

Received December 25, 2020, accepted January 14, 2021, date of publication January 18, 2021, date of current version January 26, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3052489

Particle Swarm Optimization for Automatically Evolving Convolutional Neural Networks for Image Classification

TOM LAWRENCE¹, LI ZHANG² , (Senior Member, IEEE), CHEE PENG LIM³ ,
AND EMMA-JANE PHILLIPS¹

¹Department of Computer and Information Sciences, Faculty of Engineering and Environment, Northumbria University, Newcastle upon Tyne NE1 8ST, U.K.

²National Subsea Centre, Robert Gordon University, Aberdeen AB10 7GJ, U.K.

³Institute for Intelligent Systems Research and Innovation, Deakin University, Waurn Ponds, VIC 3216, Australia

Corresponding author: Li Zhang (li.zhang@northumbria.ac.uk)

This work was supported in part by the European Regional Development Fund, and in part by Ocucon Ltd., for jointly funding a Ph.D. studentship.


ABSTRACT Designing Convolutional Neural Networks from scratch is a time-consuming process that requires specialist expertise. While automated architecture generation algorithms have been proposed, the underlying search strategies generally are computationally expensive. The existing methods also do not explore the search space efficiently, and often lead to sub-optimal solutions. In this research, we propose a novel Particle Swarm Optimization (PSO)-based model for deep architecture generation to address the above challenges. Our proposed solution incorporates three new components. Firstly, a group-based encoding strategy is devised, which enforces the candidate networks to always follow the best practices. Specifically, it ensures that the number of groups can be adjusted in accordance with the input image size. By restricting the number of groups, we can adapt the frequency of the pooling operations toward the input image size. As such, it ascertains the position and maximum frequency of the pooling operations always result in a valid network architecture without the need for additional complex governing rules. Secondly, a new velocity updating mechanism is devised, which creates new network architectures by identifying the key network configuration differences. Thirdly, a new position updating mechanism using weighted velocity strengths is devised. Both the velocity and position updating mechanisms facilitate the proposed PSO-based model to search the intermediate positions of the particles' trajectories, allowing a better trade-off between diversification and intensification to be achieved. We employ eight well-known data sets, including Convex, Rectangles, MNIST and its variants, for model evaluation. The proposed PSO-based model achieves up to 7.58% improvement in accuracy and up to 63% reduction in computational cost, in comparison with those from the current state-of-the-art methods.

INDEX TERMS Convolutional neural network, encoding, evolutionary computation, image classification, particle swarm optimization.

I. INTRODUCTION

Convolutional Neural Networks (CNNs) have shown superior performance on computer vision applications. Traditionally, networks such as LeNet [1], AlexNet [2] and ResNet [3] are designed by manual processes. The resulting models are often repurposed to solve challenges in new problem domains using methods such as transfer learning [4]. Transfer learning involves taking a pre-trained model and

training some of the layers further using a new data set [5]. Such a transfer learning process relies heavily on existing certain well-known deep architectures. They are often overly complex as the original model intention has been focused on obtaining state-of-the-art performance on complex and varied large-scale data sets. On the other hand, hand-crafting a new model from scratch requires specialist knowledge and trial-and-error owing to a vast number of design choices and hyperparameter settings. It is a bottleneck of designing a network for a new application domain.

The associate editor coordinating the review of this manuscript and approving it for publication was Kaitai Liang .

As an example, a typical CNN can be divided into several specific types of layers. In general, each layer has a specific type. There are three distinctive types of layers, i.e. convolutional layers for extracting deep features, average or max pooling layers for reducing the feature map sizes, and fully connected layers for model classification. The convolutional layers perform convolutional operations on the input images for deep feature learning while the pooling layers down-sample an input dimension to achieve better spatial invariance by reducing the feature map size and capturing invariances in image-like data [6]. A reduction in the size of a feature map also results in fewer parameters, mitigating the overfitting issue [7] and lowering the computational cost. The final layer is a fully connected layer for classification purposes.

A convolution is performed by sliding a kernel of size K over an input feature map of size M and taking the element-wise product. The distance at which the kernel is moved is known as stride S and the input feature map can also be given a P padding. The process can be repeated multiple times using different kernels to increase the number of output channels, which is denoted as the model width, e.g. in Wide Resnet [8]. Using larger strides reduces the output dimensions, however a common practice is to perform all convolution operations based on the same convolution formulation, i.e., use an appropriate padding to ensure matching between the input and output dimensions. The required padding for each side of the input volume to perform the same convolution formulation can be found using Equation 1, with the most common setting of $S = 1$.

$$P = \frac{K - S}{2} \quad (1)$$

When creating a new model from scratch, one of the main challenges is to select the appropriate number of convolutional layers, kernel sizes, and the number of output channels. The design process also needs to consider appropriate positions, frequencies and types of pooling layers. The complexity of this task is evident with the large body of studies developed for the construction of new CNN architectures. The related work in this field has resulted in numerous hand-crafted architectures including VGG [9], GoogLeNet [10], ResNet [3], and DenseNet [11]. Through parameter turning, Wide Resnet [8] was able to improve its predecessor [3] by constructing a wider variant of the originally proposed ResNet model. The vast parameter combinations indicate that it is impractical to identify the most effective combinations using a manual process, therefore, the need to automate the process of parameter settings.

Specifically, the concept of automatically evolving CNNs refers to an automatic procedure for the generation of CNN models with diverse architectures for tackling different problems [12]–[14]. Many existing studies on automatic CNN architecture generation adapt the existing meta-heuristic algorithms [15] for deep architecture search [12], [16]. Examples include Genetic Algorithm (GA) [17], [18] which employs selection, crossover and mutation operations to

evolve the search process. On the other hand, Particle Swarm Optimization (PSO) [19] utilizes the personal and global best solutions to explore the search space. IPPSO [13] has been one of the first studies to apply PSO for deep network design. Inspired by network IP addresses, the method adopts a new encoding scheme to overcome network representation constraints, so that complex models can be easily encoded. EvoCNN [14] presents a GA-based approach that uses a variable-length gene encoding strategy to represent the building blocks of a CNN model. In view of the impressive global search capabilities of PSO and its computational efficiency (e.g. as compared with the GA), psoCNN [20] has been proposed recently for deep network generation. The model adapts a traditional PSO algorithm to behave more like the GA, where a particle position is updated by copying the layers at random from either the personal or global best solutions. Although psoCNN outperforms a number of existing methods, e.g. IPPSO and EvoCNN, for deep architecture generation, the model suffers from a number of constraints. The main weaknesses is that the search space is not thoroughly explored as new particle positions can only be constructed from the layers embedded in either the global or personal best solutions. This indicates that only the new combinations of layers are explored, rather than the entirely new architectures. Another weakness of the model is the obligation to design rules to guard against invalid CNN architectures. Examples of invalid model guarding rules include the restriction of employing a fully connected layer as the first layer, and the limitations in pooling so that the resulting feature map sizes are not too small.

A. RESEARCH PROBLEMS

Owing to the difficulty in identifying efficient deep networks for tackling different problems, this research aims to provide an automatic procedure for deep CNN model generation. The research problems addressed in this study include:

- how to encode a CNN model architecture in a way that it is ensured to be both architecturally valid and reasonably built to avoid the need for additional hardcoded governing rules [20] or wasteful function evaluations;
- how to effectively guide each particle through a complex search space efficiently in order to construct more effective networks, in the meanwhile faster than the current state-of-the-art algorithms [20], [21];
- how to design efficient CNN models using an algorithm which is both easy to understand and fast to run, so that the approach can be easily exploited in both academia and industry settings with limited specialised knowledge, while not compromising the overall performance.

Fig. 1 illustrates an overview of the proposed deep architecture generation model.

B. CONTRIBUTIONS

This research has the following contributions, which aim to address the aforementioned research problems.

- A new group-based encoding strategy is proposed. Each group contains at least one convolutional layer. Its final

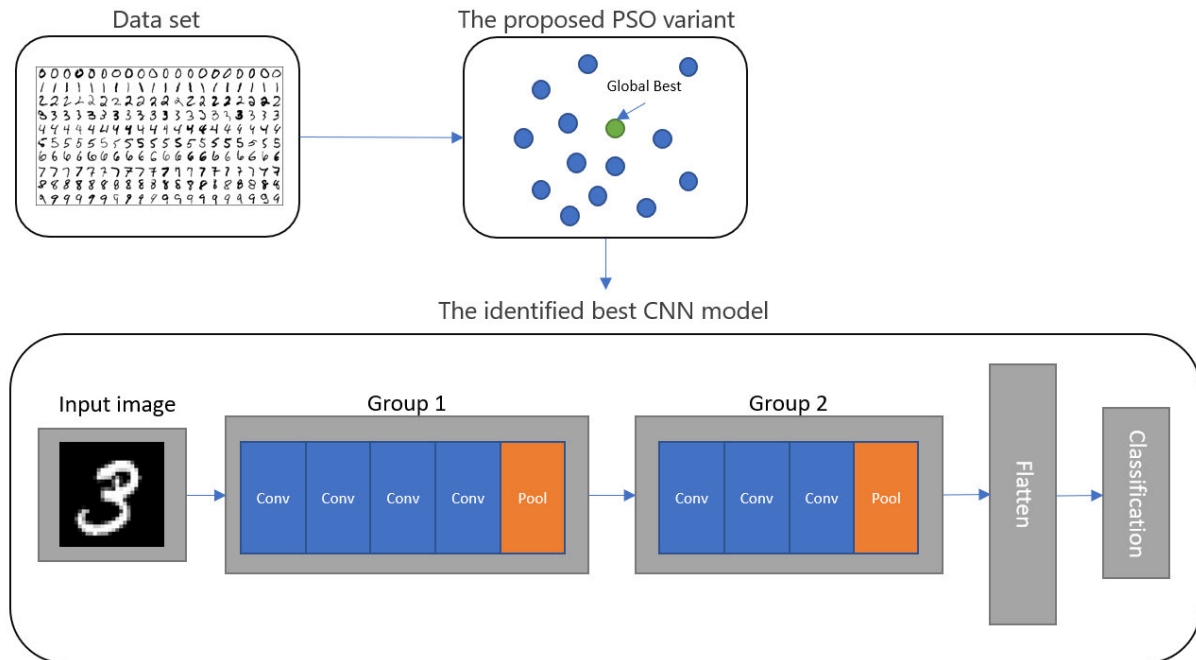


FIGURE 1. The proposed system architecture where the identified best model is indicated by the global best solution.

layer is reserved as an optional pooling layer. The number of groups can be adjusted in accordance with the input image size. By restricting the number of groups, we can adapt the frequency of the pooling operations toward the input image size. As such, it ascertains the position and maximum frequency of the pooling operations always result in a valid model architecture without the need for additional governing rules.

