectures were manually designed using extensive expert knowledge, as seen in models like VGG [7], ResNet [8], and DenseNet [9]. Achieving an efficient network requires repeated experimentation, improvements, and adjustments, thus consuming substantial time and GPU resources. Neural Architecture Search (NAS) aims to transform the task of DNN architectural design into an automated process to optimize the structural design of neural networks. This reduces reliance on deep learning experts and significantly decreases the time required to design efficient networks, enhancing the efficiency of finding superior architectures. NAS algorithms allow us to explore and evaluate many network architectures to identify the model best suited for a specific task.

✉ Sheng Su
susheng@uestc.edu.cn

¹ School of Information and Software Engineering, University of Electronic Science and Technology of China, Chengdu 610000, China

² Yangtze Delta Region Institute, University of Electronic Science and Technology of China, Huzhou 313000, China

³ School of Economic and Management, Chongqing University of Art and Science, Chongqing 404100, China

Fig. 1 Algorithm flowchart

Existing Neural Architecture Search (NAS) algorithms can be broadly categorized into three types based on the search strategies used: NAS algorithms based on Reinforcement Learning (RL) [10], gradient-based NAS algorithms, and those using Evolutionary Computation (EC) [11], specifically Evolutionary NAS (ENAS) algorithms. Specifically, RL-based algorithms typically require thousands of Graphics Processing Units (GPUs), even when executed on moderately sized datasets such as the CIFAR-10 image classification

benchmark [12]. Gradient-based algorithms are more efficient than those based on RL. However, researchers often encounter pathological architectures due to the incorrect relation of gradient-based optimization, which has not yet been mathematically proven. Additionally, gradient-based methods require the prior construction of a supernet, which demands significant expertise. Evolutionary algorithms are renowned for their ability to perform global searches, meaning they can explore a broad area of the search space, not

just local optima. This capability allows ENAS to discover more diverse and innovative network architectures that traditional gradient-based optimization methods might overlook. However, despite their effectiveness in many domains, evolutionary algorithms have drawbacks and limitations, such as the potential need for extended periods to find an optimal or acceptable solution, especially in cases with large search spaces or high problem complexity. Algorithms may need to iterate through many generations before converging. Although evolutionary algorithms have strong global search capabilities, they might still get trapped in local optima, potentially preventing them from finding the global optimum. In response to the analysis and limitations of NAS algorithms, this paper proposes an efficient and flexible social development algorithm to design efficient, compact CNN architectures. By examining the experiences of human social development, we identify three critical factors influencing societal progress: individual quality, social systems, and developmental resources. We argue that in a continuously evolving and optimizing society, the quality of individuals should improve over time. A well-functioning social system includes excellent capabilities for social collaboration, innovation mechanisms, and a tolerance for failure. Drawing on these insights from social development, we introduce the Social Development Algorithm to address the challenges of neural network architectural design. The main contributions are as follows:

1. The concept of social collaborative operation is proposed in this paper. In the process of societal development, people collaborate to solve problems. The social collaborative operation of this algorithm randomly selects multiple parents from the population and places a portion of each parent's genes into a new individual. This ensures that the offspring contain genes from multiple parents, thereby more effectively maintaining genetic diversity within the population. Furthermore, this approach introduces more variation and diversity, which assists in escaping local optima.
2. A failure tolerance mechanism is introduced in this algorithm. After executing fusion and innovation operations, the algorithm assesses the fitness of the newly generated individuals, comparing it to the fitness levels of existing members of the population. If the offspring's fitness does not surpass that of any individual in the population, the failure tolerance mechanism is activated. This involves repeatedly performing the fusion or innovation operations until a superior offspring is produced or the maximum number of failure tolerance attempts is reached. The failure tolerance mechanism enables the algorithm to persistently attempt to generate more optimal individuals, thereby enhancing its search capabilities and aiding in discovering superior solutions. By continuously striving to

generate superior individuals, the failure tolerance mechanism can accelerate the optimization process and help find solutions closer to the optimal in a limited number of iterations.

3. The algorithm incorporates individual selection operations and developmental resource provision. As the number of iterations increases, the algorithm escalates the quantity of individuals culled, thereby accelerating the evolutionary process of the population. Once a certain threshold is reached, the number of eliminations remains constant. After eliminating certain individuals, new ones are added to the population. This approach serves two primary purposes: firstly, to maintain a constant population size, and secondly, to introduce new individuals. These newly added individuals may possess different genetic combinations, enhancing the diversity of the population, improving the quality of its members, and preventing the algorithm from converging on local optima.

The remainder of this paper is organized as follows. Section 2 provides an overview of the background and related work. Section 3 details the proposed Social Development Algorithm. Section 4 presents the experimental design and results. Finally, Sect. 5 concludes the paper.

2 Background and related work

Elsken et al. [13] divide NAS (Neural Architecture Search) into three phases: search space, search strategy, and performance estimation strategy. The ENAS (Efficient Neural Architecture Search) algorithm utilizes Evolutionary Computation (EC) techniques to address NAS challenges. EC is a family of algorithms that simulate natural evolutionary processes such as natural selection and genetic mutation to solve optimization and search problems. These algorithms draw from the concept of biological evolution, iteratively developing a population through selection, replication, mutation, and recombination to continuously refine solutions. Notably, Genetic Algorithms (GA) [14], Genetic Programming (GP) [15], and Particle Swarm Optimization (PSO) [16] are commonly used EC methods in practice. Due to their robustness against local minima and independence from gradient information, EC methods have been widely applied to solve complex non-convex optimization problems [17], even when the mathematical form of the objective function is not explicitly available [18]. EC methods have been employed for over two decades to search for optimal neural architectures and evolve the weights of neural networks, a process known as neuroevolution [19]. Compared to neuroevolution, ENAS focuses more on searching for architectures, subsequently using gradient-based algorithms to optimize weight values. The first instance of ENAS is often considered to be

the LargeEvo algorithm [19], which was proposed by Google and published in an early version on arXiv in March 2017. This paper was later accepted at the 34th International Conference on Machine Learning in June 2017. The LargeEvo algorithm utilizes GA to search for the best architectures of CNNs, and its effectiveness has been demonstrated through experimental results on CIFAR-10 and CIFAR-100 [12]. Subsequently, a multitude of ENAS algorithms have been proposed. The key to ENAS lies in its strategy methods for sampling parents, updating the population, and generating offspring. For instance, Real et al. (2017) [20], Real et al. (2019) [21], and Liu et al. (2018b) [22] employ tournament selection for sampling parents, whereas Elsken et al. (2019) [23] use inverse density sampling from the multi-objective Pareto front. Real et al. (2017) [20] eliminate the worst-performing individuals from the population, while Real et al. (2019) [21] find that removing the oldest individuals (reducing greed) is beneficial, and Liu et al. (2018b) [22] do not remove any individuals at all. For offspring generation, most methods initialize sub-networks randomly, whereas Elsken et al. (2019) [23] adopt Lamarckian inheritance, transferring knowledge (in the form of learned weights) from parent networks to offspring networks through network morphisms. Real et al. (2017) [20] also allow offspring to inherit all parameters from their parents unaffected by the applied mutations. However, this inheritance does not strictly preserve functionality and can potentially accelerate learning compared to random initialization. We propose a new neural network architecture search strategy, the Social Development Algorithm.

3 Algorithm introduction

3.1 Algorithm description

By analyzing the experiences of human social development, we extract three crucial factors that influence the development of a society: individual quality, social system, and development resources. We believe individual quality should improve progressively in a continuously developing and optimizing society. Such a society should have a sound social system, including excellent organizational problem-solving abilities, a free innovation mechanism, and a failure tolerance mechanism. Drawing from social development, we propose the Social Development Algorithm (SDA) for neural network architecture search. This algorithm incorporates individual selection operations, fusion operations, individual innovation exploration operations, failure tolerance mechanisms, and development resource supply operations. For the problem of neural network architecture search, conventional EA algorithms often encounter issues such as getting trapped in local optima, failing to find the global optimal solution, and

slow convergence. The social development algorithm introduces a trapezoidal elimination strategy and development resource supply operations to address these shortcomings. These innovations promptly eliminate poorly performing neural network architectures while simultaneously supplementing the population with new architectures that contain excellent genetic fragments, thus avoiding local optima and improving the chances of finding the global optimum. The failure tolerance mechanism maximizes the effectiveness of the fusion and individual innovation exploration operations, maintaining the evolutionary speed of the population and facilitating rapid convergence.

3.2 Algorithm process

The process of the social development algorithm is illustrated in Fig. 1. The algorithm begins by randomly initializing a population of neural network architectures, with each individual representing the encoding of a network architecture. Following this, the fitness of each architecture is computed. The details of the random initialization process and fitness calculation will be introduced later. After computing the fitness, the algorithm enters the iteration process, where we first randomly select several architectures from the population for fusion operations. The fusion process combines parts of each selected architecture to generate a new individual, which may alter the individuals within the population. After the fusion process, several individuals are randomly selected for innovation exploration operations. The innovation exploration operation modifies the selected individuals' architectures, including replacing certain neural network blocks (but not all) with other blocks from the search space, and setting different output channel numbers, sub-layer numbers, and bottleneck channel numbers.

Following this, we enter a conditional elimination phase. When the number of iterations exceeds a certain threshold, we eliminate a certain number of individuals from the current population. The number of individuals eliminated increases with the number of iterations, and this increment can be controlled by setting parameters. After the elimination step, the population size decreases, requiring us to add the same number of individuals to the population to maintain its size. Once the addition is complete, an iteration cycle is completed. At this point, we check if the algorithm meets the termination conditions, such as reaching the iteration limit or observing no change in the highest fitness of the population for N consecutive generations. The iteration terminates if the conditions are met, yielding an optimized population. We then select the individual with the highest fitness from this population as the result of our algorithm, representing the most optimal architecture.

