2021 IEEE Congress on Evolutionary Computation (CEC) | 978-1-7281-8393-0/21/$31.00 ©2021 IEEE | DOI: 10.1109/CEC45853.2021.9504716

A Flexible Variable-length Particle Swarm Optimization Approach to Convolutional Neural Network Architecture Design

Junhao Huang[1], Bing Xue[1], Yanan Sun[2] and Mengjie Zhang[1]
[1] *School of Engineering and Computer Science, Victoria University of Wellington*, Wellington 6140, New Zealand
[2] *School of Computer Science, Sichuan University*, Chengdu 610065, China
Emails: {junhao.huang, bing.xue}@ecs.vuw.ac.nz, ysun@scu.edu.cn, mengjie.zhang@ecs.vuw.ac.nz

***Abstract*—The great success of convolutional neural networks (CNNs) in image classification benefits from the powerful feature learning abilities of their architectures. However, arbitrarily constructing these architectures is very costly in terms of manpower and computation resources. Recently, particle swarm optimization (PSO), as a promising evolutionary computation (EC) method, has been used to automatically search for the CNN architectures and achieved encouraging results on image classification, but these existing methods are not well-designed and/or do not have good search capabilities. In this paper, a flexible variable-length PSO algorithm is proposed for automated CNN architecture design for image classification tasks. Particularly, an improved encoding scheme is proposed to truly break the fixed-length representation constraint of PSO and encode the parameters in a more meaningful way. In addition, a novel velocity and position updating approach is developed to update variable-length particles. Experiments on four benchmark datasets are carried out to confirm the superiority of the proposed algorithm. It is shown that the proposed method leads to better results comparing to the state-of-the-art algorithms in terms of classification performance, parameter size and computational complexity.**

***Index Terms*—Convolutional neural network, particle swarm optimization, neural architecture search, image classification.**

I. Introduction

Convolutional neural networks (CNNs), have achieved remarkable success in various fields such as computer vision [1], [2], natural language processing [3], [4] and speech recognition [5], [6]. However, most of the excellent networks are handcrafted and they require a lot of labour and domain expertise, which hinders the promotion and application of CNN technology [7]. To solve such an arduous task of neural architecture search (NAS), in recent years, evolutionary computation (EC) based methods have been applied to automatically search for the optimal CNN architectures without human intervention. Due to the insensitivity to local optima and adaptivity to the dynamic training environment, EC algorithms have shown their promising performance in dealing with challenging optimization/learning problems in network architecture construction [8]–[11].

Specifically, particle swarm optimization (PSO), as one of the most widely-used EC algorithms, is attracting increasing attention in this research area [10]–[13]. PSO is good at global optimization and has lower computational complexity, so it is very suitable for evolving neural networks whose architecture and configurations are generally represented by discrete vectors in evolutionary computation-based neural architecture search (ENAS). Intuitively, the length of the particle is related to the depth of the encoded network. However, PSO uses a fixed-length encoding, which makes it hard to be fully utilized for evolving CNN architecture, since the optimal depth of a CNN is unknown. Wang *et al.* [11] introduce PSO to ENAS for the first time, and the method, named as IPPSO, encodes the CNN architecture into the format of an IP address and evolves the particles by standard PSO. A disabled layer is developed to obtain the variable-length particles. Nevertheless, this method essentially evolves the network architecture by a fixed-length code and suffers from the need for predefining maximum length of particle. Another PSO-based method [12], called psoCNN, evolves the particles by measuring the differences between them instead of using standard updating methods. psoCNN is actually a combination method and fails to mutate the configurations of the network building blocks in the evolutionary process.

As a result, how to effectively achieve variable-length PSO for CNN architecture design is still a challenging problem. In this paper, we propose a flexible variable-length PSO to automatically evolve CNNs for image classification. A new encoding scheme based on the IP-Based Encoding Strategy (IPES) in IPPSO [11] is developed to achieve the exploration of CNN architecture more reasonably. Besides, a novel particle alignment operation is employed to update the variable-length particles during the evolutionary process. Without human intervention and extra genetic operators, this method can automatically evolve compact and high-performing CNN architecture in a relatively short time. Our specific goals of this work are listed as follows:

1) Build a fully-convolutional feature extractor where some hyperparameters regarding the convolutional and pooling layers will be well presented to effectively shrink the search space.
2) Propose a new encoding scheme to improve the IP-Based Encoding Strategy used in IPPSO which will

978-1-7281-8393-0/21/$31.00 ©2021 IEEE

break the constraint of traditional PSO on fixed-length encoding.

3) Design an effective operator to flexibly update the velocity and position of particles with different lengths.
4) Unify the evolution of CNN's hyperparameters and depth. The invalid values in the PSO evolution will be deemed as a depth exploration of the model and assigned reasonable meanings.

The remainder of the paper is organized as follows. In Section II, background of the CNN architectures and PSO, and the related work on PSO-based ENAS are presented. Section III elaborates the proposed flexible variable-length PSO in details. Experimental design and results are provided in Sections IV and V, respectively. Finally, the conclusions are drawn in Section VI.