- A new velocity updating mechanism based on the key network configuration differences is developed. Existing models such as psoCNN [20] copy the layers randomly from the global and personal best solutions for architecture generation. This indicates that new models are always generated based on the combinations of existing layer configurations. To overcome such limitation, the new velocity updating mechanism creates new network architectures by identifying the key layer configuration differences between particles. This proposed mechanism is capable of devising new network architectures by exploring the intermediate positions of the particles' trajectories. It is also less dependent on the requirement of a good random swarm initialization.
- A new position updating mechanism with weighted velocity strengths is devised. This granular position updating mechanism enables a thorough exploration of the search space and increases the likelihood of generating diversified network configurations. It employs a weighted strength of the velocity updates for new position generation, leading to the exploration of the search space in various forces and scales to increase the chances of formulating diversified networks.

Such a granular movement also enables a better balance between intensification and diversification, in order to increase the chances of finding global optimality. Both proposed encoding and search strategies illustrate significant capabilities in enhancing performance and computational efficiency.

- A comprehensive evaluation of the proposed model with a number of data sets is conducted. Our proposed model compares favourably with the state-of-the-art models such as psoCNN [20] and notable alternative methods including EvoCNN [14]. Serving as a practical alternative to deep network generation, our proposed model achieves up to 7.58% improvement in accuracy and up to 63% reduction in computational cost, in comparison with those from the current state-of-the-art methods. Importantly, the proposed model is repeatable and easy to implement with limited hardware resources.

This paper is organized as follows. We discuss the classical search methods such as PSO and recent related studies on deep architecture generation in Section II. Section III introduces the proposed algorithm for deep architecture generation. A comprehensive evaluation is presented in Section IV. Section V concludes this research and presents the directions for further research.

II. RELATED STUDIES

There are many recent applications of CNN models [3], e.g. detecting material defects in industrial settings [22] and addressing medical problems such as skin lesion segmentation [23], fall detection [24] and health monitoring [25], [26]. CNN models have also shown groundbreaking results in

classifying handwritten digits [1]. Object localization models e.g. YOLO [27], [28], Fast R-CNN [29] and Faster R-CNN [30] and object segmentation methods e.g. Mask R-CNN [31] all make use of CNNs as the backbones to perform detection, localization and segmentation tasks. Advances in automated CNN architecture generation, therefore, have a significant impact in many domains.

Evolutionary algorithms such as PSO show superior search capabilities in solving diverse optimization problems. Related studies adapt the PSO algorithm to CNN architecture generation. Before discussing recent deep architecture generation methods, we first introduce the PSO algorithm.

Proposed by [19], PSO is a popular swarm intelligence algorithm which simulates natural social behaviours, e.g. bird flocking or fish schooling. The concept of PSO is to create a swarm of particles, where each particle explores a search space guided by the best-known position of the entire swarm, g_{best} , as well as its individual best experience, p_{best} . In each iteration, the particle position is updated by adding a velocity to the current position vector. The formula for velocity calculation can be divided into three main components, i.e. inertia, cognitive, and social components.

The inertia component shown in Equation 2 multiplies the current velocity V for particle i in the t -th iteration with a weight w , which controls the impact of the previous velocity on the new velocity calculation.

$$inertia = wV_i^t \quad (2)$$

The cognitive component shown in Equation 3 multiplies the distance between the current particle position X and its personal best solution P by a cognitive acceleration coefficient c_1 as well as a random parameter r_1 .

$$cognitive = c_1 r_1 (P_i^t - X_i^t) \quad (3)$$

Similarly, the social component shown in Equation 4 multiplies the distance between the current particle position X and the global best solution G by a social acceleration coefficient c_2 and a random value r_2 .

$$social = c_2 r_2 (G^t - X_i^t) \quad (4)$$

The acceleration coefficients c_1 and c_2 control the degrees at which a given position update is guided by the cognitive or social component. The complete velocity updating formula shown in Equation 5 produces the final velocity for the $(t+1)$ -th iteration.

$$V_i^{t+1} = wV_i^t + c_1 r_1 (P_i^t - X_i^t) + c_2 r_2 (G^t - X_i^t) \quad (5)$$

The new position in the $(t+1)$ -th iteration is produced using Equation 6, based on the velocity yielded from Equation 5.

$$X_i^{t+1} = X_i^t + V_i^{t+1} \quad (6)$$

In comparison with other optimization methods, PSO shows impressive search capabilities in solving single- and multi-objective optimization problems [32]–[35]. It is also

relatively easy in implementation and computation. Because of this, it has been widely adopted for optimizing CNN models [36], [37].

One of the first studies of applying PSO to CNN generation was IPPSO [13]. The IPPSO model adopts a flexible encoding scheme to address the limitation of the traditional PSO model where particles are required to have a fixed length. The encoding scheme is inspired by IP addressing and subnetting in computer network research. Specifically, an IP address strategy is used to represent the layer parameters as a series of bits. As an example, the kernel sizes within the search range between 1 and 8 are encoded into 3 bits, i.e. a kernel size of 1 is encoded as 001. A subnet is used to identify a layer type, i.e. a convolutional, pooling, or fully connected layer. Evaluated using a variant of the MNIST data set, i.e. MNIST-BI (MNIST with background images), IPPSO has achieved state-of-the-art performance. Experiments were reported as taking on average 2.5 hours to complete for each data set.

A novel PSO variant namely psoCNN [20] was introduced for deep architecture generation. Based on a selection criterion for position updating in a swarm, psoCNN selects the candidate layers from either the global or personal best solution. It outperforms several state-of-the-art models for architecture generation, including GA-based methods such as EvoCNN [14], PSO-based methods such as IPPSO [13], and reinforcement learning-based methods such as MetaQNN [21]. It also depicts a low computational cost. One weakness of psoCNN is that the particles are not able to fully explore a search space, as compared with case in the traditional PSO search operation. Instead, layers are copied from the personal and global best solutions directly which significantly reduces search diversity and increases the likelihood of being trapped in local optima.

Regardless of the search strategies, the bottleneck in optimizing CNN models is the considerable computational cost in fitness evaluation of the candidate models. Such a fitness evaluation procedure needs to be repeatedly performed over a significant number of times during the optimization process. Related studies such as [38] indicated that only 1.14% of the candidate models achieve good results, and 88% provide reasonable results, while the remaining illustrate poor performances. The study indicates that the majority of the computational cost has been spent evaluating less optimal networks during the search process. As such, a linear prediction model was proposed in [38] as the performance predictor of deep networks. Such techniques can be combined with any optimal architecture generation process to improve computational efficiency while effectively exploring the search process.

Recently, Sun *et al.* [39] proposed a GA-based deep architecture generation model, namely CNN-GA, to automatically devise networks for image classification. Since the GA-based optimization process commonly employs a fixed-length encoding strategy, where the length of chromosomes is fixed and must be specified beforehand, CNN-GA introduces a variable-length encoding operation to overcome

this restriction. The encoding scheme considers both the skip blocks and pooling layers. A skip block contains two convolutional layers with fixed kernel sizes of 3×3 and 1×1 , respectively, along with a skip connection. A binary tournament selection mechanism [40] is adopted, whereby two individuals are selected at random based on the fitness scores. A crossover operation is subsequently performed with respect to a random threshold. Mutations are conducted by adding, removing, or modifying layers. Evaluated using CIFAR-10 and CIFAR-100 data sets, CNN-GA outperforms several existing methods such as NASNet [41] and DARTS [42].

Another deep architecture generation model, namely Automatically Evolving CNN (AE-CNN), was proposed in [43]. For deep network generation, the AE-CNN model employs either a ResNet block [3] with three convolutional layers and skip connections, or a DenseNet block [11] with four convolutional layers and skip connections. The pooling layer is designed to perform mean and max-pooling using a 2×2 kernel. The computational cost of the search process is handled by introducing asynchronous computation and caching, which successfully reduces the cost of the fitness evaluation. Based on the CIFAR-10 and CIFAR-100 data sets, the proposed method is able to reduce the computational cost from the 100 GPU days as required by MetaQNN [21] to 35 GPU days. However, their proposed strategies have not explored kernel sizes other than 3×3 in the convolutional layers.

Gao *et al.* [44] proposed a gradient-priority PSO algorithm for deep network generation for undertaking EEG-based emotion recognition. It addresses the efficiency limitations of automated architecture search in a high-dimensional search space. A hybrid model based on PSO and the gradient descent method is proposed for carrying out a weighted exploration in dimensions of greater importance. The method identifies the optimal settings of both the convolutional and pooling layers. To be specific, the kernel sizes and the number of output channels are optimized for the convolutional layers. For pooling layers, the optimal pooling types and the kernel sizes are identified. Instead of calculating the distance between the current particle and the global best solution, the model computes the maximum gradient position for each particle, which is subsequently used to accelerate the particle movement in the direction illustrating the most impact. The method achieves an impressive performance for deep network generation for emotion recognition in comparison with those from existing studies.

Within the family of neural architecture search (NAS) based algorithms, Kwasigroch *et al.* [45] aimed to reduce the computational cost when generating CNN model architectures from scratch. A novel network morphism operation is combined with a greedy search algorithm (i.e. hill-climbing) for generating an optimal architecture in undertaking malignant melanoma classification. The search mechanism incrementally increases the model size over a number of iterations. Savings in the computational cost are achieved by inheriting the previously trained weights over to the new offspring.

The architecture search initially constructs a base model. It consists of a 3×3 convolutional layer, followed by a max-pooling layer, a second 3×3 convolutional layer and finally a Sigmoid activation function. This base model is trained before multiple offspring solutions are created. The offspring networks are created by applying an operation to extend the parent model. The available extension methods include inserting new layers, altering the number of output channels on existing layers, or stacking two layers side by side and applying an addition or concatenation operation to the output of each layer. The offspring models are trained when training of the parent model is completed. The best offspring model from the current iteration is selected to become the parent for the next iteration. Impressive results are achieved. The entire search process requires 18 GPU hours, as compared with 38 GPU hours required by a related method reported in [43].

Wang *et al.* [46] proposed an efficient PSO model, namely EPSOCNN, for deep architecture generation. It employs the classical PSO algorithm to optimize a single network block only, i.e. a dense block, to accelerate the evolutionary process. Proposed in DenseNet [11], a dense block contains 3×3 convolutions and skip connections. Specifically, the EPSOCNN model optimizes the number of output channels within a block using a widening factor, as well as the number of convolutional layers with a search range between 6 and 32 within a single block. Once the best dense block is found, a second-stage process progressively stacks the optimized dense block to identify the optimal number of blocks, in order to construct the final network. With fewer than 4 GPU days, EPSOCNN yields an error rate of 3.58% on the CIFAR-10 data set, with an improvement of 1.12% over that in [43]. Note the study conducted has been based on the optimization of an existing state-of-the-art network block.

There are also other related studies, such as [47] and [48], which adopt non-evolutionary techniques such as pruning to remove insignificant weights for devising deep networks.

