Electronic Research Archive

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

ERA, 32(3): 2016–2032.
DOI: 10.3934/era.2024092
Received: 18 October 2023
Revised: 23 February 2024
Accepted: 27 February 2024
Published: 06 March 2024

Research article

Evolving blocks by segmentation for neural architecture search

Xiaoping Zhao[1,*], Liwen Jiang[1], Adam Slowik[2], Zhenman Zhang[1] and Yu Xue[1,*]

[1] School of Computer Science, Nanjing University of Information Science and Technology, Nanjing 210044, China

[2] Department of Electronics and Computer Science, Koszalin University of Technology, Koszalin 75-453, Poland

* **Correspondence:** Email: zxp@nuist.edu.cn, xueyu@nuist.edu.cn; Tel: +8618951802528, +8613776601258.

Abstract: Convolutional neural networks (CNNs) play a prominent role in solving problems in various domains such as pattern recognition, image tasks, and natural language processing. In recent years, neural architecture search (NAS), which is the automatic design of neural network architectures as an optimization algorithm, has become a popular method to design CNN architectures against some requirements associated with the network function. However, many NAS algorithms are characterised by a complex search space which can negatively affect the efficiency of the search process. In other words, the representation of the neural network architecture and thus the encoding of the resulting search space plays a fundamental role in the designed CNN performance. In this paper, to make the search process more effective, we propose a novel compact representation of the search space by identifying network blocks as elementary units. The study in this paper focuses on a popular CNN called DenseNet. To perform the NAS, we use an ad-hoc implementation of the particle swarm optimization indicated as PSO-CNN. In addition, to reduce size of the final model, we propose a segmentation method to cut the blocks. We also transfer the final model to different datasets, thus demonstrating that our proposed algorithm has good transferable performance. The proposed PSO-CNN is compared with 11 state-of-the-art algorithms on CIFAR10 and CIFAR100. Numerical results show the competitiveness of our proposed algorithm in the aspect of accuracy and the number of parameters.

Keywords: convolutional neural network; neural architecture search; particle swarm optimization; block-based

1. Introduction

Convolutional neural networks (CNNs) are applied in many meaningful real-world tasks such as image recognition [1, 2], video classification [3], semantic segmentation [4], and natural language processing [5]. Many remarkable CNN models have been proposed, such as AlexNet [6], VGG [7], GoogleNet [8], ResNet [1], and DenseNet [9]. Although hand-crafted models have achieved excellent performance in image tasks [10, 11], the design of architectures requires major effort from human experts. Moreover, hand-crafted models usually lack flexibility and cannot be easily transferred to work on multiple datasets.

To solve the above problems, neural architecture search (NAS) was proposed [12]. NAS aims to search the best architectures of networks by means of a designing algorithm instead of performing the design manually. Effectively, NAS formulates the architecture design problem as an optimization problem and searches for a solution with high performance [13–16]. NAS methods can be divided into three types on the basis of the type of search approach: 1) reinforcement learning (RL), 2) gradient search, and 3) global search heuristics. Since RL approaches [12] require a large number of attempts and long observations to generate rewards, their application to NAS is impractical as it requires a large amount of computational resources. Gradient method approaches [17–19] are usually much faster than the NAS based on RL. However, gradient-based methods are inappropriate for NAS as they are likely to get trapped in a local optimum.

To overcome this limitation, evolutionary neural architecture search (ENAS), i.e., a global search heuristic to progressively select neural architectures, has become a prominent approach in neural network design. As highlighted in [20], ENAS is an umbrella name that includes all global heuristic approaches, such as evolutionary algorithms and swarm intelligence algorithms. In addition, ENAS can be viewed as a modern reinterpretation and implementation of neuroevolution which integrates knowledge-based information to encode and solve the network design problem. While neuroevolution attempts to evolve the structure and weights of a neural network, ENAS [21] evolves the structure and then tends to use a gradient-based method to quickly identify promising weights. Furthermore, traditional neuroevolution represents the search space as long vectors of weights that directly encode the neural network. On the other hand, ENAS often uses clever encoding strategies that aim at removing redundancies in the representation, and proposes a compact encoding of the problem. Thus, while neuroevolution can be applied to small networks, ENAS is better suited to design deep neural networks, such as CNNs. LargeEvo [22], proposed by Real et al., applies a genetic algorithm (GA) to evolve CNN architectures, which explicitly represents the fully-connected layers and uses only one mutation operator. The resulting search space is thus very high-dimensional and the search operators tend not to be efficient at detecting new promising solutions. Inspired by popular architectures such as ResNet [1] and DenseNet [9], Sun et al. use a GA to evolve the architectures in block-based search spaces in [23]. Xue et al. [24] proposed a block-based adaptive mutation neural architecture search algorithm, adaptively adjusting the mutation strategies during the evolution process to achieve better exploration. Song et al. [25] proposed an efficient residual dense block search algorithm, exploiting the variation of feature scale adequately to find lightweight and accurate networks for image super-resolution. Fang et al. [26] introduced a densely connected search space capable of searching block counts and block widths, thus expanding more possibilities on the basis of traditional DenseNet. Although these methods have achieved good performance, the encoded search space is still

very large due to different kinds of blocks being individually encoded.

In this paper, inspired by [23], we propose a method to further reduce the search space and thus enable a successful and computationally inexpensive application of ENAS. The contributions of this paper are as follows:

1) We propose an encoding strategy for the DenseNet block that aims to compress the search space and simplify the search process, resulting in significant efficiency improvements.

2) We propose a cutting method, which greatly reduces the size and flops of the final searched model without affecting the classification accuracy and facilitates the transfer learning of the designed architecture.