II. Background

A. CNN Architecture

A typical CNN architecture for image classification consists of two components, namely the feature extraction part and the classification part [14]. The former component is generally constructed by convolutional layers and pooling layers, while the latter usually contains fully-connected layer(s) only. The input data are internally transformed in the feature extractor part layer by layer to generate high-level abstract features, which will be fed into the following classification part to complete classification prediction. Features are learned through convolution operation in the convolutional layers and nonlinear activation attached to each convolutional layer, and the functionality of the pooling layers that have no weight parameter is to downsample the size of the feature maps. Generally, the configurations of a convolutional layer include the filter size, the stride size, and the number of output feature maps, while those of a pooling layer include the kernel size, the stride size, and the pooling type (e.g., max or average). Note that in the proposed method, a global average pooling plus a fully-connected layer is used for classification, and we only evolve the feature extractor because it is the determinant of the model's performance.

B. Particle Swarm Optimization

Particle swarm optimization (PSO) [15] is a population-based heuristic EC algorithm, which has been widely used for solving optimization problems owing to its powerful search ability, low computational complexity, simplicity in implementation, and the characteristic of fast convergence. At each iteration, each individual also known as a particle in the population is adjusted to a new position in the search space based on its corresponding updated velocity. The calculations of the $(t+1)$-th iteration in PSO are formulated as

$$\begin{aligned} v_{i,j}^{t+1} = \omega \cdot v_{i,j}^{t} + c_1 \cdot r_1 \cdot (pBest_{i,j} - x_{i,j}^{t}) \\ + c_2 \cdot r_2 \cdot (gBest_j - x_{i,j}^{t}) \end{aligned} \tag{1}$$

$$x_{i,j}^{t+1} = x_{i,j}^{t} + v_{i,j}^{t+1} \tag{2}$$

where $x_{i,j}$ represents the j-th dimension of the i-th particle position vector, $v_{i,j}$ denotes the velocity of x_i at j-th dimension, $pBest_{i,j}$ represents the j-th dimension of the historical best of the i-th particle, and $gBest_j$ is the global best particle's the j-th dimension. ω, c_1 and c_2 are three tunable parameters, while r_1 and r_2 are random numbers ranging from 0 to 1. In PSO, the particles are iteratively updated, and the $gBest$ in the last iteration will be selected as the final solution to the target optimization problem.

C. Related Work

Neural architecture search (NAS) is a complicated problem that usually involves intensive computing resources, such as 28 days of training on 800 GPUs for [16]. Due to the inexpensive computational cost and strong search ability, PSO can ensure the efficiency of NAS while maintaining good performance. Wang *et al.* [11] represent the CNN architectue by a fixed-length encoding scheme based on the Internet Protocol address (IP address). Each type of layers as well as the parameters are assigned different value ranges according to the idea of subnetting. This method, called IPPSO, also introduces disabled bits to produce networks of different sizes. The main shortcoming is that the encoding strategy is not a completely variable-length method which is of low flexibility. Moreover, there are too many invalid values in the search space of IPPSO, and simply replacing them with randomly generated values might destroy the evolving process. Prior to this, Sun *et al.* [10] devise a PSO-based architecture optimization (PSOAO) algorithm to address the updating of particles with variable lengths. An x-reference velocity calculation method is applied by padding or trimming to keep the same length of global best particle to the current evolved particle. This method, however, is only employed on convolutional auto-encoder (CAE), and is incapable of changing the particle length during the evolution because it is fixed after the swarm initialization. Another work [12], named as psoCNN, evolves the variable-length particles by measuring the differences between the current particle and the two best particles, and the layers are randomly selected from the vector of differences to form the updating velocity. Obviously, this updating mechanism is not carried out in a smooth space. The whole evolutionary process is actually a recombination of the current particles which to some degree affected the evolving effectiveness.

This thereby motivates us to develop a flexible and truly variable-length PSO algorithm for CNN architecture search. In the proposed method, all the aforementioned problems of the previous work will be well addressed.

III. Proposed Algorithm

In this section, the proposed flexible PSO algorithm (FPSO) is presented in detail. To be specific, we first introduce the framework of the algorithm in SubSection III-A and then exploit the main steps in Subsections III-B to III-E.

A. Framework of FPSO

The framework of the proposed FPSO algorithm is shown in Fig. 1. The algorithm starts from a randomly generated

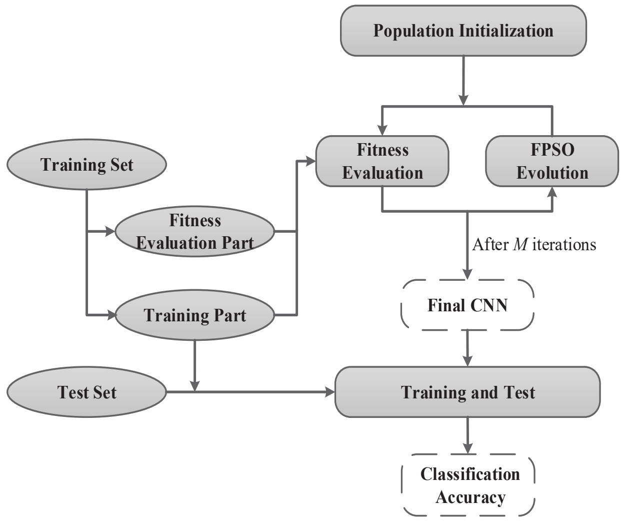

Fig. 1. Framework of the proposed FPSO algorithm.

