Complex & Intelligent Systems (2023) 9:3211–3228
https://doi.org/10.1007/s40747-022-00919-y

ORIGINAL ARTICLE

Evolutionary convolutional neural network for image classification based on multi-objective genetic programming with leader–follower mechanism

Qingqing Liu[1,2] · Xianpeng Wang[3] · Yao Wang[2] · Xiangman Song[1]

Received: 24 July 2022 / Accepted: 6 November 2022 / Published online: 29 November 2022

Abstract

As a popular research in the field of artificial intelligence in the last 2 years, evolutionary neural architecture search (ENAS) compensates the disadvantage that the construction of convolutional neural network (CNN) relies heavily on the prior knowledge of designers. Since its inception, a great deal of researches have been devoted to improving its associated theories, giving rise to many related algorithms with pretty good results. Considering that there are still some limitations in the existing algorithms, such as the fixed depth or width of the network, the pursuit of accuracy at the expense of computational resources, and the tendency to fall into local optimization. In this article, a multi-objective genetic programming algorithm with a leader–follower evolution mechanism (LF-MOGP) is proposed, where a flexible encoding strategy with variable length and width based on Cartesian genetic programming is designed to represent the topology of CNNs. Furthermore, the leader–follower evolution mechanism is proposed to guide the evolution of the algorithm, with the external archive set composed of non-dominated solutions acting as the leader and an elite population updated followed by the external archive acting as the follower. Which increases the speed of population convergence, guarantees the diversity of individuals, and greatly reduces the computational resources. The proposed LF-MOGP algorithm is evaluated on eight widely used image classification tasks and a real industrial task. Experimental results show that the proposed LF-MOGP is comparative with or even superior to 35 existing algorithms (including some state-of-the-art algorithms) in terms of classification error and number of parameters.

Keywords Evolutionary neural architecture search · Multi-objective genetic programming · Leader–follower mechanism

Introduction

Convolutional neural network (CNN) is a typical representative of neural network architectures. Due to its powerful feature extraction capability, it has shown excellent advantages on a variety of competing vision-related tasks, including images classification [1,2], text detection[3], industrial data analysis [4], etc. In the past few years, researchers have conducted a variety of interesting studies on CNNs, including the design of architectures (depth, width, etc.) and the enhancement of learning capabilities (feature extraction and exploitation, propagation of loss functions, etc.). Representatives of CNNs include AlexNet [5], GoogLeNet [6], VGG [7], ResNet [8], DenseNet [9], etc., all of which have achieved quite remarkable results.

Although these CNNs have been designed with great success, the construction of CNN architectures is by no means an easy task. For machine learning scholars, CNNs are like a finely crafted work of art that requires constant tuning to opti-

✉ Xianpeng Wang
wangxianpeng@ise.neu.edu.cn

Qingqing Liu
liuqingqing@stumail.neu.edu.cn

Yao Wang
yaowang@stumail.neu.edu.cn

Xiangman Song
songxiangman@ise.neu.edu.cn

1 National Frontiers Science Center for Industrial Intelligence and Systems Optimization, Northeastern University, Shenyang 110819, China

2 Liaoning Engineering Laboratory of Data Analytics and Optimization for Smart Industry, Shenyang 110819, China

3 Key Laboratory of Data Analytics and Optimization for Smart Industry (Northeastern University), Ministry of Education, Shenyang 110819, China

mize the mechanisms and parameters of the network until a good combination, a suitable regular planner, and optimized parameters are found. This process is inseparable from a wealth of prior knowledge and a huge amount of work, and many scholars have put considerable efforts just to design a satisfactory network architecture [10].

Fortunately, the potential of evolutionary algorithm (EA) in neural architecture searching (ENAS) has attracted considerable attention in the last few years and has yielded very promising results [11,12]. Systematic reviews of ENAS can be found in [13,14], where several representative algorithms are introduced, such as Genetic CNN and Evo-CNN (CNN architectures searching based on genetic algorithm) [15,16], EAS and Meta-QNN (CNN architectures searching based on Q-learning) [17,18], Large-scale Evolution [19], CGP-CNN (CNN architectures searching based on Cartesian genetic programming) [20], NAS (CNN architectures searching based on reinforcement learning) [21], and CNN-GA and AE-CNN (CNN architectures searching based on genetic algorithm and block encoding) [22,23], etc. Experimental results of these methods have shown that pretty good performance in searching for the optimal network structures and excellent results have been achieved. But there are still some important limitations that need to be addressed. First, some of these algorithms still require considerable expertise, e.g., the EAS algorithm is based on a primary network, while the selection of the primary network still requires considerable empirical knowledge. Second, the performance of some ENAS algorithms is highly dependent on computational resources, e.g., it takes 28 days to train NAS on CIFAR10 even with 800 GPUs. Third, some of these methods, such as genetic CNNs, employ the encoding strategy with a fixed length or a fixed width, which means that the depth or width of CNNs is fixed. Since the performance of CNNs depends heavily on their depth and width, it is desirable that the architecture of CNNs could be flexible and versatile to ensure good generalization ability to different tasks.

The purpose of this paper is to design an algorithm that can autonomously evolve neural networks for the task of image classification, which can overcome the limitations of the existing methods described above. In view of the excellent performance of genetic programming in many practical applications, the idea of the binary diagram of Cartesian genetic programming (CGP) [24] is introduced into this paper to encode the structure of CNN architectures. The main contributions of this paper are as follows:

(1) A flexible coding strategy of variable length and width is proposed based on CGP, and 22 alternative function blocks are designed. Where the tree-like structure of CGP with fewer structural constraints can represent the topology of CNNs well. Meanwhile, the efficient alternative function blocks can make the algorithm more efficient and can expand the search space, which in turn provides more possibilities for finding more high-quality architectures.
(2) A multi-objective genetic programming with a leader–follower mechanism is designed, where the external archive of non-dominated solutions and the elite population act as leader and follower, respectively. This mechanism can speed up the convergence while preventing the algorithm from getting trapped in a local optimum to a large extent.
(3) The effectiveness of the proposed algorithm is validated on eight benchmark datasets that are widely adopted in images classification tasks and a real-world industrial dataset. Computational results illustrate the superior performance of the proposed algorithm over various state-of-the-art algorithms.

The rest of this paper is organized as follows. The next section presents an introduction of the background and related works on the design of CNNs. Then the details of the proposed LF-MOGP are presented in the subsequent section followed by which the experiment design is introduced to evaluate the performance of the proposed algorithm. Then the results and analysis of the experiments are given. The penultimate section presents a case of LF-MOGP applied to a real-world industrial problem of slab number recognition. Finally, the conclusion and future works are presented.

Background

Cartesian genetic programming algorithm

As an evolutionary computation technique, genetic programming (GP) is capable of automatically evolving models to solve real-world problems based on the principles of evolution and natural selection in the biological world [25], which is well-known for its flexibility and high interpretability compared to other evolutionary algorithms and is widely adopted in image classification, scheduling and regression tasks [26]. As a highly representative branch of GP, CGP can flexibly encode various computing structures while avoiding the bloat problem in GP [27–29]. Which is capable of achieving high robustness and generalization due to its remarkable self-organization, self-learning, and self-adaptive properties, and is more effective than other GP algorithms in many complex problems such as parameters optimization, scheduling, resource allocation, and complex network analysis [30,31].

The general form of CGP is shown in Fig. 1, which is represented by a directed graph with index nodes. In this form, there are n inputs and m outputs, and the output is obtained from the nodes of the last column. The size of the

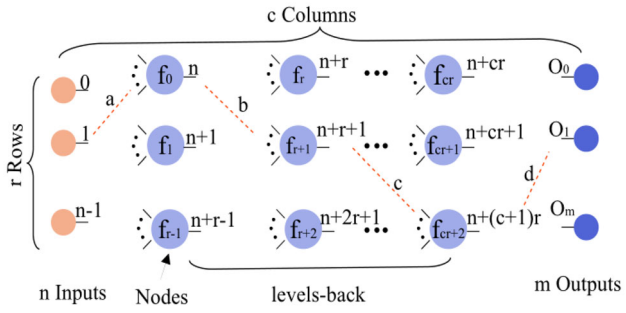

Fig. 1 General form of Cartesian-GP

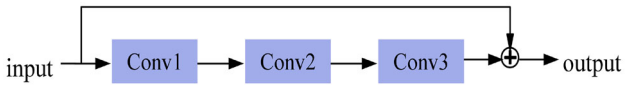

Fig. 2 Examples of ResNet

directed graph shown in Fig. 1 is $n \times c$. The nodes of the same column cannot be connected to each other, and the connection between columns is also restricted by the level-back (e.g. level-back = 3 means that the nodes in ith column can connect to column $i - 3$ at most). The red dashed path 'a-b-c-d' represents a simple neural network with five nodes.

ResNet block

The creation of ResNet is a landmark event in the development of CNNs, which has made extraordinary contributions to mitigating gradient loss and utilizing features. The success of ResNet is mainly attributed to the design of its building blocks, especially the shortcut connections. During forward propagation, the presence of the hopping structure allows the input signal to propagate directly from any lower level to the higher level, and the loss will not be attenuated by any intermediate weight matrix. For the hopping structure, when the input and output dimensions are the same, the corresponding dimensions can be added directly and without any other operation. But if the two dimensions are inconsistent, the smaller dimension will be expanded by 0-padding or 1×1 convolution to make the input and output dimensions consistent. Figure 2 shows a typical example of ResNet consisting of three convolution layers and a skip connection, where the line chart shows the jump propagation of loss values.

