psoResNet: an Improved PSO-Based Residual Network Search Algorithm

Dianwei Wang[a], Leilei Zhai[a], Jie Fang[a], Yuanqing Li[a], Zhijie Xu[b]

a. School of Telecommunication and Information Engineering, Xi'an University of Posts and Telecommunications, Xi'an, 710121, P. R. China.

b. School of Computing and Engineering, University of Huddersfield, Huddersfield, HD1 3DH, UK.

Abstract

Neural Architecture Search (NAS) methods are widely employed to address the time-consuming and costly challenges associated with manual operation and design of deep convolutional neural networks (DCNNs). Nonetheless, prevailing methods still encounter several pressing obstacles, including limited network architecture design, excessively lengthy search periods, and insufficient utilization of the search space. In light of these concerns, this study proposes an optimization strategy for residual networks that leverages an enhanced Particle swarm optimization algorithm. Primarily, low-complexity residual architecture block is employed as the foundational unit for architecture exploration, facilitating a more diverse investigation into network architectures while minimizing parameters. Additionally, we employ a depth initialization strategy to confine the search space within a reasonable range, thereby mitigating unnecessary particle exploration. Lastly, we present a novel approach for computing particle differences and updating velocity mechanisms to enhance the exploration of updated trajectories. This method significantly contributes to the improved utilization of the search space and the augmentation of particle diversity. Moreover, we constructed a crime-dataset comprising 13 classes to assess the effectiveness of the proposed algorithm. Experimental results demonstrate that our algorithm can design lightweight networks with superior classification performance on both benchmark datasets and the crime-dataset.

Index Terms — Residual networks; particle swarm optimization; neural network optimization; image classification.

1 Introduction

Convolutional neural networks (CNNs) have shown great advantages in the field of image classification **Error! Reference source not found.**. Compared to traditional image classification algorithms that depend on feature extraction and classifiers, excellent CNN architectures can achieve optimal results in almost all benchmark classification datasets, even beyond the level of manual recognition [2]. However, when designing new high-performance networks, both the optimal design of the network connection and the selection of parameters within each network layer require significant time for tuning. Moreover, to obtain more competitive performance from the model, increasing numbers of researchers are working on developing deeper CNNs [3]. As the network model grows larger, the greater difficulties to handle the huge combination of parameters manually. Therefore, an automated method is eagerly needed to implement the design of CNN architectures [4].

Evolutionary computation [5] (EC) is a simulation of evolutionary principles in biology that can effectively address optimization and search problems, and its flexible encoding and parameter search capabilities play a crucial role in the automatic design of CNN architectures. Therefore, the thought of adaptively adjusting the model architecture and parameters using EC methods has been widely used [6]. The genetic algorithms (GAs) [7] and particle swarm optimization (PSO) [8] are commonly used methods in EC.

EvoCNN [9] introduced a novel search approach based on GAs, aiming to optimize the initialization of CNN architectures and connection weights through the utilization of a variable-length encoding strategy. However, such algorithms required a significant amount of computation time (2-4 days on 2-3 GPUs for small datasets) to achieve effective network architecture design. In comparison to GAs, the PSO algorithm demonstrated enhanced search capabilities and computational efficiency. IPPSO [10] was the first method to apply PSO to evolving CNN architectures, which achieved optimization of the network architecture by encoding the internet protocol (IP) addresses assigned to particles at each layer. However, the algorithm could only optimize particles that reach a pre-defined maximum length and its results have only been validated on three benchmark datasets. In response to these issues, a new particle update rule was proposed in the psoCNN [11] method. The particle positions were updated by copying the parameters

between different layers in the individual or global optimal solution, which can search for the optimized solution of the internal parameters, the number of layers, the connection methods of the convolutional as well as the fully connected layers. Based on psoCNN, the psoGroup [12] proposed a group-based coding strategy, which achieved the limit of the number of pooling layers by setting the number of groups. Also, they designed a particle velocity update mechanism based on differences in network configuration to improve the diversity of particle search. Although each of these methods has made a significant contribution to the automatic search of architectures, there remain the following problems. First, the existing methods typically use traditional convolutional layers or residual building blocks as the fundamental unit for architecture search [13], which neglects the design of other stacking blocks and often leads to limited variety of discovered network connectivity architectures [14]. Second, current methods take long time for searching optimal network architectures, and there are still many issues to be addressed for further optimization. Finally, PSO-based methods rely excessively on global best particles and individual best particles when calculating particle velocities and updating positions, which restricts the effective utilization of the search space and tends to make the search process fall into a local optimum [12].

To address these issues, a residual network design method based on PSO (psoResNet) is proposed in this paper, with the main purpose to design an optimal architecture search strategy for deep residual networks based on an optimized Particle swarm optimization algorithm. In our method, automatic search of low-complexity CNN architectures can be accomplished in a relatively short time, while having excellent performances in image classification tasks. The proposed psoResNet algorithm is inspired by psoCNN [11], and the best network architecture is finally selected by encoding different network architectures with the global optimal and individual optimal as the reference basis for network update. To address the issue that the current algorithms are often limited to a single search architecture, in this paper, a residual architecture is adopted as the architectural search cell to enhance the exploration depth of the CNN network. In addition, the search process of the current algorithm is prone to fall into local optimum. To this end, we optimize the difference calculation method and the velocity update mechanism of the PSO algorithm, which is dedicated to exploring the search space more deeply and increasing the diversity of particles in the search process. To further improve the search efficiency of the algorithm, we speed up the adaptation evaluation process by reducing the number of parameters of the search cell. Moreover, we optimize the initialization range setting problem to find the optimal search space. The main contributions of this paper are listed as follows:

- A depth initialization strategy is devised to replace the manual setting of depth ranges. Additionally, we introduce residual block incorporating depth-wise separable convolutions and the convolutional block attention module (CBAM) into the task of automatic network search, serving as architectural search unit.
- A novel method of particle difference calculation and a particle velocity update mechanism are proposed to achieve blank region search in the search space, improving the flexibility of the exploration process and the diversity of particles.
- We build a crime-dataset containing a variety of criminal images, the classification of which provides important clues for criminal investigation and crime cracking.

The remainder of the paper is organized as follows. Section 2 describes the research methods related to the evolutionary neural architecture search. Section 3 describes the framework and details of the steps of the proposed algorithm. Sections 4 and 5 describe the experimental design and the experimental results and analysis, respectively. Section 6 describes the conclusion and future work.

2 Related Work

The EC based deep neural architecture search methods focus on designing network architectures and parameters by simulating the evolution of species or the behavior of populations in nature. In this section, we will mainly discuss GAs, PSO, and other related EC methods for deep architecture optimization.

2.1 Deep architecture search via GAs methods

Genetic algorithms have been employed for handling search tasks in neural network architectures [17]. Xie et al. [18] previously connected GAs to CNN structural design by introducing a fixed-length binary encoding strategy, where different characters represent the connectivity between nodes. However, the fixed-length encoding strategy searched for a network architecture with a single depth, potentially resulting in suboptimal network performance. Due to the unpredictable depth of the network, Sun et al. [19] proposed a variable-length encoding strategy to search for superior network architectures by evaluating CNN architectures of different depths. However, one drawback is that the fitness evaluation process is time-consuming, leading to a less efficient search. To address this issue, Sun et al. [20] introduced an asynchronous computation component and cache component to reduce the frequency of fitness evaluations. The algorithm employed skip connection blocks as search

cells to explore deeper neural networks. During fitness evaluation, individuals are first queried within the cache files to prevent duplicate evaluations. To design CNN architectures under computational resource constraints, Li et al. [21] proposed an adaptive penalty algorithm and an adaptive repair method. This algorithm constrained the complexity of the CNN architecture by correcting the architecture of infeasible solutions and increasing the selection probability of feasible solutions. The GAs-based search method has high performance in global network search tasks, with the drawback that the method is slow to converge.

2.2 Deep architecture search via PSO methods

Particle swarm optimization algorithm [8] is one of the evolutionary computations that is essentially inspired by the flight pattern of swarms of birds. PSO randomly generates initial populations, and the individuals in these populations are called particles. During the iterative optimization process, the particles continuously adjust the direction and position of their search for the optimal value by tracking the individual optimal value (P_{best}) and the global optimal value (g_{best}). Due to its simplicity of implementation and fast convergence speed, particle optimization has gradually been applied in automated neural network design [22].

Sun et al. [23] introduced a flexible convolutional self-encoder (FCAE) to overcome the limitations on the number of convolutional and pooling layers imposed by traditional self-encoders. FCAE utilized the PSO algorithm to update the network architecture, where the parameter values of different layers are represented by each particle, along with a variable-length encoding speed update mechanism. The algorithm was evaluated on four datasets, and the results demonstrated the highest classification accuracy. However, the network architecture was solely composed of multi-layer convolution and pooling layers, making the network model relatively simple. Li et al. [24] employed the binary quantum particle swarm optimization (BQPSO) algorithm to simplify the architecture search process and minimize manual intervention. They proposed a new binary encoding strategy and quantum behavioral evolution strategy to ensure the classification performance of the search results. Experimental evidence confirmed the stability and robustness of the algorithm. However, the focus of the algorithm lay on optimizing PSO and overlooked the exploration of more complex network architectures. Jiang et al. [25] proposed a decomposition-based multi-objective Particle swarm optimization algorithm (MOPSO/D). The network classification error rate and the number of parameters were also optimized to achieve the design of a low-complexity CNN architecture. Wang et al. [26] proposed an alternative-assisted CNN structural block search method based on the PSO algorithm. The algorithm reduced computational costs by reducing the data volume and resolution of the dataset. It employed SVM as a proxy model to screen underperforming particles and avoid unnecessary fitness evaluation. However, the algorithm searched for architectures by solely stacking dense blocks, which could limit the exploration of flexible architectures. Considering the ease of implementation of the PSO search process, Zhang et al. [27] proposed the PSO-PRCN method to classify transmission line faults, facilitating the maintenance of faulty lines. This suggested potential applications of Particle swarm optimization algorithm-based architecture search methods in the engineering field to enhance the efficiency of network architecture design.