3) We propose an ENAS method based on particle swarm optimization for CNN design (PSO-CNN). It combines early stopping and training subset methods to reduce computational cost and increase convergence speed.

The remainder of the paper is designed as follows. Section 2 introduces the related work. Section 3 presents the proposed method. Sections 4 and 5 respectively describe the implementation details of the experiment and analyze the results. Finally, Section 6 gives the conclusion and describes our future work.

2. Related works

2.1. Particle swarm optimization algorithm

Let us consider a set $D \subseteq \mathbb{R}^n$, namely the domain or decision space. Let us indicate with $x = (x_1, x_2, \ldots, x_n) \in D$ the generic vector of the domain and with $f(x)$ a scalar function $f : D \to \mathbb{R}$.

Particle swarm optimization (PSO) is a population-based optimization algorithm belonging to the family of swarm intelligence algorithms [27]. To optimize $f(x)$, PSO uses a population of individuals i, namely particles. Each particle is associated with two attributes, velocity v^i and position x^i. The position represents the candidate solution, while the velocity is a perturbation vector. The update formulas to generate a new position x^{i+1} are given by

$$\begin{aligned} v^{i+1} &= wv^i + c_1 r_i(x^i_{lbest} - x^i) + c_2 r_i(x_{gbest} - x^i), \\ x^{i+1} &= x^i + v^{i+1}, \end{aligned} \tag{2.1}$$

where c_1, c_2 are parameters of the algorithm called the acceleration constants, r_i is a random number between 0 and 1 (without 0 or 1), x^i_{lbest} is the local best, i.e. the best position previously visited by the particle i, and x_{gbest} is the global best, i.e. the best position ever visited by the entire population.

If the newly explored position x^{i+1} outperforms x^i_{lbest}, the local best is updated. If x^{i+1} also outperforms x_{gbest}, the global best is updated too.

2.2. Block-based design method

The efficiency of NAS depends heavily on the chosen search space. Modern studies on NAS indicate the advantages of using a block representation of neural networks, simplifying the space complexity [23, 24]. Inspired by these studies, we design a search space where the elementary unit is the 2-dimensional DenseNet block [9] for image classification, here indicated as DB. Figure 1

illustrates the structure of DB. Three different convolutional layers with corresponding feature maps are sequentially connected. The DB terminates with a transition layer to connect to the following DB.

An important feature of DB which is exploited by our NAS framework is that the DB like the one in Figure 1 is univocally identified by two parameters: the number of layers N and the growth rate K. Once these two parameters are identified, the entire block is designed. The growth rate K is used to calculate the size of every input and output layer.

If the input size of the first layer is a_0, then the input size of the N-th layer is $a_0 + (N-1)K$ since each layer accepts inputs from all previous layers. This is done for all N layers composing the block. Due to its impact on network growth, the parameter K is called the *growth rate*. Further details of the DB and the search space in the proposed NAS algorithm are provided in Section 3.2.

This section describes and discusses the details of the proposed approach. The overall framework will first be provided in Section 3.1. The encoding strategy will be described in Section 3.2. The fitness evaluation method will be given in Section 3.3. Section 3.4 will describe the evolutionary search with the PSO algorithm. Finally, the cutting method after finding the optimal architecture will be provided in Section 3.5.

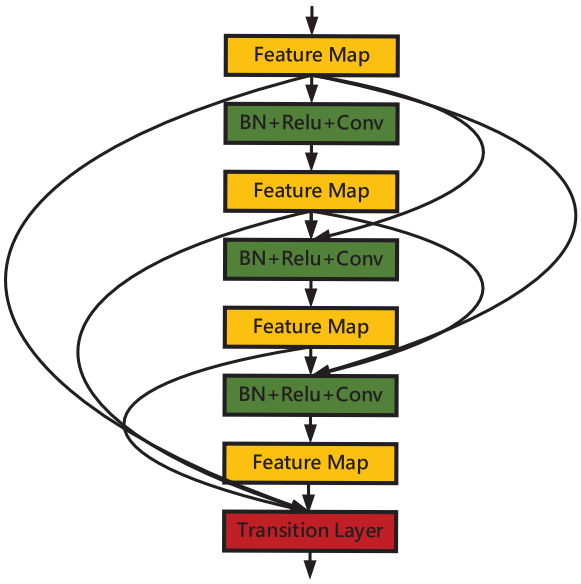


Figure 1. An example of a DB.

3. Proposed algorithm

3.1. Algorithm overview

The whole framework of PSO-CNN is shown in Figure 2. First, according to [12], the final CNN architectures that perform well in smaller datasets can be transferred to other datasets. In order to accelerate the entire training process and save computing resources, the training set is randomly cut into sub-training sets, and each CNN architecture is trained by the subset of a training set. Second, we need to set the encoding strategy for the architecture of the search space, which will be described in

Section 3.2. Then, in order to accelerate the entire evolution process, PSO is used to evolve the DB so as to find the best-performing individual on the sub-training sets. Next, the optimal architecture found is segmented using the splitting method in Section 3.5. Finally, the optimal individual obtains its accuracy after full training. In the following sections, the encoding strategy, fitness evaluation, optimization of the DenseNet block by PSO, and segmentation block are introduced.