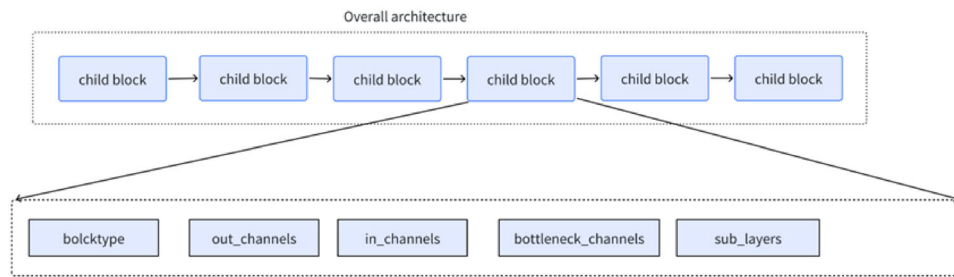


Fig. 2 Encoding strategy of architecture. The overall architecture is constructed by sequentially connecting multiple sub-blocks. Each sub-block is defined by a string that specifies the sub-block's network block type, output channel number, input channel number, bottleneck channel number, number of sub-layers, and stride. For example, if a sub-block string is SuperResK1K7K1(8,656,1,16,5), it indicates that the sub-block

type is SuperResK1K7K1, with an input channel number of 8, an output channel number of 656, a stride of 1, a bottleneck channel number of 16, and 5 sub-layers. The overall architecture string is composed of multiple such substrings, representing a neural network architecture formed by serially connecting the blocks corresponding to these substrings

3.3 Encoding method and search space

The search space is constructed based on references [24] and [25], consisting of residual and bottleneck blocks defined in ResNet. Using string encoding, we represent the solution to the neural network architecture search problem, i.e., the network architecture. The encoding comprises multiple substrings, each representing a neural network block. A substring includes the name of the neural network block and numbers enclosed in parentheses. For network blocks of the SuperConvKXBNRELU type, the numbers in parentheses indicate the input channel number, output channel number, and stride of the block. For SuperResKXXK and SuperResK1KXXK1 type blocks, the numbers represent the input channel number, output channel number, stride, and a number of sub-layers, respectively. A parent string formed by concatenating multiple substrings in sequence represents the complete architecture of a neural network, where each substring corresponds to a sequentially connected neural network block. The overall neural network architecture encoding and the encoding of local sub-blocks are shown in Fig. 2. Figures 3 through 5 illustrate the specific structural compositions of SuperResKXXK and SuperResK1KXXK1 network blocks and how different strings represent these structures.

3.4 Algorithm framework

Algorithm 1 presents the overall framework of the Social Development Algorithm. The main components of the algorithm include social collaborative operation, social innovation operation, trapezoidal elimination, and development resource supply operation. The algorithm searches the search space through social collaborative operation and innovation operation, avoids ineffective searches through the failure tolerance mechanism, promptly eliminates individuals with

no development potential using the trapezoidal elimination strategy, and adds individuals with excellent genetic fragments to the population through the development resource supply operation.

Algorithm 1

Input: Maximum number of iterations σ , Tolerance for unchanged highest fitness over generations τ , Baseline parameter for the number of individuals to be removed θ , Iteration threshold for removing individuals η , Individuals are removed at iterations that are integer multiples of ϵ , Upper limit of tolerance for individual innovation γ , Maximum tolerance for social collaboration operation failures β , Number of individuals randomly selected for fusion operations α , Upper limit on the number of individuals to be eliminated h .

Output: The optimal architecture s

```

1: Randomly initialize the algorithm to obtain the initial population  $P$ .
2: Number of iterations, denoted as  $i \leftarrow 0$ .
3: Count of consecutive iterations with unchanged maximum fitness, denoted as  $n \leftarrow 0$ .
4: Maximum fitness  $l \leftarrow 0$ .
5: while  $i < \sigma$  &  $n < \tau$  do
6:   Social collaborative operation tolerance frequency blend  $\leftarrow 0$ .
7:   while  $blend < \beta$  do
8:     Randomly select  $\alpha$  individuals from population  $P$ .
9:     Perform social collaboration operations on these  $\alpha$  individuals.
10:    Compute the fitness of the new individual  $q$ .
11:    if the fitness of  $q >$  the lowest fitness in the population then
12:      Replace the individual corresponding to the lowest fitness with  $q$ .
13:    BREAK
14:  end if
15:  ++blend
16: end while
17: Innovation tolerance frequency explore  $\leftarrow 0$ .
18: while  $explore < \gamma$  do
19:   Randomly select an individual  $p$  from  $P$ .
20:   Innovate network blocks for  $p$ .
21:   Innovate the number of output channels for  $p$ .
22:   Innovate the number of sub layers for  $p$ .
23:   Innovate the number of bottleneck channels for  $p$ .
24:   Calculate the fitness of the new individual  $q$ .
25:   if the fitness of  $q >$  the lowest fitness in the population then
26:     Replace the individual corresponding to the lowest fitness with  $q$ .

```

```

27:     BREAK
28:   end if
29:   ++explore
30: end while
31:  $r \leftarrow i/\epsilon$ 
32: if  $r$  is an integer & the number of individuals in the population
  > threshold  $\eta$  then
33:   if  $r * \theta < h$  then
34:      $u \leftarrow r * \theta$ 
35:   else
36:      $u \leftarrow h$ 
37:   end if
38:   Identify the oldest  $2*u$  individuals from the existing popula-
   tion  $P$  as a group.
39:   Eliminate the  $u$  individuals with the lowest fitness from the
   group.
40:   Randomly generate  $u$  new solutions and add them to the exist-
   ing population  $P$ .
41: end if
42:  $i++$ 
43: if Maximum fitness of existing population  $o > l$  then
44:    $l \leftarrow o$ 
45:    $n \leftarrow 0$ 
46: else
47:    $++n$ 
48: end if
49: end while
50: output the optimal architecture  $s$ .

```

3.5 Social collaborative operation

Randomly select N individuals from the population for the collaborative operation, in which each individual is an encoded neural network architecture. Begin by selecting two individuals, A and B, from these N individuals. Create a mixed neural network architecture encoding, blend1 , by combining the first substring (first neural network block) of A with the second substring and subsequent parts of B. Next, blend blend1 with a randomly selected third individual, C, by connecting the first two substrings of blend1 with the third substring and subsequent parts of C, resulting in blend2 . Repeat this process until all N individuals have been fused. After fusion, adjust the input and output channel numbers of the entire string. As shown in Fig. 6, we first select 6 individuals from the population, labeled A, W, X, D, G, and I. The substrings a, w, x, d, g, and i represent the sub-neural network blocks of these 6 individuals. These 6 network blocks are sequentially connected to form a new individual. After the social collaborative operation, the new architecture string includes parts of each old string.

Algorithm 2

Input: Maximum tolerance for social collaboration operation failures β , number of individuals randomly selected for fusion operations α , maximum number of individuals to be eliminated h , Social collaborative operation tolerance frequency blend .

Output: The new architecture generated by social collaboration operations e

```

1: Initialize a new empty architecture  $e$ 
2: while  $\text{blend} < \beta$  do
3:   Randomly select  $\alpha$  individuals from population  $P$ 
4:   for  $i = 0$  to  $\alpha$  do
5:     if the  $i$ -th individual has  $i$  sub-blocks then
6:       Randomly select the  $i$ -th sub-block  $e_i$  of the  $i$ -th individual
7:       Set the number of output channels of the last block of  $e$  as
       the input channel count of  $e_i$ .
8:       Add sub-block  $e_i$  to  $e$ .
9:     else
10:      continue
11:    end if
12:  end for
13:  Compute the fitness of the new individual  $q$ .
14:  if the fitness of  $q >$  the lowest fitness in the population then
15:    Replace the individual corresponding to the lowest fitness
    with  $q$ .
16:  break
17: end if
18: ++blend
19: end while

```

3.6 Social innovation operation and random initialization of population

The social innovation operation involves modifying several neural network blocks within the architecture. For each randomly selected neural network block, i.e., a specific substring in the encoding, we primarily alter the block's type, number of output channels, and number of sub-layers. We randomly select a replacement block from the search space for a given neural network block, thereby changing the block's type to another type. For the block's output channels, we multiply the number indicated in the block's substring by 1, 1.25, 1.5, 2, and 2.5, and divide by 1.25, 1.5, 2, and 2.5. These numbers are then processed to be multiples of 8, deduplicated, and added to a set. A random number from this set is selected to replace the number of original output channel in the block's substring. The innovation operation for the bottleneck channel number follows the same procedure as for the number of output channel. We adjust the original number by +1, +2, -1, -2, and +0 for the number of sub-layers. These numbers are deduplicated, added to a set, and a random number from this set is selected to replace the original number of sub-layer in the block's substring. After replacement, the number of input channel of the new block is set to the number of output channel of the preceding block. After several social innovation operations, we explore various blocks within the neural network. We then remove any potentially empty blocks and adjust the channel relationships between

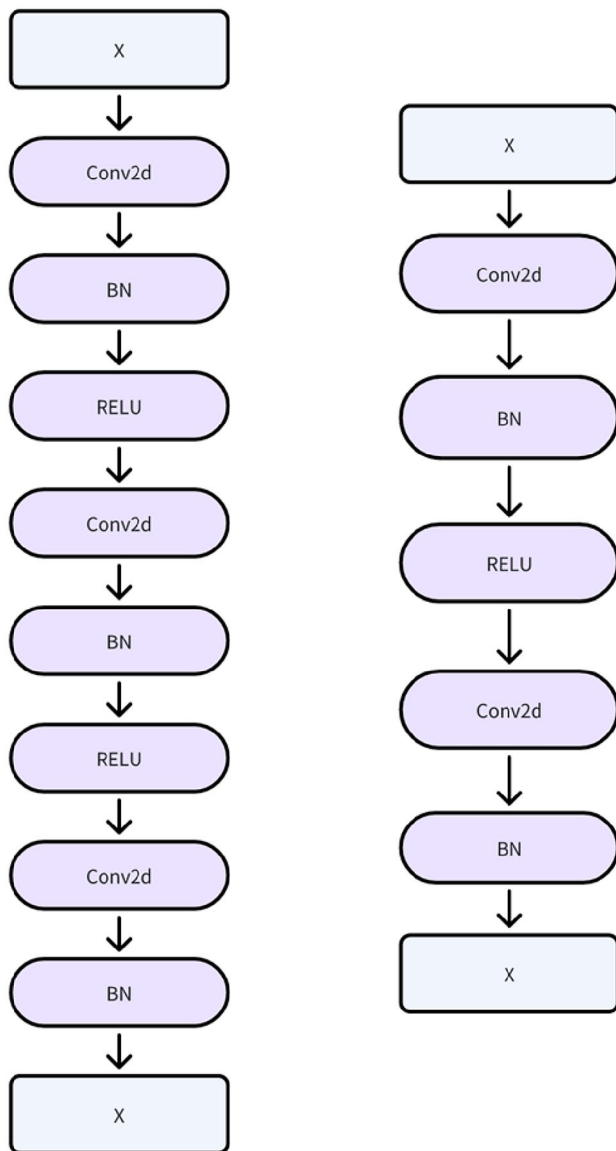


Fig. 3 The structure of basic blocks a and b, with a on the left and b on the right

blocks to form a new individual, q . We compute the fitness score of the new individual, q . If q 's fitness score exceeds that of some individuals in the population P , we replace the worst-performing individual in P with q . Otherwise, we implement a failure tolerance mechanism, continuing the neural network architecture innovation operation until the maximum number of operations ϕ is reached or finding a new individual q that outperforms some individuals in P , replacing the worst-performing individual in P with q .