III. THE PROPOSED APPROACH FOR DEEP ARCHITECTURE GENERATION

We propose a novel PSO-based approach for deep architecture generation, which addresses the drawbacks of the existing methods. The proposed PSO variant incorporates a group-based encoding strategy as well as search operations motivated by network configuration variations and weighted velocity strengths to increase search diversity. Specifically, the proposed model employs a group-based encoding strategy to stimulate particle natural movement while guarding against invalid architectures. It also employs a novel particle distance computation strategy for calculating the differences between the current position of particle X and the global best g_{best} and personal best p_{best} solutions, respectively. Such a search strategy enables the swarm to thoroughly explore the search space between the particles and the local and global optimal signals, in an attempt to identify the network configuration gaps, therefore increasing the chances of achieving

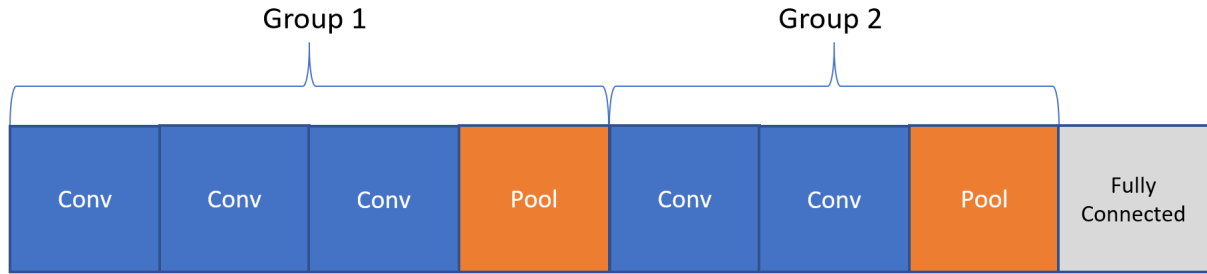


FIGURE 2. An example model containing two groups where each group contains convolutional layers and an optional final pooling layer.

global optimality. To increase diversification, a new velocity updating mechanism is adopted to randomly select the layers from either the distance between X and g_{best} , or the distance between X and p_{best} . Moreover, the proposed model weights the strength of velocity updates to implement a granular movement to balance between exploration and exploitation.

The pseudo-code of the proposed PSO algorithm for deep architecture generation is illustrated in Algorithm 1. Firstly, the training and test sets are obtained for each image database. A swarm of particles is initialized, where the position of each particle presents a potential network architecture. The proposed PSO algorithm is used to evolve the architecture and parameter settings of deep networks based on the proposed search operations. Specifically, the particles explore the search space by using a new weighted position updating procedure. During fitness evaluation, each particle encoded position is converted into a CNN model, which is subsequently trained using the training data set. The average training entropy loss is used as the fitness score to update those of p_{best} and g_{best} accordingly, if the current solution is fitter. The best architecture is obtained based on the global best position identified during the search process. It is then fully trained using the training set with a comparatively larger training epoch, and tested using the unseen test set. We introduce each key proposed component in detail in the following subsections.

A. THE PROPOSED ENCODING STRATEGY

A search space consists of all possible combinations of available settings including the number of layers and layer configurations. A particle represents one instance of a particular set of encoded settings which are used to describe a model architecture. The proposed encoding strategy adopts a group-based structure for describing a network, as shown in Fig. 2.

The underlying rational of the group-based encoding strategy is as follows. It is designed by embedding human knowledge to ensure that the convolutional layers will always be followed by optional pooling layers. The number of pooling layers can be adjusted in accordance with the input image size. In other words, the encoding process ensures that the position of the pooling layers and the frequency of pooling operations will be valid, corresponding to the input image size. It simplifies implementation and does not artificially disrupt the natural particle movement.

Algorithm 1 The Proposed PSO-Based Deep Architecture Generation Model

```

1: procedure PSO-based CNN model generation
2:   Initialize training and test data sets
3:   Initialize a swarm population
4:   for (each  $t$  iteration) do
5:     for (each particle  $X$  in swarm) do
6:       Construct a new model based on the current
       particle position
7:        $loss \leftarrow trainModel()$ 
8:       if  $loss < p_{best}.fitness$  then
9:          $p_{best}.fitness \leftarrow loss$ 
10:        Update the personal best position
11:      end if
12:      if  $loss < g_{best}.fitness$  then
13:         $g_{best}.fitness \leftarrow loss$ 
14:        Update the global best position
15:      end if
16:      Update the particle position by using the pro-
      posed weighted position updating procedure
17:    end for
18:  end for
19:  Save  $g_{best}$  and initialize the identified best model
  based on  $g_{best}$ 
20:  Train the final network using the training set and a
  larger training epoch
21:  Test the final model with the unseen test set
22:  Output the classification error rate
23: end procedure

```

Specifically, a group contains a number of convolutional layers and an optional pooling layer. A network contains multiple groups to vary down-sampling, but is limited by g_{max} to ensure down-sampling does not occur too frequently. The final group in a network is always followed by a fully connected layer for classification. This proposed group-based deep network generation strategy ensures all the formulated CNN models are valid whilst still providing sufficient flexibility for search space exploration without the requirement to specify the additional guarding rules.

In addition to the layer type, hyperparameter meta-data are encoded. In the case of a convolutional layer, we encode the kernel size as $\{k \in \mathbb{R} | k_{min} \leq k \leq k_{max}\}$, and the number

TABLE 1. The optimized network parameters and their corresponding search ranges. The settings of the search ranges adopted in our experiments are detailed in Section IV.

Layer	Parameter	Range
Convolution	Kernel k	$\{k \in \mathbb{R} k_{min} \leq k \leq k_{max}\}$
	Number of channels c_{out}	$\{c_{out} \in \mathbb{R} out_{min} \leq c_{out} \leq out_{max}\}$
Pooling	Pooling type p_{type}	$\{p_{type} \in \mathbb{R} 0 \leq p_{type} \leq 1\}$
Depth	Number of layers l	Automatically optimized during the velocity update evolving process

of output channels as $\{c_{out} \in \mathbb{R} | out_{min} \leq c_{out} \leq out_{max}\}$. Pertaining to the pooling layers, we encode the pooling type as $\{p_{type} \in \mathbb{R} | 0 \leq p_{type} \leq 1\}$. We assign the type of the pooling layer according to the value of p_{type} . The types of pooling include the max and average poolings, or none in which case pooling is skipped. We assign different types of pooling based on a pre-determined threshold setting (see Section IV for detail).

Our proposed algorithm encodes the initial network depth as $\{d \in \mathbb{R} | 1 \leq d \leq d_{max}\}$ which is subsequently split into groups. A network contains g groups where $\{g \in \mathbb{R} | 1 \leq g \leq g_{max}\}$. We initialize the number of convolutional layers in each group by setting $l_{initial}$ using Equation 7 so that the initial number of layers is evenly distributed between groups. During the evolving process, the number of convolutional layers for each group is further optimized. The final layer of a group is always a pooling layer, where the pooling type or whether pooling occurs is determined by the current value of p_{type} . As down-sampling halves the input dimension size, another key advantage of the proposed group-based method is that g_{max} can be set to reflect the dimension of the input images, thus ensuring down-sampling is omitted in situation that can cause the model to become invalid, mitigating the need for additional rules. In other words, we ensure the position and maximum frequency of the pooling always result in a valid model architecture without the need for complex governing rules such as those imposed by psoCNN, which interrupt the natural particle movements and complicate the implementation. In this research, we limit the number of groups to 2, owing to the input image size (i.e. 28×28). However, for data sets with larger images, the number of groups can be increased accordingly.

$$l_{initial} = \frac{d}{g} \quad (7)$$

As the final group is always followed by a fully connected layer, the model maps the output channel in the final network layer c_{out} to the number of target classes n_{class} automatically. The parameters optimized by the proposed PSO algorithm including their search ranges are provided in Table 1.

B. INITIALIZATION

A swarm consists of N particles initialized with randomly assigned positions. The first step of initialization is to randomly set the numbers of groups g and depth d for each particle. For each group, we initialize l convolutional layers according to Equation 7 with one pooling layer. Initializing a convolutional layer is performed by randomly assigning the kernel size k and the number of output channel c_{out} ,

respectively. The value of p_{type} is also randomly selected for the pooling layer.

C. FITNESS EVALUATION

When evaluating the fitness of a particle, we first construct a new model based on the hyperparameter settings for a given particle. We train the model on the training set for 1 epoch. Next, we compute the average loss of the Adam optimizer [49] during the training phase. This average training loss is used as the fitness score. The overall objective of the PSO algorithm during the optimization process is to minimise the average loss. The model with the most optimal network configuration is recommended as the global best solution. It is subsequently trained with a larger number of epochs using the training set and evaluated with the unseen test set for performance comparison.

D. PARTICLE DISTANCE CALCULATION

In the PSO algorithm, an individual particle X moves in the search space by following the personal and global best solutions. The calculation of the distance from the particle's personal best position p_{best} , and the distance from the global best position g_{best} is vital for search space exploration. We introduce the proposed position distance computation between two particles, as follows.

Particles often have different lengths owing to different architecture configurations. As such, on a group-by-group basis, the shallower group is padded to the same length by temporarily appending the empty layers. Once both particle lengths match, the distance of particle $X2$ with respect to particle $X1$ defined as $X1 - X2$ is calculated depending on the layer types. For the convolutional layers, $X1 - X2$ is computed by subtracting the current values of k and c_{out} of $X2$ from those of $X1$ to return the distances represented as Δk and Δc_{out} . Two special cases exist. Specifically, if a convolutional layer of $X1$ is empty, the output is empty. Conversely, if a convolutional layer of $X2$ is empty, the configurations of the corresponding convolutional layer for $X1$ are copied. Pertaining to the pooling layer, $X1 - X2$ is computed by subtracting the current value of p_{type} of $X2$, from that of $X1$ to return the distance represented as Δp_{type} . Fig. 3 visually demonstrates the proposed distance computation mechanism between two particles, including the special and normal cases. The proposed particle difference computation mechanism is applied to the distance computation between the current particle and its personal and the global best solutions by replacing $X1$ with p_{best} and g_{best} , respectively.

In comparison with the existing studies such as psoCNN [20], where the distance between two particles is

	Group				
X1	Conv $k = 7$ $c_{out} = 100$	Conv $k = 4$ $c_{out} = 20$	Conv $k = 6$ $c_{out} = 20$	None	Pool $p_{type} = 0.6$
X2	Conv $k = 3$ $c_{out} = 30$	Conv $k = 7$ $c_{out} = 90$	None	Conv $k = 5$ $c_{out} = 40$	Pool $p_{type} = 0.2$
X1 – X2	Conv Diff $\Delta k = 4$ $\Delta c_{out} = 70$	Conv Diff $\Delta k = -3$ $\Delta c_{out} = -70$	Conv Diff $\Delta k = 6$ $\Delta c_{out} = 20$	None	Pool Diff $\Delta p_{type} = 0.4$

FIGURE 3. Distance between particles calculated as $X1 - X2$.