Proposed algorithm

Algorithm overview with leader-and-follower mechanism

Algorithm 1 Framework of LF-MOGP

Input: Population size N, elite population size K, maximum generations T;
Output: Ensemble model EM^*;
$t \leftarrow 0$;
$\mathbb{E} = \emptyset, \mathbb{F}_t = \emptyset$; // Set the external archive $\mathbb{E}$ and the elite population $\mathbb{F}_t$ to be empty
$\mathbb{P} \leftarrow$ Initialize a population by Algorithm 2;
Evaluate each solution in $\mathbb{P}$ for accuracy and complexity;
Sort solutions in $\mathbb{P}$ by the non-dominated sorting method;
$\mathbb{E} \leftarrow$ Put all non-dominated solutions in $\mathbb{P}$ into $\mathbb{E}$;
$\mathbb{F}_t \leftarrow$ Select K elite individuals from the rest dominated solutions in $\mathbb{P}$ into $\mathbb{F}_t$ by crowding distance;
while $t < T$ **do**
  $\xi \leftarrow$Uniformly generate a number from [0,1];
  if $\xi > t/(t+T)$ **then**
    $\{p_1, p_2\} \leftarrow$ Randomly select two different solutions from $\mathbb{E}$;
    $\{p_1', p_2'\} \leftarrow$ **MUTATION**(p_1), **MUTATION**(p_2);
    $\{q_1, q_2\} \leftarrow$ Randomly select two different solutions from $\mathbb{E}$;
    $\{q_1', q_2'\} \leftarrow$ **CROSSOVER**(q_1, q_2);
  end
  else
    $\{p_1, p_2\} \leftarrow$ Randomly select p_1 from $\mathbb{E}$ and p_2 from $\mathbb{F}_t$;
    $\{p_1', p2'\} \leftarrow$**MUTATION**(p_1), **MUTATION**(p_2);
    $\{q_1, q_2\} \leftarrow$ Randomly select q_1 from $\mathbb{E}$ and q_2 from $\mathbb{F}_t$;
    $\{q_1', q_2'\} \leftarrow$ **CROSSOVER**(q_1, q_2);
  end
  $\widetilde{E} = \mathbb{E} \bigcup \{p_1', p_2'\} \bigcup \{q_1', q_2'\}$;
  $\mathbb{E} \leftarrow$ Update $\mathbb{E}$ with $\widetilde{E}$ by non-dominated sorting;
  $\widetilde{F} = \mathbb{F}_t \bigcup \{p_1', p_2'\} \bigcup \{q_1', q_2'\} \bigcup \widetilde{E} \setminus \mathbb{E}$;
  $\mathbb{F}_t \leftarrow$ Select K elite individuals from the solutions in $\widetilde{F}$ by non-dominated sorting and crowding distance;
  $t \leftarrow t + 1$;
end

The framework of the proposed LF-MOGP is shown in Algorithm 1, which consists of the following steps. In the beginning, the items such as $\mathbb{E}, \mathbb{F}_t, \mathbb{P}$ are initialized, where the initialization of the population based on the basic functional blocks is explained in detail in Algorithm 2, and the basic blocks designed in our algorithm are described in Sect. 3.2.

Then, the CNN corresponding to each solution in the population is constructed and evaluated separately. Here two evaluation metrics are introduced, the maximization of classification accuracy (Acc) and the minimization of complexity of the model, their detailed descriptions can be found in Sect. 3.6.

After that, the comparisons of the non-dominated relationship of all the solutions are performed, and then the leader–follower mechanism comes into play, the flow diagram of the leader–follower mechanism is shown in Fig. 3 (the red line represents the leader $\mathbb{E}$ and the yellow circle represents the follower $\mathbb{F}_t$, and the black arrow represents the update of the solution set). Specifically, the non-dominated solutions are stored in $\mathbb{E}$, which will act as the leader during evolution. For the rest solutions in $\mathbb{P}$, K elite solutions will be selected based on the crowding distance and stored in $\mathbb{F}_t$, which will act as the follower during evolution. In the early stage of the algorithm, the non-dominated solutions in external archive $\mathbb{E}$ will act as the leader. That is, the parent solutions are mainly selected from $\mathbb{E}$, which helps to achieve a fast convergence speed. While in the later stage of the algorithm, the parent solutions are selected from both the leader $\mathbb{E}$ and follower $\mathbb{F}_t$, which can improve the search diversity thus preventing the algorithm from falling into local optimum. In the main loop of the LF-MOGP algorithm, the leader and the follower will be updated iteratively and will collaborate to achieve optimization of the relevant metrics with higher accuracy and lower model complexity, as shown in the third graph of Fig. 3, migrating to the upper right corner.

For the generation of new solutions, based on the characteristics of CGP encoding, we design both mutation and crossover operators. Obviously, the leader–follower mechanism allows the evolutionary process to focus on non-dominated solutions rather than the whole population at the early stage, which can greatly reduce the computational resource requirements and also speed up the convergence rate. The new solutions generated are also used to update the solutions in $\mathbb{F}_t$, thus allowing $\mathbb{F}_t$ to follow the changes of $\mathbb{E}$, which can improve the individuals' diversity in the later stage of the algorithm. That is, the leader–follower mechanism is able to achieve a better balance between exploration and exploitation while reducing computational resources.

Encoding and decoding strategy

Since the optimal structure of a CNN (including width and depth) is unknown when dealing with a specific problem, the encoding strategy employed must satisfy the variability of the depth and width of the CNN so that the proposed LF-MOGP algorithm has the opportunity to find the optimal structure of the CNN without the limitation of the search space. In this regard, CGP has exactly such flexibility, and each node function in CGP can be easily replaced with an efficient function block, which makes CGP a good representation of CNN structure. In addition, the CGP-based encoding strategy has fewer restrictions on crossover and variation operations. It does not have the traditional restrictions on the length or width of the parent individuals by the crossover and variation operators, which can further expand the search space and provide the possibility of finding the optimal structure of CNNs.

As mentioned above each node in CGP is replaced with an efficient function block. To increase the efficiency of the algorithm, the ResNet block mentioned in Sect. 2.2 is introduced as a basic block. Besides, another four types of functional blocks, namely, ConvBlock, Pooling, Concat, and Sum are also designed. These blocks contain several sub-blocks depending on their internal parameter settings. Table 1 shows the specific design of parameters in each functional block and its corresponding sub-blocks.

These functional blocks follow the following naming rules. Taking ConvBlocks as an example, their names are changed from C1 to C9, and the number and size of convolution kernels are increased accordingly. That is, these 9 ConvBlocks CB_32_1, CB_32_3, CB_32_5, CB_64_1, CB_64_3, CB_64_5, CB_128_1, CB_128_3, CB_128_5 are denoted as C1, C2,..., C9 in that order. ResNet Blocks are also named in a similar way. The average and max pooling are denoted by P1 and P2, respectively. The names of the other function blocks are consistent with their symbols.

ConvBlock is used for feature extraction. The parameters of the convolution are as follows. The kernel size is chosen from $\{1 \times 1, 3 \times 3, 5 \times 5\}$, the step is set to 1, and the input will be padded with 0 before the convolution operation. To alleviate the gradient dispersion during training, a batch normalization is performed after each convolution.

ResNet Block is processed by the standard convolution. The size of the convolution kernel is selected from $\{1 \times 1, 3 \times 3, 5 \times 5\}$, padding for half of the size of the convolution kernel size. After convolution, the batch normalization and ReLU function will be adopted. The specific form of ResNet is shown in Fig. 2 above.

Pooling includes the maximum one and the average one. The size of the filter is set to 2×2, and the step is set to 2. Since the pooling process is equivalent to downscaling the features, the number of pooling layers is bounded by the size of the input image of $d \times d$, and the maximum number of the adopted pooling block is $log_2 d$.

Concat is designed to merge the feature maps at the channel level. If the two feature maps to be concatenated have

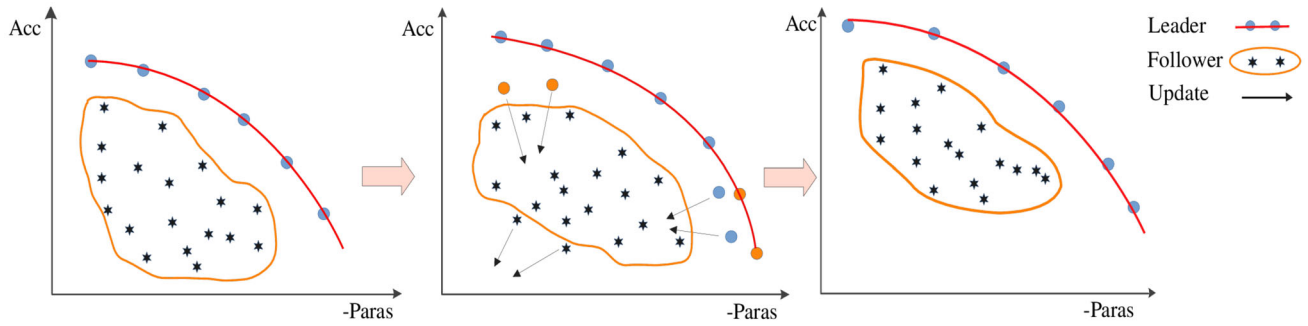

Fig. 3 Illustrative diagram of the leader-and-follower mechanism

Table 1 Detailed settings of the alternate blocks

Type symbol	ConvBlock CB(c, k)	ResNet Block RB(c, k)	Pooling AP	Pooling MP	Concat Concat	Sum Sum
Variations	$c \in \{32, 64, 128\}$ $k \in \{1, 3, 5\}$	$c \in \{32, 64, 128\}$ $k \in \{1, 3, 5\}$	k=2		–	–
Name	C1, C2, ... ,C9	R1, R2, ..., R9	AP	MP	Concat	Sum

the same number of rows and columns, they will be merged directly at the channel level, otherwise, we will down-sample the one with a larger feature map by the maximum pooling to make the two features have the same size. Then the final feature map $\mathcal{F}$ can be represented as follows:

$$\mathcal{F} = \min(M_1, M_2) \times \min(N_1, N_2) \times (C_1 + C_2), \quad (1)$$

where the $M_1 \times N_1 \times C_1$ and $M_2 \times N_2 \times C_2$ are two dimensions of the input images.