Algorithm 3

Input: Maximum number of individual innovation tolerance γ , number of individual innovation operations δ .

Output: The new architecture generated by social innovation operations e

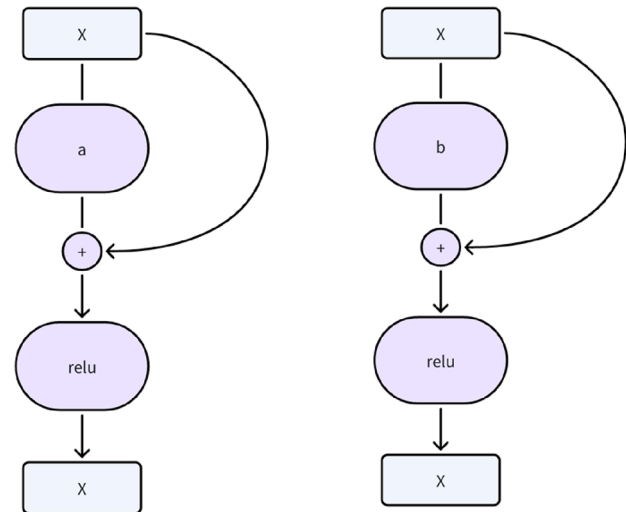


Fig. 4 The structure of blocks RESA and RESB, with RESA on the left and RESB on the right

```

1: Number of social innovative operations  $i \leftarrow 0$ 
2: Number of social innovative operation tolerances  $explore \leftarrow 0$ 
3: while  $i < \delta$  do
4:   while  $explore < \gamma$  do
5:     Randomly select an individual  $p$  from population  $P$ .
6:     Randomly select a sub-block  $p_i$  from  $p$ .
7:     Randomly replace the neural network block in  $p_i$  with a new
       neural network block.
8:     Calculate a set of potential number of output channel by mul-
       tiplying and dividing the original output channel count of  $p_i$  by 1,
       1.25, 1.5, 2, 2.5.
9:     Normalize all new numbers of output channel in the set to
       multiples of 8 and remove duplicates.
10:    Randomly select one number of output channel from the set
       to replace the original number.
11:    Generate a set of potential sub-layer numbers by adding and
       subtracting 1, 2, -1, -2, and 0 to/from the original sub-layer count of
        $p_i$ .
12:    Remove duplicates from the new sub-layer numbers set.
13:    Randomly select one sub-layer number from the set to replace
       the original layer count.
14:    Calculate a set of potential bottleneck channel counts by mul-
       tiplying and dividing the original bottleneck channel count of  $p_i$  by
       1, 1.25, 1.5, 2, 2.5.
15:    Normalize all new numbers of bottleneck channel in the set
       to multiples of 8 and remove duplicates.
16:    Randomly select one bottleneck channel number from the set
       to replace the original count.
17:    Compute the fitness of the new individual  $q$ .
18:    if the fitness of  $q >$  the lowest fitness in the population then
19:      Replace the individual corresponding to the lowest fitness
       with  $q$ .
20:    break
21:    end if
22:     $++explore$ 
23:  end while
24:   $++i$ 
25: end while

```

Random initialization is used before the algorithm's iteration process. We input a seed architecture containing fewer

Fig. 5 The structure of blocks SuperResK1KXXK1 and SuperResKXXKX. Assuming the string is Super-ResK1K7K1(8,656,1,16,5), the convolution kernel within the block is 7, and the number of RES block is 4 because the number of sub-layers is 5. The value 656 represents the output channels of the SuperResK1K7K1 block, 1 is the stride of the convolution block, and 16 is used to set the input and output channels within the sub-blocks of SuperResK1K7K1

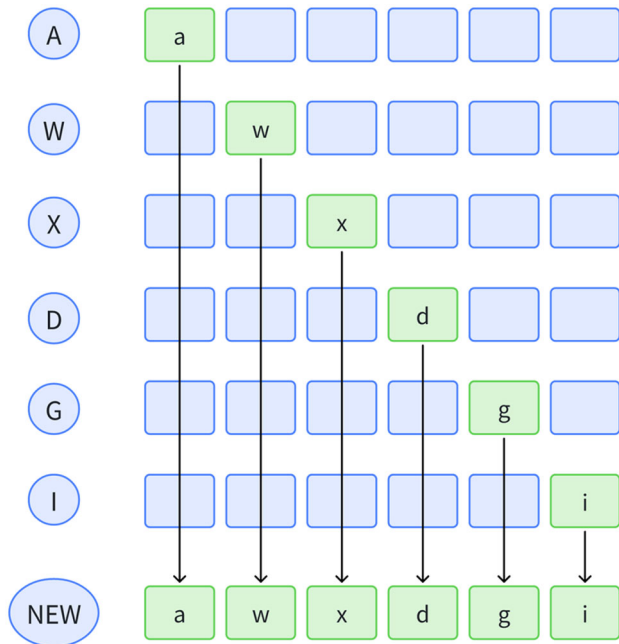
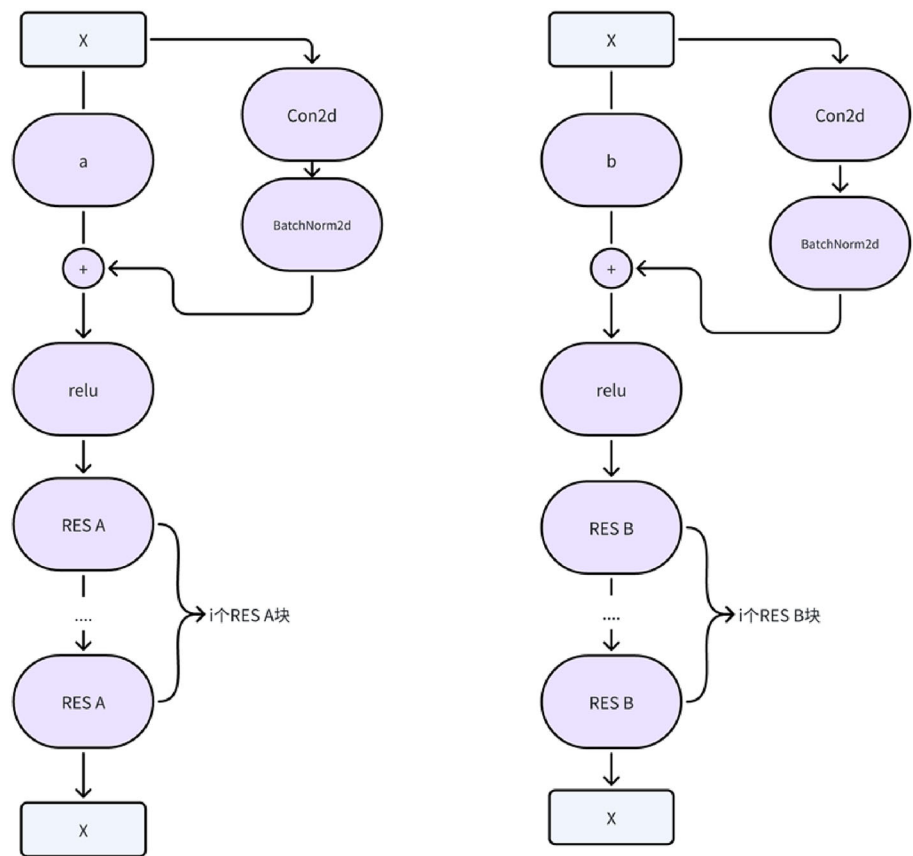