2.3 Deep architecture search via other EC methods

Apart from the commonly used PSO and GAs, various other approaches have been employed for NAS tasks. Based on genetic programming (GP), Bi et al. [28] developed a feature learning algorithm for binary and multi-classification tasks. To design a flexible architecture, they introduced a program architecture based on strongly typed GP (STGP). Additionally, new function sets, and terminal sets were designed for the program architecture based on convolution and pooling operations. For the task of skin lesion classification, Kwasigroch et al. [29] utilized the hill-climbing algorithm (HCA) and network morphological operations to explore the search space and design neural network architectures. The algorithm progressively expanded the initial network and reused weights, resulting in performance improvements and computational cost reductions. Suganuma et al. [30] employed Cartesian Genetic Programming (CGP) to encode CNNs. Convolutional layers and tensor connections were utilized as node functions to reduce the search space. Additionally, the authors proposed rich initialization and early network termination strategies to accelerate the search process. The method effectively identified well-performing architectures in a short time. Tian et al. [31] introduced an architecture search method for Liquid State Machines (LSMs). The method employed a simulated annealing algorithm to search for multi-liquid architectures, determine the number of neurons, and optimize the parameters of each small liquid. The design of low-complexity architectures was achieved by balancing the number of neurons against accuracy.

3 Proposed Algorithm

In this study, the entire search process consists of three main parts: initialization, fitness evaluation, and

particle update. During the initialization phase, we incorporate a depth initialization strategy to constrain the search space. Simultaneously, a new residual architecture block is designed to enable diverse network architecture exploration and reduce the number of parameters. In the fitness evaluation phase, we train the dataset for a small number of epochs to shorten the evaluation time. In the particle update phase, we introduce a novel particle difference calculation method and velocity calculation method to effectively utilize the search space and overcome the issue of local optimal search results. The algorithm focuses on designing an efficient network architecture by continuously updating the particle architecture and parameters based on fitness evaluation results. When the number of iterations reaches a predefined value, the current optimal particle architecture represents the final network architecture. The specific algorithm flow is illustrated in Figure 1.

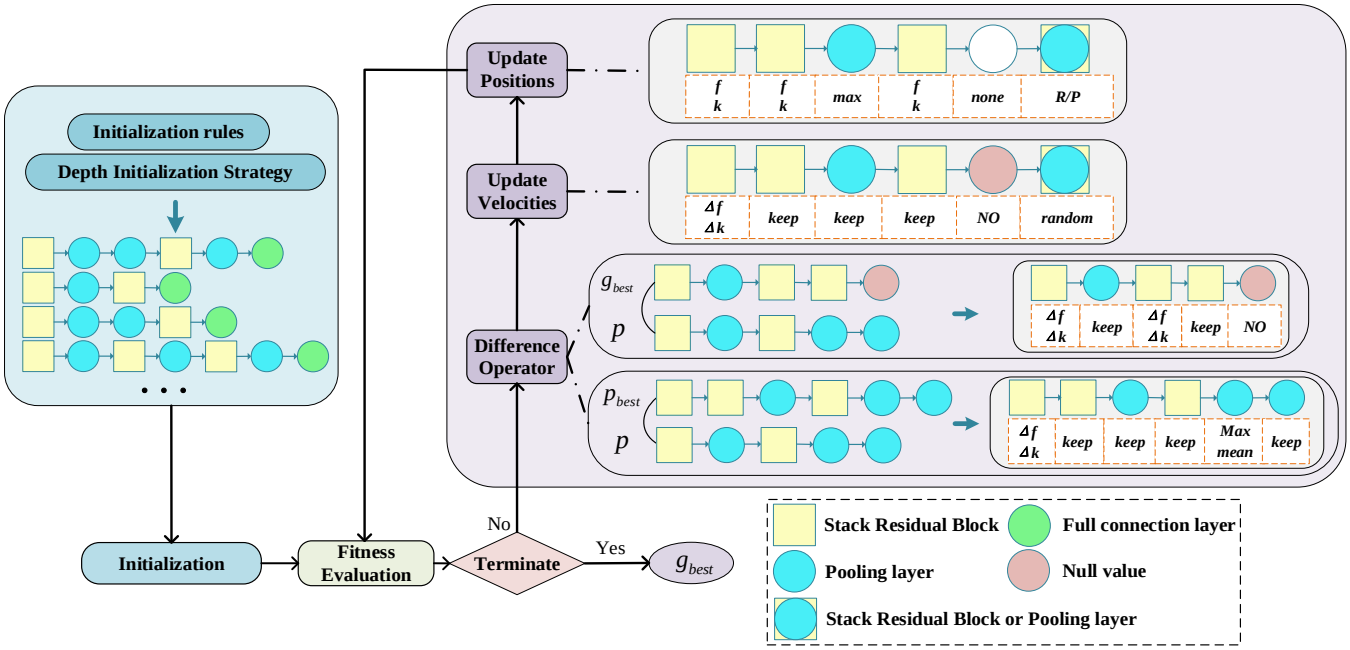


Figure 1 Framework of the proposed psoResNet algorithm.

3.1 Particle initialization

Before initializing the particles, the resulting particles require special encoding. This enables it to compile the architecture and parameters of each layer of the particle based on the encoding when it enters the fitness evaluation phase.

The coding architecture in this paper consists of three main types of layers: Stack Residual blocks, pooling layers, and fully connected layers. In this paper, the different types of layers within the particle architecture are denoted by R, P, and FC respectively. One of the model architectures for Stack Residual blocks is shown in Figure 2, which uses residual connections to address the issue of a single network design. To speed up the network evaluation and achieve a more lightweight architecture design, both convolutional layer 1 and convolutional layer 2 use depth-wise separable convolution. CBAM [16] is a lightweight attention mechanism combining the channel attention mechanism with the spatial attention mechanism, which can assign higher weights to information-rich regions while suppressing redundant feature information. Therefore, this paper connects CBAM with depth-wise separable convolution to form a residual architecture for feature extraction with fewer parameters. The connection of the residual blocks in the network, the size of the convolutional kernel of the corresponding convolutional layer, and the number of output feature maps are the parameters that the algorithm will search for in this paper. The initial pooling layer contains two main types, max-pooling and average pooling.

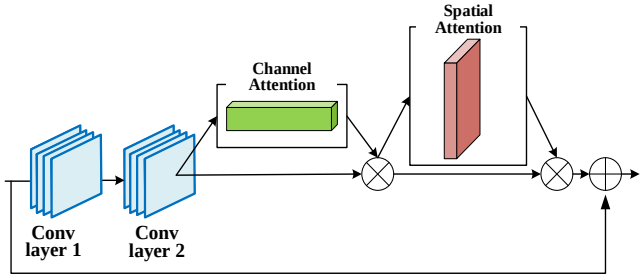


Figure 2 Architecture of the Stack Residual Block

Different datasets vary in terms of image size, data

volume, and classification difficulty, so a targeted design of network architectures with different depths can achieve better classification performance. Since the optimal depth of the network cannot be predicted, this study uses a variable-length coding strategy to represent different particle architectures. Specifically, a range of particle depth initialization is given in advance, and each particle chooses a random value from this range as the initial architecture depth. The variable-length encoding strategy allows the particles to adaptively adjust the network depth during the iterative process, and thus find the optimal network depth by evaluating networks of different lengths. Figure 3 plots the encoding format for two particles of different lengths.

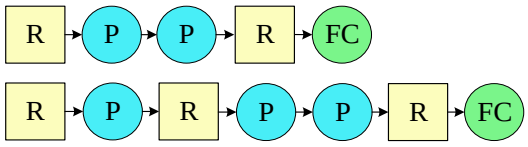


Figure 3 The particle encoding format

The variable-length encoding strategy addresses the challenge of searching for the optimal network depth. However, if the depth range is predefined solely based on empirical knowledge during initialization on different datasets, it may lead to issues such as setting the range too small, preventing the discovery of the optimal depth, or setting it too large, resulting in an excessively large search space and increased search time. Therefore, we draw inspiration from manually designed networks and introduce a depth initialization strategy as follows:

- We replace the convolutional layers of the residual blocks in the ResNet network with depth-wise separable convolutions **Error! Reference source not found.**, denoted as R_N. Five different depths of architectures are employed, including ResNet18, ResNet34, ResNet50, ResNet101, and ResNet152. In ResNet, the definition of the number of layers is calculated based on individual layers (e.g., a residual block containing two convolutional layers is counted as 2 layers) [32]. However, in our research, we consider one residual block as one layer. Therefore, we convert the number of layers in ResNet to the layers encoded in psoResNet. To ensure that the network can adequately learn data features, we set the minimum number of layers to 4. The depth ranges encoded are shown in Figure 4.
- A binary tree traversal is used to train on the R_N network architecture, and the training sequence is shown in Figure 4. It is worth noting that the whole traversal process is not to train each network architecture once. To minimize the time used for traversal, the dataset is first trained on R_N50, and the decision to reduce the number of network layers is made by determining whether the training results are overfitting. The training process utilizes the Adam optimizer with a learning rate of 0.001 and runs for 200 epochs.
- We determine the structural depth that produces the best classification results. Using this depth range as a reference, we define the initial depth range for network initialization.