	Group						Group		
p_{best}	Conv $k = 7$ $c_{out} = 100$	Conv $k = 5$ $c_{out} = 30$	Conv $k = 3$ $c_{out} = 60$	Pool $p_{type} = 0.2$	g_{best}		Conv $k = 6$ $c_{out} = 80$	None	Pool $p_{type} = 0.2$
X	Conv $k = 3$ $c_{out} = 30$	Conv $k = 7$ $c_{out} = 90$	None	Pool $p_{type} = 0.5$	X		Conv $k = 3$ $c_{out} = 30$	Conv $k = 7$ $c_{out} = 90$	Pool $p_{type} = 0.2$
$p_{best} - X$	Conv Diff $\Delta k = 4$ $\Delta c_{out} = 70$	Conv Diff $\Delta k = -2$ $\Delta c_{out} = -60$	Conv Diff $\Delta k = 3$ $\Delta c_{out} = 60$	Pool Diff $\Delta p_{type} = -0.3$	$g_{best} - X$		Conv Diff $\Delta k = 3$ $\Delta c_{out} = 50$	None	Pool Diff $\Delta p_{type} = 0$

FIGURE 4. Distance calculation between particle X and p_{best} and g_{best} respectively.

yielded by directly copying the layer configurations from p_{best} or g_{best} , the proposed movement mechanism identifies the configuration variations between two particles as indicated in Fig. 4 and effectively explores the search space between the current particle and the local and global best solutions to avoid stagnation. In other words, the proposed strategy is able to devise new layer configurations, instead of inheriting the existing layer structures from p_{best} and g_{best} directly, in order to increase search diversity. Therefore, the resulting model is able to better explore the search space and attain global optimality. We explain how these position differences are used with respect to the velocity calculation in the next section.

E. VELOCITY CALCULATION

To calculate velocity V , we compute the distances of X with respect to p_{best} and g_{best} . Velocity is calculated for each group respectively using the proposed distance calculation

mechanism. Fig. 4 illustrates an example for the distance calculation between the current particle and the personal and global best solutions.

Next, both resulting $g_{best} - X$ and $p_{best} - X$ distances are padded to the same length by temporarily appending the empty layers to the shallower group so that the same depth is achieved.

We iterate over m layers and choose whether to keep the resulting velocity calculation from $g_{best} - X$ or $p_{best} - X$ in the final velocity by generating a random number $\{r \in \mathbb{R} | 0 \leq r \leq 1\}$ and comparing it against a threshold α . The resulting velocity for each element is determined using Equation 8, as shown in Fig. 5, where g represents the group number and m denotes the number of layers in each group. In addition, we set $\alpha=0.5$ to best match the setting of [20], in order to facilitate a direct comparison with the existing methods.

$$V_m^g = \begin{cases} p_{best_m}^g - X_m^g & \text{if } r \leq \alpha, \\ g_{best_m}^g - X_m^g & \text{otherwise} \end{cases} \quad (8)$$

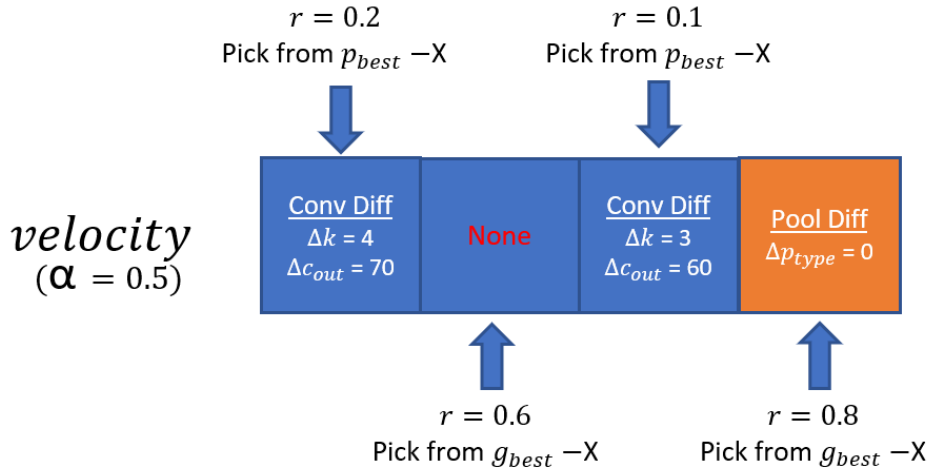


FIGURE 5. Calculating the final velocity by picking at random from $p_{best} - X$ or $g_{best} - X$.

F. PARTICLE UPDATE

Velocity V obtained from the process is subsequently used to update the position of particle X . To facilitate a thorough exploration of the search space as well as increase the likelihood of generating more diversified layer configurations, we adopt a weighted velocity strength for position updating. Specifically, unlike the original PSO algorithm where the full velocity is used for position updating, as indicated in Equation 6, we adopt a weighting factor β to apply partial velocity for new position generation in the $(t + 1)$ -th iteration. The proposed position updating formula is defined in Equation 9.

$$X^{t+1} = \beta V + X^t \quad (9)$$

where β is the weighting factor used to control the degree at which the position of a particle is changed with respect to the velocity. We set $\beta=0.5$ in Equation 9 based on trial-and-error. Moreover, position updates with respect to the kernel size k are bound purely by k_{min} . Likewise, position updates with respect to the number of channels c_{out} are bound only by out_{min} , in order to ensure the values remain valid. Such a position updating mechanism provides a granular and thorough exploration of the search space to increase the likelihood of finding global optimality and avoiding being trapped in local optima.

IV. EXPERIMENTAL STUDIES

In our experimental studies, we initialize a swarm of 20 particles, a maximum of 10 iterations, and the settings in Table 2, for identifying the optimal network configuration. The proposed model is implemented using PyTorch v1.5 [50]. During the optimization process, we train each model recommended by each particle for 1 epoch with a mini-batch size of 64. We adopt the cross-entropy loss as the fitness criterion and Adam as the optimizer with a learning rate of 0.001. The final best model indicated by the global best solution is re-trained for 100 epochs using the training set, which is subsequently evaluated using the test set.

TABLE 2. Algorithm settings and the search space used in our experiments. We adopt the settings to closely match those of existing studies [20] so that a fair comparison can be made.

Name	Description	Value Used
k_{min}	Minimum kernel size	3
k_{max}	Maximum kernel size	7
out_{min}	Minimum number of channels	3
out_{max}	Maximum number of channels	256
d_{max}	Maximum depth	20
g_{max}	Maximum number of groups	2
α	Layer selection boundary threshold	0.5
β	Weighting factor	0.5

All convolutions are performed with the same convolution formulation where padding is applied to the input to ensure matching between the input and output dimensions. Each convolution within the CNN, except for the first one, is preceded with a dropout layer with a dropout probability of 0.5. A convolutional layer is always followed by batch normalisation [51] and a ReLu activation function [52]. For the pooling operations, we adopt a non-overlapping 2×2 pooling mechanism. The final layer of each yielded model is a fully connected layer, which maps the outputs of the final convolutional layer to the number of classes of the respective data set.

A. ALGORITHM PARAMETER SETTINGS

We adopt the settings shown in Table 2. The values and constraints in Table 2 are manually selected to best match those proposed in [20], in order to facilitate a direct comparison between the competing methods. As an example, the α parameter in Equation 8 is used as the threshold to determine if each dimension of the new velocity is generated using the position difference from the personal or global best solution. We set $\alpha=0.5$ to best match the setting adopted by psoCNN [20] to ensure a fair comparison. Such a setting (i.e. $\alpha=0.5$) gives an equal consideration of the position difference from both best solutions as a reasonable trade-off. In our experiments, we adopt $\beta=0.5$ in Equation 9 based on

manual trial-and-error. β is the weight factor used to control the degree at which the position of a particle is changed with respect to the velocity. We set $\beta=0.5$ in Equation 9 to determine the effects of the updated velocity for new position generation. As the benchmark data sets used have a size of 28×28 , we set $g_{max} = 2$ so that down-sampling does not cause the dimension to reduce below 8×8 . Moreover, the pooling parameter, p_{type} , is optimized during the optimization process as shown in Table 1. It has the search range of $[0, 1]$. Equation 10 is used to define the pooling type according to the optimized value of p_{type} .

$$pooling = \begin{cases} NoPooling & \text{if } p_{type} \leq 0.33, \\ AvePooling & \text{if } p_{type} > 0.33 \ \& \ p_{type} \leq 0.66, \\ MaxPooling & \text{otherwise} \end{cases} \quad (10)$$

B. BENCHMARK MODELS

A number of hand-crafted networks and deep architecture generation methods are employed for performance comparison. In particular, several PSO-based algorithms are used in our experiments. As an example, IPPSO [13] is adopted which employs a test methodology consisting of 20 particles over 10 iterations constrained to a maximum of 9 convolutional layers and 3 fully connected layers.

Another state-of-the-art PSO-based architecture generation method, i.e. psoCNN [20], is also selected for comparison. It adopts a test methodology consisting of 20 particles optimized over 10 iterations. The search space constraints include a kernel size between 3×3 and 7×7 inclusive, a maximum of 256 channels and an upper limit of 20 layers comprising a combination of convolutional, pooling and fully connected layers. During the training phase, each candidate model is trained for 1 epoch using the Adam optimizer, where the swarm objective is to minimise the average loss.

In addition to PSO-based approaches, we select the GA-based EvoCNN [14] model as another method for performance comparison. EvoCNN employs the selection, crossover and mutation operators for CNN architecture generation. It generates offspring chromosomes from an initial parent pool size of 100. The variable-length gene encoding strategy is used to represent the CNN architectures with diverse lengths where convolutional, pooling and fully connected layers are embedded.

Moreover, MetaQNN [21] is used for performance comparison. It is a meta-modeling algorithm based on reinforcement learning for generating high-performing CNN architectures. Finally, three variants of the LeNet model [1] are also adopted as examples of hand-crafted networks for performance comparison. We select LeNet-1 which contains twelve convolutional layers, two average pooling layers and one linear layer, LeNet-4 which contains twenty convolutional layers, two average pooling layers and two linear layers, and LeNet-5 which contains twenty-two convolutional layers, two average pooling layers and three linear layers.

C. DATA SETS

We employ a total of eight well-known benchmark data sets for performance comparison, namely Rectangles, Rectangles-I, Convex, MNIST, and four MNIST variant data sets. These data sets are selected to aid direct comparison between the proposed method and the aforementioned recent models. Specifically, the MNIST data set [1] contains images of handwritten digits ranging from 0 to 9, with size-normalized and centered. The data set consists of 60,000 training and 10,000 test samples.

We also employ several MNIST variant data sets for evaluation. These data sets are comparatively more challenging owing to the additional distraction factors. As an example, MNIST-RD [53] comprises rotated MNIST digits, while MNIST-RB [53] contains MNIST digits with random background. MNIST-BI [53] consists of MNIST digits against background images, while MNIST-RD+BI [53] contains rotated MNIST digits against background images.