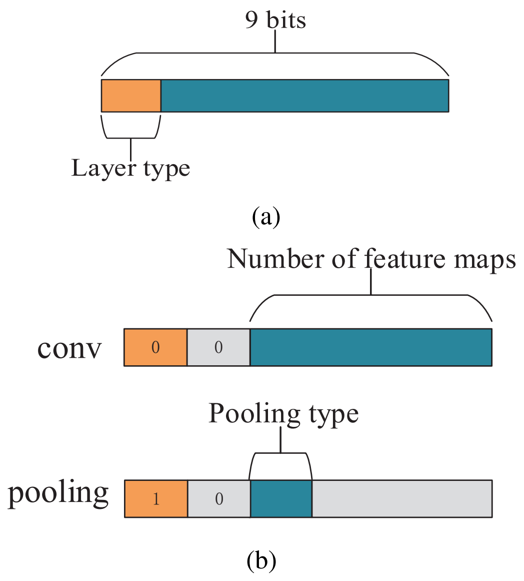

Fig. 2. Representation of layer in binary form. (a) 9-bits representation of one layer. (b) Representations of convolutional and pooling layers and their configurations.

initial swarm, which is also known as the initial population in EC. Then the core part, that is, the evolutionary process is performed, where the updating of the velocity and position of the particles in the population, and fitness evaluations are iteratively performed until a stopping criterion is satisfied, say, M iterations. As mentioned before, this method aims at evolving the optimal CNN architecture for image classification problems, therefore, the classification accuracy of the evolved CNN on the training data is used as the fitness value. The 10% of the training set is randomly extracted as the fitness evaluation set in the fitness evaluation process, and the test dataset is only involved in the test process to output the final classification accuracy of the evolved CNN, which is global best individual in the last PSO iteration.

B. Particle Encoding Scheme

The evolved CNN architecture in this method is only composed of two building blocks, namely the standard convolutional layer and the pooling layer, where some of the configurations are empirically fixed as commonly used values in the deep learning community [14], [17] in advance in order to minimize the search space efficiently. To be specific, the filter size and the stride size of the convolutional layer are set to 3 and 1, respectively. The kernel size and the stride size of the pooling layer are both set to 2. Hence, following the IPES, the configuration of the convolutional layer involved in evolution is the number of output feature maps with a range [1, 128] which can be represented by a 7-bits binary string, while that of the pooling layer is the pooling type (maximal or average) represented by 1 bit. Note that the maximum number of the output feature maps for a convolutional layer can be flexibly adjusted and the range here is a general setting based on the task complexity.

Obviously, the longest binary representation of the above two types of layers is 7 bits, so generally an 8-bit address is sufficient to accommodate any building layer where one extra bit is used to distinguish two different types of layers. However, we add one more bit to the current binary code to increase the encoding space for the purpose of architecture length exploration. In this case, the final encoding length of one layer is 9 bits and the representation is shown in Fig. 2. Each layer is encoded into a 9-bits binary string where the first bit is used to decide the layer type, i.e., 0 for convolutional layer and 1 for pooling layer. The second bit is a reserved bit which is used to generate a search space for exploring different particle lengths. For this purpose, this bit is always 0 for the encoding of the convolutional layer and the pooling layer, and the length change process will be started when it becomes 1. The last 7 bits are the value space allocated to the configuration of the layer - number of output feature maps for convolutional layer and pooling type for pooling layer. According to this encoding method, the ranges of the convolutional layer and the pooling layer are [0, 127] and [256, 383], respectively[1].

As for those values that do not fall into the ranges of both layers, we regard it as an attempt to change the length of architecture rather than the so-called invalid values that are usually discarded and reassigned [11], [18]. During the PSO evolution, the layer is considered to be removed if the updated value is less than 0, else if the second bit of the updated value turns into 1, a corresponding layer is added after the current layer with their parameters adjusted. The ranges of different layers and operations as well as their parameters are tabulated in Table I. The resulted parameter value after PSO updating and "floor" rounding operation is assumed to be k. It can be seen that the proposed encoding strategy covers the entire value space by assigning them the meanings of building blocks or operations.

For example, if the value is updated to 201 by PSO, then

[1]Note: the value 127 here means 128 output feature maps because the binary string starts from 0.

TABLE I
THE RANGES AND PARAMETERS OF DIFFERENT TYPES OF LAYERS AND OPERATIONS

Range of k	Layer type/Operation	Adjusted Parameter(s)
$(-\infty$, -1]	Remove the layer	-
[0, 127]	Conv layer	k
[128, 255]	Add conv layer	$\lfloor \frac{k}{2} \rfloor$, $\lceil \frac{k}{2} \rceil$
[256, 319]	Maximal pooling layer	k
[320, 383]	Average pooling layer	k
[384, 511]	Add pooling layer	319 + $\lceil \frac{k-383}{2} \rceil$, 320 - $\lfloor \frac{k-383}{2} \rfloor$
[512, $+\infty$)	Add pooling layer	$\lfloor \frac{k}{2} \rfloor$, $\lceil \frac{k}{2} \rceil$

according to our encoding strategy, the value needs to be split into 100 and 101, representing two convolutional layers with 100 and 101 output feature maps, respectively. Note from Table I that we have two "Add pooling layer" operations. The difference between them is that when the value is in range [384, 511], the resulting two pooling layers are of different types, while by the other operation they are of the same type. Each representation code of the network layer is regarded as one dimension of the particle, which ensures that the inherent encoding structure of the network configuration is not destroyed during evolution. Through this setting, we have truly realized the variable-length representation of particles.