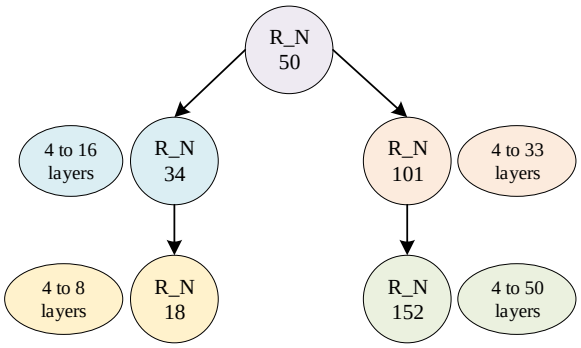


Figure 4 Binary tree traversal search depth process

The initialization process is shown in Algorithm 1. A different length is randomly selected for each particle based on a pre-defined range of network depths. In the initialization process, we adopted an initialization rule to ensure the viability of each generated particle. In this process, the first layer of the particle is a residual layer, and the last layer is a fully connected layer. Regarding the number of pooling layers within the particle, we have introduced a limiting rule based on the size of the input image. The rule is specifically formulated as follows: $x \le \log_2(size)$, where x represents the number of pooling layers, and size denotes the size of the input image. This constraint ensures that the algorithm can effectively handle various input images. In Algorithm 1, Add_R(f, k) represents adding a residual block to a particle, where the number (f) and size (k) of the convolution kernels are randomly selected within the specified range. Add_P(max, $mean$) represents adding a max-pooling or average-pooling layer to a particle. Add_FC(·) denotes adding a fully connected layer to a particle.

3.2 Fitness evaluation

Fitness evaluation plays an essential role in PSO that provides a quantifiable measure of quality differences between particles. This step is helpful in selecting the best architectures during iterations and subsequently adjusting particle velocities and positions. Specifically, each particle is decoded and constructed into a CNN model. We then feed the training dataset into this network and train it using an Adam optimizer

with a learning rate of 0.001. Finally, we output the classification accuracy on the testing dataset as the fitness value. Considering that the primary purpose of fitness evaluation is to compare the relative quality of particles and to reduce the iteration time required, we perform evaluations for each particle using only one epoch.

Algorithm 1 Particle Swarm initialization

Input: Swarm size(N), minimum number of layers (l_{min}), maximum number of layers (l_{max}), Minimum number of feature maps (f_{min}), Maximum number of feature maps (f_{max}), minimum convolution kernel size (k_{min}), maximum convolution kernel size (k_{max}),

Output: The initialized particles.

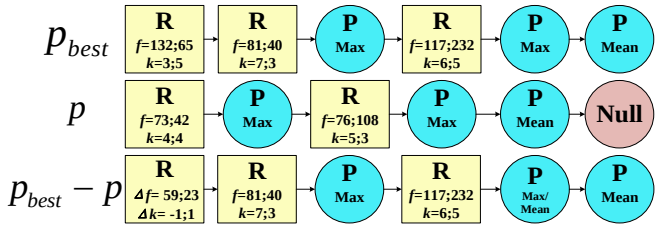
```
for i in N do
    l = rand [l_min , l_max]
    for j in l do
        if j==1 then
            P_i(j)=Add_R (f , k);
        else if j==l then
            P_i(j)=Add_FC (·);
        else
            type = rand (R , P);
            if type = R then
                P_i(j)=Add_R (f , k);
            else if type = P then
                P_i(j)=Add_P (max , mean);
            end
        end
    end
end
Return P_1⋯P_N
```

3.3 Particle difference calculation

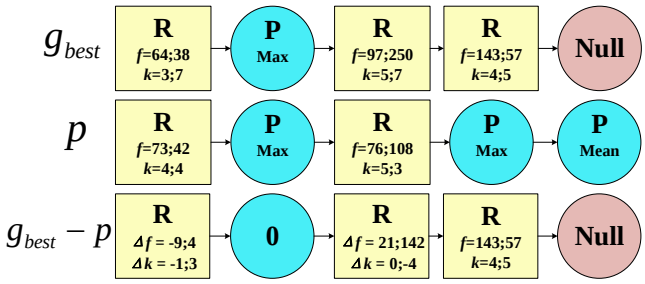
After completing the particle fitness evaluation, each particle can select P_{best} and g_{best} based on its fitness value and calculate its difference from P_{best} and g_{best}. As different particles usually have different lengths, null values are used to fill in the shorter particles when calculating the difference and prioritize the architecture of P_{best} and g_{best} as the benchmark for evaluation.

To make the exploration trajectories of particles more diverse, the calculation process not only considers the differences in layer types between particles but also the differences between parameters when the corresponding types are the same. Where f and k denote the number of output feature maps and the size of the convolution kernel of the two convolutional layers in Stack Residual Blocks, respectively. Figure 5 presents the results of the difference calculation for particles with P_{best} and g_{best} .The specific calculation rules are as follows:

- When the layer type is the same between two particles, the difference between the layer parameters is calculated. For convolutional layers, the value of the difference between the parameters of the current layer of the particle compared to P_{best} or g_{best} is denoted by Δf and Δk. For pooling layers, when the layer types between two particles are different, denoted by Random, which indicates that the type of the pooling layer will be reselected;
- When the layer type differs between particles, the layer type and parameters corresponding to P_{best} or g_{best} are output for the corresponding particle difference.
- When the P_{best} or g_{best} particle is shorter, the difference is filled with null values and the corresponding particle difference is also output with null values.
- When the length of particle P is shorter, the null filled part corresponds to the difference value of P_{best} or g_{best} for the layer type and parameters.



(a) The difference between particles and P_{best}



(b) The difference between particles and g_{best}

Figure 5 Particle difference calculation.

3.4 Particle velocity calculation

Based on the rules for particle difference calculation described above, the difference between each particle relative to the global optimal g_{best} and the individual optimal P_{best} can be derived. The velocity of each particle can then be calculated based on the difference between it and the best particle. The original Particle swarm optimization algorithm only considers $(g_{best} - P)$ and $(P_{best} - P)$ when calculating particle velocities. Building upon this, we propose a velocity

calculation method based on random terms. The null values are first used to complement the shorter difference group so that $(g_{best} - P)$ and $(P_{best} - P)$ have the same length. Next, each block in the two difference groups is assigned a random value of [0,1] and the velocity of each particle is calculated based on equation (2).

$$h = rand\ [0,1], \tag{1}$$

$$v = \begin{cases} g_{best} - P, & h \le C_g \\ p_{best} - p, & C_g < h \le 2C_g \\ rand(R,P), & 2C_g < h \le 1 \end{cases}, \tag{2}$$

Where h is a random floating-point number belonging to [0,1]. To balance the consideration between the global best particle and individual best particle, we introduce a decision factor, C_g, to control the similarity between particle updates and the best particle. Furthermore, we introduce a random term that increases the chance of exploring new architectures during the particle update process. In this process, we allow particles to select the global best particle and individual best particle with equal probability as a reference, ensuring the comprehensive consideration of both global and local information. Before calculating the velocity, the value of h needs to be assigned to each of the blocks corresponding to $(g_{best} - P)$ and $(P_{best} - P)$, as shown by h in Figure 6.

Next, the final velocity is assigned by iterating through the h values of each corresponding block. When $h \le C_g$, the particle velocity inherits the properties of $(g_{best} - P)$. When $C_g < h \le 2C_g$, the particle velocity inherits the properties of $(P_{best} - P)$. When $2C_g < h \le 1$, the velocity of the particle is output as Rand, which means that the particle velocity is a randomly taken value. This thought of random number-based velocity calculation is similar to the mutation process in GAs, which inherits some of the excellent structural information while exploring more possibilities of particle architecture and parameter matching. To better understand the particle velocity calculation method proposed in this paper, the solution process for the velocity calculation is plotted in Figure 6. It will be described later how to update the particle position information based on the obtained particle velocities.

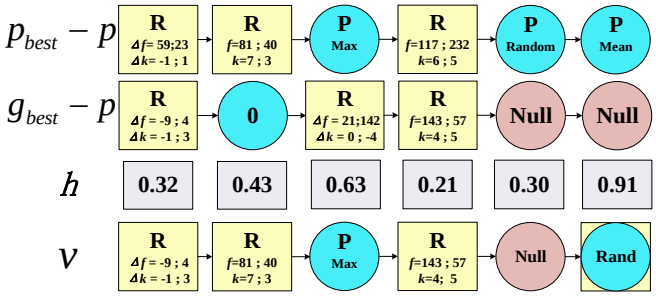


Figure 6 Particle velocity calculation

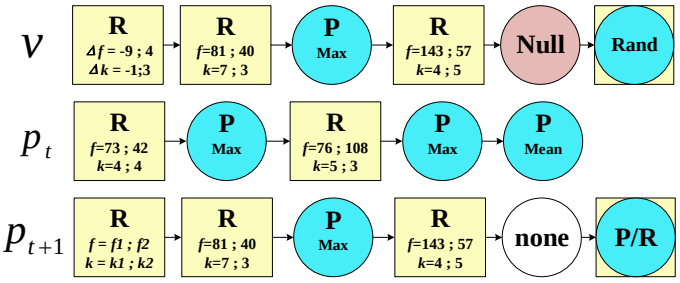


Figure 7 Example of particle position update process

3.5 Particle position update

The process of particle position updating is also the process of changing the architecture of the network. By calculating the particle velocities, we obtain a particle velocity value that incorporates the global and individual extremes. Based on the corresponding particle velocities, the particle architecture and parameters can be optimized.