Sum is designed to merge the feature maps at the pixel level. Similar to the Concat block, if the two feature maps to be summed are in different numbers of rows or columns, we will down-sample the one with a larger feature map by the maximum pooling to make the two features have the same size. In addition, if the two features to be summed have different channels, the one with the smaller number of channels will be expanded with a 1×1 convolution operator to make them have the same dimension. The output of the feature map $\mathcal{F}$ can be represented as follows:

$$\mathcal{F} = \min(M_1, M_2) \times \min(N_1, N_2) \times \max(C_1, C_2) \quad (2)$$

where the $M_1 \times N_1 \times C_1$ and $M_2 \times N_2 \times C_2$ are two dimensions of the input images.

Figure 4 illustrates the procedure of how to decode the genotype of a solution into its corresponding CNN architecture. As shown in Fig. 4, a solution consists of three items, $node\_id$, is_active and $gene$. Specifically, the gene of the first node is $(C4, 0, 0)$, and according to the naming rules mentioned earlier, $C4$ indicates that Convblock is selected, the number of convolution kernels is 64, and the size of the convolution kernel is 1×1. The blocks and their specific symbols can be referenced in Table 1. With respect to the pooling block in the dotted box in the corresponding CNN architecture below, it does not actually work here because the sixth value of is_active is $False$.

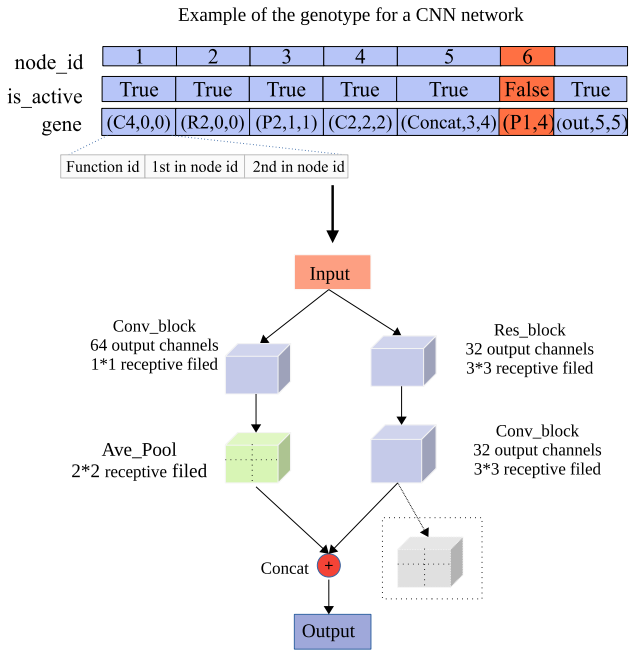

Fig. 4 Illustrative diagram of encoding and decoding strategy

Population initialization

The initialization procedure of the population is illustrated in Algorithm 2, where each solution is produced according to the employed encoding and decoding strategy.

Algorithm 2 Population Initialization

Input: Population size N, size of the training instance $d \times d$, minimum nodes N_l, maximum nodes N_u;
Output: Initialized population $\mathbb{P}$;
, $\mathbb{P} \leftarrow \emptyset$;
$n_p \leftarrow \log_2 d$; `// Get the maximum number of pooling layers`
for $i = 1, \cdots, N$ **do**
  $n_{pi} \leftarrow 0$;
  $k \leftarrow$ Uniformly generate the number of nodes (depth of the network) in $(N_l, N_u]$;
  $gene \leftarrow$ Initialize an empty array of size $(k, 3)$ to store the block type and connection IDs;
  $is\_active \leftarrow$ Initialize a boolean array of size k to store the activation type of the nodes;
  for $j = 1, \cdots, k$ **do**
    $b \leftarrow$ Randomly choose a block from the alternative blocks;
    if b *is* POOLING BLOCK **then**
    end
    $n_{pi} \leftarrow n_{pi} + 1$;
    if $n_{pi} > n_p$ **then**
      $b \leftarrow$ Reselect one block from remaining blocks;
    end
    $(c_m, c_n) \leftarrow$ Randomly generate connection ID from [0, j) for c_m and c_n respectively;
    $gene_j \leftarrow$ Determine the j_{th} value of $gene$ ($gene_j$) based on b and (c_m, c_n);
  end
  $S_i \leftarrow$ Get the solution by combining $gene$ and is_ative;
  $\mathbb{P} \leftarrow \mathbb{P} \cup S_i$;
end
Return $\mathbb{P}$

The generation of a solution consists of the following steps. First, the parameters and an empty solution are initialized. The next step is to select the type and connections for each block of the solution. Notice that, since the pooling is a downsampling operation, so the number of the adopted pooling blocks must be less than $\log_2 d$ [20], otherwise, b must be re-selected from the rest of the blocks except the pooling ones. Furthermore, each block can only be connected to the nodes whose position precedes it, that is, both c_m and c_n must be less than j. When all the genotypes of the k nodes are determined, they will be encoded into the corresponding blocks according to the encoding strategy described in Sect. 3.2. Finally, the solution S_i corresponding to a CNN network is obtained by combining the $gene$ and is_active.

Mutation

The mutation operation proposed in our method includes the following three types, $Adding$, $Removing$, and $Modifying$, respectively. The detailed process of the mutation operation is presented in Algorithm 3. The process of mutation operation consists of the following steps. First, the number of nodes in the parent individual p is counted and one position is randomly selected for the mutation. Second, a mutation type is randomly selected from the three alternative types, and then the mutation operation is performed. Please note that this mutation process may result in a dimension mismatch in the mutated offspring. For this possible case, a dimension matching discriminant and restoration module is proposed to determine if there is a mismatch in the dimension or the image size in the mutated offspring. If so, the 1×1 convolution will be adopted to make the two dimensions consistent, and the maxpool operation will be used to downsample the large-size input so that the two inputs have the same size. Finally, the mutated offspring q is obtained and returned.

Algorithm 3 Mutation Operation of LF-MOGP

Input: Parent solution p;
Output: Mutated offspring q;
$L \leftarrow$ Count the number of nodes in p;
$p_m \leftarrow$ Randomly choose a position to mutate from (0-L);
type $\leftarrow$ Randomly choose a mutation type from {Adding, Removing, Modifying};
if *type is* $Adding$ **then**
  Perform $Adding$: Randomly select a block and insert it into position p_m
end
else if *type is* $Removing$ **then**
  Perform $Removing$: Remove the block at position p_m
end
else if *type is* $Modifying$ **then**
  Perform $Modifying$: Modify the associated parameters of the block at position p_m
end
$q \leftarrow$ Evaluate and repair dimensions of p;
Return q

Figure 5 illustrates the implementation of the mutation process, where the third mutation strategy $Modifying$ includes two methods, which are modifying connection nodes of block and modifying the configuration of block (size or number of convolutional kernels, etc.)