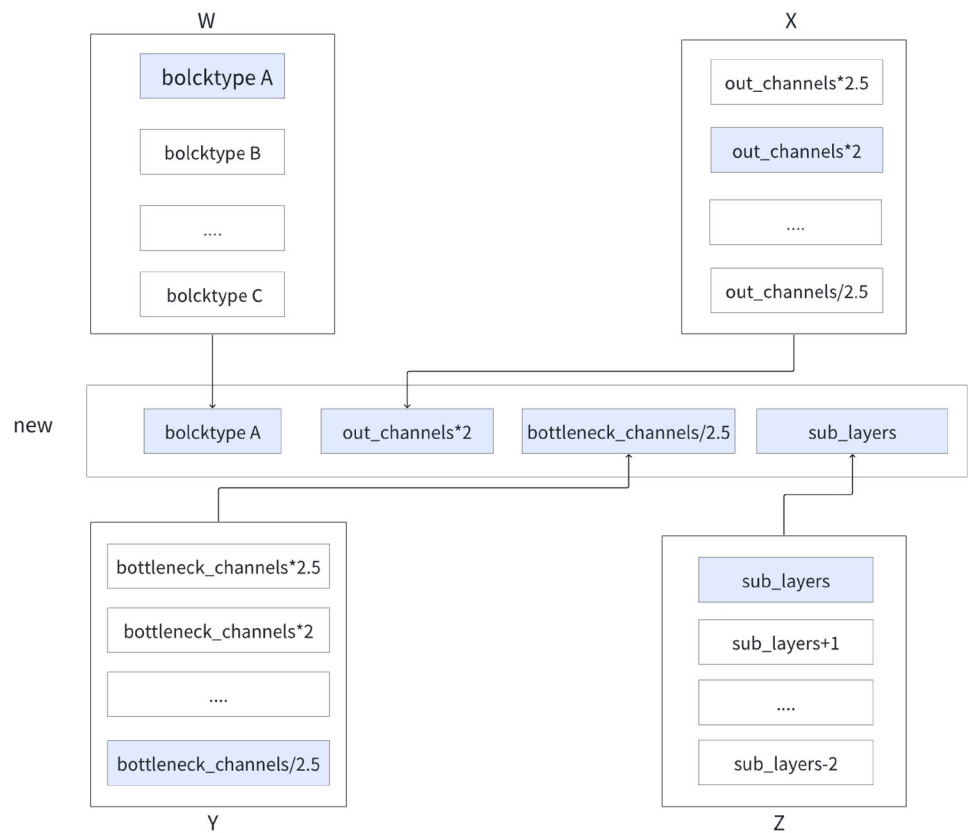
Fig. 6 Example of social collaborative operation. Individuals A, W, X, D, G, and I were randomly selected from the population. The substrings a, w, x, d, g, and i represent the sub-neural network blocks of these six individuals

neural network blocks than the desired final architecture. This seed architecture includes only the essential neural network blocks to ensure that during the social innovation process, the architecture does not transform into only one type or a single neural network block from the search space. This is important because during the exploration of neural network blocks, candidate blocks are derived based on the state of the original blocks. If the seed string contains only a subset of the neural network blocks, it may lead to an incomplete search of the search space. When generating a new individual, the exploration operation differs slightly: the new neural network blocks are added before the original blocks rather than replacing them. This approach increases the number of blocks in the network architecture. A parameter can be set to control the maximum number of blocks in the neural network architecture. This process generates and adds a new architecture to the population. By executing this process multiple times, a population is created that consists of random variations of the seed architecture.

3.7 Elimination and supplementation strategy

To maintain population stability and gradually eliminate low-fitness neural network structures, the algorithm needs to eliminate certain individuals from the population at specific

Fig. 7 Example of the social innovation operation. Involves sets W, X, Y, and Z, which represent the candidate neural network block types, output channel numbers, bottleneck channel numbers, and sub-layer counts, respectively. The innovation operation randomly selects from these four sets to form new neural network blocks



intervals. First, calculate the absolute difference between the current number of individuals in the population and the specified population size to obtain the number m . Then, identify the $2*m$ structures with the lowest fitness in the population. Among these $2*m$ structures, find the m structures that have existed the longest and eliminate them. This approach eliminates low-fitness individuals while retaining some of the newer structures, thereby preventing the algorithm from getting stuck in local optima and affecting the final search result. The number of individuals removed and supplemented increases gradually with the iteration process and eventually stabilizes.

4 Experiments and discussions

In this chapter, we design a series of experiments to validate the effectiveness of the proposed algorithm. We primarily use two datasets, CIFAR-10, CIFAR-100 and ImageNet-1k for the experiments.

4.1 Benchmark datasets

We use three widely utilized benchmark datasets in NAS to quantify the algorithm performance: CIFAR-10, CIFAR-100 and ImageNet-1K. These datasets are commonly used for

CNN-based classification tasks, providing numerous challenging competitors for performance comparison. CIFAR-10 consists of 60,000 images, with 50,000 used for training and 10,000 for testing. Each image is a 32×32 RGB image, uniformly distributed across 10 different classes, such as airplanes, cars, birds, and cats. The CIFAR-100 dataset is similar to CIFAR-10 but contains 100 classes, each with 500 training images and 100 testing images. The larger number of classes and the fewer images per class make the classification of CIFAR-100 more challenging. The ImageNet-1K dataset is a widely used benchmark in computer vision, consisting of 1000 object categories with 1.2 million training images, 50,000 validation images, and 100,000 test images. It serves as a foundational dataset for training and evaluating deep learning models in image classification. Compared to the CIFAR datasets, ImageNet-1K contains higher-resolution images and a significantly larger number of categories, making it more challenging for neural network training.

4.2 Experiment settings

The hyperparameters of the algorithm mainly involve the social development algorithm part and the training part. The parameters of the social development algorithm are summarized in Table 1. For all training experiments, we use the Stochastic Gradient Descent (SGD) optimizer with a momen-

tum of 0.9. The weight decay is set to $5e-4$ for CIFAR-10/100 and $4e-5$ for ImageNet. The initial learning rate is $1e-1$, with a batch size of 256. The learning rate follows a cosine decay schedule [26]. Models are trained for up to 1440 epochs on CIFAR-10/100 and 480 epochs on ImageNet. All experiments are conducted on an RTX 4090 GPU.

4.3 Overall results

In this section, we compare our algorithm with numerous state-of-the-art algorithms on the CIFAR datasets and ImageNet-1K. Table 2 presents the experimental results for the CIFAR datasets, while Table 4 showcases the results for ImageNet-1K.

4.3.1 Results on CIFAR datasets

Table 2 presents the experimental results of the social development algorithm and selected peer competitors on CIFAR-10 and CIFAR-100 in terms of architecture complexity, classification performance, and search cost.

In the Table 2, we classify the selected competing methods into three groups based on the degree of automation in the design of network architecture. Note that the degree of automation of the original components (e.g., convolution operators) is not considered. The first Group consists entirely of human-designed state-of-the-art compact CNN architectures, including ResNet110 [27], DenseNet-BC ($k = 12$) [28], MobileNetV2 [29], ShuffleNetV2 [30] and Ghost-ResNet-56 [31]. These compact CNN architectures perform well in image classification tasks, and their modules are widely used in many NAS algorithms. The algorithms in the second group are fully automated and do not require extensive domain knowledge to assist in the architecture search. This group includes NAS V3 [32], MetaQNN [33], LS-Evolution [20], CGP-CNN (ResSet) [34], FPSO [35], AE-CNN [36], and CNN-GA [37]. Note that GD-based NAS algorithms require manual construction of supernet in advance, involving extensive prior knowledge of CNNs. Therefore, the second group only includes RL- and EC-based NAS algorithms. The third set of algorithms belongs to the "automatic+manual" NAS method. The CNN architectures in the set are designed by NAS algorithms and require manual intervention, such as manually predefined macro-architectures or supernet. This group includes NAS algorithms based on reinforcement learning (RL), gradient descent (GD), and evolutionary computing (EC), focusing on compact architecture design such as LEMONADE [23], NSGA-Net [38] and FPNASNet [39]. Other algorithms include Block-QNN-S [40], NASNet-A [41], ENAS [42], DARTS [43], NAO [44], SNAS [45], ProxylessNAS [46], GeNet [47], Hierarchical Evolution [48], AmoebaNet-A [21], EffPNet [49], SI-EvoNet-S [?], and PNAS [50]. Algorithms such as LEMONADE [23], NSGA-

Net [38], GeNet [47], Hierarchical Evolution [48], EffPNet [49], SI-EvoNet-S [51] and AmoebaNet-A [21] use evolutionary computing. Since our algorithm requires manually predefined search spaces, it belongs to the third group, the "Auto + Manual" NAS methods.

Specifically, we use the number of parameters (#Parameters) to indicate architecture complexity and GPU days to measure search cost. A dash ("–") indicates that the corresponding algorithm did not report results. The three groups of peer competitors are separated by horizontal lines, and the best results of the proposed method are shown at the bottom of the table. The resulting three architectures are referred to as SDA-A, SDA-B, and SDA-C.

As shown in Table 2, SDA-A achieves an accuracy of 96.96% on the CIFAR-10 dataset, outperforming all manually designed networks, including ResNet-110 (93.57%) [27], DenseNet BC (95.49%) [28], and MobileNetV2 (94.56%) [29]. Compared to the evolutionary algorithm-based method ZenNet [52], which adopts the same fitness criteria and similar parameter size (0.99M vs. 1M), SDA-A demonstrates superior accuracy. Furthermore, SDA-A ranks ninth among all methods evaluated, with only eight hybrid (manual and automated) methods achieving higher accuracy. However, these top-performing methods, such as ProxylessNAS (97.92%) [46] and NASNet-A (97.35%) [41], require significantly more parameters (5.7M and 3.3M, respectively). Moreover, the search cost of SDA-A is only 0.05 GPU-days, which is 80 times lower than that of DARTS (4 GPU-days) [43] and 4000 times lower than NAO (200 GPU-days) [44]. These results suggest that SDA-A achieves a near-optimal balance between accuracy, model size, and computational efficiency, making it highly competitive for practical deployment.

On the CIFAR-100 dataset, SDA-C achieves outstanding performance, surpassing all manually designed networks such as ResNet-110 (74.84%) [27] and MobileNetV2 (77.09%) [29], with significantly fewer parameters (1.22M compared to 1.7M and 2.1M, respectively). In addition, SDA-C outperforms all automated methods, exceeding EPCNAS-C [53] by 1.37%, while maintaining a lower number of parameters (1.22M vs. 1.29M) and reduced search cost (0.213 vs. 1.1 GPU-days). Among the hybrid (manual and automated) methods, SDA-C outperforms most existing algorithms. For instance, NAO [44] achieves an accuracy of 84.33% using 10.6M parameters, while SDA-C achieves a competitive accuracy of 83.0%, reducing the parameter count by a factor of 8.8. Additionally, SDA-C's search cost is only 0.213 GPU-days, approximately 1000 times lower than that of NAO (200 GPU-days) [44].