Our particle update process is depicted in equation (3). In this equation, i and j represent the j-th layer of the i-th particle. In the next iteration, the changes in the type and parameters of layers within a particle will depend on the current particle's parameters and velocity. Specifically, we determine the layer type of $P(t+1)$ by examining the layer type of velocity in the j-th layer of the particle. We then determine the layer parameters of $P(t+1)$ by considering both velocity parameters $v_{i,j}(t)$ and the current parameters $P_{i,j}(t)$ of the particles.

$$P_{i,j}(t+1) = P_{i,j}(t) + v_{i,j}(t). \tag{3}$$

For a more intuitive understanding of how the parameters within the particles change, an example graph of the position update of a single particle is given in Figure 7. For the corresponding R in velocity, if the parameters are represented by Δf and Δk, the updated particle parameters f and k will be randomly selected in the range $[f, f + \Delta f]$ and $[k, k + \Delta k]$ respectively. Otherwise, the corresponding types and parameters in the velocity are directly replaced. Where

$f_1 = rand\ [73-9\ ,\ 73]$, $f_2 = rand\ [42\ ,\ 42+4]$, $k_1 = rand\ [4-1\ ,\ 4]$, $k_2 = rand\ [4\ ,\ 4+3]$. Null means that the corresponding layer of this particle should be deleted, and Rand means that the particle reselects the type and parameters of the corresponding layer at random. Once the particle positions have been updated, the FC layer is added afterwards to form the particle architecture for the next iteration.

4 Experimental Studies

The proposed algorithm was tested not only with commonly used benchmark image classification datasets, but also with a newly constructed crime-dataset. The datasets used for the experiments, the parameter settings, and the comparison models are described in detail next.

4.1. Datasets

In this experiment, we have chosen four public benchmark datasets to evaluate the proposed algorithm, which are MNIST-RD [33], Convex, MNIST-Fashion [34], and CIFAR10 [35].

Table 1 Overview of the datasets.

Dataset	Input size	classes	training	test
MNIST-RD	28×28×1	10	12,000	50,000
Convex	28×28×1	2	8,000	50,000
MNIST-Fashion	28×28×1	10	60,000	10,000
CIFAR10	32×32×3	10	50,000	10,000
crime-dataset	64×64×3	13	9680	2429

The MNIST-RD contains 10 categories of rotated Figures, which contain more additional information not related to classification information than the MNIST dataset, making the network more challenging. The MNIST-Fashion dataset is used for classifying different clothing categories. It shares the same image size and number of categories as the MNIST-RD dataset, but it poses a greater challenge than recognizing digits. The Convex dataset is a black-and-white image containing geometric shapes and is used to determine whether an image is convex or not. The CIFAR10 dataset contains ten categories of natural objects, which are difficult to classify in terms of image size, classification category, noise, and image rotation, and can be used to evaluate the performance of the algorithms in this paper.

Besides evaluating our algorithms by using a benchmark dataset, we also employ a customized dataset called the crime dataset. Images captured at crime scenes provide important clues for forensic analysis and thus play a pivotal role in criminal investigations. We collected numerous crime scene images under different time, lighting conditions, and shooting distances. After data cleaning (including removing duplicate images and standardizing image size), we obtained a dataset of 12,109 images covering 13 different categories. These categories include bloodstains, cars, telephotos, doors and windows, fingerprints, interior scenes, floor plans, shoe prints, skin, tattoos, tools, tires, and tire indentations. In the experiment, 80% of the data is used for training and the remaining 20% is used for validation. Figure 8 shows the sample images from the crime scene image database. Table 1 describes the input size, the number of categories and the data size used for training and testing of the above dataset.

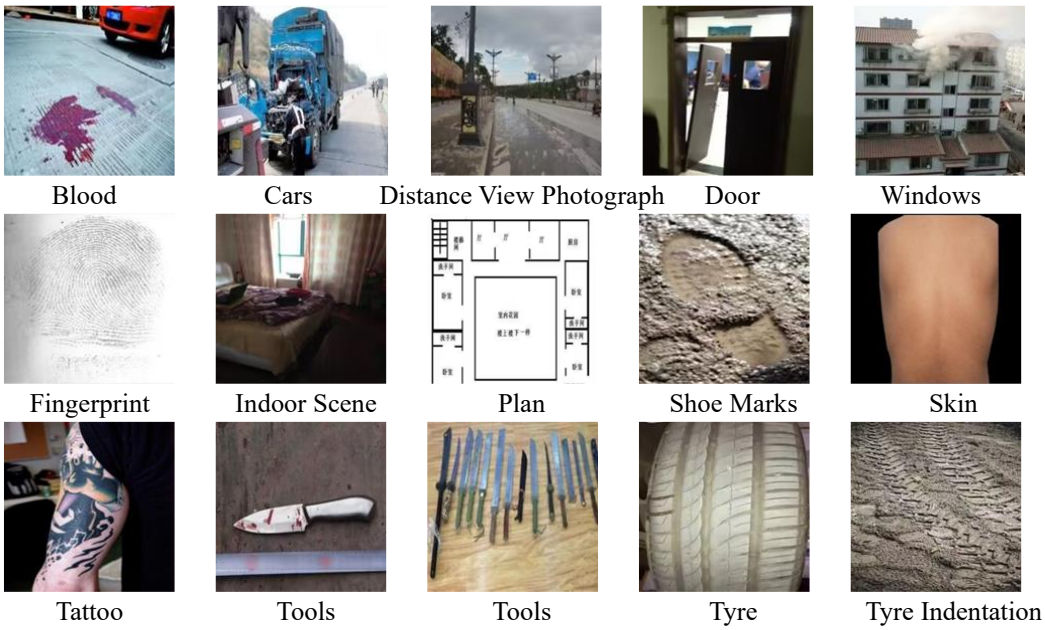


Figure 8 Examples of images from various types of crime-dataset

4.2 Experimental parameter settings

The experiments were accelerated by NVIDIA GeForce RTX 2080Ti, the framework for deep learning was TensorFlow 1.14.0, and the software programming environment was Python 3.6. For the algorithm proposed in this paper, the parameter settings were divided into two categories: different datasets with the same parameter settings and different datasets with different parameter settings. Table 2 lists the settings that have the same parameters for all datasets.

Table 2 Algorithm parameter settings. For different datasets with the same parameter settings.

Iteration parameters	Value
Swarm size	20
Number of iterations	10
$C_g(0, 0.5)$	0.4
Minimum number of feature maps	3
Maximum number of feature maps	256
Minimum convolution kernel size	3×3
Maximum convolution kernel size	7×7
Minimum number of layers	4
Epochs of fitness evaluation	1

Swarm size represents the size of the initialized particle population, and each particle represents a network architecture. To search for the optimal network architecture as quickly as possible, the swarm size is set to 20, and the number of iterations represents the number of times each particle changes position during the search process, which is set to 10. The first step of the algorithm is to initialize the particles, and the architecture parameters in the initial swarm are selected from a set range. To reduce the search time of the particle iteration process, 1 epoch is used to train the particles when they are evaluated for fitness. The C_g is used to effectively adjust the similarity of particles to the best particle. A lower C_g value increases the randomness of particles, while a higher C_g value causes particles to overly rely on the best value, thereby increasing the likelihood of getting stuck in local optima. Therefore, we set the value of C_g to 0.4, which ensures a good balance between optimal search and random exploration.

In the process of network depth initialization, different datasets should correspond to different network depths. Based on the depth setting strategy proposed in this paper, we use the binary tree traversal method to experiment on each of the five datasets and obtain the setting values for the parameter ranges. Table 3 lists the network depth ranges corresponding to different datasets.

Table 3 Depth range settings. Different parameter settings for different datasets.

Dataset	Range of depth
MNIST-RD	4 to 8
Convex	4 to 8
MNIST-Fashion	4 to 8
Crime	4 to 8
CIFAR10	4 to 16

4.3 Peer Competitors

To fully illustrate the ability of the algorithms in this paper to efficiently search for high-performance CNN architectures, we use several excellent manually designed architectures and network architecture search algorithms based on optimization methods for performance comparison.

For different datasets, the manual design architectures compared in this paper include the PCANet-2, RandNet-2 and LDANet-2 algorithms designed by Chan et al[36], VGGNet [37], ResNet [32], MobileNet [38], GoogleNet [39], AlexNet [40], SqueezeNet [41], ShuffleNetV2 [42] and Ghost-ResNet-56 [43], which are currently the most mainstream feature learning methods.