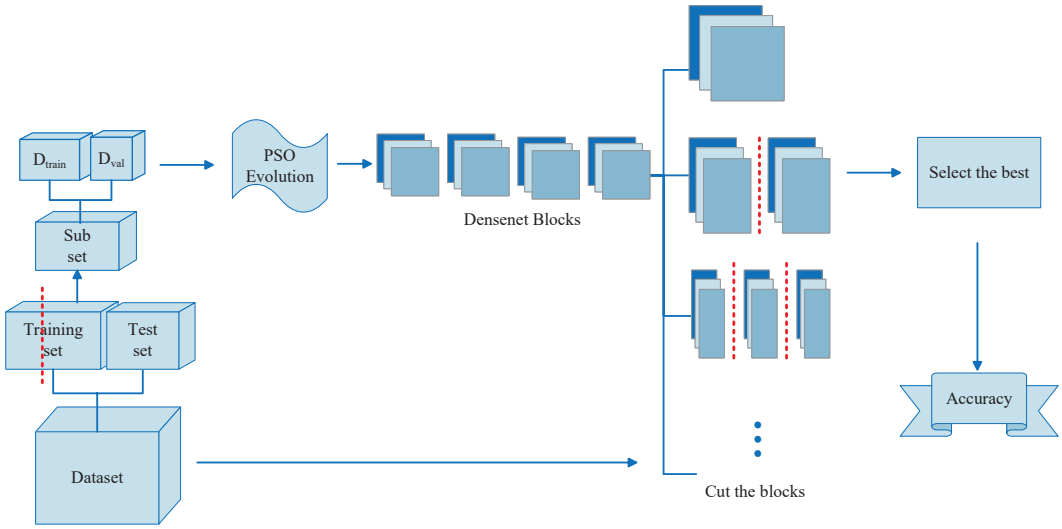


Figure 2. The framework of the proposed PSO-CNN method.

3.2. Encoding strategy

This encoding strategy is designed for the network structure of DB. In the DenseNet block, there are two hyperparameters introduced in Section 2.2. The growth rate K refers to the number of additional channels in each layer, which is one of the hyperparameters. The other one is the number of layers N. By simplifying the encoding method, the search process of PSO can be greatly improved. The encoding strategy of a DenseNet block used by the proposed algorithm is

$$D_i = (K_i, N_i), \tag{3.1}$$

where K_i is the growth rate of the i-th DenseNet block, and N_i represents the number of layers in the i-th DenseNet block. Figure 3 gives a specific example of the DenseNet block encoding strategy. From the figure we can see that $N = 3$, so there are 3 layers in each block. The growth rate $K = 3$, so the number of additional channels for each layer or the convolution kernel in each layer is 3. Eventually, this block will have $3 * 3 = 9$ additional channels.

DenseNet consists of DenseNet blocks, with a general number of 4. The layers between two adjacent DenseNet blocks are called transition layers, which mainly serves as a connection and reduces the size of the feature map. Therefore, a Densenet can be represented as

$$P_i = (D_1, D_2, D_3, D_4), \tag{3.2}$$

where D_i represents the i-th block. Figure 4 shows an example of a DenseNet consisting of four blocks.

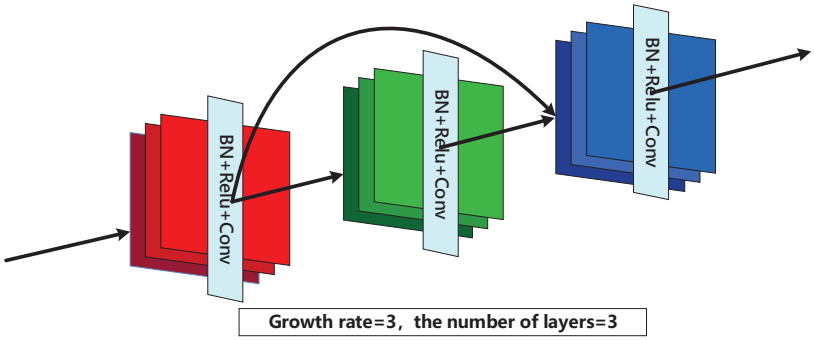


Figure 3. The encoding strategy of a DenseNet block.

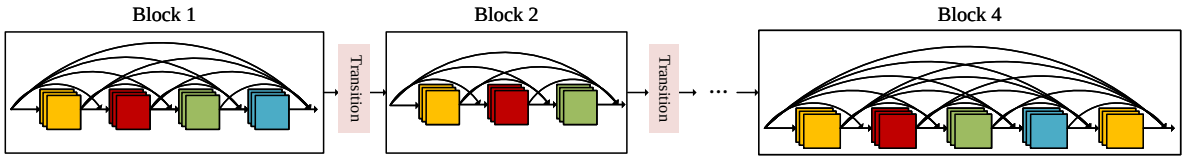


Figure 4. A deep DenseNet with four blocks.

3.3. Fitness evaluation

Algorithm 1 Evaluate the fitness

Require: a subset of training set d_t, population pop, $best_{acc} = 0$, $acc = 0$;
Ensure: fitness of population $fitness$, $best_{acc}$;

```
d_train, d_val ← Randomly divide d_t into 80% as d_train, 20% as d_val;
fitness ← []
for p in pop do
    Decode p into a network structure;
    Use the SGD optimizer to train the particle on d_train;
    acc ← Evaluate the particle on d_val ;
    Add acc to fitness;
    if acc > best_acc then
        best_acc ← acc;
    end if
end for
Return fitness, best_acc;
```

After using the proposed encoding strategy, Algorithm 1 describes the evaluation process of evaluating the fitness for searching for the optimal individual. According to [28], a subset of the training set d_t is used to train the DenseNet blocks. It is worth noting that the samples in each subset are chosen randomly without necessarily maintaining an even split of the classes. It also conforms to the theory of transferable blocks [29], which proves that one architecture which is learned from one

dataset can be transferred to another larger dataset. The particle p in the population pop represents a DenseNet individual. Then, d_t will be split into a training set d_{train} and a validation set d_{val}. In lines 3–10 of Algorithm 1, each particle is decoded into the corresponding network architecture. Next, the SGD optimizer that has been widely proven effective is used to train the particle on d_{train}. Finally, the fitness value acc of the architecture represented by each particle is calculated on d_{val}, and the fitness values of all individuals in the population $fitness$ and the optimal value $best_{acc}$ are recorded.