Overall, traditional manual design methods, such as ResNet-110 [27] and MobileNetV2 [29], struggle to achieve competitive accuracy and require expert knowledge, limiting their scalability. Fully automated methods, like NAO

Table 1 Social Development Algorithm Parameters

Parameters	Values
Loop count	Counts the number of iterations.
Population size	512 (If the population size falls below the specified threshold, a new network is randomly created.)
Iteration limit	50 (If the maximum fitness score does not improve over 50 consecutive iterations, the iteration stops.)
ϵ	20
η	Half of the population size (i.e., 256). If the loop count is a multiple of 20 and the population size exceeds 256, existing individuals are removed, and new individuals are added.
θ	10 (During individual removal, the $\theta \times \text{Loop count} / \epsilon$ individuals with the lowest fitness among the $2\theta \times \text{Loop count} / \epsilon$ longest-existing individuals are removed.)
β	256 for the social collaboration operation, indicating up to 256 innovation operations. If the fitness of the newly generated individual does not exceed the lowest fitness in the population, collaboration continues until a better individual is found or the 256-iteration limit is reached.
ϕ	50 for the social innovation operation, indicating up to 50 innovation operations. If the fitness of the newly generated individual does not exceed the lowest fitness in the population, innovation continues until a better individual is found or the 50-iteration limit is reached.

[44] and EPCNAS-C [53], often suffer from excessive computational costs, with search times exceeding hundreds to thousands of GPU days, making them impractical for real-world applications. In contrast, semi-automated approaches (Auto + Manual), including SDA, successfully integrate domain knowledge with search algorithms, enabling efficient exploration of the architecture space. SDA-A and SDA-C achieve state-of-the-art performance with significantly lower search costs, outperforming most fully automated methods while requiring only a fraction of their computational budget.

Table 3 summarizes the comparative analysis between SDA and state-of-the-art methods across different search paradigms. While NASNet-A [41] and ProxylessNAS [46] achieve higher CIFAR-10 accuracy, they require significantly more parameters and computational resources. In contrast, SDA-A achieves competitive accuracy with a minimal parameter count and the lowest search cost. On CIFAR-100, SDA-C achieves top-tier accuracy while being one of the most computationally efficient models.

4.3.2 Results On ImageNet-1K

Table 4 presents a comparative analysis of the social development algorithm and other neural architecture search (NAS) methods based on Top-1 accuracy (%) and search cost, measured in GPU days, on the ImageNet-1K dataset. The results highlight the trade-offs between classification performance and computational efficiency among different architectures. Among the listed methods, EfficientNet-B7 [54] achieves the highest Top-1 accuracy of 84.4%, but at the expense of an

extremely high computational cost (3800 GPU days), making it less practical for resource-constrained environments. In contrast, the proposed SDA-D architecture achieves a competitive Top-1 accuracy of 82.6% while requiring only 0.42 GPU days, demonstrating superior efficiency in neural architecture search. Compared to other NAS approaches, SDA-D significantly reduces search costs while maintaining high classification performance. For instance, PNAS [50] achieves a lower 74.2% accuracy with a substantially higher search cost (224 GPU days), while MetaQNN [33] attains 77.4% accuracy but requires 96 GPU days, both of which are considerably less efficient than SDA-D. Moreover, reinforcement learning-based NAS methods, such as NASNet-A [41], demand 1800 GPU days yet achieve only 74.0% accuracy, illustrating their high computational overhead relative to the proposed approach. Overall, SDA-D achieves an optimal balance between accuracy and efficiency, outperforming many state-of-the-art NAS methods in terms of computational cost while maintaining competitive classification performance. These results underscore the effectiveness of SDA-D as a practical and resource-efficient solution for NAS.

4.4 Algorithm analysis

4.4.1 Algorithm performance statistical analysis

In this section, we set the parameters as follows: $\theta = 10$, $\epsilon = 10$, $\eta = 256$, $\beta = 384$, $\phi = 64$, and conducted 20 executions of the algorithm. The maximum execution time for each iteration was set at 5 hours. Upon completion of the

Table 2 Experimental Results On CIFAR datasets

Method	#Params (M)	CIFAR-10	CIFAR-100	GPU Days	Type
ResNet-110 [27]	1.7 M	93.57	74.84	—	Manual
DenseNet-BC (k = 12) [28]	0.8 M	95.49	77.72	—	Manual
MobileNetV2 [29]	2.1 M	94.56	77.09	—	Manual
ShufteNetV2 [30]	2.47M	94.17	—	—	Manual
Ghost-ResNet-56 [31]	0.43 M	92.7	—	—	Manual
NAS V3 [32]	7.1M	95.53	—	22400	Automatic
MetaQNN [33]	11.2 M	93.08	72.86	90	Automatic
LS-Evolution [20]	5.4 M	94.6	—	2750	Automatic
LS-Evolution [20]	40.4 M	—	77	2750	Automatic
CGP-CNN (ResSet) [34]	2.01 M	94.99	—	30	Automatic
FPSO [35]	0.7M	93.72	—	1.65	Automatic
AE-CNN [36]	2.0 M	95.7	—	27	Automatic
AE-CNN [36]	5.4 M	—	79.15	36	Automatic
CNN-GA [37]	2.9 M	96.78	—	35	Automatic
CNN-GA [37]	4.1 M	—	79.47	40	Automatic
EPCNAS-A [53]	0.12 M	95.05	—	1.17	Automatic
EPCNAS-B [53]	0.31 M	96.27	—	1.17	Automatic
EPCNAS-C [53]	1.16 M	96.93	—	1.1	Automatic
EPCNAS-A [53]	0.15 M	—	74.63	1.25	Automatic
EPCNAS-B [53]	0.35 M	—	79.44	1.13	Automatic
EPCNAS-C [53]	1.29 M	—	81.67	1.1	Automatic
LEMONADE [23]	0.5 M	95.43	—	80	Auto + Manual
LEMONADE [23]	1.1 M	96.31	—	80	Auto + Manual
NSGA-Net (macro) [38]	3.3 M	96.15	79.26	8	Auto + Manual
FPNASNet [39]	5.76 M	96.99	—	0.83	Auto + Manual
Block-QNN-S [40]	6.1 M	96.7	82.95	90	Auto + Manual
NASNet-A [41]	3.3 M	97.35	—	2000	Auto + Manual
ENAS [42]	4.6 M	97.11	—	0.45	Auto + Manual
DARTS [43]	3.3 M	97.24	—	4	Auto + Manual
NAO [44]	10.6 M	97.52	—	200	Auto + Manual
NAO [44]	10.8 M	—	84.33	200	Auto + Manual
SNAS [45]	2.8 M	97.15	—	1.5	Auto + Manual
ProxylessNAS [46]	5.7 M	97.92	—	1500	Auto + Manual
GeNet [47]	—	92.9	70.97	17	Auto + Manual
Hierarchical Evolution [48]	15.7 M	96.37	—	300	Auto + Manual
AmoebaNet-A [21]	3.2 M	96.66	81.07	3150	Auto + Manual
EffPNNet [49]	2.54 M	96.51	81.51	< 3	Auto + Manual
SI-EvoNet-S [51]	1.84 M	97.31	—	0.458	Auto + Manual
SI-EvoNet-S [51]	3.32 M	—	84.3	0.813	Auto + Manual
PNAS [50]	3.2 M	96.37	80.47	150	Auto + Manual
ZenNet [52]	0.5 M	96.2	—	—	Auto + Manual
ZenNet [52]	1 M	96.2	—	—	Auto + Manual
ZenNet [52]]	0.5 M	—	79.9	—	Auto + Manual
ZenNet [52]	1.0 M	—	80.1	—	Auto + Manual
SDA-A	0.99 M	96.96	—	0.05	Auto + Manual
SDA-B	0.992 M	—	81.27	0.06	Auto + Manual
SDA-C	1.22 M	—	83.04	0.213	Auto + Manual

Table 3 Comparative Summary of SDA and Peer Methods

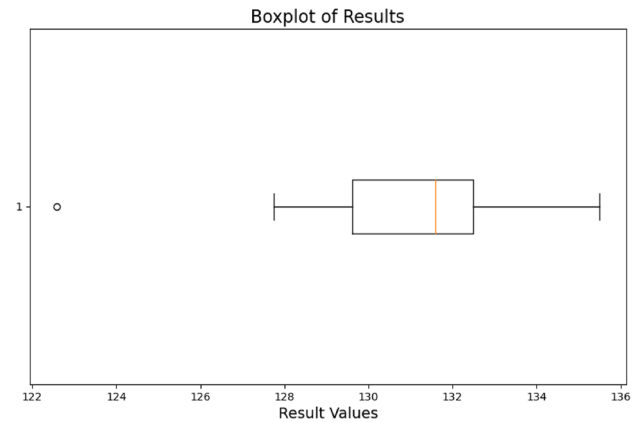
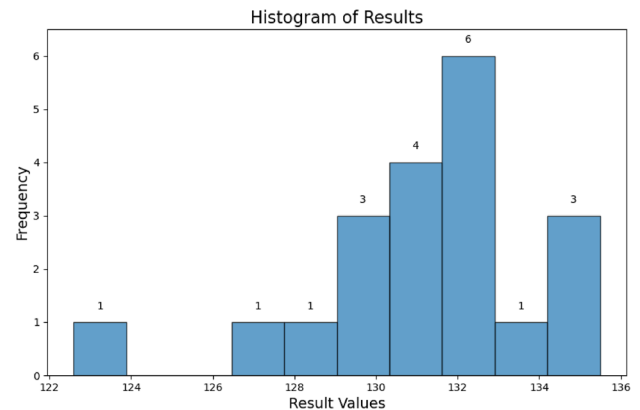
Method	Type	Params (M)	CIFAR-10(%)	CIFAR-100(%)	GPU Days
ResNet-110 [27]	Manual	1.7	93.57	74.84	—
EPCNAS-C [53]	Auto	1.29	—	81.67	1.1
NASNet-A [41]	Auto+Manual	3.3	97.35	—	2000
NAO [44]	Auto+Manual	10.8	97.52	84.33	200
ProxylessNAS [46]	Auto+Manual	5.7	97.92	—	1500
SDA-A (Ours)	Auto+Manual	0.99	96.96	—	0.05
SDA-C (Ours)	Auto+Manual	1.22	—	83.04	0.213

Table 4 Performance of different NAS methods.