In addition to MNIST and its variant data sets, other additional data sets adopted for comparison include Convex [53], Rectangles [53], and a variant of Rectangles, i.e. Rectangles-I [53], which consists of rectangles against image backgrounds.

Table 3 provides a summary of the aforementioned data sets.

D. RESULTS

1) PERFORMANCE COMPARISON WITH EXISTING STUDIES

Table 4 presents the experimental results for eight data sets. For the three LeNet models, i.e. LeNet-1, LeNet-4 and LeNet-5, the results for the MNIST data set are taken from the original publication [1]. We conduct experiments using the three LeNet models for the remaining data sets and present the results in Table 4. For all other benchmark methods, i.e. MetaQNN [21], EvoCNN [14], IPPSO [13] and psoCNN [20], we provide the results reported in their original studies, in order to ensure a fair comparison.

The last two rows in Table 4 present the mean classification error rate achieved by the proposed method over 10 runs, as well as the best result from the 10 runs, for each data set. The remaining rows are the mean and best error rates reported by the compared methods. The results in which the proposed method outperforms the competitors are highlighted in bold. As illustrated in Table 4, our proposed approach achieves a superior performance, indicated by a reduction in the error rates reported across nearly all the test data sets in comparison with all the baseline models, with the same swarm size, iterations and constraints. In addition, psoCNN is the best performing baseline method across all data sets. In Table 5, we compare the error rates of our proposed model against those of psoCNN [20], to clearly indicate performance improvements.

MNIST represents a relatively simple benchmark data set with limited room for improvement. The networks devised by our proposed method achieve a mean error rate of 0.38%

TABLE 3. A summary of the data sets used in our experiments, all of which have an input size of 28 x 28 x 1.

Data set	Description	Classes	Training Samples	Test Samples
MNIST	Handwritten digits	10	60,000	10,000
MNIST-RD	MNIST with rotated digits	10	12,000	50,000
MNIST-RB	MNIST with random backgrounds	10	12,000	50,000
MNIST-BI	MNIST with background images	10	12,000	50,000
MNIST-RD+BI	MNIST with rotated digits and background images	10	12,000	50,000
Rectangles	Rectangle border shapes	2	12,000	50,000
Rectangles-I	Rectangle border shapes against image backgrounds	2	12,000	50,000
Convex	Convex shapes	2	8,000	50,000

TABLE 4. Experimental results compared against various benchmark methods in terms of error rates. Results in bold indicating a reduction in error rate when compared with the benchmark methods. For the LeNet models, we report the results for MNIST taken from the original study [1] alongside our own results for the remaining data sets. The results of MetaQNN, EvoCNN, IPPSO and psoCNN are extracted from their original studies, i.e. [13], [14], [21] and [20], respectively.

Model	MNIST	MNIST-RD	MNIST-RB	MNIST-BI	MNIST-RD+BI	Rectangles	Rectangles-I	Convex
Hand-crafted architectures								
LeNet-1	1.70% [1]	19.3%	7.50%	9.80%	40.06%	0.08%	16.92%	10.61%
LeNet-4	1.10% [1]	11.79%	6.18%	8.96%	33.83%	0.05%	16.09%	8.39%
LeNet-5	0.95% [1]	11.10%	5.99%	8.70%	34.64%	0.07%	12.48%	8.40%
Reinforcement learning techniques								
MetaQNN (best) [21]	0.44%	-	-	-	-	-	-	-
Evolutionary optimization techniques								
EvoCNN (best) [14]	1.18%	5.22%	2.80%	4.53%	35.03%	0.01%	5.03%	4.82%
EvoCNN (mean) [14]	1.28%	5.46%	3.59%	4.62%	37.38%	0.01%	5.97%	5.39%
IPPSO (best) [13]	1.13%	-	-	-	33%	-	-	8.48%
IPPSO (mean) [13]	1.21%	-	-	-	34.50%	-	-	12.06%
psoCNN (best) [20]	0.32%	3.58%	1.79%	1.90%	14.28%	0.03%	2.22%	1.70%
psoCNN (mean) [20]	0.44%	6.42%	2.53%	2.40%	20.98%	0.34%	3.94%	3.90%
Our system								
ours (best)	0.35%	3.23%	1.8%	2.2%	11.61%	0%	1.01%	1.36%
ours (mean)	0.38%	3.9%	1.88%	2.46%	13.4%	0%	1.57%	1.63%

TABLE 5. The mean error rates over 10 runs for the proposed method and psoCNN [20], along with the performance differences between the two methods (where the (-) symbol indicates that the proposed model is better and the (+) symbol indicates that the proposed model is worse).

Model	MNIST	MNIST-RD	MNIST-RB	MNIST-BI	MNIST-RD+BI	Rectangles	Rectangles-I	Convex
ours (best)	0.35%	3.23%	1.8%	2.2%	11.61%	0%	1.01%	1.36%
ours (mean)	0.38%	3.9%	1.88%	2.46%	13.4%	0%	1.57%	1.63%
psoCNN (best) [20]	0.32%	3.58%	1.79%	1.90%	14.28%	0.03%	2.22%	1.70%
psoCNN (mean) [20]	0.44%	6.42%	2.53%	2.40%	20.98%	0.34%	3.94%	3.90%
error difference (best)	0.03%(+)	0.35%(-)	0.01%(+)	0.30%(+)	2.67%(-)	0.03%(-)	1.21%(-)	0.34%(-)
error difference (mean)	0.06%(-)	2.52%(-)	0.65%(-)	0.06%(+)	7.58%(-)	0.34%(-)	2.37%(-)	2.27%(-)

for MNIST, which is an improvement of 0.06% as compared with those from the networks yielded by psoCNN. The best CNN model devised by our proposed method achieves the best error rate of 0.35%, which is within a reasonable error margin as compared with the best error rate of 0.32% from psoCNN.

With respect to the MNIST-RD data set, the psoCNN method obtains a mean error rate of 6.42%. Our method produces a mean error rate of 3.9%, which illustrates an improvement of 2.52%. The devised best network achieves the best error rate of 3.23%, outperforming the best model from psoCNN by 0.35%.

Moreover, for MNIST-RB, again psoCNN is the leading baseline method with a mean error rate of 2.53%. Our proposed method yields a mean error rate of 1.88%, and outperforms psoCNN by 0.65%.

For the MNIST-BI data set, our devised networks achieve a mean error rate of 2.46%. The psoCNN model achieves a slightly better performance with a mean error rate of 2.4%.

For MNIST-RD+BI, our generated networks achieve a mean error rate of 13.4%, which shows a significant improvement of 7.58% over the mean result of 20.98% obtained by the networks devised by psoCNN. Our identified best network also produces the best error rate of 11.61%, and outperforms that yielded by psoCNN by 2.67%.

For the Rectangles data set, our approach achieves the mean and best error rates of 0%. This indicates that our approach is able to devise 10 CNN networks, all of which achieve a 100% accuracy rate. On the other hand, psoCNN and EvoCNN obtain the mean error rates of 0.34% and 0.01%, respectively.

For the Rectangles-I data set, our optimized networks achieve a mean error rate of 1.57%, and outperform those generated by psoCNN by 2.37%. Moreover, our identified best CNN model achieves the best error rate of 1.01%, which shows an improvement of 1.21% as compared with the best network yielded by psoCNN.

TABLE 6. Result comparison between our model using the encoding strategy only, and our model using both encoding and search strategies, over 10 runs.

Model	MNIST	MNIST-RD	MNIST-RB	MNIST-BI	MNIST-RD+BI	Rectangles	Rectangles-I	Convex
psoCNN (best) [20]	0.32%	3.58%	1.79%	1.90%	14.28%	0.03%	2.22%	1.70%
psoCNN (mean) [20]	0.44%	6.42%	2.53%	2.40%	20.98%	0.34%	3.94%	3.90%
ours (best) encoding + copy	0.36%	3.32%	2.07%	2.44%	15.68%	0.02%	1.58%	1.47%
ours (mean) encoding + copy	0.42%	4.76%	2.25%	3.43%	18.00%	0.47%	2.30%	1.94%
ours (best)	0.35%	3.23%	1.8%	2.2%	11.61%	0%	1.01%	1.36%
ours (mean)	0.38%	3.9%	1.88%	2.46%	13.4%	0%	1.57%	1.63%

TABLE 7. The mean search time in minutes of our experiments using (1) purely the proposed encoding strategy, and (2) the overall proposed model for the training and search phase over 10 runs and their corresponding improvements against those of psoCNN. The (-) symbol indicates that the proposed strategies are better in computational costs. All experiments have been conducted using one NVIDIA GeForce RTX 2080Ti consumer GPU.

Model	MNIST	MNIST-RD	MNIST-RB	MNIST-BI	MNIST-RD+BI	Rectangles	Rectangles-I	Convex
psoCNN (mean) [20]	276	47	49	56	43	14	46	33
ours (mean) encoding + copy	268	43	44	29	29	7	42	30
Speed improvement encoding + copy	-3%	-9%	-10%	-48%	-33%	-49%	-8%	-6%
ours (mean)	192	22	23	21	26	5	33	29
Speed improvement	-30%	-53%	-53%	-63%	-40%	-61%	-29%	-11%

For the Convex data set, our generated CNN models achieve a mean error rate of 1.63%, which outperforms those produced by psoCNN by 2.27%.

We subsequently conduct a convergence curve comparison between the proposed model and psoCNN. Fig. 6 plots the mean entropy loss scores of the g_{best} solutions for both methods in the training stage over 10 runs with respect to all data sets. Our proposed model illustrates a faster reduction in loss and continuous improvements in subsequent iterations as compared with those of psoCNN. As an example, for the MNIST-RD+BI data set, the mean loss of psoCNN flattens after iteration 4. Comparatively, our proposed model continues to achieve improvements until iteration 8. In short, our model shows faster convergence rates in comparison with those of psoCNN pertaining to the architecture search for all the data sets.

2) EFFECTIVENESS OF THE PROPOSED ENCODING AND SEARCH STRATEGIES

To further indicate the effectiveness of the proposed encoding and search strategies, we conduct experiments using purely the proposed encoding strategy as well as the overall proposed model. Specifically, Table 6 illustrates the mean results of our model (1) with only the proposed encoding strategy, and (2) with both the proposed encoding and search mechanisms, over a set of 10 runs. The results from the proposed encoding strategy only are shown in rows 3 and 4 in Table 6. When evaluating the encoding strategy in isolation, we adopt the same copying strategy of psoCNN [20] as the search operations. Specifically, we copy the layers randomly from either the global or personal best position using the same decision boundary setting of $\alpha = 0.5$, as that used in psoCNN. When using the proposed encoding strategy only, the results indicate a reduction of the mean error rates for 6 out of 8 data sets, as compared with those of psoCNN, as presented in Table 6. Furthermore, the results indicate a further improvement across all 8 data sets when both the proposed encoding and search strategies are combined, as illustrated in rows 5 and 6 in Table 6. This clearly ascertains

the capability of the proposed search mechanisms in enhancing the performance over and above the improvements from adopting only the proposed encoding strategy.