C. Population Initialization

The first step of the proposed algorithm is to obtain the initial population. Given the population size, particles are randomly generated. For each particle, the initial length is first determined which is randomly sampled from a Gaussian distribution with the mean being a predefined task-dependent number and the standard deviation of one. Then the two building layers with different parameters are randomly stacked to form a CNN architecture until the length requirement is reached. Note that based on the convention of the CNN construction, the network should start from a convolutional layer, and the number of the pooling layers, num_p, must satisfy the rule of $num_p \le \lfloor log_2 d \rfloor$ where d denotes the input image size. This initialization process, resulting in a swarm of particles with various lengths, is summarized in Algorithm 1.

D. Fitness Evaluation

In regard to fitness evaluations which are composed of two parts - training and evaluating, we decode each particle into a CNN architecture, where each convolutional layer is followed by batch normalization (BN) [19] and ReLU [20], and the classification layer (one global average pooling + one fully-connected layer) mentioned before is added to the tail of the decoded CNN architecture. It is worth noting that we adopt an early-stopping mechanism to ease the computational burden that only a small number of epochs is used to train the evolved individuals and after the training, the evaluated accuracies are collected as their fitness values accordingly. Here, the fitness evaluation set is randomly extracted from the training set with the proportion of 10%, which is to speed up the training process and avoid overfitting. Note that the evolving process

Algorithm 1: Population Initialization

```
Input: The population size $N$, the input image size
       $d \times d$, mean value of the Gaussian distribution $\mu$.
Output: The initial population $P_0$.
1  $P_0 \leftarrow \emptyset$;
2  for $i = 1$ to $N$ do
3      $p_i \leftarrow \emptyset$;
4      $len_i \leftarrow$ Randomly generate an integer from
       $X \sim N(\mu, 1)$;
5      $num_p \leftarrow$ Randomly generate an integer between
       [0, $\lfloor log_2 d \rfloor$];
6      Randomly select $len_i - num_p$ integers between [0,
       127];
7      Randomly select $num_p$ integers between [256, 383];
8      $p_i \leftarrow$ Randomly combine the $len_i$ integers;
9      $P_0 \leftarrow P_0 \cup p_i$;
10 end
11 Return $P_0$.
```

generates the particles with different lengths that represent the CNN architectures with different capacities. Simply estimating the performances of those networks trained with one fixed number of epochs might be unfair and inaccurate. We hereby dynamically set different training epochs for individuals in the evaluation by introducing a performance improvement threshold γ. Specifically, all the individuals are trained with at least a specific number of epochs and continue to be trained until the performance improvement is smaller than γ. This practice ensures that the performance tendency of the individual can be accurately captured while maintaining relatively high efficiency. The procedure of a fitness evaluation is tabulated in Algorithm 2.

E. Particle updating

In the velocity and position updating of PSO, the current particle, personal best solution as well as global best solution should be with the same length. To this end, a novel variable-length PSO method is proposed to deal with the updating of particles with different lengths. Before the standard velocity calculation, a particle alignment operation is conducted to obtain the vectors of equal length. In the PSO evolution, the personal best solution and the global best solution are extended or trimmed to match the length of the current particle. Moreover, to increase population diversity, we introduce more randomness by applying a random offset number to get different components of the particle.

Fig. 3 depicts an example of particle alignment operation. It is assumed that the personal best solution $pBest$ with the length of len_{pBest} is shorter than the current particle x with the length of len_x, while the global best solution $gBest$ with the length of len_{gBest} is longer than x. Firstly, two numbers, $offset1$ and $offset2$, are randomly generated between [0, $len_x - len_{pBest}$] and [0, $len_{gBest} - len_x$], respectively. Then, a vector $pBest'$ with the same length as x is constructed, where

Algorithm 2: Fitness Evaluation

```
Input: The population $P$, the training part of the training
       set $D_{train}$, the fitness evaluation set $D_{fitness}$, the
       number of training epochs $k$, performance
       improvement threshold $\gamma$.
Output: The population $P$ with fitness.
1  for each individual $p$ in $P$ do
2      $i \leftarrow 1$;
3      $acc_{best} \leftarrow 0$;
4      $toContinue \leftarrow False$;
5      while $i \leq k$ or $toContinue$ do
6          Train $p$ on $D_{train}$ for one epoch;
7          $acc \leftarrow$ Evaluate $p$ on $D_{fitness}$;
8          if $acc - acc_{best} \geq \gamma$ then
9              $toContinue \leftarrow True$;
10         else
11             $toContinue \leftarrow False$;
12         end
13         if $acc > acc_{best}$ then
14             $acc_{best} \leftarrow acc$;
15         end
16         $i \leftarrow i + 1$;
17     end
18     $fitness \leftarrow acc_{best}$;
19     $P \leftarrow$ Update the fitness of $p$ in $P$;
20 end
21 Return $P$.
```

the values in indices of $[offset1, offset1 + len_{pBest} - 1]$ are the same as $pBest$, and the other positions are padded with zeros. In terms of the longer solution $gBest$, a sub-vector $gBest'$ is extracted from it with indices $[offset2, offset2 + len_x - 1]$. Consequently, we obtain three solutions of equal length in the updating process. Finally, the updated velocity and position of the current particle can be successfully calculated by the standard PSO formulas (1) and (2).