Method	Top-1 (%)	GPU Days
AmoebaNet-A [21]	74.5	3150
EcoNAS [55]	74.8	8
CARS-I [56]	75.2	0.4
GeNet [47]	72.1	17
DARTS [43]	73.1	4
SNAS [45]	72.7	1.5
PC-DARTS [57]	75.8	3.8
ProxylessNAS [46]	75.1	8.3
GDAS [58]	74.0	0.8
FBNetV2-L1 [59]	77.2	25
NASNet-A [41]	74.0	1800
Mnasnet-A [60]	75.2	—
MetaQNN [33]	77.4	96
PNAS [50]	74.2	224
SemiNAS [61]	76.5	4
TE-NAS [62]	76.7	0.2
OFANet [63]	80.1	51.6
EfficientNet-B7 [54]	84.4	3800
SDA-D	82.6	0.42

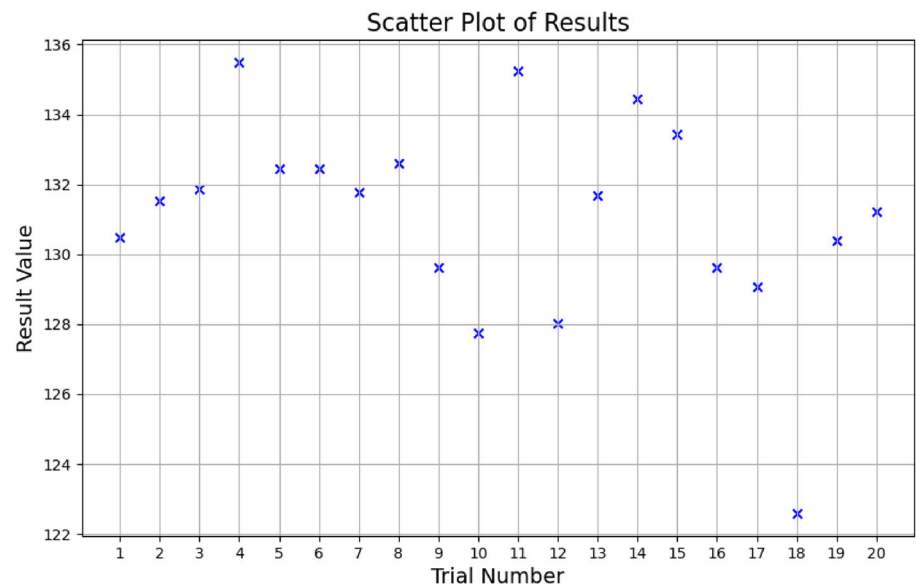
executions, we recorded the fitness of the optimal architecture within the population, which served as the performance metric for the algorithm. Finally, we created a scatter plot to illustrate the performance outcomes of the algorithm.

The box plot (Fig. 8) illustrates the distribution of the results from 20 experimental trials. The plot uses the box and whiskers to depict the data's dispersion and central tendency. The lower edge of the box represents the first quartile (Q1), the upper edge represents the third quartile (Q3), and the central horizontal line denotes the median, or the 50th percentile. According to the data shown in the plot, the majority of the algorithm execution results are concentrated between 130 and 134, indicating that the algorithm's performance is relatively stable. The two whiskers outside the box represent the minimum and maximum values (excluding outliers). Outliers, marked by the individual points, indicate that a few

**Fig. 8** Boxplot of results**Fig. 9** Histogram of results

experimental results deviate significantly from the overall distribution, which may be due to certain specific factors in the experiment. The minimum and maximum values of the results are 122.59 and 135.49, respectively, suggesting that the fluctuation range of the results across the 20 trials is relatively small. The first and third quartiles are 129.08 and 133.42, respectively, further confirming that the majority of the data is distributed within this range.

The histogram (Fig. 9) further illustrates the frequency distribution of the algorithm's execution results. The height

Fig. 10 Scatter Plot of results

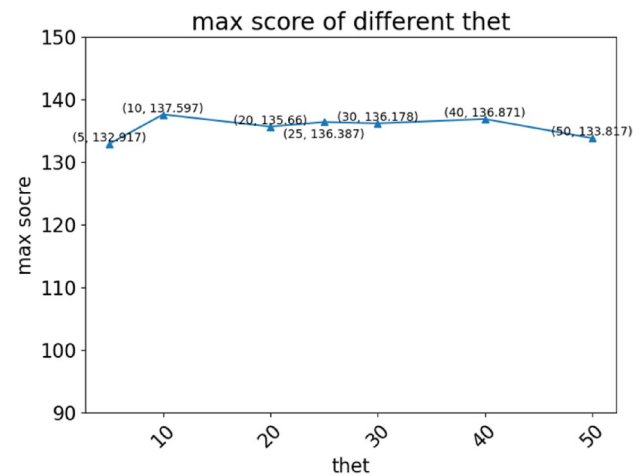
of each bar represents the frequency of experimental results within the corresponding range. From the figure, it is evident that the majority of the results are concentrated in the 130 to 134 range, which is consistent with the analysis from the boxplot. The histogram also reveals a skewness in the data distribution, indicating that the algorithm's execution results tend to fall within a relatively stable range. To enhance the readability of the histogram, the frequency of each bar has been annotated within the figure.

The scatter plot (Fig. 10) presents the specific results of each experimental trial. From the scatter plot, we can observe that the experimental results fluctuate within a certain range, with the majority of the points clustered in the middle range (between 130 and 134). Grid lines have also been added to the scatter plot to further assist readers in identifying the distribution of the data points.

4.4.2 Parameter sensitivity analysis

In this section, we perform a sensitivity analysis for the parameters ϵ , η , θ , β , and ϕ . The search space is constructed based on [24] and [25], consisting of residual blocks and bottleneck blocks defined in ResNet. The parameter limit is set to 1M, the population size is 512 individuals, and the total search time is set to 5 hours. Figure 11, 12, 13, 14 illustrate the performance of the algorithm under different parameter values. In each figure, the horizontal axis represents the parameter value, while the vertical axis denotes the fitness of the optimal architecture within the population after the execution of the algorithm.

From the above illustrations, we can summarize that an excessively large or small value of θ decreases the highest fitness. When the number of individuals eliminated is too small,

**Fig. 11** The fitness of the best individual in the population under different values of θ , with other parameters set as follows: $\epsilon = 15$, $\beta = 384$, $\phi = 64$

many underperforming architectures remain, and the population's genetic diversity decreases, making individuals too similar and thus reducing the algorithm's exploration capability and the speed at which it optimizes the population's overall fitness. Conversely, when the number of individuals eliminated is too large, some potentially excellent architectures might be removed, preventing them from contributing to future generations, which also slows down the evolutionary process. The parameter ϵ behaves similarly to θ but also influences the evolution speed by affecting the frequency of individual elimination. Generally, the more fusion operations performed, the higher the maximum fitness we achieve. However, as the number of fusion operations increases, the cost of fusion operations rises while the corresponding benefits diminish, which can also lead to a decline in maximum fit-

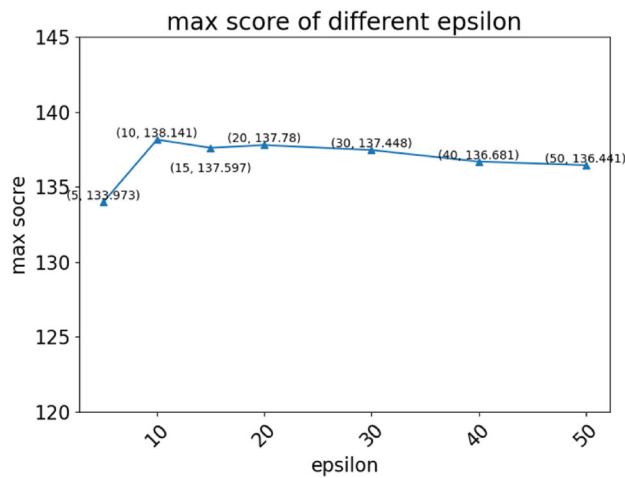


Fig. 12 The fitness of the best individual in the population under different values of ϵ , with other parameters set as follows: $\theta = 10$, $\beta = 384$, $\beta = 64$

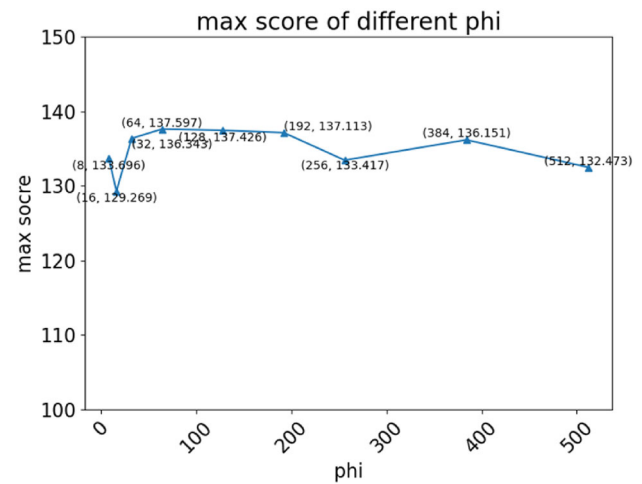


Fig. 14 The fitness of the best individual in the population under different values of ϕ , with other parameters set as follows: $\theta = 10$, $\epsilon = 15$, $\beta = 384$

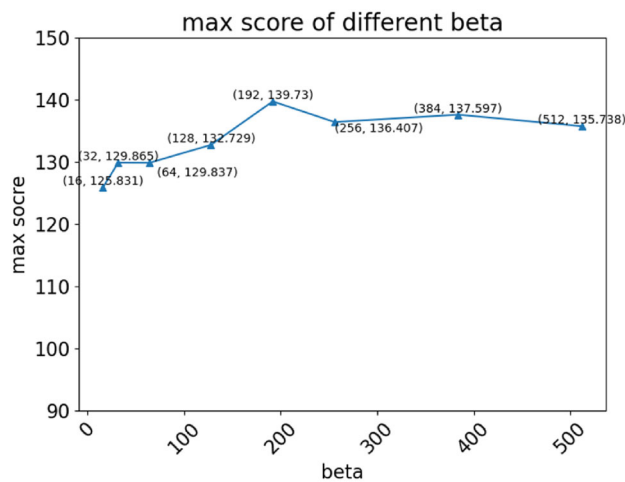


Fig. 13 The fitness of the best individual in the population under different values of β , with other parameters set as follows: $\theta = 10$, $\epsilon = 15$, $\phi = 64$

ness. The number of social innovation operations (ϕ) behaves similarly to the number of fusion operations.