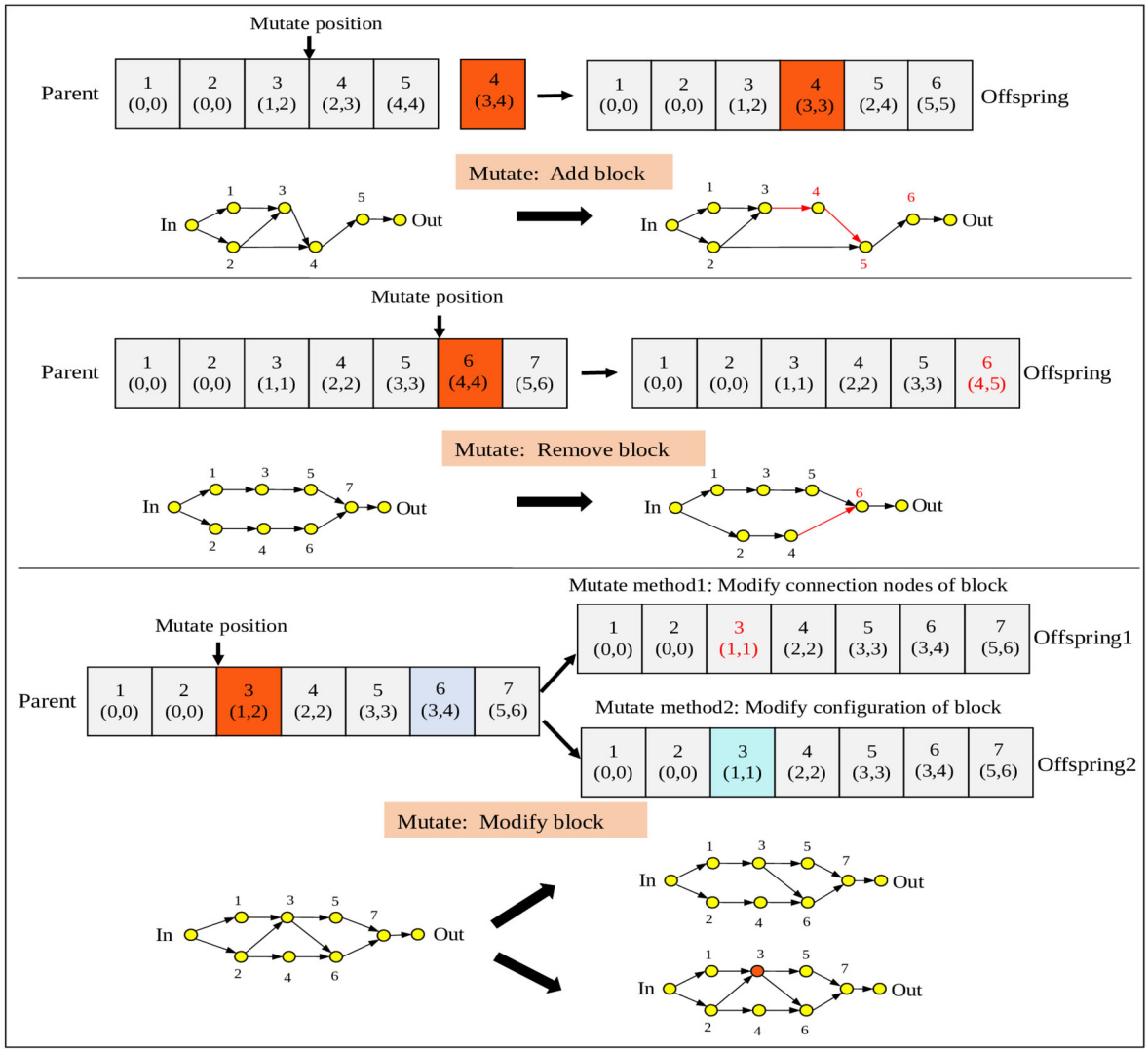

Fig. 5 Illustration of the mutation process implementation

Crossover

Unlike the traditional crossover operation that requires two parent individuals to have the same length, in our algorithm, the two parent individuals can have different lengths. Fewer constraints here, so a larger search space is gained. The detailed procedure of this crossover operation is presented in Algorithm 4.

Specifically, the crossover operation consists of the following steps. First, based on the two random crossover positions the single-point crossover operation is performed. After restructuring, new offsprings are preliminary generated. A simple example to illustrate the crossover process is presented in Fig. 6. Next, a dimension matching discriminant is carried out, and if necessary, their dimensions are repaired in a similar way like the mutation operator. Finally, the offspring solutions generated by the crossover operation are obtained.

Algorithm 4 Crossover Operation of LF-MOGP

Input: genotype of parents P_1, P_2;
Output: Crossed offspring Q_1, Q_2;
pos_1, pos_2 ← Randomly choose two crossover positions of P_1, P_2;
P_{11}, P_{12} ← Separate genotype of P_1 at position pos_1;
P_{21}, P_{22} ← Separate genotype of P_2 at position pos_2;
Q_1 ← Combine genotypes P_{11}, P_{22};
Q_2 ← Combine genotypes P_{21}, P_{12};
Q_1, Q_2 ← Evaluate and repair dimensions of the new offspring;
Return Q_1, Q_2

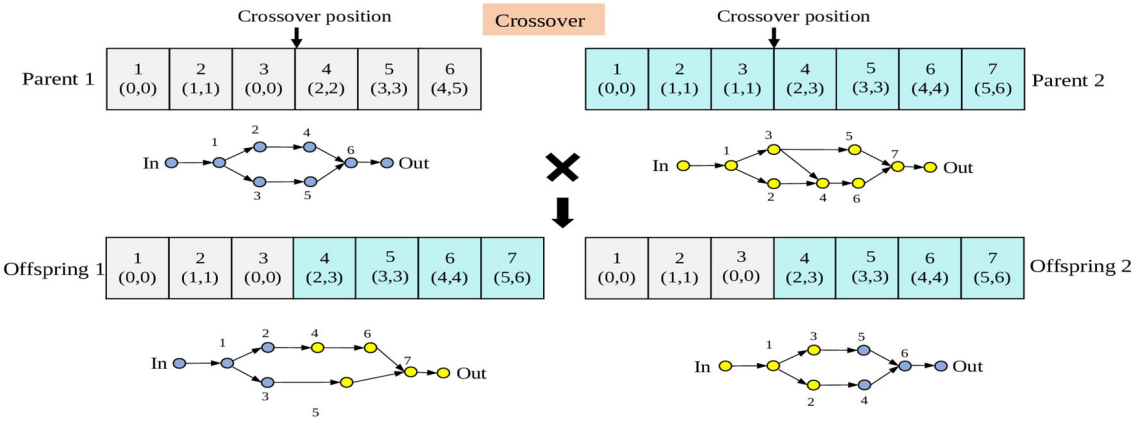

Fig. 6 Illustration of the crossover process implementation

Fitness evaluation

The evaluation of a solution consists of two conflicting objectives: the classification accuracy (Acc) and the complexity of the corresponding CNN. The classification accuracy is calculated by the ratio of the number of incorrectly classified images to the total number of images, which is widely adopted in image classification tasks. The complexity is calculated by the number of trainable parameters of the corresponding CNN, as done by Sun et al. [22]. The purpose of choosing these two metrics is to obtain CNN architectures with high accuracy but low complexity.

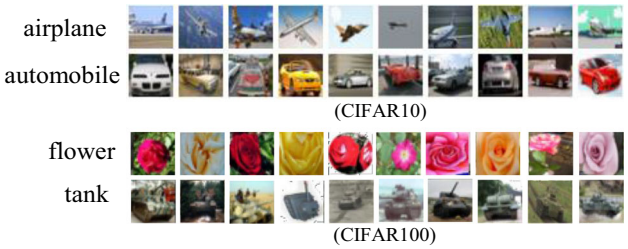

Fig. 7 Examples for benchmarks dataset of CIFAR10 and CIFAR100

Fig. 8 Examples for benchmarks datasets of Fashion

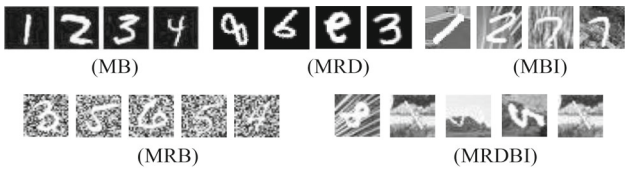

Fig. 9 Examples for benchmarks datasets of MNIST and its variants

Experiment design

Benchmark datasets

In the experiments, eight benchmark datasets that are widely used for images classification tasks are adopted to evaluate the performance of the proposed LF-MOGP. These benchmark datasets include CIFAR10, CIFAR100, Fashion, MB, MRI, MRB, MRD, and MRDBI. CIFAR10 and CIFAR100 cover the colorful images of objects such as cars and boats. The difference between them is that CIFAR100 covers 100 categories of objects, and each image of CIFAR100 contains a fine label except for super label. Fashion covers some fashion objects such as coats and shirts, and images covered in Fashion are grayscale. MNIST and its variants are established for the classification of ten hand-written digits (i.e., 0-9). The variants MRI, MRB, MRD, and MRDBI introduce different obstacles to MB (e.g., rotation, random noise, background images), which significantly increase the complexity of the classification tasks. The examples of these benchmark datasets are shown in Figs. 7, 8, and 9, respectively. A more detailed description of these datasets is provided in Table 2.

Experimental setting

The LF-MOGP algorithm is implemented in Pytorch, and all the experiments are carried out on a personal computer with one GeForce RTX 3090 GPU, Intel(R) Xeon(R) Silver 4110 CPU, and 32 GB RAM. The relevant details of the experiments can be described as follows. The maximum of the rows, columns, level_back, the minimum, and maximum of active nodes allowed in the neural networks are set to 5, 30, 10, 7, 30, respectively, which are determined by preliminary experimental experiences. Additionally, 22 alternative basic blocks are designed, which are shown in Table 1. The population size of $\mathbb{P}$ is set to 30, and the maximum of generations for population evolution is set to 100. The maximum of the

Table 2 Summary of benchmark datasets used in the experiments

Dataset	Description	Number of categories	Train-test($\times$1e3)
CIFAR10	Common objects	10	50–10
CIFAR100	Common objects	100	50–10
Fashion	Fashion objects	10	50–20
MB	MNIST with basic digits	10	12–50
MBI	MNIST with background image	10	12–50
MRB	MNIST with random background	10	12–50
MRD	MNIST with rotated digits	10	12–50
MRDBI	MNIST with rotated digits and background image	10	12-50

training epoch is set to 100. The learning rate is initially set to 1e-3 and is adjusted by cosine with an adjustment period of 80. The Adaptive Gradient method (Adam), where the beta is set to (0.9, 0.999), is chosen as the optimizer.

Experiment results

Overall results

To verify the effectiveness of the proposed LF-MOGP algorithm, a series of comparison experiments with 36 powerful competitor algorithms including the state-of-the-art ones are conducted.

Since the re-implementation of the competitor algorithms may not be able to achieve the same performance reported in the original papers. To make a fair comparison, for each dataset in our experiment, we collected the experimental results of these algorithms from the original papers. Furthermore, the competitor algorithms were often experimented with different datasets. Therefore, different competitor algorithms may be chosen for different datasets. More specifically, the best classification performance results of the proposed LF-MOGP and its competitors are shown in Tables 3, 4, and 5, which correspond to the MNIST and its variant datasets, Fashion, and CIFAR, respectively. Note that all the results on the competitors given in the tables are reported in their papers, where the symbol '–' implies that there is no public recorded result by the corresponding peer competitors.

In Table 5, not only the classification error but also the number of trainable parameters and 'GPU Days' are investigated to evaluate both the accuracy and complexity. Note that 'GPU Days' is just a reference indicator of computational consumption because the performance of different GPUs is different, and the specific experimental environment of each algorithm can be seen in the last row of Table 5. Specifically, if the algorithm employs 3 GPUs and runs for 7 days, then the corresponding 'GPU Days' will be 21. In contrast, the classification errors and training epochs are given in Table 4, while Table 3 only gives the classification errors due to fact that the competitors only reported the best classification errors of their algorithms and do not report other relevant results.