When a new position vector is obtained by velocity and position updating, we further decode the vector according to Table I to get the updated architecture. As a result, the architecture configurations and the length of CNN can be flexibly evolved by the proposed variable-length PSO algorithm.

IV. Experiment Design

In this section, we introduce our experiment design. The four benchmark datasets, peer competitors and parameter settings of the proposed FPSO are introduced.

A. Datasets and Peer Competitors

The proposed FPSO algorithm is examined on four different image classification datasets, namely, the MNIST Basic (MB) [21], the MNIST with Rotated Digits plus Background Images (MRDBI) [21], Fashion-MNIST [22] and CIFAR-10 [23]. The former two datasets - MB and MRDBI, are variants of the original MNIST [24], which is a 10-classes handwritten digits dataset. Differently, these variant datasets contains only

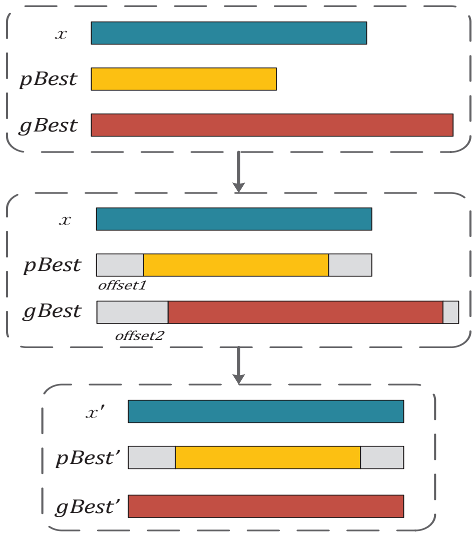

Fig. 3. Example of the particle alignment operation.

12,000 training instances but 50,000 test instances. Fewer training samples and the interference of noise added to the datasets make their classification task more difficult than the original MNIST dataset. The Fashion-MNIST dataset consists of 55,000 training and 10,000 testing grayscale 28×28 images of 10 fashion products, which has the same image size and data format as MNIST, but its image content is more complicated. The CIFAR-10 dataset is also a 10-class classification dataset, containing 50,000 training and 10,000 testing color images of dimensions 32×32.

Additionally, the results of some state-of-the-art algorithms on the above datasets are collected for comparisons to prove the superiority of the proposed method. They are SVM+RBF [21], CAE-2 [25], PCANet-2 [26], LDANet-2 [26], EvoCNN [27], IPPSO [11], DECNN [18], psoCNN [12] and HGAPSO [13] on MB and MRDBI, HOG+SVM, 2Conv+3FC, VGG16 [17], GoogleNet [28], MobileNet [29], EvoCNN [27] and psoCNN [12] on Fashion-MNIST. The peer competitors on the CIFAR-10 are NIN [30], VGG16 [17], RseNet-110 [31], DenseNet ($k = 12$) [14], GeNet [8], LS-Evolution [32], CGP-CNN (ConvSet) [9], NAS [16] and MetaQNN [33].

B. Parameter Settings

The parameters involved in the experiments are shown in Table II, most of which are specified based on the community conventions [34], [35] or the previous work [11], [13]. The experimental parameters on the four datasets are basically the same, except for the items “minimum number of training epochs in evaluation” and “number of training epochs in test”. For the evaluation process, we used 10, 10, 3 and 10 training epochs on MB, MRDBI, Fashion-MNIST and CIFAR-10, respectively. Note that only 3 epochs are used on the training of Fashion-MNIST because it has more training images than MB and MRDBI, and lower complexity than CIFAR-10. In the final training and test for evolved CNN, the number of training epochs is set to 100 for the previous three datasets and 250 for the CIFAR-10 dataset according to their task difficulties. In order to avoid errors caused by the randomness of the algorithm, the experiments on each dataset (except for CIFAR-10) will be independently run 15 times, and their results will be statistically analyzed. Due to the limited computational resources, we only run 10 times on CIFAR-10 and take the best result.

TABLE II
PARAMETER SETTINGS OF THE EXPERIMENTS

Parameter	value
population size N	20
number of generation M	20
mean value of the length initialization μ	10
standard deviation of the length initialization σ	1
c_1, c_2	1.49618
ω	0.7298
minimum number of training epochs in evaluation k	10/10/3/10
performance improvement threshold γ	0.01
number of training epochs in test l	100/100/100/250

V. EXPERIMENTAL RESULTS

In this section, we present our experimental results and analysis. Firstly, the classification performance on MB, MRDBI and Fashion-MNIST is shown. Then, we provide examples of the architectures evolved on these three datasets that are of the same scale. Finally, the FPSO algorithm is applied to CIFAR-10 to verify the versatility of the proposed method, and compared with peer competitors in terms of the classification error rate, parameter size, and computational cost.