4.4.3 Evolution Trajectory

In this section, we visualize the evolutionary trajectory of the proposed method to observe the behavior of the network architecture search and analyze the algorithm's convergence under the current parameter settings. We conducted 28 experiments, each with different parameters, but controlled the iteration time to be around 5 hours. This resulted in the evolutionary trajectories of the population's optimal architecture fitness under different parameter settings. In Figs. 15 and 16, we plot the change in fitness of the best architecture over time and compare it with the evolutionary curve

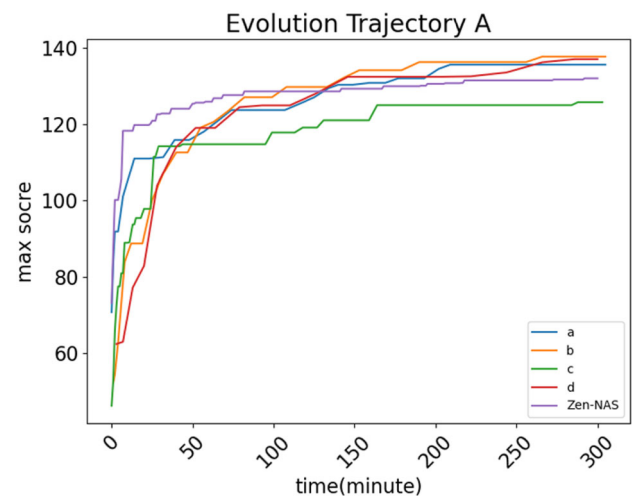


Fig. 15 The evolutionary trajectories of four different parameter sets are plotted with iteration time in minutes on the horizontal axis and the fitness of the best individual in the population on the vertical axis: Group a ($\theta = 20$, $\epsilon = 15$, $\beta = 384$, $\phi = 64$), Group b ($\theta = 10$, $\epsilon = 20$, $\beta = 384$, $\phi = 64$), Group c ($\theta = 10$, $\epsilon = 15$, $\beta = 16$, $\phi = 64$), and Group d ($\theta = 10$, $\epsilon = 15$, $\beta = 384$, $\phi = 192$)

of Zen-NAS. At the start of the iterations, the best architecture in the population is the seed architecture. In the first 1.5 hours of evolution, the evaluation values of the population's architecture rapidly increase in both algorithms. Zen-NAS shows a faster initial improvement compared to our algorithm. Still, as the evolution progresses, the improvement rate of Zen-NAS declines more quickly and converges to a lower maximum value than our algorithm. In the final generations of evolution, our algorithm's population converges to very similar and relatively high fitness levels, strongly demonstrating the effectiveness of the proposed evolutionary algorithm. Our algorithm achieves a rapid improvement in the initial

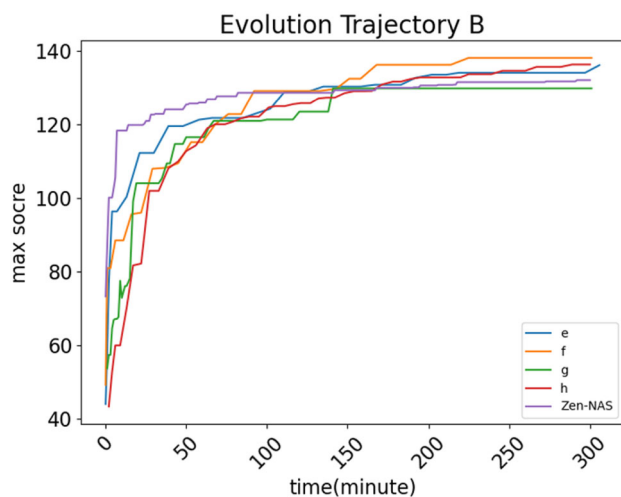


Fig. 16 The evolutionary trajectories of four different parameter sets are shown, with the horizontal axis representing iteration time in minutes and the vertical axis representing the fitness of the best individual in the population: Group e ($\theta = 30$, $\epsilon = 15$, $\beta = 384$, $\phi = 64$), Group f ($\theta = 10$, $\epsilon = 10$, $\beta = 384$, $\phi = 64$), Group g ($\theta = 10$, $\epsilon = 15$, $\beta = 32$, $\phi = 64$), and Group h ($\theta = 10$, $\epsilon = 15$, $\beta = 384$, $\phi = 32$)

generations. Still, the rate of improvement becomes very slow and small in later stages, indicating that the evolutionary search converges to a relatively stable architecture within the first 1.5 hours. It can be anticipated that setting a larger population size and longer iterations (time) could potentially allow the algorithm to find better architectures. However, based on current observations, classification performance may not significantly improve, and additional search time would result in higher computational costs, which may not be worthwhile. Our algorithm uses a variable-length encoding strategy, theoretically capable of evolving CNN architectures of any depth (relationship between architecture depth and iteration count).

5 Conclusion and future research directions

In this paper, we proposed the Social Development Algorithm (SDA) for evolving CNN architectures for image classification tasks inspired by the experiences of human social development. The algorithm incorporates individual selection operations, fusion operations, individual social innovation operations, a failure tolerance mechanism, a trapezoidal elimination strategy, and development resource supply operations. Fusion and individual social innovation operations are designed to maximize the algorithm's ability to generate new architectures during evolution. The trapezoidal elimination strategy and development resource supply operations promptly eliminate poorly performing neural network architectures and simultaneously supplement the population with new architectures containing excellent genetic fragments, avoiding local optima and discovering global optima.

The failure tolerance mechanism ensures the effectiveness of fusion and individual social innovation operations, maintaining the population's evolutionary speed and facilitating rapid convergence.

The first limitation of this work is that the method depends on a predefined set of neural network sub-block types, which means the search results may be closely tied to these predefined components. The second limitation is that the network structure components (neural network sub-blocks) are connected in a sequential manner, which limits the algorithm's ability to explore more complex network architectures.

Therefore, for future work, one promising direction for further exploration and improvement lies in the connectivity between network components (neural network sub-blocks). Future efforts may focus on replacing the current sequential connections with more complex and diverse network topologies, such as mesh-like structures. Another potential avenue is to investigate more flexible and dynamic strategies for the predefined configuration of network components.

Acknowledgements This research was supported by the Pre-research on Common Technologies under Grant No. 50901010101, Science and Technology Support Plan of Sichuan Province of China under Grant No. 2021YFG0024, Luzhou Science and Technology Fund, Sichuan Province of China under Grant No. 2022-XDY-192, Project of Chongqing University of Arts and Sciences, China under Grant No. R2018JG15, Humanities and Social Sciences Research Project of Education Commission of Chongqing, China under Grant No. 21SKGH210, National Natural Science Foundation of China under Grant No. 62302081.

References

- Jin, B., Xu, X.: Carbon emission allowance price forecasting for china guangdong carbon emission exchange via the neural network. *global finance review*. 2024; 6 (1): 3491 (2016)
- Jin, B., Xu, X.: Price forecasting through neural networks for crude oil, heating oil, and natural gas, *Measurement: Energy* 1 (1) 100001 (2024)
- Jin, B., Xu, X.: Wholesale price forecasts of green grams using the neural network, *Asian Journal of Economics and Banking* (ahead-of-print) (2024)
- Jin, B., Xu, X.: Forecasting wholesale prices of yellow corn through the gaussian process regression. *Neural Computing and Applications* **36**(15), 8693–8710 (2024)
- Jin, B., Xu, X.: Pre-owned housing price index forecasts using gaussian process regressions. *Journal of Modelling in Management* **19**(6), 1927–1958 (2024)
- Jin, B., Xu, X.: Predictions of steel price indices through machine learning for the regional northeast chinese market. *Neural Computing and Applications* **36**(33), 20863–20882 (2024)
- Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition, *arXiv preprint arXiv:1409.1556* (2014)
- He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition, in: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778 (2016)
- Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K.Q.: Densely connected convolutional networks, in: *Proceedings of the IEEE*