3) COMPUTATIONAL COST COMPARISON

We present a computational cost comparison for the architecture search during the training stage between the proposed model and psoCNN over 10 runs in Table 7. To indicate the effectiveness of the proposed strategies in reducing the computational cost, we present the mean search time (in minutes) using (1) solely the proposed encoding strategy, and (2) the overall proposed model. Our model and psoCNN adopt the same experimental settings, i.e., with the mean training entropy loss as the fitness score and a training epoch of one. In addition, we use the same swarm size of 20, and the same maximum iteration cycle of 10 so that the search duration of all methods can be compared directly and fairly. All the experiments are conducted using one NVIDIA GeForce RTX 2080Ti consumer GPU.

When evaluating the encoding strategy only, we employ the same copying search operation as that of psoCNN. We report the mean computational costs of our model using purely the encoding strategy in row 2 and its cost reduction against those of psoCNN in row 3. As indicated in Table 7, our model with purely the encoding strategy illustrates enhanced computational efficiency in comparison with those of psoCNN for all data sets. This is owing to the initialization of the swarm with valid and comparatively reasonable network architectures using the proposed encoding strategy that are able to accelerate the search process. In addition, we report the mean computational costs of our model using both the proposed encoding and search strategies in row 4 and its cost reduction against those of psoCNN in row 5. The results indicate that the overall proposed model shows a greater computational cost reduction across all data sets, owing to the efficiency of the proposed search mechanisms for architecture evolution during the search process.

Because of the adoption of different fitness evaluation strategies, we do not present a direct computational cost

comparison with other baseline methods, i.e. MetaQNN, EvoCNN and IPPSO. Specifically, these models train and test each optimized candidate network using the training and validation sets, respectively, which increases the search times due to the use of additional steps (which are computationally heavy). According to the original studies, MetaQNN [21] indicates a search time of 100 GPU days for the MNIST data set, EvoCNN [14] requires average 2-3 days using two GTX 1080 GPU cards for each data set, while IPPSO [13] consumes average 2.5 hours for the MNIST, MNIST-RD+BI and Convex data sets, respectively. Our proposed model and psoCNN illustrate better computational efficiency in comparison with those of the aforementioned models in most of the test cases.

E. DISCUSSION

1) THEORETICAL JUSTIFICATION

In this research, we propose a group-based encoding strategy as well as new velocity and position updating mechanisms based on the key network configuration differences and weighted velocity strengths. The advantages of our proposed model in comparison with the most closely related one, i.e. psoCNN [20], are as follows.

The search process of the psoCNN model [20] generates a particular type of layer in any number (e.g. multiple convolutional or pooling layers). It also generates any combination of convolutional, pooling and fully connected layers for devising CNN architectures. Therefore, psoCNN requires additional governing rules to ensure the validity of the generated networks, which interrupt the exploration capability of the particles in the search space. In order to tackle such limitation, the proposed group-based encoding strategy ensures all the formulated CNN models are valid without the requirement of additional governing rules. Specifically, each group contains at least one convolutional layer, which is followed by an optional pooling layer. In other words, the pooling layer is always positioned as the final layer in the group. Moreover, the number of groups can be adjusted in accordance with the input image size. By restricting the number of groups, we can adapt the frequency of pooling operations toward the input image size. This ensures the position of pooling layers and the frequency of pooling operations are valid with respect to the input image size without the need for any additional governing rules. Instead of generating several fully connected layers as in psoCNN, our model only attaches one fully connected layer as the last layer of the final group in a network, which is similar to the approach in [3].

Moreover, in psoCNN [20], instead of creating new layer configurations, the existing layer configurations from the personal and global best solutions are copied directly in the velocity updating operation (as illustrated in the right-hand side in Fig. 7). In contrast, our model employs the key layer configuration differences between the current particle and the personal and global best solutions for new velocity generation (as illustrated in the left-hand in Fig. 7). Therefore,

the proposed model is more capable of devising new network architectures using intermediate positions of the particles' trajectories. In addition, we employ a new position updating mechanism that takes a partial strength of velocity updates to generate new positions. This provides the capabilities for the particles to explore the search space with various momentums and scales, in order to increase search diversity. Such a search mechanism allows a more granular exploration pertaining to the in-between positions, increasing the chances of devising more diversified networks. Therefore, the proposed model is less reliant on initialization, along with enhanced capabilities in evolving architecture generation.

2) EXPERIMENTAL OBSERVATIONS IN COMPARISON WITH RELATED STUDIES

In this section, we discuss how our encoding strategy and search mechanisms lead to improvements in both speed and accuracy in comparison with those of psoCNN [20] through experimental observations. We firstly explain the rational of our encoding strategy in enhancing both speed and accuracy. Then, we clarify the rational of the combined search and encoding strategies in further improving accuracy and speed of the final model.

To identify the contribution of our encoding strategy in terms of computational efficiency, we isolate the encoding strategy from the proposed search operations by combining it with the copying search strategy used in psoCNN [20]. We record the time taken to perform the architecture search in the training phase. The detailed computational costs are presented in Table 7. As discussed earlier, the results indicate that across all 8 data sets, our encoding strategy improves the computational costs as compared with those of psoCNN [20]. Specifically, we observe a reduction in computational costs in the architecture search stage by up to 49% on the Rectangles data set. As indicated in Table 7, our proposed encoding strategy is the primary reason for the enhancement of computational efficiency. As an example, the encoding method of psoCNN allows the construction of models containing a large number of fully connected layers. In some cases, the models created by psoCNN contain 8 consecutive fully connected layers, resulting in a long training time. Subsequent iterations attempt to eliminate the additional fully connected layers and finally recommend a model with one fully connected layer towards the end of the search process. In contrast, as discussed above, the proposed encoding strategy avoids such a problem by fixing the number of fully connected layers to one, as well as fixing its position to the final layer of the model, therefore contributing toward the reduction in computational costs.

To identify the contribution of our encoding strategy in terms of accuracy, Table 6 shows that by applying our encoding strategy on its own and adopting the same velocity updating mechanism as that of psoCNN (i.e. the layer copying strategy from p_{best} or g_{best}), it results in an improvement in the mean accuracy rates (of 10 runs) pertaining to the MNIST, MNIST-RD, MNIST-RB, MNIST-RD+BI, Rectangles-I and

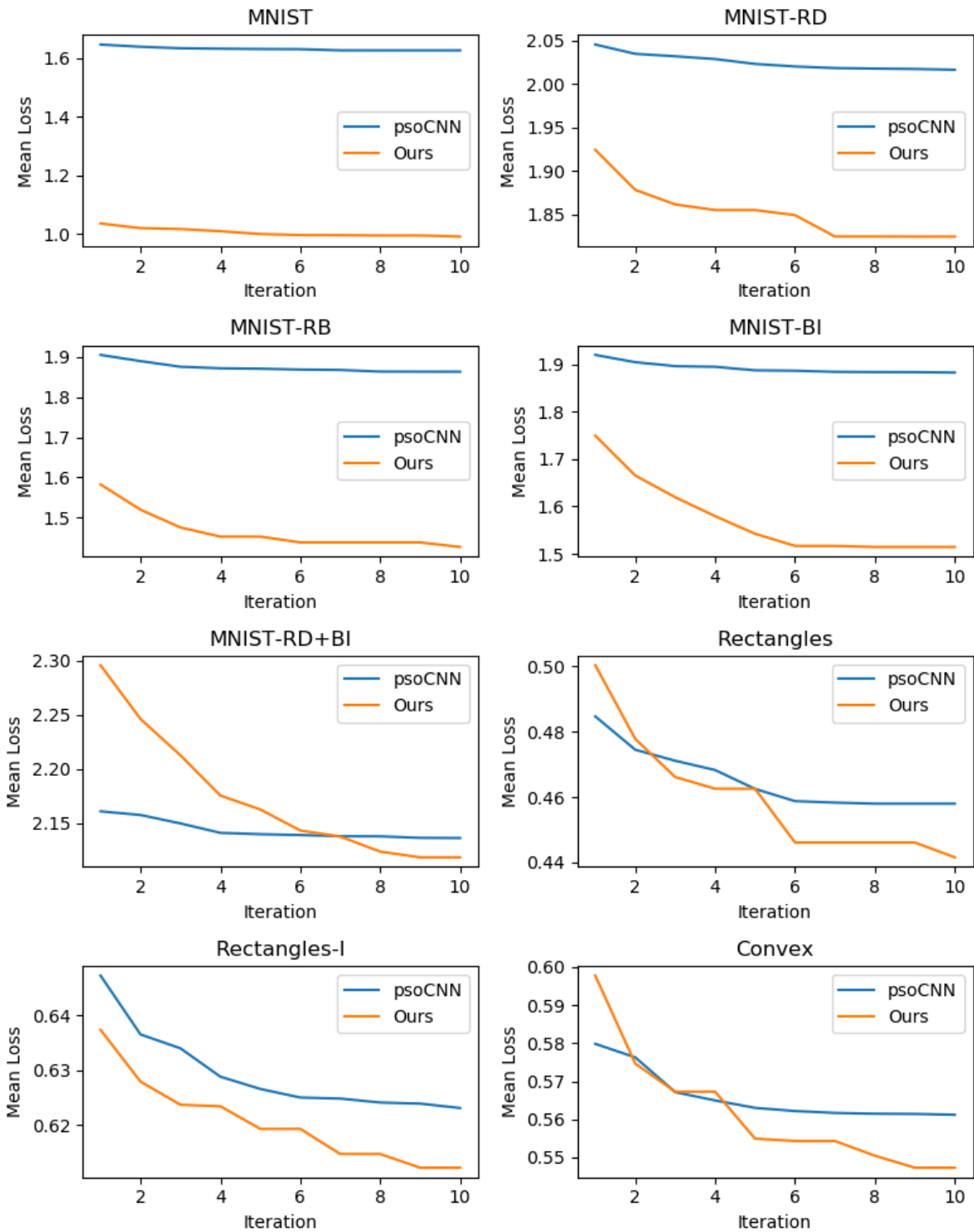


FIGURE 6. Convergence curves of the proposed algorithm and psoCNN. The mean losses of 10 runs are plotted over 10 iterations for all data sets.