A. Overall Results

Table III exhibits the experimental results on MB and MRDBI, from which it can be seen that the proposed FPSO algorithm has considerable performance on both datasets. Note that the values following the mean error rates are the standard deviations of these results after multiple independent runs. According to Table III, the best result of FPSO on MRDBI outperforms all the peer competitors while that is second on MB to HGAPSO. Moreover, the mean error as well as the standard deviation of FPSO is even better than the best results of some methods, which, on the one hand, indicates that the proposed method is not only effective but also stable. It is worth noting that the HGAPSO algorithm [13] uses Denseblock [14], which is commonly considered as a very effective handcrafted module to construct CNN models. The classification performance of the FPSO algorithm, however, surpasses it on the MRDBI dataset with the basic building blocks, which strongly proves the superiority of the proposed method.

Regarding the experimental results on Fashion-MNIST listed in Table IV, it is shown that FPSO is the best method among the eight peer competitors in terms of the classification error. When comparing the number of parameters, the proposed method shows significantly fewer parameters than VGG16, GoogleNet and EvoCNN, slightly fewer parameters than MobileNet and psoCNN, and similar to the number of parameters obtained by 2Conv + 3FC but lower error rate.

TABLE III
THE CLASSIFICATION ERROR RATES (%) OF THE PROPOSED FPSO COMPARED WITH THE PEER COMPETITORS ON MB AND MRDBI

Model	Error rate (%)	
	MB	MRDBI
SVM+RBF	3.03	55.18
CAE-2	2.48	45.23
PCANet-2	1.4	35.86
LDANet-2	1.05	38.54
EvoCNN (best)	1.18	35.03
EvoCNN (mean)	1.28 (0.15)	37.38 (1.75)
IPPSO (best)	1.13	33
IPPSO (mean)	1.21 (0.1)	34.5 (2.96)
DECNN (best)	1.03	32.85
DECNN (mean)	1.46 (0.11)	37.55 (2.45)
psoCNN (best)	-	14.28
psoCNN (mean)	-	20.98
HGAPSO (best)	0.74	10.53
HGAPSO (mean)	0.84 (0.07)	12.23 (0.86)
FPSO (best)	0.96	10.17
FPSO (mean)	1.08 (0.06)	11.91 (0.79)

TABLE IV
THE CLASSIFICATION ERROR RATES (%) OF THE PROPOSED FPSO COMPARED WITH THE PEER COMPETITORS ON FASHION-MNIST

Model	Error rate (%)	#parameters (M)
Human performance	16.5	-
HOG + SVM	7.4	-
2Conv + 3FC	6.6	0.5
VGG16	6.5	26
GoogleNet	6.3	23
MobileNet	5	4
EvoCNN (best)	5.47	6.68
EvoCNN (mean)	7.28 (1.69)	6.52
psoCNN (best)	5.53	2.32
psoCNN (mean)	5.90	2.5
FPSO (best)	4.93	0.53
FPSO (mean)	5.22 (0.14)	0.61

B. Evolved CNN Architectures

In this subsection, the best evolved architectures on MB, M-RDBI and Fashion-MNIST are given in Tables V, VI and VII, respectively, which is expected to help users in designing CNN architectures. It is shown that the best CNN architectures on MB and MRDBI have 21 and 22 network layers, respectively, each of which encompasses 15 and 17 convolutional layers. Surprisingly, the best CNN architecture on Fashion-MNIST has only 13 network layers, including 9 convolutional layers.

Here we briefly describe the potentially helpful information observed from these architectures. First of all, the three evolved network architectures have the same pattern, that is, the stacking of convolutional layers with similar number of output feature maps. This construction form is similar to many manually designed CNN architectures, such as VGGNet [17] and ResNet [31]. Furthermore, all of the number of feature maps of these network architectures present a changing trend of increasing first and then decreasing, which is a trick worth learning in adjusting the network channels. Last but not least, it is observed that the structure of two consecutively stacked pooling layers appeared during the evolutionary process, which can quickly reduce the spatial size of the feature maps. This

TABLE V
THE BEST CNN ARCHITECTURE SEARCHED ON MB

Model	Configuration
conv	filter size: 3×3, stride: 1, feature maps: 27
conv	filter size: 3×3, stride: 1, feature maps: 30
conv	filter size: 3×3, stride: 1, feature maps: 32
conv	filter size: 3×3, stride: 1, feature maps: 106
conv	filter size: 3×3, stride: 1, feature maps: 111
conv	filter size: 3×3, stride: 1, feature maps: 111
pool	kernel size: 2×2, stride: 2, type: Average
conv	filter size: 3×3, stride: 1, feature maps: 126
conv	filter size: 3×3, stride: 1, feature maps: 126
conv	filter size: 3×3, stride: 1, feature maps: 29
conv	filter size: 3×3, stride: 1, feature maps: 78
conv	filter size: 3×3, stride: 1, feature maps: 79
conv	filter size: 3×3, stride: 1, feature maps: 66
pool	kernel size: 2×2, stride: 2, type: Average
pool	kernel size: 2×2, stride: 2, type: Average
conv	filter size: 3×3, stride: 1, feature maps: 89
conv	filter size: 3×3, stride: 1, feature maps: 89
pool	kernel size: 2×2, stride: 2, type: Max
conv	filter size: 3×3, stride: 1, feature maps: 77
pool	global average pooling
fc	output neurons: 10