3.4. Evolving the blocks by PSO

Algorithm 2 Evolving the blocks by PSO

Require: the set of number of layers N, the set of the growth rate K;
Ensure: g_{best};
1: *population* ← Initialize the particle (the number is set to 10) with N and K;
2: *fitness* ← Use the fitness evaluation method in Algorithm 1;
3: g_{best}, $best_{epoch}$, generation g ← null, 0, 1;
4: **while** g <= 30 and $g - best_{epoch} < 10$ **do**
5: p_{best} ← Update the best fitness of current *population*;
6: **if** $p_{best} > g_{best}$ **then**
7: $g_{best} \leftarrow p_{best}$;
8: Update the $best_{epoch}$;
9: **end if**
10: **for** particle p in *population* **do**
11: p ← Use PSO to update the position (optimize with floating numbers);
12: Convert the encoding of p to integer encoding;
13: *fitness* ← Use the fitness evaluation method in Algorithm 1;
14: **end for**
15: $g \leftarrow g + 1$;
16: **end while**
17: Return g_{best};

Algorithm 2 illustrates the proposed algorithm for using PSO to evolve DenseNet blocks. When particles are initialized, the number of layers N and growth rate K are generated in a random range, and then fitness evaluation is used for each particle. It should be noted that these two hyperparameters have upper and lower bounds that need to be set, thereby meeting the memory constraints of the device and avoiding the inability to build proper models and lose important information about the architecture. In addition, an early stop mechanism is also used. When the accuracy of the particles does not improve within 10 epochs, the entire evaluation process stops, and $best_{epoch}$ represents the epoch in which the historically optimal particle is found. In lines 5–9, the optimal fitness value of the current population p_{best} and the historical optimal particle g_{best} are updated. In lines 10–14, the PSO algorithm is used for all particles in the population to update their positions. It should be noted that when updating the particle positions in continuous space, we first use floating point numbers and then convert to integer codes, finally using the fitness evaluation method. The final output is the structure that exhibits the most competitive performance throughout the entire iterative process. By combining the reduction of

the training set with the early stop mechanism, this method can effectively accelerate the search process and reduce computational complexity.

3.5. Progressively splitting DenseNet blocks

[30] showed that early feature reuse is redundant for later layers, and that feature reuse between neighboring feature maps is a more efficient strategy. Therefore, as shown in Algorithm 3, after obtaining the best-performing individual using PSO, we attempt to use the same cutting method for each block. The cutting method makes more efficient use of features between adjacent layers than the stacking method [28, 29], optimizing computational resources and improving model performance. The dense connectivity of the DB helps to maintain the coherence of the information during the cutting process, which mitigates the information loss that may be caused by the cutting. The cutting method can reduce the potential redundancy of information that may be caused by global sharing, and enhance the efficiency of feature utilization with a resistance to overfitting. The entire training set d_t is used to train the split structure, and the d_t is divided into a training set d_{train} and a validation set d_{val}. The number of cuts i starts from 1, and Figure 5 shows the process of cutting a block. It should be noted that the cutting will be as uniform as possible, with the front part having the same size and the last part being appropriately sized. The purpose of splitting blocks is to reduce the size of the final searched model. If the final model is formed by stacking, the number of parameters will be several times larger than the model obtained by splitting. In addition, once the final accuracy of the split CNN candidates is not improved compared to before splitting, the splitting process will automatically stop. Finally, the best structure will be obtained.

Algorithm 3 Progressively splitting DenseNet blocks

Require: the evolved blocks b, the optimal accuracy acc_{best} , the training dataset d_t;
Ensure: acc_{best};

d_{train}, d_{val} ← Randomly split d_t into 80% as d_{train} (training set), 20% as d_{val} (validation set);
i, acc, n ← 0, 0, 5;
while $i < n$ **do**
    $i \leftarrow i + 1$;
    c ← Split b for i times;
    Decode c into a network structure;
    Use SGD optimizer to train c on the d_{train};
    acc ← Evaluate c on d_{val};
    if $acc > acc_{best}$ **then**
        $acc_{best} \leftarrow acc$;
    else
        Break;
    end if
end while
Return acc_{best};

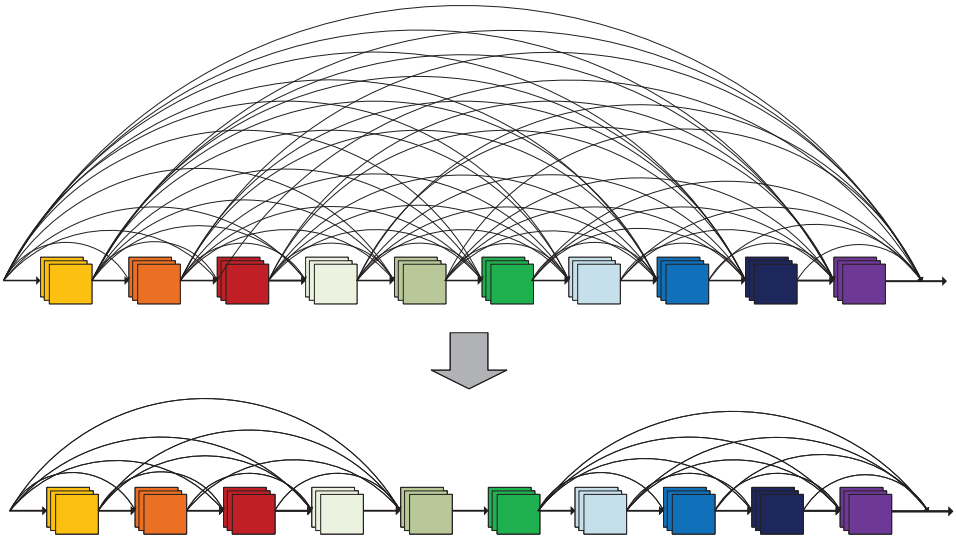


Figure 5. An example of cutting a DenseNet block.