A. MNIST and its variants

As shown in Table 3, regarding the best classification performance, LF-MOGP can outperform all the compared competitors on the MNIST and its variants, except for the third-best performance on the MRD dataset. More specifically, the best classification error for MB, MRB, MBI and MRDBI obtained by the ENAS algorithms, are 0.79%, 2.44%, 4.06% and 17.92%, respectively. However, our LF-MOGP can further reduce the best classification error to 0.52%, 2.41%, 3.08% and 14.98%. Especially for the MB dataset, our LF-MOGP achieves a classification accuracy of nearly 99.5%. Moreover, for the MRDBI dataset whose difficulty of the classification is highest, the best performance among the compared algorithms is 17.92% obtained by SEECNN, but our LF-MOGP reduces the classification error to 14.98%, which demonstrates the advantage of LF-MOGP in dealing with complex classification tasks. When compared with the two GP-based algorithms(IEGP and FGP), the proposed LF-MOGP also demonstrates an absolute advantage, with significantly lower classification errors than both algorithms on all queryable datasets.

B. Fashion

Nine peer competitors including the four methods (2C1P2F, 2C1P, 3C2F, 3C1P2F+Dropout) collected from the website (https://github.com/zalandoresearch/fashion-mnist) of the Fashion dataset are adopted here to evaluate the performance of LF-MOGP, and the statistical results are shown in Table 4. As shown in Table 4,

LF-MOGP obtained the second-best classification error, and among all the competitors, the lowest classification error is 3.09% obtained by Fine-Tuning DARTS. However, it is

Table 3 The best classification error rates of LF-MOGP and its competitors on MNIST and its variants

Method	Model	MB	MRD	MRB	MBI	MRDBI
Handcrafted	CAE-2 [32]	2.48	9.66	10.90	15.50	45.23
	TIRBM [33]	–	**4.20**	–	–	35.5
	PGBM+DN-1 [34]	–	–	6.08	12.25	36.76
	RandNet-2 [35]	1.25	8.47	13.47	11.65	43.69
	PCANet-2 [35]	1.06	7.37	6.19	10.95	35.86
	LDANet-2 [35]	1.05	7.52	6.81	12.42	38.54
	SVM+RBF [36]	3.03	10.38	14.58	22.61	32.62
	SVM+Poly [36]	3.69	13.61	16.62	24.01	37.59
	NNet [36]	4.69	17.62	20.04	27.41	42.17
	SAA-3 [36]	3.46	11.43	11.28	23.00	24.09
	DBN-3 [36]	3.11	12.30	6.73	16.31	28.51
ENAS	IPPSO [37]	1.13	–	–	–	33
	SEECNN [38]	0.79	4.33	2.44	4.06	17.92
	EvoCNN [16]	1.18	5.22	2.8	4.53	35.30
	IEGP [39]	1.18	5.72	6.41	10.59	–
	FGP [40]	1.18	7.37	6.54	7.48	–
	LF-MOGP (ours)	**0.52**	5.45	**2.41**	**3.08**	**14.98**

worth pointing out that Fine-Tuning DARTS is a handcrafted model, where the cutout and random erasing data augmentation techniques were adopted during training. Besides, Fine-Tuning DARTS is a DARTS-based fine tuning algorithm, while the proposed LF-MOGP does not use any additional data augmentation techniques and is trained from scratch instead of fine tuning. Except for the Fine-Tuning DARTS, LF-MOGP reduces the best two classification errors by 2.52% and 2.72%, respectively, compared to VGG16 and GoogleNet. Compared to the three ENAS algorithms (EvoCNN, SEECNN and FPSO), LF-MOGP achieves the highest classification accuracy without significantly increasing parameters. Compared with FPSO, the number of parameters of our model increases by 0.12M, but the accuracy of our model is higher. Besides, LF-MOGP reduces the classification error by 1.68% compared to EvoCNN, while the number of parameters is reduced by 1.24M, which is very promising, and the classification error is reduced by 1.6% compared to SEECNN.

C. CIFAR10 and CIFAR100

The comparison results of LF-MOGP against the competitors are presented in Table 5. For CIFAR10, the classification errors of LF-MOGP are lower than that of all the handcrafted models. Besides, the best CNN evolved by LF-MOGP has a smaller number of parameters, which demonstrates the significant superiority of the proposed algorithm over the handcrafted models. In comparison to the ENAS methods on CIFAR10, the proposed LF-MOGP is not inferior, except that the classification error of LF-MOGP is 0.72% higher than that of PNAS, while the running time is only 4% of PNAS.

Regarding the GPU Days and parameters, the best results of the ENAS algorithm are 1.65 and 0.7M obtained by FPSO. However, its accuracy is not very promising, due to the fact that its classification error is 2.15% higher than that of our LF-MOGP.

In addition to PNAS, the top five ENAS algorithms with the minimal classification errors are CNN-GA, Genetic CNN, Large-scale Evolution, EvoCNN, NATS-Bench, and their classification errors are 4.78%, 5.01%, 5.40%, 5.47%, 5.63%, respectively. Compared with them, the proposed LF-MOGP can further reduce the classification error by 0.65%, 0.88%, 1.27%, 1.34% and 1.50%, respectively. The CNN evolved by LF-MOGP is more lightweight with only 1.07M parameters, which is conducive to extending the algorithm to practical applications. Moreover, LF-MOGP also has a significant advantage in terms of 'GPU Days'. Compared with the above ENAS algorithms, the proposed LF-MOGP algorithm takes the shortest running time to search for the optimal CNN architecture except for FPSO.

For CIFAR100, the best error obtained by LF-MOGP is about 26.37%. Compared with the Handcrafted algorithms, LF-MOGP can outperform all of them except DenseNet. More specifically, although the classification error of LF-

Table 4 The best classification error rates, parameters and train epoch of LF-MOGP and its competitors on Fashion dataset

Method	Model	Error Rates	Parameters	Epoch
Handcrafted	2C1P2F	8.40	3.27M	300
	2C1P	7.50	**100K**	30
	3C2F	9.30	–	–
	3C1P2F+Dropout	7.40	–	150
	GoogleNet [6]	6.30	101M	–
	AlexNet [5]	10.10	60M	–
	VGG16 [7]	6.50	26M	200
Fine-Tuning DARTS [41]†‖	**3.09**	3.2M	–	
ENAS	SEECNN [38]	5.38	–	–
	EvoCNN [16]	5.47	3.68M	100
	FPSO [42]	4.93	2.32M	–
	LF-MOGP (ours)	3.78	2.44M	100

† Best classification error for handcrafted neural network models available on Fashion dataset, which uses the cutout and random erasing data augmentation techniques during training

MOGP is a little higher than that of DenseNet, the number of parameters of the model obtained by our LF-MOGP is only one-third of that obtained by DenseNet. Compared with the ENAS algorithms, the classification error of LF-MOGP is slightly inferior to those obtained by CNN-GA, Large-scale Evolution and Genetic CNN. However, the number of parameters of our evolved model is also much less than those obtained by the three algorithms. Moreover, our LF-MOGP can reduce the error by 0.77%, 0.74%, and 0.12% compared to ME-HDSS, MetaQNN, and NATS-Bench algorithms, respectively. Thus, it can be seen that our LF-MOGP is still very competitive with these state-of-the-art algorithms on the CIFAR100 dataset.

Analysis of strategy effectiveness and evolutionary behavior

Effectiveness regarding the leader–follower mechanism

The leader–follower mechanism (details in Sect. 3.1) is an important strategy proposed in this study. To verify its necessity and effectiveness, further experiments are carried out on CIFAR10 in this section. The comparison of the evolutionary behavior of LF-MOGP with and without the leader–follower mechanism is illustrated in Fig. 10.)

As can be seen from Fig. 10a, the Pareto front obtained by LF-MOGP is much superior to that obtained by the one without the leader–follower mechanism. In addition, according to Fig. 10b, the evolutionary process of the mean ACC of top 5 individuals obtained by LF-MOGP also outperform that obtained by the one without the leader–follower mechanism. The experiment results show that the leader–follower mechanism is effective and can guide the search to more promising regions in the search space.

Evolutionary behavior

To analyze the evolutionary behavior (especially convergence) of the proposed LF-MOGP, we show the evolutionary process of the Pareto front obtained by our algorithm in Fig. 11, using the MBI dataset as an example. The figure contains three parts, namely the evolutionary process of Pareto front (with a sampling period of 20 generations), and the two convergence trajectories of the ACC and complexity (number of parameters) metrics obtained by our algorithm.

As can be seen from the Fig. 11a, the quality of the Pareto front steadily improves as the number of generations increases, with the highest accuracy of 97.6% obtained by the non-dominated solutions on the validation set. According to Fig. 11b, both the mean ACC of the population and the best ACC of the individual show a steady increase against the generation, and tend to converge at the 80th generation. Similarly, from Fig 11c, it appears that the parameters of the individual with the best accuracy, as well as the mean parameters of the population, decreases obviously with the increase of generation, and their trajectories also tend to converge at around the 80th generation.