TABLE VI
THE BEST CNN ARCHITECTURE SEARCHED ON MRDBI

Model	Configuration
conv	filter size: 3×3, stride: 1, feature maps: 122
conv	filter size: 3×3, stride: 1, feature maps: 31
conv	filter size: 3×3, stride: 1, feature maps: 59
conv	filter size: 3×3, stride: 1, feature maps: 75
conv	filter size: 3×3, stride: 1, feature maps: 91
conv	filter size: 3×3, stride: 1, feature maps: 97
pool	kernel size: 2×2, stride: 2, type: Average
conv	filter size: 3×3, stride: 1, feature maps: 108
conv	filter size: 3×3, stride: 1, feature maps: 89
conv	filter size: 3×3, stride: 1, feature maps: 86
conv	filter size: 3×3, stride: 1, feature maps: 87
conv	filter size: 3×3, stride: 1, feature maps: 87
conv	filter size: 3×3, stride: 1, feature maps: 87
conv	filter size: 3×3, stride: 1, feature maps: 115
conv	filter size: 3×3, stride: 1, feature maps: 115
pool	kernel size: 2×2, stride: 2, type: Average
pool	kernel size: 2×2, stride: 2, type: Average
conv	filter size: 3×3, stride: 1, feature maps: 119
conv	filter size: 3×3, stride: 1, feature maps: 70
conv	filter size: 3×3, stride: 1, feature maps: 24
pool	global average pooling
fc	output neurons: 10

TABLE VII
THE BEST CNN ARCHITECTURE SEARCHED ON FASHION-MNIST

Model	Configuration
conv	filter size: 3×3, stride: 1, feature maps: 59
conv	filter size: 3×3, stride: 1, feature maps: 74
conv	filter size: 3×3, stride: 1, feature maps: 91
pool	kernel size: 2×2, stride: 2, type: Max
conv	filter size: 3×3, stride: 1, feature maps: 66
conv	filter size: 3×3, stride: 1, feature maps: 66
pool	kernel size: 2×2, stride: 2, type: Max
conv	filter size: 3×3, stride: 1, feature maps: 121
conv	filter size: 3×3, stride: 1, feature maps: 115
conv	filter size: 3×3, stride: 1, feature maps: 119
conv	filter size: 3×3, stride: 1, feature maps: 9
pool	global average pooling
fc	output neurons: 10

may also be a useful practice worth adopting in feature learning.

C. Experiments on CIFAR-10

With the encouraging performance on previous datasets, in this subsection, the proposed method is conducted on more complex dataset - CIFAR-10. Note here that we simply use the same parameter settings as those in the previous experiments without refinement and this experiment is served as an example of the proposed algorithm on medium-scale dataset. As can be seen from Table VIII, compared with the peer competitors, the proposed FPSO algorithm shows competitive performance with much fewer parameters and consumed only 1.65 GPU days. It is worth noting that most of these comparison methods are performed based on effective handcrafted building blocks, such as Inception Block [28], ResBlock [31] and DenseBlock [14], but the FPSO algorithm does not involve these advanced knowledge and achieves similar or even slightly better classification performance. This result, together with those above, fully demonstrates the effectiveness and efficiency of our proposed algorithm, as well as its versatility especially on small and medium-scale datasets.

TABLE VIII
THE CLASSIFICATION ERROR RATES (%) OF THE PROPOSED FPSO COMPARED WITH THE PEER COMPETITORS ON CIFAR-10

Model	Error rate (%)	#parameters (M)	GPU Days
NIN	8.81	35.68	-
VGG16	6.66	20.04	-
ResNet-110	6.43	1.7	-
DenseNet ($k = 12$)	5.24	1.0	-
GeNet	7.10	-	17
LS-Evolution	5.40	5.4	2750
CGP-CNN (ConvSet)	6.34	1.75	30.4
NAS	6.01	2.5	22400
Meta-QNN	6.92	-	100
FPSO	6.28	0.70	1.65

VI. CONCLUSION

In this paper, a flexible PSO algorithm, referred to as FPSO, is proposed to evolve the CNN architecture for image classification automatically and efficiently. The method includes a flexible variable-length encoding scheme which is developed based on an existing strategy, a more complete search space construction strategy, and an effective variable-length particle updating approach. The proposed method is examined on four popular benchmark datasets and compared with many state-of-the-art algorithms, and shows better or competitive performance in the classification accuracy, parameter size and computational cost due to its more reasonable search space design and the fast and stable search strategy. As the experiments showing, the proposed method allows PSO to evolve completely automatically under variable particle lengths, which is the most important contribution of this paper.

Since the building blocks in this method are the basic layers, this, to some extent, might limit the feature learning ability of the evolved model. Therefore, in the future, more effective building blocks (e.g. shortcut connections in ResNet and DenseNet, feature fusion in GoogleNet) can be utilized to improve the model ability. In addition, we will also explore ways to combine the proposed method with a surrogate model to further reduce the computational complexity.