4. Experiment design

This section describes the experimental design related to our proposed PSO-CNN. First, Section 4.1 describes the benchmark dataset used, then Section 4.2 introduces the algorithms for comparison, and finally Section 4.3 presents the setup of the experimental parameters.

4.1. Benchmark datasets

In order to validate the effectiveness of our proposed algorithm, experiments are carried out on two benchmark datasets, CIFAR10 and CIFAR100, which are widely used in NAS methods. CIFAR10 is a color image dataset containing ten categories, including cat, bird, dog, airplane, deer, automobile, frog, horse, ship, and truck. Each category has 6000 images, so the entire dataset has a total of 60,000 images, of which 50,000 are training images and 10,000 are test images. The size of each image is 32×32. The difference between CIFAR100 and CIFAR10 is that CIFAR100 has 100 categories and each category has 500 training pictures and 100 test pictures. In addition, the performance results of the comparison algorithm can be easily collected from other papers. Meanwhile, the images will be pre-processed in the same way as the other methods and clipped to the original size, flipped horizontally and finally input into the proposed algorithm.

4.2. Comparison methods

In the experiments, our proposed PSO-CNN will be compared with existing NAS methods to show oue method's competitiveness. Among the algorithms chosen for comparison, we chose the gradient-based NAS method DARTS [17] and reinforcement learning (RL) NAS methods such as EAS [31], NAS [12], and MetaQNN [32]. There are also evolutionary computational NAS methods that are consistent with our algorithm, for instance, Genetic CNN [33], Hierarchical Evolution [34], AE-CNN [23], Firefly-CNN [35], NSGANet [36], and EPSO-CNN [28].

4.3. Parameter setting

Due to the tasks and evaluations being identical to the benchmarks, we directly cited the results of the comparison algorithms used in our study from other papers. In this paper, the inertia weight w is set to 0.7, the acceleration coefficient c_1 is set to 1.5, and the acceleration coefficient c_2 is set to 1.5. The population size and generation are set to 10 and 30, respectively. The number of convolutional layers in the DenseNet block is in the range of [6, 32], and the growth rate range is in [12, 32]. The optimizer used is SGD, the weight decay is set to e^{-4}, the momentum is set to 0.9, and the initial learning rate is set to 0.1.

5. Analysis of experimental results

In this section, the results of our experiments are analyzed. First, in Sections 5.1 and 5.2 we analyze the results of our comparison experiments with other algorithms on CIFAR10 and CIFAR100, respectively. Then, in Section 5.3 we analyze the ablation experiments of our algorithm.

5.1. Experimental results of the algorithm on CIFAR10

Table 1 displays the competitiveness of the proposed PSO-CNN compared to other algorithms on CIFAR10 and CIFAR100. The second and third columns represent the tested error rates of the algorithms. C10 and C100 represent the two datasets. In addition to this, the number of parameters of the architectures searched by the different algorithms and the search time overhead spent are also shown. The sixth column shows the type of construction process for the final model, where "RL" represents reinforcement learning, "Gradient" stands for gradient-based methods, and "EC" represents evolutionary computation.

Table 1. Comparison experiments of our proposed PSO-CNN with other algorithms.

Algorithms	Error (%) C10	Error (%) C100	Parameters (M)	Times GPU/Days	Method
EAS [31]	4.23	-	23.4	10	RL
NAS [12]	6.01	-	2.5	22,400	RL
MetaQNN [32]	6.92	27.14	-	100	RL
DARTS (first order) [17]	3.14	20.58	3.3	1.5	Gradient
Large-scale Evolution [22]	5.4 -	- 23	5.4 40.4	2750	EC
Genetic CNN [33]	7.1	29.05	-	17	EC
Hierarchical Evolution [34]	3.63	-	-	300	EC
AE-CNN [23]	4.3 -	- 20.85	2.0 5.4	27 36	EC
NSGANet [36]	4.67	25.17	0.2	27	EC
EPSOCNN [28]	3.58	18.56	6.74	< 4	EC
Firefly-CNN [35]	3.3	22.3	3.21	-	EC
PSO-CNN (our proposed)	3.28	18.24	1.07	3	EC

As can be obtained from Table 1, the proposed algorithm PSO-CNN obtains the lowest classification error rate, except for DARTS, compared to the comparison algorithms, but with a much smaller model size. Since the hand-crafted networks have no search process, they are represented by '-' on GPU/Days. Compared with the three reinforcement learning based search methods, the proposed algorithm is 0.95% lower than EAS, and has an obvious advantage in computing time. Compared with the evolutionary calculation method, the proposed algorithm is 2.12% lower than Large-scale Evolution, and 1.02% lower than AE-CNN. The number of parameters of PSO-CNN is less than all comparison algorithms except NSGANet, because NSGANet is a multi-objective NAS method and takes the number of parameters as one of the optimization objectives. Therefore, due to the small number of parameters, its classification error rate is 1.39% higher than the proposed algorithm. As PSO is used for optimization, the proposed algorithm has similar classification performance with EPSOCNN, but the number of parameters is much fewer.