Discussion

In summary, LF-MOGP achieves promising performance on the eight datasets. Compared with the ENAS algorithms,

Table 5 The best classification error rates, parameters, GPU Days, and experimental environment of LF-MOGP and its competitors on CIFAR10 and CIFAR100

Method	Model	CIFAR10		CIFAR100		Parameters	Experimental environment
		Error	GPU Days	Error	GPU Days		
Handcrafted	Maxout [43]	9.38	–	38.57	–	–	–
	All-CNN [44]	9.08	–	33.71	–	–	NVIDIA modern
	NIN [45]	8.81	–	35.68	–	–	–
	Highway Network [46]	7.76	–	–	–	–	NVIDIA Tesla K40
	VGG [7]	6.66	–	–	–	20.04 M	NVIDIA Titan Black
	ResNet(depth=101) [8]	6.60	–	44.74	–	1.70 M	–
	DenseNet($K = 24$) [9]	5.19	–	19.64	–	15.30 M	–
ENAS	Genetic CNN [15]	5.01	30	25.10	–	– / 4.6 M	–
	MetaQNN [47]	6.92	21000	27.14	–	–/–	NVIDIA GeForce GTX 2080
	CGP-CNN [48]	6.34	30	–	–	1.75M/4.60M	NVIDIA GeForce GTX 1080Ti
	NAS [21]	6.01	22400	–	–	2.50 M/–	–
	PNAS [49]	**3.41**	225	–	–	3.20 M/–	–
	EvoCNN [16]	5.47	–	–	–	6.68 M/–	NVIDIA Tesla Volta V100
	Large-scale Evolution [19]	5.40	–	23.00	–	5.40 M/–	–
	CNN-GA [22]	4.78	35	22.03	40	2.90 M/4.10M	–
	NATS-Bench [50]	5.63	–	26.49	–	–/–	–
	ME-HDSS [51]	6.35	–	27.11	–	–/–	–
	FPSO [42]	6.28	**1.65**	–	– **0.70 M**/–	–	
	LF-MOGP (ours)	4.13	10	26.37	13	1.07 M/4.12M	NVIDIA GeForce 3090

* For the parameter, it is same in CIFAR10 and CIFAR100 for the handcrafted model, but the ENAs algorithm will get different models on different datasets, so there will be two parameter quantities, of which the left and right are the parameters of CIFAR10 and CIFAR100 respectively

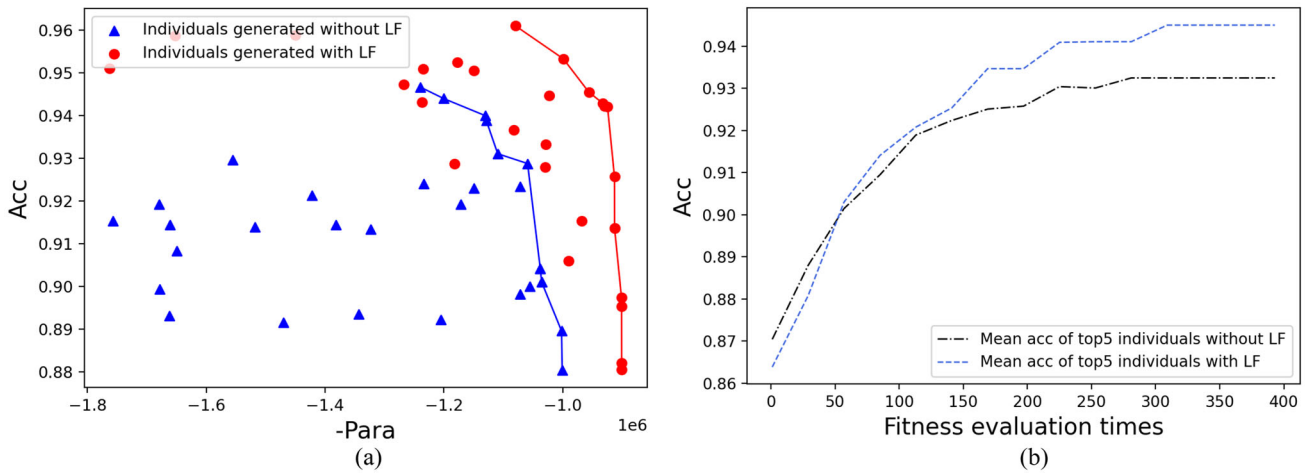

Fig. 10 Evolutionary behavior of LF-MOGP with and without the leader–follower mechanism

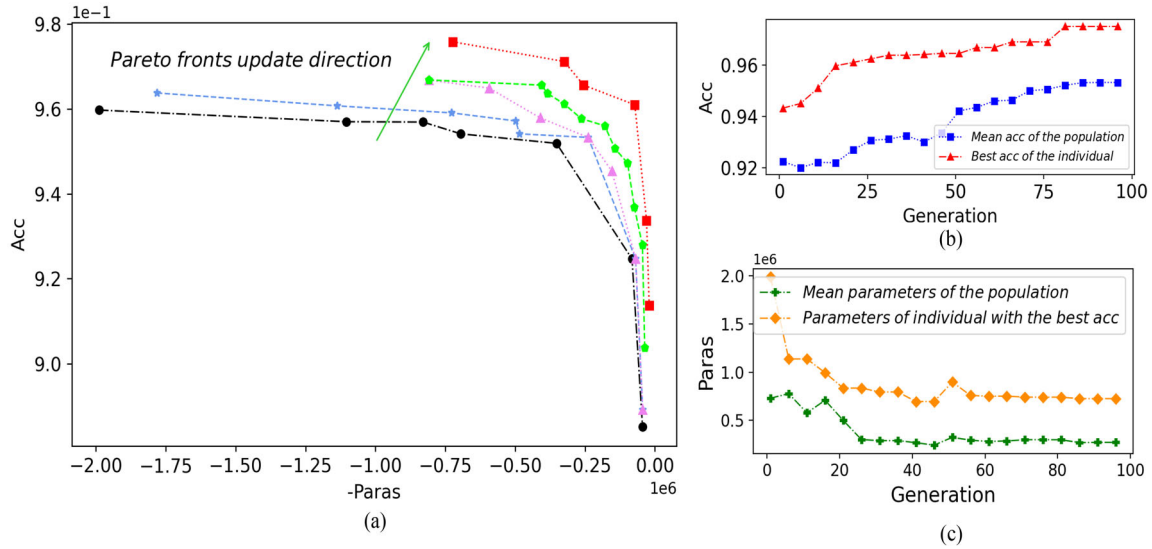

Fig. 11 LF-MOGP evolutionary behavior on MBI

LF-MOGP performed best on five datasets, second best on two datasets, and third best on one dataset. The advantages are more obvious when compared with the Handcrafted algorithms. The main reasons for the good performance of LF-MOGP can be analyzed as follows:

(1) Benefit from the encoding strategy of variable length-width designed in LF-MOGP. This encoding strategy is conducive to feature extraction and ensures that richer features can be extracted, and rich features are the basic guarantee for completing the classification tasks. In addition, LF-MOGP is essentially a block-based algorithm, while CGP is a tree-like structure with fewer structural constraints on the CNNs, so encoding the blocks with CGP can provide a larger search space for the algorithm.

(2) The proposed leader–follower mechanism has the advantage of accelerating the convergence speed of the algorithm as well as fewer resource requirements. An external archive of non-dominated solutions acts as the leader and the evolution operation is mainly performed on solutions from the external archive, which can greatly reduce the computational resources. In addition, an elite population is updated by new solutions that cannot enter the external archive, which means that the elite population can be viewed as a follower of the external archive. During evolution, if the diversity of the external archive tends to deteriorate, the solutions from the elite population will be selected to generate new solutions with good diversity, which can help the algorithm avoid getting trapped in the local optimum.

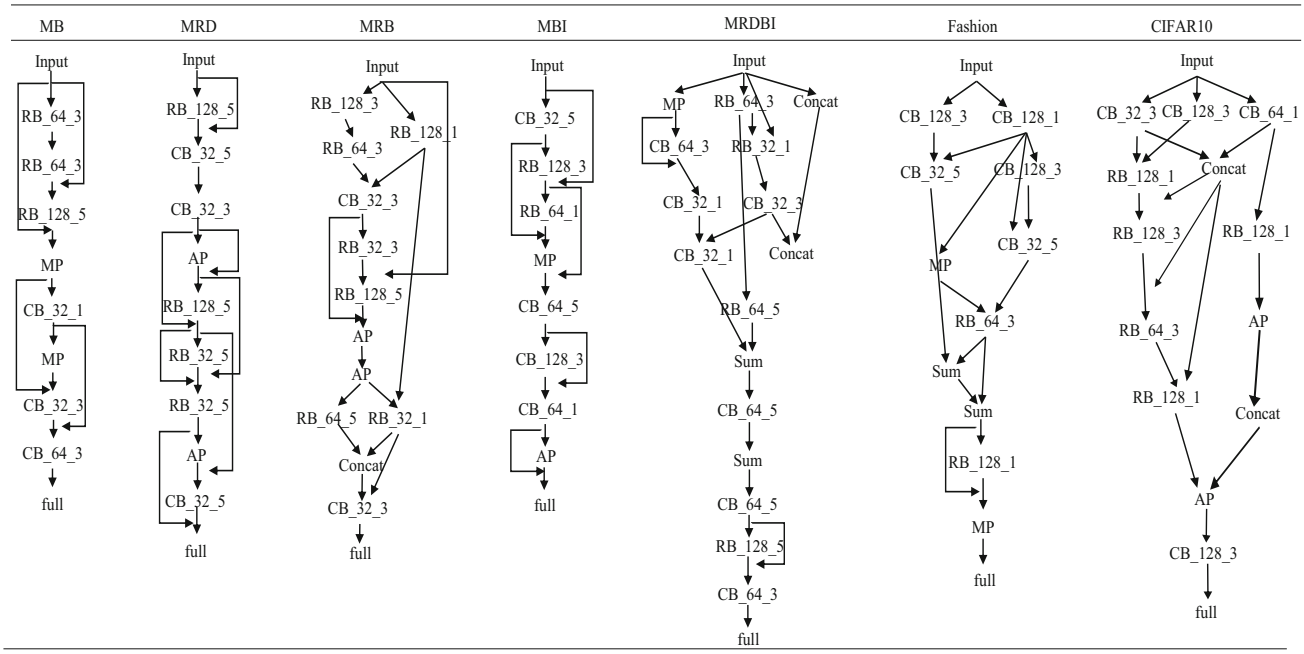

Fig. 12 Best CNNs evolved by LF-MOGP

Best CNNs evolved by LF-MOGP

Evolved CNN architectures

The best CNNs evolved by LF-MOGP on the seven benchmark datasets are presented in Fig. 12. Based on these best CNNs, the following conclusions can be drawn. First, the depth and width of these CNNs are different on different datasets, which indicates that LF-MOGP can design CNNs in a targeted manner according to the data characteristics of different tasks. Second, it breaks the limitations of traditional CNNs construction. For example, the fully connected layer can also use features from previous layers, which is significantly different from the classical fully connected layer that only completes the classification task based on the features obtained in the last layer, which improves the utilization of features. In addition, in conventional CNNs, pooling after convolution is a regular operation, whereas here they can be performed simultaneously, which makes the scale of features richer and thus the CNNs constructed by LF-MOGP become more flexible and powerful for feature extraction and utilization. Finally, the best CNNs evolved by LF-MOGP on the seven benchmark datasets are relatively lightweight, and thus it is more friendly to resources and hardware. Such CNNs are more suitable for practical applications such as applications on mobile devices.