Several automatic generation methods for deep network architectures are also been selected for comparison in this paper, including IPPSO [10], EAS [44], EvoCNN [9], psoCNN [11], psoGroup [12], ME-HDSS [45], CH-CNN [21], sosCNN [46] and FGP [47]. The above listed are the better algorithms in the current NAS approach, which do not require extensive domain knowledge to aid the design and can be used to evaluate the performance of the psoResNet.

5 Experimental results and analysis

In this section, we present the results of psoResNet on different datasets and compare them with the state-of-the-art network models and EC-based NAS methods. We analyze the performance of the psoResNet search architecture from three perspectives: the accuracy of the search results, the number of parameters in the search results, and the search time. Additionally, we utilize statistical testing methods for a more comprehensive model evaluation and design ablation experiments to analyze the algorithm's performance.

5.1 Overall Results

Referring to the above experimental parameter settings, the classification network is optimized for different datasets. The average accuracy and best accuracy values obtained by PsoResNet after ten runs on different datasets are listed in the experimental results. Table 4 compares the results of the proposed

algorithm (psoResNet) with other benchmark methods on the MNIST-RD and Convex datasets. To ensure fairness in the comparison, the data we compared are the original results as they are publicly reported.

From Table 4, it can be observed that, compared to traditional CNN algorithms, psoResNet achieves the best classification accuracy in terms of both mean and best accuracy. Furthermore, the search performance of the psoResNet is at a better level compared to the five CNN structural search methods based on EC. While the psoGroup algorithm, which is the best-performing baseline method among the compared algorithms, achieves similar classification accuracy to psoResNet on the Convex dataset (outperforming psoResNet by 0.02%), psoResNet achieves a maximum classification accuracy of 96.84% on the MNIST-RD dataset, surpassing psoGroup by 0.16%.

Table 4 psoResNet classification accuracy (%) for the MNIST-RD and Convex benchmark datasets compared to different models. The symbol "-" indicates that there are no publicly reported results.

Model	MNIST-RD ↑	Convex ↑
PCANet-2 [36]	91.48%	95.81%
RandNet-2 [36]	91.53%	94.55%
LDANet-2 [36]	95.48%	92.78%
EvoCNN(mean) [9]	94.54%	94.61%
EvoCNN(best) [9]	94.78%	95.18%
IPPSO(mean) [10]	-	87.94%
IPPSO(best) [10]	-	91.52%
psoCNN(mean) [11]	93.58%	96.10%
psoCNN(best) [11]	96.42%	98.30%
FGP(mean) [47]	91.56%	98.16%
FGP(best) [47]	92.63%	98.46%
psoGroup(mean) [12]	96.10%	98.37%
psoGroup(best) [12]	<u>96.68%</u>	**98.64%**
psoResNet(mean)	96.11%	97.67%
psoResNet (best)	**96.84%**	<u>98.62%</u>

*The best and second-best results are bolded and underlined, respectively.

Table 5 presents the validation results on the MNIST-Fashion dataset, including the classification accuracy and network parameters for psoResNet and other comparative algorithms. The network generated by the psoResNet method achieves the lowest number of parameters, with only 0.17 million parameters. This is even 0.33 million less than that of SqueezeNet [41] which has the lowest number of parameters. While the classification accuracy is improved by 4.36%. This indicates that our method significantly reduces network redundancy during the CNN architecture optimization process. Furthermore, the network with extremely low parameters searched by psoResNet performs impressively, boasting not only the highest average accuracy compared to other networks but also the best classification accuracy which is only 0.17% lower than EvoCNN [9].

Table 5 psoResNet test accuracy (%) and parameters(M) on the MNIST-Fashion dataset compared to different models.

Model	Parameters ↓	Accuracy ↑
GoogleNet [39]	101M	93.7%
AlexNet [40]	60M	89.9%
SqueezeNet-200 [41]	0.5M	90.0%
psoCNN(mean) [11]	1.8M	90.85%
psoCNN(best) [11]	1.4M	91.90%
EvoCNN(mean) [9]	6.52M	92.72%
EvoCNN(best) [9]	6.68M	**94.53%**
sosCNN(mean) [46]	3.42M	93.83%
sosCNN(best) [46]	2.3M	94.32%
psoResNet (mean)	<u>0.45M</u>	93.92%
psoResNet (best)	**0.17M**	<u>94.36%</u>

*The best and second-best results are bolded and underlined, respectively.

To show the impact on computational cost arising from the search strategy proposed in this study and the stacked low-complexity blocks, the computational time utilized by the algorithms in the architecture search is compared. For fairness of comparison, the same experimental setup as the comparison algorithms is used, where the particle population size is 20 and the particle iteration period is 10. All calculations are also performed on an NVIDIA GeForce RTX 2080Ti. Table 6 shows the search times (in minutes) and time compression ratios of the algorithms in this paper and the two compared algorithms of the same type on the three datasets MNIST-RD, Convex, and MNIST-Fashion.

As shown in Table 6, our algorithm consumes less computational resources compared to the other two algorithms during the architecture search. Specifically, on the MNIST-RD dataset, we only require 40 minutes to discover the best-performing network, surpassing the performance of the other two algorithms. This represents a 14.9% reduction in search time compared to the psoCNN algorithm and a 7.0% reduction compared to the psoGroup algorithm. For the MNIST-Fashion dataset, the psoResNet algorithm's search time is reduced by 20.4% compared to the psoCNN algorithm.

Referring to Table 4, we can deduce that for the Convex dataset, the psoResNet algorithm exhibits slightly lower accuracy than psoGroup. However, as indicated in Table 6, we can identify a relatively efficient architecture in just 24 minutes, signifying a 20% reduction compared to psoGroup. These results

illustrate that the particle initialization method and particle update mechanism proposed in this paper can evolve effective classification architectures with reduced computational costs.

Table 6 Comparison of the mean search time in minutes for our algorithm and other benchmark methods. The data in brackets indicates the compression ratio of the search time. The symbol "-" indicates that there are no publicly reported results. All experiments have been run on one NVIDIA GeForce RTX 2080Ti**.**

Model	MNIST-RD	Convex	MNIST-Fashion
psoCNN[11]	47(14.9%)	33(27.3%)	157(20.4%)
psoGroup [12]	43(7.0%)	30(20%)	-
psoResNet	**40**	**24**	**125**

We next test the algorithms using the CIFAR10 dataset. Table 7 compares the classification accuracy, the number of structural parameters, and the search time of the architecture search algorithm for the different algorithms on the CIFAR10 dataset.

We divide the compared models into two categories, manually designed networks and automatically designed networks. As can be seen from Table 7 shows that the psoResNet algorithm achieves a substantial improvement in both classification accuracy and reduction in the number of parameters compared to the manually designed network architecture. Although Ghost-ResNet-56 has fewer parameters than psoResNet, its classification accuracy is much lower than that of psoResNet. This indicates that the proposed algorithm can solve the difficulties of manually designed networks and design more novel and better network architectures.

Table 7 Comparison of experimental results with various benchmark methods in terms of accuracy, parameters and search time for the CIFAR10 dataset.

Model	Accuracy ↑	Parameters ↓	Time ↓
ResNet101	93.57%	1.7M	-
VGGNet	94.34%	20.34M	-
ShuffleNet V2	94.17%	2.47M	-
Ghost-ResNet-56	92.70%	**0.43M**	-
ME-HDSS(C100)[45]	93.65%	0.86M	-
EAS [44]	**95.77%**	23.4M	5(days)
CH-CNN(M=1.0)[21]	94.50%	1.04M	10(days)
psoResNet(mean)	94.42%	<u>0.81M</u>	**8.41(hours)**
psoResNet(max)	<u>94.73%</u>	1.12M	<u>8.94(hours)</u>

*The best and second-best results are bolded and underlined, respectively.

For architecture auto-search algorithms, the number of parameters of the psoResNet algorithm is roughly comparable to ME-HDSS and CH-CNN, but it has the highest classification accuracy. Although the EAS algorithm has a slightly higher accuracy than psoResNet, its network architecture has 18.16 times more parameters than psoResNet. Furthermore, psoResNet has the shortest search time among the baseline algorithms. The proposed algorithm can minimize the number of parameters without affecting the classification performance.

For the CIFAR10 dataset, there are several reasons why our algorithm completely wins in terms of search time. The first aspect is that the optimization approach is different; EAS use the idea of reinforcement learning, which explores the architecture space by reusing the weights of previous networks. CH-CNN use a genetic algorithm, which selects the best results by training and testing each network with 350 epochs. psoResNet is based on a Particle swarm optimization algorithm that uses only one epoch for training and evaluating the particles to select the best ones. The second aspect is that our algorithm uses a stacked network base block. This base block is a residual block of fused depth-wise separable convolution, which has fewer parameters than normal convolutions and contributes significantly to the reduction of computation time. Thirdly, when initializing the depth of particles, we abandon the conventional manual approach. This measure further reduces the risk of excessively long network search times caused by inappropriate parameter settings.

5.2 Overall evaluation of the psoResNet

The independent samples t-test [48] is used to test whether there is a statistically significant difference between two independent samples. Typically, we use the P-value to indicate this significance. When the P-value is less than 0.05, we consider that there is a significant difference between the two samples; otherwise, we believe there is not enough evidence to suggest a significant difference between them. To further evaluate the performance of the psoResNet, we used the independent samples t-test method to analyze our algorithm against other algorithms. We ran our algorithm ten times, treating the results as one sample dataset, and compared them against the results of the comparative algorithm, which formed another sample dataset. We conducted significance analyses separately for accuracy, parameter count, and search time across four datasets, and the specific results are presented in Table 8.