Convex data sets. These empirical results reveal that psoCNN has constructed pooling layers one after another, leading to poor performing networks and wasteful function evaluations. The experimental results also indicate that the use of

hardcoded rules in psoCNN for ensuring model validity interrupts the natural particle movement. This is ascertained by the aforementioned accuracy improvements when applying our encoding strategy alone. The proposed encoding strategy

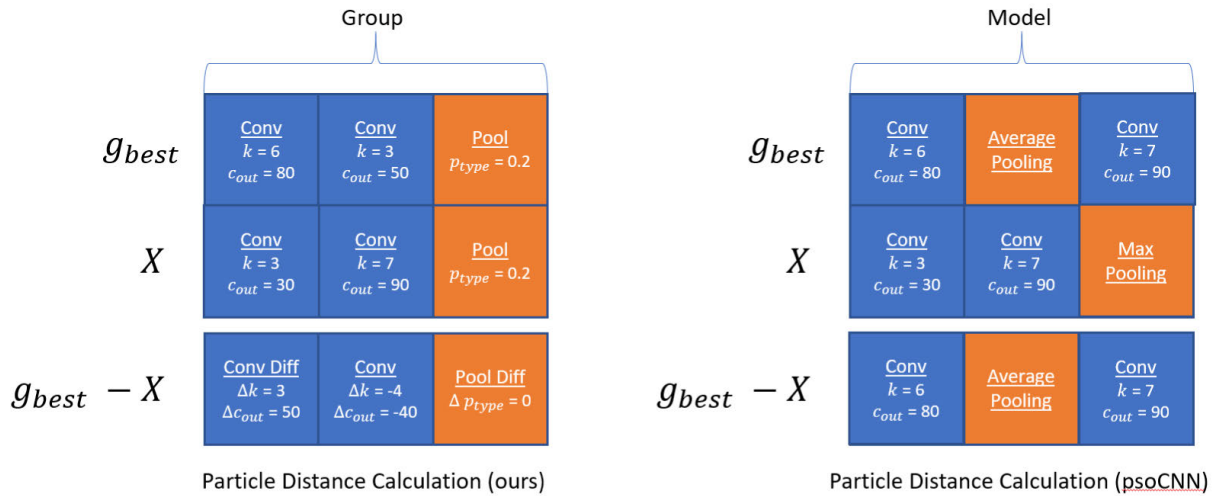


FIGURE 7. Comparison of particle distance calculations between our proposed approach and the psoCNN method.

ensures that undesirable candidate models such as those containing stacks of pooling layers are never constructed, since the pooling position is determined by a group-based structure in our scheme. Each group can only contain at most one pooling layer, which is always positioned as the final layer in the group after at least one convolutional layer. Pooling layers, therefore, are never stacked. Furthermore, rules that can interrupt the natural particle movement such as eliminating excessive pooling layers are not required in our scheme, as the maximum number of pooling layers is determined by the maximum number of groups. Therefore, our model eliminates the possibility of having excessive down-sampling operations that can cause the network to become invalid, mitigating the need for additional governing rules. Indeed, our encoding strategy ensures that each devised model is reasonably constructed that helps exploit the search space in a comprehensive manner.

As indicated in Table 6 and Table 7, combining our proposed search mechanisms (both velocity and position updating operations) and our encoding strategy together results in further improvement of accuracy and computational efficiency as compared with those yielded by using the proposed encoding strategy alone. We observe that by combining both strategies together, the mean accuracy rate increases by up to 7.58% pertaining to the MNIST-RD+BI data set and the mean computational cost reduces by up to 63% pertaining to the MNIST-BI data set. This suggests that the copying search strategy adopted by psoCNN is less flexible, because such a strategy heavily relies on a good initialization. It does not create new layer structures for new velocity generation, but simply copying the existing layer configurations from either the p_{best} or g_{best} solutions, as illustrated in the right-hand part of Fig. 7. Comparatively, our proposed velocity updating mechanism combined with a granular weighted movement operation is able to formulate completely new particle solutions by identifying the key layer configuration differences between particles, as shown

in the left-hand part of Fig. 7. In other words, our search strategies are able to yield new layer configurations which cannot be achieved by the copying operation of psoCNN. In addition, owing to the capabilities of searching the intermediary positions, our velocity and granular weighted position updating mechanisms have better diversification and intensification, leading to better devised networks, as ascertained by the empirical results. In short, both proposed encoding and search mechanisms illustrate superior efficiency in terms of speed and performance in comparison with those of psoCNN.

We have also observed that the proposed model and psoCNN are significantly faster than the reinforcement learning and other evolutionary optimization methods for deep architecture generation, whilst not sacrificing accuracy. The search cost of our proposed model consumes 5 to 192 minutes, while that of psoCNN consumes 14 to 276 minutes, for all the test data sets. On the other hand, MetaQNN [21], which is based on reinforcement learning, requires 100 GPU days for processing the MNIST data set, while EvoCNN [14], which uses a GA-based method, needs on average 2-3 days of processing time with two GTX1080 GPU cards for each data set.

3) DISCUSSION OF IDENTIFIED MODELS

The best CNN models identified by our proposed method for all the test data sets are shown in Table 8. In some instances, such as on the Rectangles data set, numerous models that achieve the same best performance (i.e. 0% error rate) are identified. While the computational cost is not an objective in our fitness function, significantly smaller models are generated in this study as compared with those reported by psoCNN. As an example, the best model contains two convolutional layers with 70 and 60 channels, respectively, for the Rectangles data set, whereas the network produced by psoCNN contains three convolutional layers with 139, 113, and 226 channels respectively. Average pooling

TABLE 8. The best CNN models discovered by the proposed approach for the eight test data sets.

Data set	Group	Layer Type	Layer Setting
MNIST	1	Convolution	$k = 5, c_{out} = 165$
	1	Convolution	$k = 6, c_{out} = 220$
	1	Convolution	$k = 5, c_{out} = 215$
	1	Pooling	Average Pooling
	2	Convolution	$k = 4, c_{out} = 120$
	2	Pooling	Average Pooling
MNIST-RD	1	Convolution	$k = 6, c_{out} = 66$
	1	Convolution	$k = 5, c_{out} = 140$
	1	Convolution	$k = 6, c_{out} = 192$
	1	Pooling	Average Pooling
	2	Convolution	$k = 7, c_{out} = 217$
	2	Pooling	Average Pooling
MNIST-RB	1	Convolution	$k = 4, c_{out} = 230$
	1	Convolution	$k = 6, c_{out} = 109$
	1	Convolution	$k = 4, c_{out} = 219$
	1	Convolution	$k = 4, c_{out} = 23$
	1	Convolution	$k = 5, c_{out} = 112$
	1	Convolution	$k = 7, c_{out} = 174$
	1	Convolution	$k = 5, c_{out} = 36$
	1	Pooling	Average Pooling
MNIST-BI	1	Convolution	$k = 4, c_{out} = 153$
	1	Convolution	$k = 4, c_{out} = 153$
	1	Convolution	$k = 7, c_{out} = 159$
	1	Convolution	$k = 7, c_{out} = 195$
	1	Convolution	$k = 7, c_{out} = 18$
	1	Pooling	Average Pooling
MNIST-RD+BI	1	Convolution	$k = 4, c_{out} = 170$
	1	Convolution	$k = 4, c_{out} = 247$
	1	Convolution	$k = 3, c_{out} = 200$
	2	Convolution	$k = 5, c_{out} = 136$
	2	Convolution	$k = 6, c_{out} = 137$
	2	Convolution	$k = 6, c_{out} = 38$
RECTANGLES	1	Pooling	Average Pooling
	1	Convolution	$k = 7, c_{out} = 70$
	1	Convolution	$k = 7, c_{out} = 60$
	1	Pooling	Average Pooling
RECTANGLES-I	1	Convolution	$k = 4, c_{out} = 204$
	1	Convolution	$k = 5, c_{out} = 202$
	1	Convolution	$k = 5, c_{out} = 210$
	1	Convolution	$k = 5, c_{out} = 44$
	1	Convolution	$k = 4, c_{out} = 49$
	1	Pooling	Average Pooling
	2	Convolution	$k = 4, c_{out} = 141$
	2	Convolution	$k = 4, c_{out} = 176$
	2	Convolution	$k = 5, c_{out} = 41$
	2	Pooling	Average Pooling
CONVEX	1	Convolution	$k = 4, c_{out} = 193$
	1	Convolution	$k = 5, c_{out} = 225$
	1	Convolution	$k = 5, c_{out} = 108$
	1	Convolution	$k = 5, c_{out} = 222$
	2	Convolution	$k = 7, c_{out} = 171$
	2	Convolution	$k = 5, c_{out} = 184$
	2	Convolution	$k = 7, c_{out} = 145$
	2	Convolution	$k = 6, c_{out} = 31$
	2	Pooling	Average Pooling

is the most common method of pooling identified in our experiments, which is the same observation found in related studies.

The empirical results indicate that performance can be further improved by using more iterations and larger population sizes. For further research, we intend to extend the proposed method by encoding the skip connections, allowing deeper models to be generated. Such networks can be used for undertaking complex data sets that require deep model architectures.

V. CONCLUSION, LIMITATIONS AND FUTURE WORK

We have proposed a novel PSO-based algorithm for automatic CNN architecture generation. The proposed algorithm addresses two weaknesses of the current state-of-the-art methods. Firstly, a group-based encoding strategy has been introduced to remove the need for additional hard-coded rules. Our strategy adopts a natural particle movement and simplifies implementation. Secondly, a novel particle distance calculation scheme has been proposed. It performs distance calculation at a parameter level to address the drawback of the existing methods which copy the layers directly at random from either p_{best} or g_{best} . Moreover, a weighted position updating mechanism has been developed, with the use of a weighting factor to provide granular movement control in the search space. Combining the proposed distance computation strategy with the new position updating mechanism, our method is equipped with superior search diversity and is less dependent on good initialization, as compared with related methods such as psoCNN.

With an identical setting, i.e. no data augmentation, the same swarm size, iteration numbers, and search ranges, we have demonstrated that our method achieves superior performance in eight benchmark data sets, and achieves up to 7.58% improvement in accuracy and up to 63% reduction in computational cost in comparison with those of existing methods. The convergence curves of the proposed model across all data sets indicate a steady reduction in loss, indicating convergence of the particles without being trapped in local optima.

One limitation of this study is the model settings shown in Table 2. The current settings are based on the constraints reported in the related paper [20], in order to ensure that a fair comparison can be made. It is envisaged that optimal settings can be formulated, e.g. with the use of adaptive strategies for α and β . Indeed, it is beneficial to devise an adaptive strategy to alter α slightly after each iteration so that the position differences to the global best solution are given more emphasis than those from earlier iterations.

In future work, we will explore adding skip connections, which are a key requirement in deep architectures, for the generation of new models. Moreover, we will explore hybrid deep networks such as the combination of CNN with Long Short-Term Memory (LSTM) networks. Such hybrid models will be useful for undertaking various image understanding and time series prediction tasks. From the algorithm perspective, the impact of larger swarm sizes and iterations will be explored, along with adaptive selection from the distance between X and g_{best} as the iterations progress for velocity updating, in order to further aid exploration. On the other hand, hybrid search strategies based on the integration of the proposed model with other swarm intelligence algorithms, e.g. Firefly Algorithm [54], Cuckoo Search [55], Dragonfly Algorithm [56] and Grey Wolf Optimizer [57], will be investigated to further enhance performance. Furthermore, we will apply the proposed algorithm for CNN model generation

in other computer vision tasks such as object detection and visual question generation.