Convergence performance of the best CNNs

To better understand the convergence performance of the best CNNs evolved by LF-MOGP, we plot the trajectories of classification accuracy and loss value on the validation set during the training process. Note that here is the validation set instead of the test set, because the test set is not allowed to participate in the training process. To organize the paper properly, the convergence curves of six representative CNNs with the best performance are provided in Fig. 13. From the observation of the convergence curves, the following conclusions can be drawn. First, these CNNs can converge within 100 epochs, so the convergence rate is relatively fast. Second, the convergence curves indicate that the settings of the relevant hyperparameters are feasible to ensure that each model can be adequately trained, such as the number of training epochs. In addition, the settings of the learning rate and its adjustment strategy are reasonable to enable the CNNs to achieve convergence as soon as possible.

Real-world application

Nowadays, intelligent methods based on computer vision have been widely used in industrial applications [52], so in this section, LF-MOGP is further validated on the online classification of the real-world industrial slab numbers. The slab

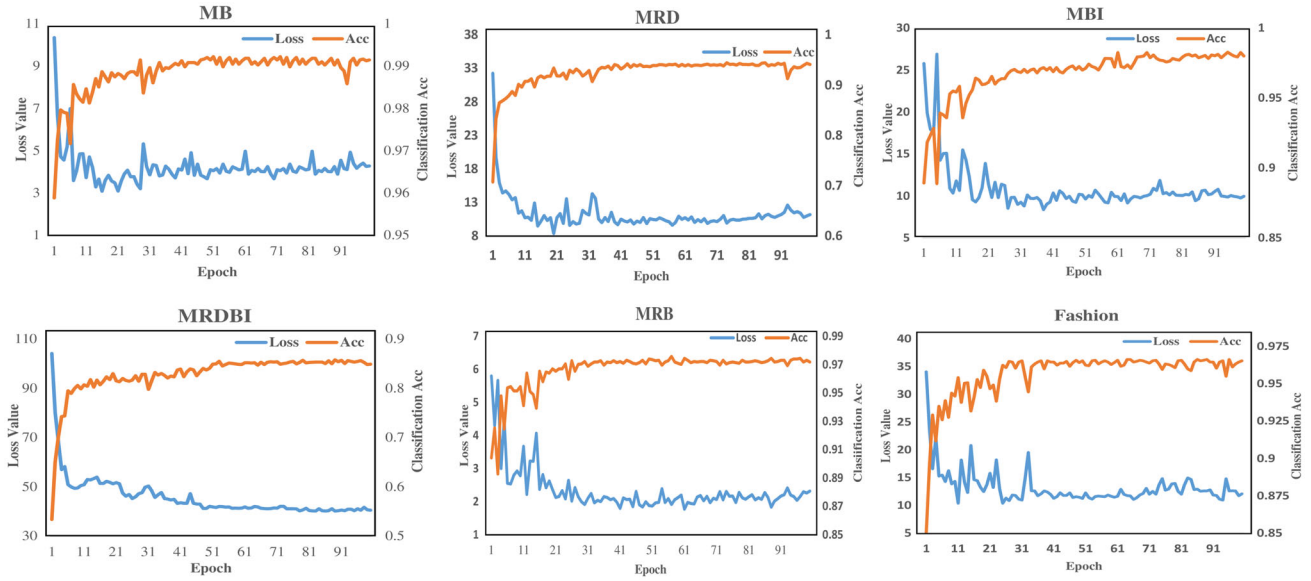

Fig. 13 Convergence performance of the best CNNs

Fig. 14 Example of real-word industrial slab data

number is used as a unique identification for each slab in the hot rolling process to help the site operator to put the slab into the designated heating furnace for heating and then complete the hot rolling production. Figure 14 shows an example of industrial slab number image data, where the sequence of slab numbers is on the left and the segmented slab number characters images are on the right. At present, the identification of slab numbers is done by operators 24 h a day, which is labor-intensive and inefficient, and any mistake will cause serious economic losses to the production line. Therefore, it is important to design a stable and efficient intelligent slab number identification algorithm to reduce labor costs and improve production efficiency.

Table 6 shows the classification results of the proposed LF-MOGP and several comparative algorithms on the real-world slab dataset. As can be seen from Table 6, the best CNN architecture evolved by LF-MOGP achieves the highest classification accuracy of 98.07% without a significant increase in the number of parameters compared to the rival algorithms. In fact, the number of parameters is less than those obtained by VGG and ResNet. The application scenarios for real-world problems are generally limited computing resources. Therefore, the evolved CNNs have a very high practical application value with high accuracy and low complexity.

The identification results for some practical slab numbers are presented in Fig. 15, where the characters marked in red mean misidentified numbers. From this figure, it can be seen that the CNN evolved by LF-MOGP can identify the slab number correctly in most cases even for those slabs with quite low quality characters. Such slab numbers are also very hard to distinguish by experienced human experts. It is worth noting that LF-MOGP is designed for single-label image classification. When dealing with slab number sequence recognition, our method can first recognize the segmented character images with a batch size of 10 without disturbing the order, and then reconvert the recognition result into a slab number sequence.

Conclusion and future work

In this paper, a CGP-based autonomous evolutionary convolutional neural network search algorithm (LF-MOGP) was proposed to evolve good CNNs for image classification tasks. In this algorithm, a flexible variable length-width encoding strategy was designed based on CGP and 22 basic functional blocks, which can help to expand the search space. To achieve convergence acceleration and reduce computational resources, a leader–follower strategy was proposed to guide the evolution process. The proposed LF-MOGP is tested on eight benchmark datasets and a real-world industrial dataset, and the experimental results illustrated that LF-MOGP outperformed 35 existing algorithms in the literature in terms of classification accuracy, model complexity, and computa-

Table 6 Classification results on the real-world slab numbers dataset

Model	LeNet [53]	VGG [7]	ResNet [8]	DenseNet [9]	LF-MOGP
Parameters(M)	0.27	13.84	11.69	6.97	7.7
Acc (%)	96.56	96.95	97.21	97.26	98.07

Fig. 15 Example of identification results of LF-MOGP and several comparative algorithms (the red characters indicate recognition errors)

Images					
Labels	8M4095L510	8M5055L520	8M5055L030	8M505L010	8M5080L520
LeNet	8M4995L510	8N5055L520	8M5055L030	8M5051010	8N50801520
VGG	8N4095L510	8N5055L520	8M5065L030	8M5051010	8M50301520
ResNet	8M4095L510	8M5055L520	8M5855L030	8M505L010	8M5030L520
DenseNet	8M4885L510	8N5055L520	8M5855L030	8M5051010	8N50801520
Ours	8M4095L510	8M5055L520	8M5055L030	8M505L010	8M5080L520

tional resource requirement. Since the CNNs constructed by LF-MOGP are relatively lightweight, it has greater potential for industrial applications, which is our main future work.

Acknowledgements This research was supported by the Major Program of National Natural Science Foundation of China (71790614), the Fund for the National Natural Science Foundation of China (62073067), the 111 Project (B16009), and the Fundamental Research Funds for the Central Universities (N2128001).

Declarations

Conflict of interest The authors declare that they do not have any commercial or associative interest that represents a conflict of interest in connection with the submitted work.