Table 8 indicates that all P-values show statistically significant differences, except for the CIFAR-10 dataset parameter for which the P-value exceeds 0.05. Notably, the significant differences in search time are particularly noticeable. On the CIFAR-10 dataset, there are no significant differences in

parameters between the algorithms, which is mainly because the comparative algorithms selected in this study focus on reducing parameters. As a result, there is not much variance in terms of parameters. However, the proposed algorithm enhances accuracy and reduces the search time. Combined with the data analysis in Section 5.1, the proposed algorithm can achieve network architecture optimization in a short period of time while maintaining high classification performance and low network complexity.

Table 8 P-values of Independent Samples t-Tests

Dataset	Accuracy-P	Parameters-P	Time-P
MNIST-RD	0.019	0.013	0.001
Convex	0.002	0.004	0.001
MNIST-Fashion	0.021	0.011	0.003
CIFAR10	0.046	0.166	0.001

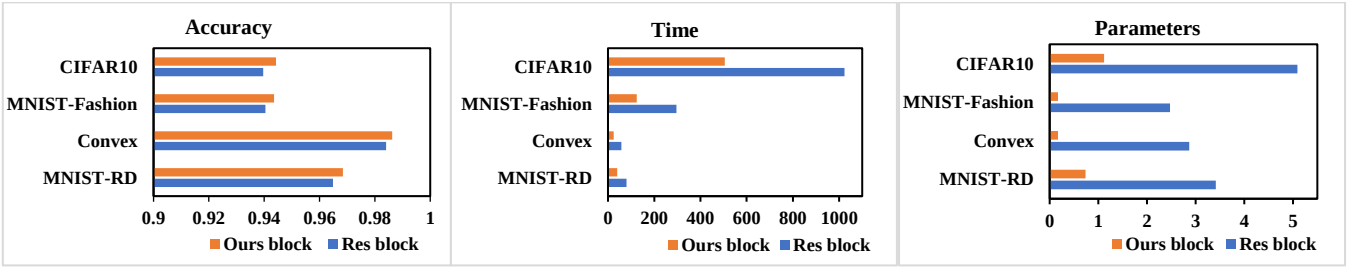
5.3 Search performance analysis of residual blocks

To validate the role played by the proposed residual blocks fusing depth-wise separable convolution and CBAM, we compare the effect of the original residual block with the fused residual block on four public datasets. Table 9 presents the performance of the two residual blocks in terms of accuracy, time and number of parameters. Further, to show the advantages of our block in structural search more visually, we plot the experimental results under different datasets as a bar chart, as shown in Figure 9.

Figure 9(a) plots the accuracy of the best particles searched, and it can be found that the Ours-block outperforms the Res-block in terms of classification performance. this mainly depends on our application of CBAM in the residual block. The attention mechanism allows the network to highlight not only the spatial level features, but also the channels with prominent features, thus the network focuses more on the object information during the iterative process. The residual connections, on the other hand, can effectively convey gradients and information flow, facilitating deeper training and optimization of the network. The experimental results show that by combining the attention mechanism and residual connectivity, we can build a more powerful and efficient network architecture. The time taken for the two block search processes and the parametric size of the search results are plotted in Figure 9(b) and Figure 9(c) respectively. Since depth-wise separable convolution is used, it is clear that Our-blocks are more lightweight compared to Res-blocks. This reduces the search time to a large extent and reduces the number of parameters in the network. Therefore, fused residual blocks can achieve low computational consumption while maintaining good feature extraction performance.

Table 9 The comparison of accuracy, time, and structural parameters for each of the two iteration blocks. Where Ours-block represents the fused residual of the proposed block and Res-block denotes the original residual block.

	Accuracy		Time(minutes)		Parameters(M)	
Dataset	Res-block	Ours-block	Res-block	Ours-block	Res-block	Ours-block
MNIST-RD	96.48%	96.84%	80	40	3.41	0.736
Convex	98.40%	98.62%	58	24	2.866	0.172
MNIST-Fashion	94.04%	94.36%	296	125	2.472	0.168
CIFAR10	93.96%	94.42%	1023	505	5.09	1.118

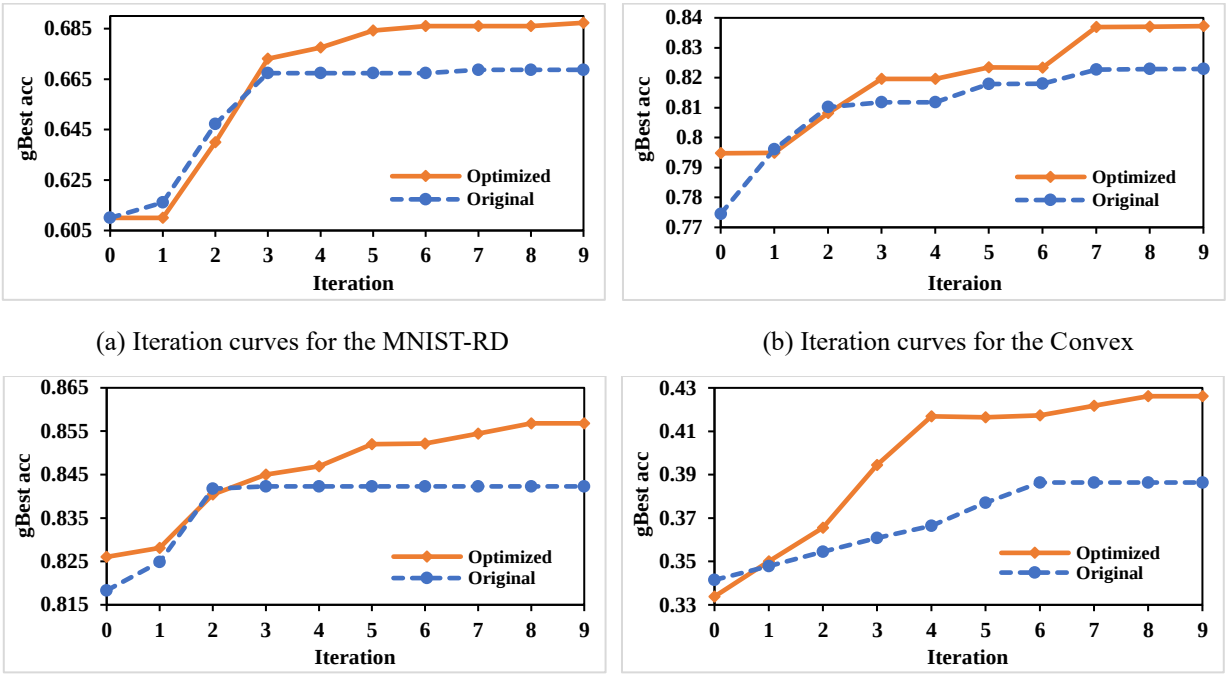


(a) (b) (c)

Figure 9 The comparison of the two iteration blocks in terms of accuracy, time, and architecture parameters respectively. Where (a) represents a comparison of the accuracy of the search results, (b) represents a comparison of the search time of the search process, and (c) represents a comparison of the number of parameters of the search results.

5.4 Discussion on the effectiveness of the particle update mechanism

Figure 10 plots the change in the global best particle fitness value over ten iterations for the four datasets MNIST-RD, Convex, MNIST-Fashion, and CIFAR10. The curve also reflects the ability of the algorithm to discover better particle architectures. To show the effectiveness of the particle update mechanism, we plot the curves before and after optimization in the same line graph. Original denotes an unoptimized particle iteration process, where only the particle difference calculation between layer types is considered and the particle velocity calculation without random term. Optimized is the particle iteration process for the psoResNet proposed in this paper. As a whole, the Optimized curves eventually achieve higher fitness values than the Original curves.



(a) Iteration curves for the MNIST-RD

(b) Iteration curves for the Convex

(c) Iteration curves for the MNIST-Fashion

(d) Iteration curves for the CIFAR10

Figure 10 The iterative change process of classification accuracy before and after optimization of the particle update mechanism under four datasets.

For the MNIST-RD dataset, the Original curve essentially stops growing from the third iteration onwards, eventually stopping at around 0.67. In contrast, the Optimized curve is always growing, and after ten iterations the final fitness value is 0.02 higher than the Original. For the Convex dataset, from the third iteration, the Optimized particle fitness value exceeds the Original particle fitness value and keeps growing. For the MNIST-Fashion dataset, the Original curve converges by the third iteration, while the Optimized curve has a clear increasing trend. For the CIFAR 10 dataset, the Optimized curve reach the Original optimal value in the early iterations. The results suggest that limiting the calculation of particle velocity to the layer architecture of g_{best} and P_{best} alone does not adequately explore promising networks. Considering the differences in parameters between particles and incorporating the variational ideas from GAs into the particle velocity update strategy can increase particle diversity, fully utilize the search space and prevent the iterative process from falling into local optimal.

5.5 Discussion on Evaluation Epochs

Evaluating the epochs required for particle training is a crucial parameter in this study. To validate whether setting the epoch to 1 is feasible, we performed particle evaluations with different epoch values (1, 5, 10 epochs) on the convex dataset.