- conference on computer vision and pattern recognition, pp. 4700–4708 (2017)
10. Kaelbling, L.P., Littman, M.L., Moore, A.W.: Reinforcement learning: A survey. *Journal of artificial intelligence research* **4**, 237–285 (1996)
 11. Bäck, T., Fogel, D.B., Michalewicz, Z.: *Handbook of evolutionary computation*. <https://api.semanticscholar.org/CorpusID:61006071> (1997)
 12. Krizhevsky, A.: Learning multiple layers of features from tiny images. <https://api.semanticscholar.org/CorpusID:18268744> (2009)
 13. Elsken, T., Metzen, J.H., Hutter, F.: Neural architecture search: A survey. *Journal of Machine Learning Research* **20**(55), 1–21 (2019)
 14. Alam, T., Qamar, S., Dixit, A., Benaida, M.: Genetic algorithm: Reviews, implementations, and applications, arXiv preprint [arXiv:2007.12673](https://arxiv.org/abs/2007.12673) (2020)
 15. Banzhaf, W., Francone, F.D., Keller, R.E., Nordin, P.: Genetic programming - an introduction: On the automatic evolution of computer programs and its applications. <https://api.semanticscholar.org/CorpusID:57323334> (1998)
 16. Kennedy, J., Eberhart, R.: Particle swarm optimization, in: *Proceedings of ICNN'95 - International Conference on Neural Networks*, Vol. 4, pp. 1942–1948 vol.4. <https://doi.org/10.1109/ICNN.1995.488968> (1995)
 17. Sun, Y., Yen, G.G., Yi, Z.: Igd indicator-based evolutionary algorithm for many-objective optimization problems. *IEEE Transactions on Evolutionary Computation* **23**(2), 173–187 (2019). <https://doi.org/10.1109/TEVC.2018.2791283>
 18. Darwish, A.A., Hassanien, A.E., Das, S.: A survey of swarm and evolutionary computing approaches for deep learning, *Artificial Intelligence Review* **53** 1767 – 1812. <https://api.semanticscholar.org/CorpusID:189763720> (2019)
 19. Floreano, D., Dürr, P., Mattiussi, C.: Neuroevolution: from architectures to learning, *Evolutionary Intelligence* **1** 47–62. <https://api.semanticscholar.org/CorpusID:2942634> (2008)
 20. Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y.L., Tan, J., Le, Q.V., Kurakin, A.: Large-scale evolution of image classifiers, in: *International conference on machine learning*, PMLR, pp. 2902–2911 (2017)
 21. Real, E., Aggarwal, A., Huang, Y., Le, Q.V.: Regularized evolution for image classifier architecture search. *Proceedings of the aaai conference on artificial intelligence* **33**, 4780–4789 (2019)
 22. Liu, H., Simonyan, K., Vinyals, O., Fernando, C., Kavukcuoglu, K.: Hierarchical representations for efficient architecture search, arXiv preprint [arXiv:1711.00436](https://arxiv.org/abs/1711.00436) (2017)
 23. Elsken, T., Metzen, J.H., Hutter, F.: Efficient multi-objective neural architecture search via lamarckian evolution, arXiv preprint [arXiv:1804.09081](https://arxiv.org/abs/1804.09081) (2018)
 24. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition, in: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778 (2016)
 25. Radosavovic, I., Kosaraju, R.P., Girshick, R.B., He, K., Dollár, P.: Designing network design spaces, 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) 10425–10433. <https://api.semanticscholar.org/CorpusID:214714446> (2020)
 26. Loshchilov, I., Hutter, F.: Sgdr: Stochastic gradient descent with warm restarts, arXiv preprint [arXiv:1608.03983](https://arxiv.org/abs/1608.03983) (2016)
 27. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition, in: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* **2016**, 770–778 (2016). <https://doi.org/10.1109/CVPR.2016.90>
 28. Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K.Q.: Densely connected convolutional networks, in: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* **2017**, 2261–2269 (2017). <https://doi.org/10.1109/CVPR.2017.243>
 29. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.-C.: Mobilenetv2: Inverted residuals and linear bottlenecks, in: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520 (2018)
 30. Ma, N., Zhang, X., Zheng, H.-T., Sun, J.: Shufflenet v2: Practical guidelines for efficient cnn architecture design, in: *Proceedings of the European conference on computer vision (ECCV)*, pp. 116–131 (2018)
 31. Han, K., Wang, Y., Tian, Q., Guo, J., Xu, C., Xu, C.: Ghostnet: More features from cheap operations, in: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* **2020**, 1577–1586 (2020). <https://doi.org/10.1109/CVPR42600.2020.00165>
 32. Zoph, B., Le, Q.V.: Neural architecture search with reinforcement learning, arXiv preprint [arXiv:1611.01578](https://arxiv.org/abs/1611.01578) (2016)
 33. Baker, B., Gupta, O., Naik, N., Raskar, R.: Designing neural network architectures using reinforcement learning, arXiv preprint [arXiv:1611.02167](https://arxiv.org/abs/1611.02167) (2016)
 34. Suganuma, M., Shirakawa, S., Nagao, T.: A genetic programming approach to designing convolutional neural network architectures, in: *Proceedings of the genetic and evolutionary computation conference*, pp. 497–504 (2017)
 35. Huang, J., Xue, B., Sun, Y., Zhang, M.: A flexible variable-length particle swarm optimization approach to convolutional neural network architecture design, in: *IEEE Congress on Evolutionary Computation (CEC)* **2021**, 934–941 (2021). <https://doi.org/10.1109/CEC45853.2021.9504716>
 36. Sun, Y., Xue, B., Zhang, M., Yen, G.G.: Completely automated cnn architecture design based on blocks. *IEEE Transactions on Neural Networks and Learning Systems* **31**(4), 1242–1254 (2020). <https://doi.org/10.1109/TNNLS.2019.2919608>
 37. Sun, Y., Xue, B., Zhang, M., Yen, G.G., Lv, J.: Automatically designing cnn architectures using the genetic algorithm for image classification. *IEEE transactions on cybernetics* **50**(9), 3840–3854 (2020)
 38. Lu, Z., Whalen, I., Boddeti, V., Dhebar, Y., Deb, K., Goodman, E., Banzhaf, W.: Nsga-net: neural architecture search using multi-objective genetic algorithm, in: *Proceedings of the genetic and evolutionary computation conference*, pp. 419–427 (2019)
 39. Cui, J., Chen, P., Li, R., Liu, S., Shen, X., Jia, J.: Fast and practical neural architecture search, in: *IEEE/CVF International Conference on Computer Vision (ICCV)* **2019**, 6508–6517 (2019). <https://doi.org/10.1109/ICCV.2019.00661>
 40. Zhong, Z., Yang, Z., Deng, B., Yan, J., Wu, W., Shao, J., Liu, C.-L.: Blockqnn: Efficient block-wise neural network architecture generation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **43**(7), 2314–2328 (2021). <https://doi.org/10.1109/TPAMI.2020.2969193>
 41. Zoph, B., Vasudevan, V., Shlens, J., Le, Q.V.: Learning transferable architectures for scalable image recognition, in: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8697–8710 (2018)
 42. Pham, H., Guan, M., Zoph, B., Le, Q., Dean, J.: Efficient neural architecture search via parameters sharing, in: *International conference on machine learning*, PMLR, pp. 4095–4104 (2018)
 43. Liu, H., Simonyan, K., Yang, Y.: Darts: Differentiable architecture search, arXiv preprint [arXiv:1806.09055](https://arxiv.org/abs/1806.09055) (2018)
 44. Luo, R., Tian, F., Qin, T., Chen, E., Liu, T.-Y.: Neural architecture optimization, *Advances in neural information processing systems* **31** (2018)
 45. Xie, S., Zheng, H., Liu, C., Lin, L.: Snas: stochastic neural architecture search, arXiv preprint [arXiv:1812.09926](https://arxiv.org/abs/1812.09926) (2018)
 46. Cai, H., Zhu, L., Han, S.: Proxylessnas: Direct neural architecture search on target task and hardware, arXiv preprint [arXiv:1812.00332](https://arxiv.org/abs/1812.00332) (2018)
 47. Xie, L., Yuille, A.: Genetic cnn, in: *Proceedings of the IEEE international conference on computer vision*, pp. 1379–1388 (2017)

48. Liu, H., Simonyan, K., Vinyals, O., Fernando, C., Kavukcuoglu, K.: Hierarchical representations for efficient architecture search, arXiv preprint [arXiv:1711.00436](https://arxiv.org/abs/1711.00436) (2017)
49. Wang, B., Xue, B., Zhang, M.: Surrogate-assisted particle swarm optimization for evolving variable-length transferable blocks for image classification. *IEEE Transactions on Neural Networks and Learning Systems* **33**(8), 3727–3740 (2022). <https://doi.org/10.1109/TNNLS.2021.3054400>
50. Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L.-J., Fei-Fei, L., Yuille, A., Huang, J., Murphy, K.: Progressive neural architecture search, in: *Proceedings of the European conference on computer vision (ECCV)*, pp. 19–34 (2018)
51. Zhang, H., Jin, Y., Cheng, R., Hao, K.: Efficient evolutionary search of attention convolutional networks via sampled training and node inheritance. *IEEE Transactions on Evolutionary Computation* **25**(2), 371–385 (2021). <https://doi.org/10.1109/TEVC.2020.3040272>
52. Lin, M., Wang, P., Sun, Z., Chen, H., Sun, X., Qian, Q., Li, H., Jin, R.: Zen-nas: A zero-shot nas for high-performance image recognition, in: *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 347–356 (2021)
53. Huang, J., Xue, B., Sun, Y., Zhang, M., Yen, G.G.: Particle swarm optimization for compact neural architecture search for image classification, *IEEE Transactions on Evolutionary Computation* **27**, 1298–1312. <https://api.semanticscholar.org/CorpusID:253316439> (2023)
54. Tan, M., Le, Q.: Efficientnet: Rethinking model scaling for convolutional neural networks, in: *International conference on machine learning*, PMLR, pp. 6105–6114 (2019)
55. Zhou, D., Zhou, X., Zhang, W., Loy, C.C., Yi, S., Zhang, X., Ouyang, W.: Econas: Finding proxies for economical neural architecture search, in: *Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition*, pp. 11396–11404 (2020)
56. Yang, Z., Wang, Y., Chen, X., Shi, B., Xu, C., Xu, C., Tian, Q., Xu, C.: Cars: Continuous evolution for efficient neural architecture search, in: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 1829–1838 (2020)
57. Xu, Y., Xie, L., Zhang, X., Chen, X., Qi, G.-J., Tian, Q., Xiong, H.: Pc-darts: Partial channel connections for memory-efficient architecture search, arXiv preprint [arXiv:1907.05737](https://arxiv.org/abs/1907.05737) (2019)
58. Zhang, M., Li, H., Pan, S., Chang, X., Su, S.: Overcoming multi-model forgetting in one-shot nas with diversity maximization, in: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 7809–7818 (2020)
59. Wan, A., Dai, X., Zhang, P., He, Z., Tian, Y., Xie, S., Wu, B., Yu, M., Xu, T., Chen, K.: et al., Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions, in: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 12965–12974 (2020)
60. Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., Le, Q.V.: Mnasnet: Platform-aware neural architecture search for mobile, in: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 2820–2828
61. Luo, R., Tan, X., Wang, R., Qin, T., Chen, E., Liu, T.-Y.: Semi-supervised neural architecture search. *Advances in Neural Information Processing Systems* **33**, 10547–10557 (2020)
62. Chen, W., Gong, X., Wang, Z.: Neural architecture search on imagenet in four gpu hours: A theoretically inspired perspective, arXiv preprint [arXiv:2102.11535](https://arxiv.org/abs/2102.11535) (2021)
63. Cai, H., Gan, C., Wang, T., Zhang, Z., Han, S.: Once-for-all: Train one network and specialize it for efficient deployment, arXiv preprint [arXiv:1908.09791](https://arxiv.org/abs/1908.09791) (2019)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.