5.2. Experimental results of the algorithm on CIFAR100

The classification difficulty ratio of CIFAR100 is higher due to having more categories. The error rate of PSO-CNN on CIFAR100 is 18.24%. Compared with gradient-based algorithms, the error rate of PSO-CNN is 3.3% lower than DARTS. In addition, compared with the method based on reinforcement learning, the proposed PSO-CNN has a lower error rate of 8.9% than MetaQNN. When compared with other well-known algorithms such as Large-scale Evolution and Genetic CNN, the proposed algorithm has a greater accuracy advantage. When compared with EPSO-CNN, the proposed algorithm has similar classification performance with it, but the size of the final model of the proposed algorithm is much smaller. The performance of PSO-CNN is mainly due to its flexibility in utilizing different parts of the features, and the local adaptation helps the network to better fit the distributional characteristics and local details of the complex input data. At the same time, the cutting process should take care to avoid loss of information that may result from localized connections.

It is worth noting that the proposed algorithm does not need to re-evolve on CIFAR100, and the user only needs to re-migrate and cut the evolved blocks on CIFAR10, which also helps reduce the computation time.

5.3. The convergence analysis

According to the analysis of the convergence performance, the performance of the block searched by the PSO-CNN is shown in Figure 6. In the early stage of evolution, the PSO can perform a fast global search. When at the 10th generation, the accuracy of the population reached 0.45. In subsequent generations, the performance of the searched block is maintained at about 0.5. The algorithm finally converges after the 20th generation, and the accuracy of the finally searched block is about 0.6. It should be noted that the training of these blocks is insufficient because we want to speed up the search efficiency of the algorithm. The best individual will be cut and undergo final training.

Each of the final blocks are cut evenly, and the final model with different cutting times is tested to get the training loss and validation accuracy. As shown in the Figure 7, the five models have extremely fast convergence speed in the early stages of training. After 50 epochs, the training loss tends to converge. At the 150th epoch, the convergence speed accelerates sharply, and almost all models tend to converge after 200 epochs. In addition, the validation accuracy experiments of the five models also show that

the performance of the final models do not degrade after being cut, and this phenomenon can validate the effectiveness of our proposed cutting method. We also found that, among all the final models, the final model which is cut 2 times has the fastest convergence rate in the entire training process.

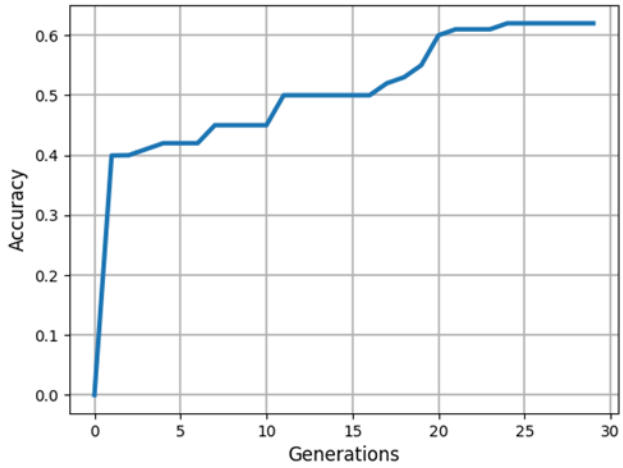
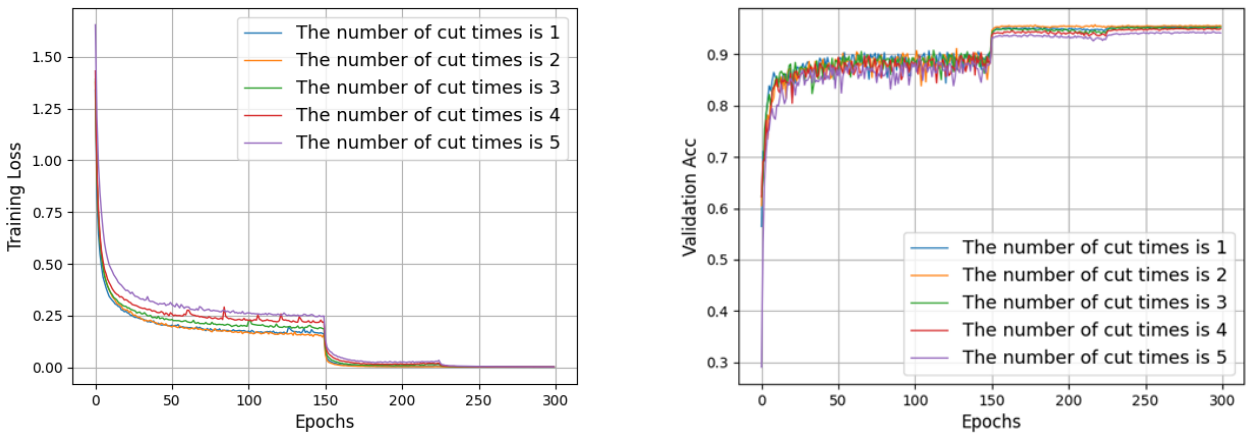


Figure 6. Convergence curve of evolving the block on CIFAR10.



(a) Training Loss (b) Validation Accuracy

Figure 7. Experimental results of different cutting times.

Figure 8 shows the final test accuracy of the final models after being cut. All models have been trained by SGD for 300 epochs. From the figure, we can see that the searched blocks without cutting have the lowest accuracy. In addition, the final model which is cut twice obtains the best classification performance, and it also matches the result of the training loss in Figure 7. It is worth noting that for the final model, more cuts do not necessarily bring better performance. The performance of models whose cutting numbers are 3, 4, and 5 are worse than cutting numbers of 2.