REFERENCES

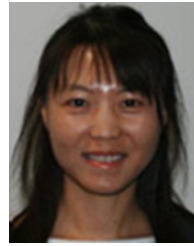
- [1] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.
- [2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.
- [3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.
- [4] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu, "A survey on deep transfer learning," in *Proc. Int. Conf. Artif. Neural Netw.* Cham, Switzerland: Springer, 2018, pp. 270–279.
- [5] S. Jialin Pan and Q. Yang, "A survey on transfer learning," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 10, pp. 1345–1359, Oct. 2010.
- [6] D. Scherer, A. Müller, and S. Behnke, "Evaluation of pooling operations in convolutional architectures for object recognition," in *Proc. Int. Conf. Artif. Neural Netw.* Cham, Switzerland: Springer, 2010, pp. 92–101.
- [7] R. Caruana, S. Lawrence, and C. L. Giles, "Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping," in *Proc. Adv. Neural Inf. Process. Syst.*, 2001, pp. 402–408.
- [8] S. Zagoruyko and N. Komodakis, "Wide residual networks," in *Proc. Brit. Mach. Vis. Conf.*, BMVA Press, Sep. 2016, pp. 87.1–87.12.
- [9] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. Int. Conf. Learn. Represent.*, 2015, pp. 1–14.
- [10] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 1–9.
- [11] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 2261–2269.
- [12] Y. Sun, G. G. Yen, and Z. Yi, "Evolving unsupervised deep neural networks for learning meaningful representations," *IEEE Trans. Evol. Comput.*, vol. 23, no. 1, pp. 89–103, Feb. 2019.
- [13] B. Wang, Y. Sun, B. Xue, and M. Zhang, "Evolving deep convolutional neural networks by variable-length particle swarm optimization for image classification," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jul. 2018, pp. 1–8.
- [14] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Evolving deep convolutional neural networks for image classification," *IEEE Trans. Evol. Comput.*, vol. 24, no. 2, pp. 394–407, Apr. 2020.
- [15] X.-S. Yang, "Nature-inspired optimization algorithms: Challenges and open problems," *J. Comput. Sci.*, vol. 46, Oct. 2020, Art. no. 101104.
- [16] B. Wang, Y. Sun, B. Xue, and M. Zhang, "A hybrid differential evolution approach to designing deep convolutional neural networks for image classification," in *Advances in Artificial Intelligence*, T. Mitrovic, B. Xue, and X. Li, Eds. Cham, Switzerland: Springer, 2018, pp. 237–250.
- [17] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1992.
- [18] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press, 1998.
- [19] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. IEEE ICNN*, vol. 4, Nov./Dec. 1995, pp. 1942–1948.
- [20] F. E. Fernandes Junior and G. G. Yen, "Particle swarm optimization of deep neural networks architectures for image classification," *Swarm Evol. Comput.*, vol. 49, pp. 62–74, Sep. 2019.
- [21] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," in *Proc. 5th Int. Conf. Learn. Represent. (ICLR)*, Toulon, France, Apr. 2017, pp. 1–18.
- [22] B. Staar, M. Lütjen, and M. Freitag, "Anomaly detection with convolutional neural networks for industrial surface inspection," *Procedia CIRP*, vol. 79, pp. 484–489, Jan. 2019.
- [23] T. Y. Tan, L. Zhang, C. P. Lim, B. Fielding, Y. Yu, and E. Anderson, "Evolving ensemble models for image segmentation using enhanced particle swarm optimization," *IEEE Access*, vol. 7, pp. 34004–34019, 2019.
- [24] X. Zhou, L.-C. Qian, P.-J. You, Z.-G. Ding, and Y.-Q. Han, "Fall detection using convolutional neural network with multi-sensor fusion," in *Proc. IEEE Int. Conf. Multimedia Expo Workshops (ICMEW)*, Jul. 2018, pp. 1–5.
- [25] P. Kinghorn, L. Zhang, and L. Shao, "A region-based image caption generator with refined descriptions," *Neurocomputing*, vol. 272, pp. 416–424, Jan. 2018.
- [26] P. Kinghorn, L. Zhang, and L. Shao, "A hierarchical and regional deep learning architecture for image description generation," *Pattern Recognit. Lett.*, vol. 119, pp. 77–85, Mar. 2019.
- [27] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 779–788.
- [28] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," 2018, *arXiv:1804.02767*. [Online]. Available: <http://arxiv.org/abs/1804.02767>
- [29] R. Girshick, "Fast R-CNN," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 1440–1448.
- [30] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 1137–1149, Jun. 2017.
- [31] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 2, pp. 386–397, Feb. 2020.
- [32] K. Mistry, L. Zhang, S. C. Neoh, C. P. Lim, and B. Fielding, "A micro-GA embedded PSO feature selection approach to intelligent facial emotion recognition," *IEEE Trans. Cybern.*, vol. 47, no. 6, pp. 1496–1509, Jun. 2017.
- [33] T. Y. Tan, L. Zhang, and C. P. Lim, "Intelligent skin cancer diagnosis using improved particle swarm optimization and deep learning models," *Appl. Soft Comput.*, vol. 84, Nov. 2019, Art. no. 105725.
- [34] T. Y. Tan, L. Zhang, and C. P. Lim, "Adaptive melanoma diagnosis using evolving clustering, ensemble and deep neural networks," *Knowl.-Based Syst.*, vol. 187, Jan. 2020, Art. no. 104807.
- [35] W. Srisukhham, L. Zhang, S. C. Neoh, S. Todryk, and C. P. Lim, "Intelligent leukaemia diagnosis with bare-bones PSO based feature optimization," *Appl. Soft Comput.*, vol. 56, pp. 405–419, Jul. 2017.
- [36] B. Fielding and L. Zhang, "Evolving image classification architectures with enhanced particle swarm optimisation," *IEEE Access*, vol. 6, pp. 68560–68575, 2018.
- [37] B. Fielding, T. Lawrence, and L. Zhang, "Evolving and ensembling deep CNN architectures for image classification," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2019, pp. 1–8.
- [38] Y. Wang, H. Zhang, and G. Zhang, "CPSO-CNN: An efficient PSO-based algorithm for fine-tuning hyper-parameters of convolutional neural networks," *Swarm Evol. Comput.*, vol. 49, pp. 114–123, Sep. 2019.
- [39] Y. Sun, B. Xue, M. Zhang, G. G. Yen, and J. Lv, "Automatically designing CNN architectures using the genetic algorithm for image classification," *IEEE Trans. Cybern.*, vol. 50, no. 9, pp. 3840–3854, Sep. 2020.
- [40] B. L. Miller and D. E. Goldberg, "Genetic algorithms, tournament selection, and the effects of noise," *Complex Syst.*, vol. 9, no. 3, pp. 193–212, 1995.
- [41] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 8697–8710.
- [42] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," in *Proc. Int. Conf. Learn. Represent.*, 2019, pp. 1729–1738.
- [43] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Completely automated CNN architecture design based on blocks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 4, pp. 1242–1254, Apr. 2020.
- [44] Z. Gao, Y. Li, Y. Yang, X. Wang, N. Dong, and H.-D. Chiang, "A GPSO-optimized convolutional neural networks for EEG-based emotion recognition," *Neurocomputing*, vol. 380, pp. 225–235, Mar. 2020.
- [45] A. Kwasigroch, M. Grochowski, and A. Mikolajczyk, "Neural architecture search for skin lesion classification," *IEEE Access*, vol. 8, pp. 9061–9071, 2020.
- [46] B. Wang, B. Xue, and M. Zhang, "Particle swarm optimisation for evolving deep neural networks for image classification by evolving and stacking transferable blocks," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jul. 2020, pp. 1–8.
- [47] S. Han, "Learning both weights and connections for efficient neural network," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 1135–1143.

- [48] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and Huffman coding," in *Proc. 4th Int. Conf. Learn. Represent. (ICLR)*, San Juan, Puerto Rico, May 2016, pp. 1–14.
- [49] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. Learn. Represent.*, Dec. 2014, pp. 1–15.
- [50] B. Van Merriënboer, O. Breuleux, A. Bergeron, and P. Lamblin, "Automatic differentiation in ML: Where we are and where we should be going," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 8757–8767.
- [51] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. Int. Conf. Mach. Learn.*, vol. 37, Lille, France, Jul. 2015, pp. 448–456.
- [52] V. Nair and G. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *Proc. 27th Int. Conf. Mach. Learn. (ICML)*, Haifa, Israel, 2010, pp. 807–814.
- [53] 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 *Proc. 24th Int. Conf. Mach. Learn. (ICML)*, New York, NY, USA, 2007, pp. 473–480.
- [54] X.-S. Yang, "Firefly algorithms," in *Nature Inspired Optimization Algorithms*, X.-S. Yang, Ed. Oxford, U.K.: Elsevier, 2014, ch. 8, pp. 111–127. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/B9780124167438000087>
- [55] X. Yang and S. Deb, "Cuckoo search via Lévy flights," in *Proc. World Congr. Nature Biologically Inspired Comput. (NaBIC)*, 2009, pp. 210–214.
- [56] S. Mirjalili, "Dragonfly algorithm: A new meta-heuristic optimization technique for solving single-objective, discrete, and multi-objective problems," *Neural Comput. Appl.*, vol. 27, no. 4, pp. 1053–1073, May 2016.
- [57] S. Mirjalili, S. M. Mirjalili, and A. Lewis, "Grey wolf optimizer," *Adv. Eng. Softw.*, vol. 69, pp. 46–61, Mar. 2014.



TOM LAWRENCE received the B.Sc. degree in computer science from Northumbria University, Newcastle upon Tyne, U.K., in 2018, where he is currently pursuing the Ph.D. degree.

His current research interests include deep learning, computer vision, and evolutionary computation.



LI ZHANG (Senior Member, IEEE) received the Ph.D. degree from the University of Birmingham.

She was an Associate Professor (Reader) of computer science with Northumbria University, U.K. She is currently a Professor with the National Subsea Centre, Robert Gordon University, U.K. She is also working as an Honorary Research Fellow with the University of Birmingham. She holds expertise in machine learning, deep learning, and evolutionary computation. She has served as an

Associate Editor for *Decision Support Systems*.



CHEE PENG LIM received the Ph.D. degree from The University of Sheffield, U.K., in 1997. He is currently a Professor of complex systems with Deakin University, Australia. He has published over 450 technical papers in books, international journals, and conference proceedings. His research interests include computational intelligence, decision support, pattern recognition, medical prognosis and diagnosis, and fault detection and diagnosis.



EMMA-JANE PHILLIPS is currently a Senior Lecturer in computer science with Northumbria University, U.K. Her research interests include image processing, machine learning, and bioinformatics.

...