Table 10 lists the final accuracy and search time for different epoch values. From the table, it can be observed that there is little difference in maximum accuracy and average accuracy across different epochs, but the search time increases significantly with the increase in epoch values. The main reason behind this phenomenon is that particle evaluation only needs to discover differences in quality between different particles, while more epochs would prolong the

architectural search time. Therefore, to minimize the time expenditure in the search process, one epoch is sufficient for our optimization.

Table 10 Results obtained with different numbers of epochs on the Convex dataset.

Epoch number	Accuracy (max)	Accuracy (mean)	Time (minutes)
1 epoch	98.62%	98.47%	24
5 epochs	98.58%	98.45%	136
10 epochs	98.64%	98.32%	293

5.6 Architecture search under crime-dataset

Tables 4, 5, and 7 statistically analyze the results of common classification benchmark datasets respectively, showing that our algorithm can find networks with high accuracy and few parameters relatively quickly. To further test the generalizability of our algorithm, we used 80% of the data for training and the remaining 20% for validation. The training process is performed using the Adam optimizer with an initial learning rate set to 0.001 and an epoch value of 300. Meanwhile, the psoCNN algorithm is used to implement the architectural optimization of the crime-dataset under the same parameter settings. The algorithm is repeated ten times and the average and maximum values are selected. Table 11 provides statistics on the classification results, the number of network parameters, and the architecture search time for different methods for the crime-dataset.

Table 11 The results of comparison experiments for the crime-dataset

Model	Accuracy ↑	Parameters ↓	Time(minutes) ↓
ResNet18	88.84%	11M	-
ResNet34	88.98%	21.3M	-
MobileNet	80.60%	3.2M	-
psoCNN(mean)	87.94%	12.6M	63
psoCNN(max)	88.43%	11.2M	59
psoResNet (mean)	<u>89.94%</u>	<u>1.05M</u>	<u>47</u>
psoResNet(max)	**90.05%**	**0.82M**	**45**

*The best and second-best results are bolded and underlined, respectively.

Table 11 shows that the proposed algorithm takes less time to search for the network with the best classification accuracy and the least number of parameters among the architectures and algorithms compared. To further illustrate the classification performance of the searched architectures on the crime-dataset, we are plotting the confusion matrix of classification results for the psoResNet algorithm, as shown in Figure 11.

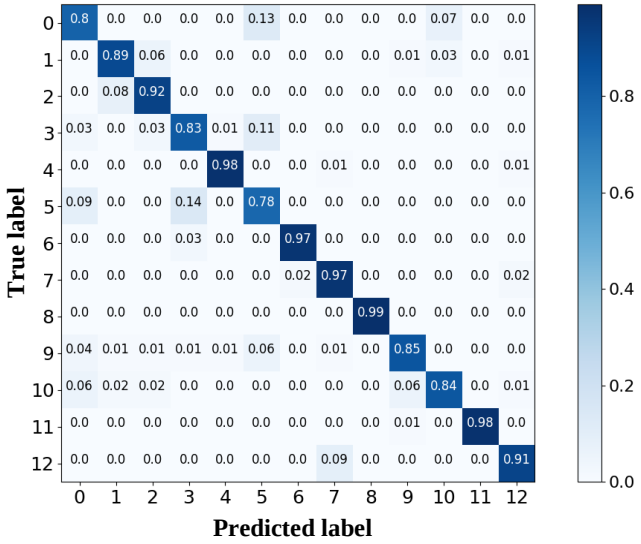


Figure 11 crime-dataset classification confusion matrix

In Figure 11, the labels 0 to 12 represent Blood, Cars, Distance View Photograph, Door-Window, Fingerprint, Indoor Scene, Plan, Shoe Marks, Skin, Tattoo, Tools, Tyre, and Tyre Indentation. The diagonal indicates the percentage of each type of criminal image that is correctly classified, while the off-diagonal indicates the percentage that is incorrectly classified. The lowest classification accuracy is 5 (Indoor Scene) because Indoor Scene contains many crime scenes, which are messy and may include bloodstains and other targets in the scene when framing. The categories Fingerprint, Skin, and Tyre have a higher classification accuracy, mainly because these images can be captured separately and are less likely to be affected by other targets.

The above experiments show that the proposed algorithm can effectively search for high-precision and low-complexity CNN architectures, which can meet the design requirements for CNN architectures in the field of image classification.

5.7 Discussion of Identified Models

Table 12 in the appendix lists the best CNN architectures searched by the psoResNet algorithm for different datasets. The third column shows the parameter values within each block. fliters1 and fliters2 indicate the number of filters for the two convolutional layers in the residual block, respectively, and the values in parentheses indicate the kernel size of the corresponding filters. The fourth column lists the number of parameters of the network architecture for the corresponding datasets.

By analyzing Table 12 we can conclude that the algorithm can make full use of the search space to design a more targeted network architecture by randomly selecting parameters in the range. The

number and size of filters in the architecture can no longer be a fixed value for different datasets. Moreover, the parameters of the architecture searched by our algorithm are small and can meet the needs of most scenarios and non-specialist researchers for network deployment.

6 Conclusion

Based on the Particle swarm optimization algorithm, we propose a novel search strategy for evolving CNN architectures more rapidly. By designing lightweight residual feature extraction block and depth setting strategy to speed up the search process of the network and solve the problem of limited network architecture design. Additionally, a more comprehensive particle update method is proposed to improve the search space utilization, which in turn leads to the design of a higher performance network architecture. The proposed algorithm is tested under several benchmark classification datasets and a self-built crime-dataset. The experimental results indicate that PsoResNet can design low-complexity architectures with relatively high accuracy for a given dataset at a low search cost.

For future work, the proposed algorithm can do adaptability studies on more neural network architectures, for example, exploring the effectiveness of the algorithm on AE [49] network design, and thus exploring more novel architectures. In addition, other EC methods such as differential evolution (DE) [50], and firefly algorithm (FA) [51] are considered to design different search strategies to handle more complex computer vision tasks in the NAS domain.

Acknowledgements

This work was supported in part by the National Natural Science Foundation of China [62201454]; and the International Science and Technology Cooperation Program of Shaanxi Province [2023-GHYB-04].

Reference

[1] Y. Zhao, K. Hao, H. He, X. Tang, B. Wei, A visual long-short-term memory based integrated CNN model for fabric defect image classification, Neurocomputing, 380(7) (2020), pp. 259-270.

[2] K. He, X. Zhang, S. Ren, J. Sun, Delving deep into rectifiers: surpassing human-level performance on imagenet classification, 2015 IEEE International Conference on Computer Vision (ICCV), Santiago, Chile, 2015, 1026-1034.

[3] C. Szegedy, L. Wei, Y. Jia, P. Sermanet, A. Rabinovich, Going deeper with convolutions, 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, USA, 2015, 1-9.

[4] E.J. Schiessler, R.C. Aydin, K. Linka, C.J. Cyron, Neural network surgery: combining training with topology optimization, Neural Networks, 144 (2021), pp. 384-393.

[5] T. Baeck, D. B. Fogel, Z. Michalewicz, Handbook of Evolutionary Computation, Taylor and Francis, CRC Press, Boca Raton, 1997.

[6] Y. Liu, Y. Sun, B. Xue, M. Zhang, G. G. Yen, K. C. Tan, A survey on evolutionary neural architecture search, IEEE Transactions on Neural Networks and Learning Systems, 34(2) (2023), pp. 550-570.

[7] C. Reeves, Genetic algorithms, Handbook of Metaheuristics, 2010, pp. 109-139.

[8] J. Kennedy, R. Eberhart, Particle swarm optimization, International Conference on Neural Networks (ICNN), Perth, Australia, 1995, pp. 1942-1948.

[9] Y. Sun, B. Xue, M. Zhang, G. G. Yen, Evolving deep convolutional neural networks for image classification, IEEE Transactions on Evolutionary Computation, 24(2) (2020), pp. 394-407.

[10] B. Wang, Y. Sun, B. Xue, M. Zhang, Evolving deep convolutional neural networks by variable-length particle swarm optimization for image classification, 2018 IEEE Congress on Evolutionary Computation (CEC), Rio de Janeiro, Brazil, 2018, pp. 1-8.

[11] F. Junior, G. G. Yen, Particle swarm optimization of deep neural networks architectures for image classification, Swarm Evolutionary Computation, 49 (2019), pp. 62-74.

[12] T. Lawrence, Z. Li, C. P. Lim, E. J. Phillips, Particle swarm optimization for automatically evolving convolutional neural networks for image classification, IEEE Access, 9 (2021), pp. 14369-14386.

[13] B. Wang, B. Xue, M. Zhang, Particle swarm optimisation for evolving deep neural networks for image classification by evolving and stacking transferable blocks, 2020 IEEE Congress on Evolutionary Computation (CEC), Glasgow, UK, 2020, pp. 1-8.

[14] T. Lawrence, L. Zhang, K. Rogage, C. P. Lim, Evolving deep architecture generation with residual connections for image classification using particle swarm optimization, Sensors, 21(23) (2021), pp. 7936(1-23).

[15] F. Chollet, Xception: deep learning with depthwise separable convolutions, 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 2017, pp. 1800-1807.

[16] S. Woo, J. Park, J.-Y. Lee, I. S. Kweon, CBAM: convolutional block attention module, 2018 European Conference on Computer Vision (ECCV), Munich, Germany, 2018, pp. 3-19.