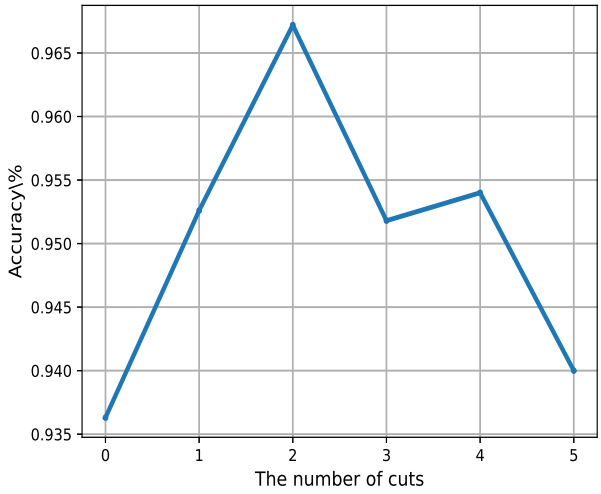


Figure 8. The classification of the final models with different cut times.

Table 2. The number of parameters and flops of the final models with different cutting times.

	1	2	3	4	5	6
Parameters	1.71 M	1.49 M	1.30 M	1.20 M	1.14 M	1.07 M
Flops	3.32 G	1.7 G	1.014 G	702 M	556 M	420 M

Table 2 shows the number of parameters and flops in the final model after each cut. These two indicators are very important because, the smaller the parameters and flops, the lighter the final searched model, which can greatly accelerate the training speed. The results in the table show that as the number of cuts increases, the parameters and flops are significantly reduced. This is because the number of cuts can effectively reduce the number of dense connections in the entire network, and it can prove the effectiveness of the cutting method. Compared with some well-known algorithms in Table 1, the final parameter of our proposed algorithm has greater advantages than all other methods, except NSGANet, because NSGANet is a multi-objective algorithm and it specifically optimizes the number of parameters. Compared with the similar EPSO-CNN, the parameters of the final model searched by the proposed algorithm is less than 1/6 of that of EPSO-CNN.

6. Conclusions

In this paper, the aim was to accelerate the design of CNN architectures using PSO and to find the smallest possible model. The goal was successfully achieved on the CIFAR10 and CIFAR100 datasets. To reduce the search space, this paper uses PSO to evolve the blocks, which has two parameters. In addition, a method of segmentation is proposed to shrink the size of the final model. As can be seen from the experimental results, PSO-CNN shows excellent competitiveness with different state-of-the-art design methods in terms of accuracy and computational resource consumption. In the future, we will try to carry out experiments with more datasets to evaluate the performance of the architecture after using the cutting method under different circumstances. Considering the current computational

resource constraints, we will further refine our model by focusing on how to effectively utilize resources to improve computational efficiency and expand the scope of testing.

Use of AI tools declaration

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

Acknowledgment

This work was partially supported by the National Natural Science Foundation of China (62376127, 61876089, 61876185, 61902281), the opening Project of Jiangsu Key Laboratory of Data Science and Smart Software (No. 2019DS301), the Natural Science Foundation of Jiangsu Province (BK20141005), the Natural Science Foundation of the Jiangsu Higher Education Institutions of China (14KJB520025).

Conflict of interest

The authors declare there is no conflict of interest.

References

1. K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* IEEE, (2016), 770–778. https://doi.org/10.1109/CVPR.2016.90
2. S. Singaravel, J. Suykens, P. Geyer, Deep-learning neural-network architectures and methods: Using component-based models in building-design energy prediction, *Adv. Eng. Inf.*, **38** (2018), 81–90. https://doi.org/10.1016/j.aei.2018.06.004
3. H. Xu, J. Kong, M. Liang, H. Sun, M. Qi, Video behavior recognition based on actional-structural graph convolution and temporal extension module, *Electron. Res. Arch.*, **30** (2022), 4157–4177. https://doi.org/10.3934/era.2022210
4. D. Peng, Y. Lei, H. Munawar, Y. Guo, W. Li, Semantic-aware domain generalized segmentation, in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, (2022), 2584–2595. https://doi.org/10.1109/CVPR52688.2022.00262
5. T. Korbak, K. Shi, A Chen, R. V. Bhalerao, C. Buckley, J. Phang, et al., Pretraining language models with human preferences, in *Proceedings of the 40th International Conference on Machine Learning*, PMLR, **202** (2023), 17506–17533.
6. A. Krizhevsky, I. Sutskever, G. Hinton, Imagenet classification with deep convolutional neural networks, *Commun. ACM*, **60** (2017), 84–90. https://doi.org/10.1145/3065386
7. K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, preprint, arXiv:1409.1556.