References

[1] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," *Proceedings of International Conference on Neural Information Processing Systems (NeurIPS)*, vol. 60, pp. 1097–1105, Dec. 2012.

[2] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and F. F. Li, "Large-scale video classification with convolutional neural networks," *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1725–1732, Jun. 2014.

[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proceedings of International Conference on Neural Information Processing Systems (NeurIPS)*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30, 2017, pp. 5998–6008.

[4] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2019, pp. 4171–4186.

[5] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, A. S. N. Jaitly, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury, "Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, Oct. 2012.

[6] W. Xiong, J. Droppo, X. Huang, F. Seide, M. Seltzer, A. Stolcke, D. Yu, and G. Zweig, "Toward human parity in conversational speech recognition," *IEEE/ACM Transactions on Audio Speech and Language Processing*, vol. 25, no. 12, pp. 2410–2423, Dec. 2017.

[7] H. Al-Sahaf, Y. Bi, Q. Chen, A. Lensen, Y. Mei, Y. Sun, B. Tran, B. Xue, and M. Zhang, "A survey on evolutionary machine learning," *Journal of the Royal Society of New Zealand*, vol. 49, no. 2, pp. 205–228, 2019.

[8] L. Xie and A. Yuille, "Genetic CNN," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[9] M. Suganuma, S. Shirakawa, and T. Nagao, "A genetic programming approach to designing convolutional neural network architectures," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, 2017, p. 497C504. [Online]. Available: https://doi.org/10.1145/3071178.3071229

[10] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "A particle swarm optimization-based flexible convolutional autoencoder for image classification," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 8, pp. 2295–2309, 2019.

[11] B. Wang, Y. Sun, B. Xue, and M. Zhang, "Evolving deep convolutional neural networks by variable-length particle swarm optimization for image classification," in *2018 IEEE Congress on Evolutionary Computation (CEC)*, 2018, pp. 1–8.

[12] F. E. Fernandes Junior and G. Yen, "Particle swarm optimization of deep neural networks architectures for image classification," *Swarm and Evolutionary Computation*, vol. 49, pp. 62–74, 2019.

[13] B. Wang, Y. Sun, B. Xue, and M. Zhang, "A hybrid GA-PSO method for evolving architecture and short connections of deep convolutional neural networks," in *16th Pacific Rim International Conference on Artificial Intelligence, Part III (PRICAI(3) 2019). Lecture Notes in Computer Science*, vol. 11672, Cuvu, Fiji, 2019, pp. 650–663.

[14] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2261–2269, Jul. 2017.

[15] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95 - International Conference on Neural Networks*, vol. 4, 1995, pp. 1942–1948 vol.4.

[16] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

[17] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *Proceedings of the International Conference on Learning Representations (ICLR)*, May. 2015.

[18] B. Wang, Y. Sun, B. Xue, and M. Zhang, "A hybrid differential evolution approach to designing deep convolutional neural networks for image classification," in *31st Australasian Joint Conference on Artificial Intelligence (AI 2018). Lecture Notes in Computer Science.*, vol. 11320, 2018, pp. 237–250.

[19] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on Machine Learning*, vol. 37, Lille, France, 07–09 Jul 2015, pp. 448–456.

[20] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, vol. 15. JMLR Workshop and Conference Proceedings, 11–13 Apr 2011, pp. 315–323.

[21] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio, "An empirical evaluation of deep architectures on problems with many factors of variation," in *Proceedings of the 24th International Conference on Machine Learning*, 2007, pp. 473–480.

[22] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[23] A. Krizhevsky, "Learning multiple layers of features from tiny images," *""*, 2009.

[24] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[25] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio, "Contractive auto-encoders: Explicit invariance during feature extraction," in *Proceedings of the 28th International Conference on Machine Learning*, 2011, pp. 833–840.

[26] T. Chan, K. Jia, S. Gao, J. Lu, Z. Zeng, and Y. Ma, "PCANet: A simple deep learning baseline for image classification?" *IEEE Transactions on Image Processing*, vol. 24, no. 12, pp. 5017–5032, 2015.

[27] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Evolving deep convolutional neural networks for image classification," *IEEE Transactions on Evolutionary Computation*, vol. 24, no. 2, pp. 394–407, 2020.

[28] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2015.

[29] A. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, Apr. 2017.

[30] M. Lin, Q. Chen, and S. Yan, "Network in network," in *Proceedings of the International Conference on Learning Representations (ICLR)*, Banff, Canada, 2014.

[31] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016.

[32] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin, "Large-scale evolution of image classifiers," in *Proceedings of the 34th International Conference on Machine Learning*, vol. 70, International Convention Centre, Sydney, Australia, Aug. 2017, pp. 2902–2911.

[33] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," in *Proceedings of the International Conference on Learning Representations (ICLR)*, Toulon, France, 2017.

[34] I. C. Trelea, "The particle swarm optimization algorithm: convergence analysis and parameter selection," *Information processing letters*, vol. 85, no. 6, pp. 317–325, 2003.

[35] D. Bratton and J. Kennedy, "Defining a standard for particle swarm optimization," in *2007 IEEE swarm intelligence symposium.* IEEE, 2007, pp. 120–127.