[17] Z. Fang, J. Ren, S. Marshall, H. Zhao, S. Wang, X. Li, Topological optimization of the densenet with pretrained-weights inheritance and genetic channel selection, Pattern Recognition, 109 (2021) pp. 107608.

[18] L. Xie, A. Yuille, Genetic CNN, 2017 IEEE International Conference on Computer Vision (ICCV), Venice, Italy, 2017, pp. 1388-1397.

[19] Y. Sun, B. Xue, M. Zhang, Automatically evolving

CNN architectures based on blocks, arXiv preprint arXiv:1810.11875v2 (2019).
[20] Y. Sun, B. Xue, M. Zhang, G. G. Yen, J. Lv, Automatically designing CNN architectures using the genetic algorithm for image classification, IEEE Transactions on Cybernetics 50 (2020) 3840-3854.
[21] S. Li, Y. Sun, G. G. Yen, M. Zhang, Automatic design of convolutional neural network architectures under resource constraints, IEEE Transactions on Neural Networks and Learning Systems, 34(8) (2021), pp. 1-15.
[22] S. Kiranyaz, T. Ince, A. Yildirim, M. Gabbouj, Evolutionary artificial neural networks by multi-dimensional particle swarm optimization, Neural Networks, 22(10) (2009), pp. 1448-1462.
[23] Y. Sun, B. Xue, M. Zhang, G. G. Yen, A particle swarm optimization-based flexible convolutional autoencoder for image classification, IEEE Transactions on Neural Networks and Learning Systems, 30(8) (2019), pp. 2295-2309.
[24] Y. Li, J. Xiao, Y. Chen, L. Jiao, Evolving deep convolutional neural networks by quantum behaved particle swarm optimization with binary encoding for image classification, Neurocomputing, 362(14) (2019) 156-165.
[25] J. Jiang, F. Han, Q. Ling, J. Wang, T. Li, H. Han, Efficient network architecture search via multiobjective particle swarm optimization based on decomposition, Neural Networks, 123(3) (2020), pp. 305-316.
[26] B. Wang, B. Xue, M. Zhang, Surrogate-assisted particle swarm optimization for evolving variable-length transferable blocks for image classification, IEEE Transactions on Neural Networks Learning Systems, 33(8) (2022), pp. 3727-3740.
[27] L. Zhang, Z. Zhao, D. Zhang, C. Luo, C. Li, Particle swarm optimization pattern recognition neural network for transmission lines faults classification, Intelligent Data Analysis, 26(1) (2022) pp. 189-203.
[28] Y. Bi, B. Xue, M. Zhang, An evolutionary deep learning approach using genetic programming with convolution operators for image classification, 2019 IEEE Congress on Evolutionary Computation (CEC), Wellington, New Zealand, 2019, pp. 3197-3204.
[29] A. Kwasigroch, M. Grochowski, A. Mikolajczyk, Neural architecture search for skin lesion classification, IEEE Access, 8 (2020), pp. 9061-9071.
[30] M. Suganuma, M. Kobayashi, S. Shirakawa, T. Nagao, Evolution of deep convolutional neural networks using cartesian genetic programming, Evolutionary Computation, 28(1) (2020), pp. 141-163.
[31] S. Tian, L. Qu, L. Wang, K. Hu, W. Xu, A neural architecture search based framework for liquid state machine design, Neurocomputing, 443(5) (2021), pp. 174-182.
[32] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, pp. 770-778.
[33] H. Larochelle, D. Erhan, A. C. Courville, J. Bergstra, Y. Bengio, An empirical evaluation of deep architectures on problems with many factors of variation, 2007 International Conference on Machine Learning (ICML), Corvalis, Oregon, USA, 2007, pp. 473-480.
[34] H. Xiao, K. Rasul, R. Vollgraf, Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms, arXiv preprint ariXiv:1708.07747 (2017).
[35] A. Krizhevsky, G. Hinton, Learning multiple layers of features from tiny images, Handbook of Systemic Autoimmune Diseases, 1 (2009).
[36] T.-H. Chan, K. Jia, S. Gao, J. Lu, Z. Zeng, Y. Ma, PCANet: a simple deep learning baseline for image classification?, IEEE Transactions on Image Processing, 24(12) (2015), pp. 5017-5032.
[37] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, arXiv preprint arXiv:1409.1556 (2014).
[38] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam, MobileNets: efficient convolutional neural networks for mobile vision applications, arXiv preprint arXiv:1704.04861 (2017).
[39] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, A. Rabinovich, Going deeper with convolutions, 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, USA, 2015, pp. 1-9.
[40] A. Krizhevsky, I. Sutskever, G. Hinton, ImageNet classification with deep convolutional neural networks, Advances in Neural Information Processing Systems, 60(6) (2012), pp. 84-90.
[41] F. Iandola, Exploring the design space of deep convolutional neural networks at large scale, arXiv preprint ariXiv:1612.06519 (2016).
[42] N. Ma, X. Zhang, H.-T. Zheng, J. Sun, ShuffleNet V2: practical guidelines for efficient CNN architecture design, 2018 European Conference on Computer Vision (ECCV), Munich, Germany, 2018, pp. 122–138.
[43] K. Han, Y. Wang, Q. Tian, J. Guo, C. Xu, C. Xu, GhostNet: more features from cheap operations, 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Seattle, WA, USA, 2020, pp. 1577-1586.
[44] H. Cai, T. Chen, W. Zhang, Y. Yu, J. Wang, Efficient architecture search by network transformation, 2018 AAAI Conference on Artificial Intelligence (AAAI), San Francisco, California, USA, 2018, pp. 2787-2794.
[45] D. O'Neill, B. Xue, M. Zhang, Evolutionary neural architecture search for high-dimensional skip-connection structures on densenet style networks, IEEE Transactions on Evolutionary Computation, 25(6) (2021), pp. 1118-1132.
[46] F. Miao, L. Yao, X. Zhao, Evolving convolutional neural networks by symbiotic organisms search algorithm for image classification, Applied Soft Computing, 109 (2021), pp. 107537.
[47] Y. Bi, B. Xue, M. Zhang, Genetic programming with image-related operators and a flexible program structure for feature learning in image classification, IEEE Transactions on Evolutionary Computation, 25(1)

(2021), pp. 87-101.
[48] B. L. Welch, The generalization of "student's" problem when several different population variances are involved, Biometrika, 34(1) (1947), pp. 28-35.
[49] G. E. Hinton, R. S. Zemel, Autoencoders, minimum description length and helmholtz free energy, 1993 International Conference on Neural Information Processing Systems (NIPS), Denver, Colorado, 1993, pp. 3-10.
[50] K. V. Price, R. M. Storn, J. A. Lampinen, Differential evolution-a practical approach to global optimization, Natural Computing, 141(2) (2005).
[51] X. S. Yang, Firefly algorithms for multimodal optimization, 2009 International Conference on Stochastic Algorithms: Foundations and Applications (SAGA), Sapporo, Japan, 2009, pp. 169-178.

Appendix

Table 12 The best CNN architectures found by the psoResNet on each dataset.

Dataset	Block	Value	Parameters
MNIST-RD	Residual block	fliters1:70(4×4); fliters2:62(3×3)	736,741
	Residual block	fliters1:255(3×3); fliters2:247(5×5)	
	Average pooling	kernel size: 3 × 3; strides: 2 × 2	
	Residual block	fliters1:250(7×7); fliters2:247(7×7)	
	FC	output neurons: 10	
MNIST-Fashion	Residual block	fliters1:143(4×4); fliters2:57(5×5)	167,716
	Residual block	fliters1:204(7×7); fliters2:208(7×7)	
	Average pooling	kernel size: 3 × 3; strides: 2 × 2	
	Maximum pooling	kernel size: 3 × 3; strides: 2 × 2	
	FC	output neurons: 10	
Convex	Residual block	fliters1:76 (5×5); fliters2:108(3×3)	171,968
	Residual block	fliters1:81(7×7); fliters2:40(3×3)	
	Residual block	fliters1:64(3×3); fliters2:38(7×7)	
	Average pooling	kernel size: 3 × 3; strides: 2 × 2	
	Residual block	fliters1:97(5×5); fliters2:250(7×7)	
	FC	output neurons: 2	
CIFAR10	Residual block	fliters1:114 (4×4); fliters2:146(3×3)	1,118,116
	Residual block	fliters1:90 (5×5); fliters2:145(4×4)	
	Residual block	fliters1:171 (3×3); fliters2:198(3×3)	
	Maximum pooling	kernel size: 3 × 3; strides: 2 × 2	
	Residual block	fliters1:35 (5×5); fliters2:205 (4×4)	
	Residual block	fliters1:230 (3×3); fliters2:153(6×6)	
	Residual block	fliters1:220 (5×5); fliters2:76(7×7)	
	Residual block	fliters1:146 (7×7); fliters2:217(7×7)	
	Residual block	fliters1:146 (7×7); fliters2:217(7×7)	
	FC	output neurons: 2	
Crime	Residual block	fliters1-68(3×3); fliters2-241(6×6)	822,610
	Residual block	fliters1-117(5×5); fliters2-232(6×6)	
	Average pooling	kernel size: 3 × 3; strides: 2 × 2	
	Maximum pooling	kernel size: 3 × 3; strides: 2 × 2	
	FC	output neurons: 13	