8. C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, et al., Going deeper with convolutions, in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, (2015), 1–9. https://doi.org/10.1109/CVPR.2015.7298594
9. G. Huang, Z. Liu, L. Van Der Maaten, K. Q. Weinberger, Densely connected convolutional networks, in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, (2017), 2261–2269. https://doi.org/10.1109/CVPR.2017.243
10. J. Xi, Z. Xu, Z. Yan, W. Liu, Y. Liu, Portrait age recognition method based on improved ResNet and deformable convolution, *Electron. Res. Arch.*, **31** (2023), 6585–6599. https://doi.org/10.3934/era.2023333
11. C. Swarup, K. U. Singh, A. Kumar, S. K. Pandey, N. Varshney, T. Singh, Brain tumor detection using CNN, AlexNet & GoogLeNet ensembling learning approaches, *Electron. Res. Arch.*, **31** (2023), 2900–2924. https://doi.org/10.3934/era.2023146
12. B. Zoph, Q. V. Le, Neural architecture search with reinforcement learning, preprint, arXiv:1611.01578.
13. T. Elsken, J. H. Metzen, F. Hutter, Neural architecture search: A survey, *J. Mach. Learn. Res.*, **20** (2019), 1997–2017.
14. Y. Xue, W. Tong, F. Neri, P. Chen, T. Luo, L. Zhen, et al., Evolutionary architecture search for generative adversarial networks based on weight sharing, *IEEE Trans. Evol. Comput.*, **2023** (2023), 1. https://doi.org/10.1109/TEVC.2023.3338371
15. Y. Xue, X. Han, Z. Wang, Self-adaptive weight based on dual-attention for differentiable neural, *IEEE Trans. Ind. Inf.*, **2024** (2024), 1–10. https://doi.org/10.1109/TII.2023.3348843
16. Y. Xue, Z. Zhang, F. Neri, Similarity surrogate-assisted evolutionary neural architecture search with dual encoding strategy, *Electron. Res. Arch.*, **32** (2024), 1017–1043. https://doi.org/10.3934/era.2024050
17. H. Liu, K. Simonyan, Y. Yang, DARTS: Differentiable architecture search, preprint, arXiv:1806.09055.
18. Y. Xue, J. Qin, Partial connection based on channel attention for differentiable neural architecture search, *IEEE Trans. Ind. Inf.*, **19** (2023), 6804–6813. https://doi.org/10.1109/TII.2022.3184700
19. Y. Xue, C. Lu, F. Neri, J. Qin, Improved differentiable architecture search with multi-stage progressive partial channel connections, *IEEE Trans. Emerging Top. Comput. Intell.*, **8** (2024), 32–43. https://doi.org/10.1109/TETCI.2023.3301395
20. Y. Liu, Y. Sun, B. Xue, M. Zhang, G. G. Yen, K. C. Tan, A survey on evolutionary neural architecture search, *IEEE Trans. Neural Networks Learn. Syst.*, **34** (2023), 550–570. https://doi.org/10.1109/TNNLS.2021.3100554
21. Y. Xue, C. Chen, A. Slowik, Neural architecture search based on a multi-objective evolutionary algorithm with probability stack, *IEEE Trans. Evol. Comput.*, **27** (2023), 778–786. https://doi.org/10.1109/TEVC.2023.3252612
22. E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, et al., Large-scale evolution of image classifiers, in *Proceedings of the 34th International Conference on Machine Learning (ICML)*, PMLR, **70** (2017), 2902–2911.

23. Y. Sun, B. Xue, M. Zhang, G. G. Yen, Completely automated CNN architecture design based on blocks, *IEEE Trans. Neural Networks Learn. Syst.*, **31** (2020), 1242–1254. https://doi.org/10.1109/TNNLS.2019.2919608

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

25. D. Song, C. Xu, X. Jia, Y. Chen, C. Xu, Y. Wang, Efficient residual dense block search for image super-resolution, in *Proceedings of the AAAI conference on artificial intelligence*, AAAI Press, **34** (2020), 12007–12014. https://doi.org/10.1609/aaai.v34i07.6877

26. J. Fang, Y. Sun, Q. Zhang, Y. Li, W. Liu, X. Wang, Densely connected search space for more flexible neural architecture search, in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, (2020), 10625–10634. https://doi.org/10.1109/CVPR42600.2020.01064

27. J. Kennedy, R. C. Eberhart, Particle swarm optimization, in *Proceedings of ICNN'95 - International Conference on Neural Networks*, IEEE, **4** (1995), 1942–1948. https://doi.org/10.1109/ICNN.1995.488968

28. B. Wang, B. Xue, M. Zhang, Particle swarm optimisation for evolving deep neural networks for image classification by evolving and stacking transferable blocks, in *2020 IEEE Congress on Evolutionary Computation (CEC)*, IEEE, (2020), 1–8. https://doi.org/10.1109/CEC48606.2020.9185541

29. E. Real, A. Aggarwal, Y. Huang, Q. V. Le, Regularized evolution for image classifier architecture search, in *Proceedings of the AAAI Conference on Artificial Intelligence*, AAAI Press, **33** (2019), 4780–4789. https://doi.org/10.1609/aaai.v33i01.33014780

30. G. Huang, S. Liu, L. v. d. Maaten, K. Q. Weinberger, CondenseNet: An efficient denseNet using learned group convolutions, in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, (2018), 2752–2761. https://doi.org/10.1109/CVPR.2018.00291

31. H. Cai, T. Chen, W. Zhang, Y. Yu, J. Wang, Regularized evolution for image classifier architecture search, in *Proceedings of the AAAI Conference on Artificial Intelligence*, AAAI Press, **32** (2018), 4780–4789. https://doi.org/10.1609/aaai.v32i1.11709

32. B. Baker, O. Gupta, N. Naik, R. Raskar, Designing neural network architectures using reinforcement learning, preprint, arXiv:1611.02167.

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

34. H. Liu, K. Simonyan, O. Vinyals, C. Fernando, K. Kavukcuoglu, Hierarchical representations for efficient architecture search, preprint, arXiv:1711.00436.

35. A. I. Sharaf, E. S. F. Radwan, An automated approach for developing a convolutional neural network using a modified firefly algorithm for image classification, in *Applications of Firefly Algorithm and its Variants*, Springer, (2020), 99–118. https://doi.org/10.1007/978-981-15-0306-1_5

36. G. Cuccu, J. Togelius, P. Cudre-Mauroux, Playing atari with six neurons (Extended abstract), in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI-20)*, International Joint Conferences on Artificial Intelligence Organization, (2020), 4711–4715. https://doi.org/10.24963/ijcai.2020/651

AIMS Press