References

1. Xue G, Liu S, Ma Y (2020) A hybrid deep learning-based fruit classification using attention model and convolution autoencoder. Complex & Intelligent Systems. https://doi.org/10.1007/s40747-020-00192-x
2. Gadekallu, T.R., Alazab, M., Kaluri, R., Maddikunta, P.K.R., Bhattacharya, S., Lakshmanna, K., M, P.: Hand gesture classification using a novel CNN-crow search algorithm. Complex & Intelligent Systems 7(4), 1855–1868 (2021). https://doi.org/10.1007/s40747-021-00324-x
3. Shaaban MA, Hassan YF, Guirguis SK (2022) Deep convolutional forest: a dynamic deep ensemble approach for spam detection in text. Complex & Intelligent Systems. https://doi.org/10.1007/s40747-022-00741-6
4. Yu J, Zhang C, Wang S (2021) Multichannel one-dimensional convolutional neural network-based feature learning for fault diagnosis of industrial processes. Neural Comput Appl 33(8):3085–3104. https://doi.org/10.1007/s00521-020-05171-4
5. Krizhevsky A, Sutskever I, Hinton GE (2012) Imagenet classification with deep convolutional neural networks. Commun ACM 60:84–90
6. Szegedy, C., Wei Liu, Yangqing Jia, Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: 2015 IEEE Conference on Computer Vision and Pattern Recognition, pp. 1–9 (2015). https://doi.org/10.1109/CVPR.2015.7298594
7. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
8. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016). https://doi.org/10.1109/CVPR.2016.90
9. Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K.Q.: Densely connected convolutional networks. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition, pp. 2261–2269 (2017). https://doi.org/10.1109/CVPR.2017.243
10. Lecun Y, Bengio Y, Hinton GE (2015) Deep learning. Nature 521(7553):436–444
11. Zhu H, Zhang H, Jin Y (2021) From federated learning to federated neural architecture search: a survey. Complex Intell Syst 7(2):639–657. https://doi.org/10.1007/s40747-020-00247-z
12. Chu J, Yu X, Yang S, Qiu J, Wang Q (2022) Architecture entropy sampling-based evolutionary neural architecture search and its application in osteoporosis diagnosis. Complex & Intelligent Systems. https://doi.org/10.1007/s40747-022-00794-7
13. Liu, Y., Sun, Y., Xue, B., Zhang, M., Yen, G.G., Tan, K.C.: A survey on evolutionary neural architecture search. IEEE Transactions on Neural Networks and Learning Systems, 1–21 (2021). https://doi.org/10.1109/TNNLS.2021.3100554
14. Hao J, Cai Z, Li R, Zhu W (2022) Saliency: a new selection criterion of important architectures in neural architecture search. Neural Comput Appl 34(2):1269–1283. https://doi.org/10.1007/s00521-021-06418-4
15. Xie, L., Yuille, A.: Genetic cnn. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 1379–1388 (2017)
16. Sun Y, Xue B, Zhang M, Yen GG (2020) Evolving deep convolutional neural networks for image classification. IEEE

Trans Evol Comput 24(2):394–407. https://doi.org/10.1109/TEVC.2019.2916183
17. Cai, H., Chen, T., Zhang, W., Yu, Y., Wang, J.: Efficient architecture search by network transformation. In: AAAI (2018)
18. Mundt, M., Majumder, S., Murali, S., Panetsos, P., Ramesh, V.: Meta-learning convolutional neural architectures for multi-target concrete defect classification with the concrete defect bridge image dataset. In: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 11188–11197 (2019). https://doi.org/10.1109/CVPR.2019.01145
19. Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y.L., Tan, J., Le, Q., Kurakin, A.: Large-scale evolution of image classifiers. arXiv preprint arXiv:1703.01041 (2017)
20. 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)
21. Zoph, B., Le, Q.V.: Neural architecture search with reinforcement learning. arXiv preprint arXiv:1611.01578 (2016)
22. Sun Y, Xue B, Zhang M, Yen GG, Lv J (2020) Automatically designing cnn architectures using the genetic algorithm for image classification. IEEE Trans Cybernet 50(9):3840–3854. https://doi.org/10.1109/TCYB.2020.2983860
23. Sun Y, Xue B, Zhang M, Yen G (2020) Completely automated cnn architecture design based on blocks. IEEE Trans Neural Netw Learn Syst 31:1242–1254
24. Miller JF, Smith SL (2006) Redundancy and computational efficiency in cartesian genetic programming. IEEE Trans Evol Comput 10(2):167–174
25. Agapitos A, Loughran R, Nicolau M, Lucas S, O'Neill M, Brabazon A (2019) A survey of statistical machine learning elements in genetic programming. IEEE Trans Evol Comput 23(6):1029–1048. https://doi.org/10.1109/TEVC.2019.2900916
26. Huynh QN, Chand S, Singh HK, Ray T (2018) Genetic programming with mixed-integer linear programming-based library search. IEEE Trans Evol Comput 22(5):733–747. https://doi.org/10.1109/TEVC.2018.2840056
27. Miller, J.F.: What bloat? cartesian genetic programming on boolean problems. (2003)
28. Miller JF, Smith SL (2006) Redundancy and computational efficiency in cartesian genetic programming. IEEE Trans Evol Comput 10:167–174
29. Fang W, Gu M (2021) FMCGP: frameshift mutation cartesian genetic programming. Complex Intell Syst 7(3):1195–1206. https://doi.org/10.1007/s40747-020-00241-5
30. Miller JF, Harding, SL (2011) GECCO 2011 tutorial: Cartesian genetic programming. In: Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation, New York, NY, USA, pp 1261–1284. https://doi.org/10.1145/2001858.2002136
31. Esparcia-Alcázar AI, Almenar F, Vos TEJ, Rueda U (2018) Using genetic programming to evolve action selection rules in traversal-based automated software testing: results obtained with the TESTAR tool. Memetic Comput 10(3):257–265. https://doi.org/10.1007/s12293-018-0263-8
32. Rifai S, Vincent P, Muller X, Glorot X, Bengio Y (2011) Contractive auto-encoders: Explicit invariance during feature extraction. In: Proceedings of the 28th International Conference on International Conference on Machine Learning, Madison, WI, USA, pp. 833–840
33. Sohn, K., Lee, H.: Learning invariant representations with local transformations. In: in Proceedings of 29 Th International Conference on Machine Learning, Edinburgh, Scotland, UK (2012)
34. Sohn K, Zhou G, Lee C, Lee H (2013) Learning and selecting features jointly with point-wise gated boltzmann machines. In: Proceedings of the 30th International Conference on Machine Learning, Atlanta, Georgia, USA, pp 217–857
35. Chan T-H, Jia K, Gao S, Lu J, Zeng Z, Ma Y (2015) PCANet: A simple deep learning baseline for image classification? IEEE Trans Image Process 24(12):5017–5032. https://doi.org/10.1109/tip.2015.2475625
36. Larochelle, H., Erhan, D., Courville, A., Bergstra, J., Bengio, Y.: An empirical evaluation of deep architectures on problems with many factors of variation. In: Proceedings of the 24th International Conference on Machine Learning.ICML '07, pp. 473–480. Association for Computing Machinery, New York, NY, USA (2007). https://doi.org/10.1145/1273496.1273556
37. Wang, B., Sun, Y., Xue, B., Zhang, M.: Evolving deep convolutional neural networks by variable-length particle swarm optimization for image classification. In: 2018 IEEE Congress on Evolutionary Computation, pp. 1–8 (2018)
38. He X, Wang Y, Wang X, Huang W, Zhao S, Chen X (2021) Simple-encoded evolving convolutional neural network and its application to skin disease image classification. Swarm and Evolutionary Computation 67:100955
39. Bi, Y., Xue, B., Zhang, M.: An automated ensemble learning framework using genetic programming for image classification. Proceedings of the Genetic and Evolutionary Computation Conference (2019)
40. Bi Y, Xue B, Zhang M (2021) Genetic programming with image-related operators and a flexible program structure for feature learning in image classification. IEEE Trans Evol Comput 25(1):87–101. https://doi.org/10.1109/TEVC.2020.3002229
41. Tanveer, M.S., Karim Khan, M.U., Kyung, C.-M.: Fine-tuning darts for image classification. In: 2020 25th International Conference on Pattern Recognition (ICPR), pp. 4789–4796 (2021). https://doi.org/10.1109/ICPR48806.2021.9412221
42. Huang, J., Xue, B., Sun, Y., Zhang, M.: A flexible variable-length particle swarm optimization approach to convolutional neural network architecture design. In: 2021 IEEE Congress on Evolutionary Computation (CEC), pp. 934–941 (2021). https://doi.org/10.1109/CEC45853.2021.9504716
43. Goodfellow, I.J., Warde-Farley, D., Mirza, M., Courville, A., Bengio, Y.: Maxout networks. Computer Science, 1319–1327 (2013)
44. Springenberg, J.T., Dosovitskiy, A., Brox, T., Riedmiller, M.: Striving for simplicity: The all convolutional net. arXiv preprint arXiv:1412.6806 (2014)
45. Lin, M., Chen, Q., Yan, S.: Network in network. arXiv preprint arXiv:1312.4400 (2014) 1312.4400 [cs.NE]
46. Srivastava, R.K., Greff, K., Schmidhuber, J.: Highway Networks. arXiv (2015). https://doi.org/10.48550/ARXIV.1505.00387
47. Baker, B., Gupta, O., Naik, N., Raskar, R.: Designing Neural Network Architectures using Reinforcement Learning. arXiv (2016). https://doi.org/10.48550/ARXIV.1611.02167
48. Suganuma, M., Kobayashi, M., Shirakawa, S., Nagao, T.: Evolution of Deep Convolutional Neural Networks Using Cartesian Genetic Programming. Evolutionary Computation 28(1), 141–163 (2020). https://doi.org/10.1162/evco_a_00253. _eprint:https://direct.mit.edu/evco/article-pdf/28/1/141/2020362/evco_a_00253.pdf
49. Liu, C., Zoph, B., Shlens, J., Hua, W., Li, L., Fei-Fei, L., Yuille, A.L., Huang, J., Murphy, K.: Progressive neural architecture search. CoRR **abs/1712.00559** (2017) arxiv:1712.00559
50. Dong X, Liu L, Musial K, Gabrys B (2022) Nats-bench: Benchmarking nas algorithms for architecture topology and size. IEEE Trans Pattern Anal Mach Intell 44(7):3634–3646. https://doi.org/10.1109/TPAMI.2021.3054824
51. O'Neill D, Xue B, Zhang M (2021) Evolutionary neural architecture search for high-dimensional skip-connection structures on densenet style networks. IEEE Trans Evol Comput 25(6):1118–1132. https://doi.org/10.1109/TEVC.2021.3083315

52. Tang L, Meng Y (2021) Data analytics and optimization for smart industry. Front Eng Manag 8(2):157–171. https://doi.org/10.1007/s42524-020-0126-0
53. Lecun Y, Bottou L, Bengio Y, Haffner P (1998) Gradient-based learning applied to document recognition. Proc IEEE 86(11):2278–2324. https://doi.org/10.1109/5.726791