A Particle Swarm Optimization-based Flexible Convolutional Auto-Encoder for Image Classification

Yanan Sun, Bing Xue and Mengjie Zhang
School of Engineering and Computer Science
Victoria University of Wellington
Wellington 6140, New Zealand
{yanan.sun,bing.xue,mengjie.zhang}@ecs.vuw.ac.nz

Abstract

Convolutional auto-encoders have shown their remarkable performance in stacking to deep convolutional neural networks for classifying image data during past several years. However, they are unable to construct the state-of-the-art convolutional neural networks due to their intrinsic architectures. In this regard, we propose a flexible convolutional auto-encoder by eliminating the constraints on the numbers of convolutional layers and pooling layers from the traditional convolutional auto-encoder. We also design an architecture discovery method by using particle swarm optimization, which is capable of automatically searching for the optimal architectures of the proposed flexible convolutional auto-encoder with much less computational resource and without any manual intervention. We use the designed architecture optimization algorithm to test the proposed flexible convolutional auto-encoder through utilizing one graphic processing unit card on four extensively used image classification datasets. Experimental results show that our work in this paper significantly outperform the peer competitors including the state-of-the-art algorithm.

1 Introduction

Auto-Encoders(AEs) [1–4] are building blocks of Stacked AE (SAE) [5, 6] that is one of the trimainstream deep learning algorithms [7] (others are Deep Belief Networks (DBN) [8] and Convolutional Neural Networks (CNNs) [9, 10]). An AE is a three-layer neural network including one input layer, one hidden layer, and one output layer, where the number of units in the input layer is identical to that in the output layer. Typically, the transformation from the input layer to the hidden layer is called the encoder, and that from the hidden layer to the output layer refers to the decoder. The encoder derives the features/representations from the input data, and the decoder reconstructs the input data from the features/representations. By minimizing the divergences between the input data and the reconstruction, one AE is trained, and multiple trained AEs are stacked to a SAE for learning hierarchical representations. These hierarchical representations have gained remarkable performance than ever before in the field of image classification [10–12].

When image data are fed to the SAE, they must be transformed into the vector-form in advance, which will change their inherent structures and reduce the consecutive performance in turn. For instance, one image is with the form $I \in \mathbb{R}^{n \times n}$, where the pixel $I_{j,k}$ $(1 < j < n, 0 < k < n)$ has the close distance to the pixel $I_{j-1,k}$. When I is vectorized to $V \in \mathbb{R}^{n^2}$, the relationship between $I_{j,k}$ and $I_{j-1,k}$ will be changed and may not be neighbours any more in V. Literatures have extensively pointed out that adjacent information is a key factor in addressing images related problems [13, 14, 9, 10, 15]. To address this issue, Masci *et al.* [16] proposed the Convolutional AEs (CAEs), where the image data is directly fed with 3-D form. In CAEs, the encoder is composed of one convolutional layer followed by one pooling layer, and the decoder comprises only

one deconvolutional layer. Multiple trained CAEs are stacked to a CNN for learning the hierarchical representations that enhance the final classification performance. Inspired by the advantages of CAEs in addressing data without the original 3-D form, variants of CAEs have been proposed subsequently. For example, Norouzi *et al.* [17] proposed the Convolutional Restricted Boltzmann Machines (RBM) [5, 18] (CRBM). Lee *et al.* [19] proposed the convolutional DBN by stacking a group of trained CRBMs. In addition, Zeiler *et al.* [20, 21] proposed the inverse convolutional ones based on the sparse coding schema [22], which inspired Kavukcuoglu *et al.* [23] to design the convolutional stacked sparse coding for solving object recognition tasks. Recently, Du *et al.* [24] proposed the Convolutional Denoising AE (CDAE) by using Denosing AE (DAE) [25] to learn the convolutional filters.

Although experimental results from the CAE and its variants have shown benefits in diverse applications, one major limitation exist in that the architectures of their stacked CNNs are inconsistent with those of state-of-the-art CNNs, such as ResNet [26] and VGGNet [27]. To be specific, because one CAE has one convolutional layer and one pooling layer in the encoder part, the stacked CNN has the same numbers of convolutional layers and pooling layers. However, state-of-the-art CNNs are with non-identical numbers of convolutional layers and pooling layers. Because the architecture of CNN is one key ingredient contributing to the final performance, the clamp on the numbers of convolutional layers and pooling layers of CAEs should be removed. However, choosing the desirable numbers of convolutional layers and pooling layers is intractable due to the non-differentiable and non-convex characteristics of practice, which is related to the architecture optimization for neural networks.

Algorithms for searching for the optimal architectures of neural networks can be classified into three different categories. The first includes Random Search (RS) [28], Bayesian-based Gaussion Process (BGP) [29, 30], Tree-structured Parzen Estimators (TPE) [31], and Sequential Model-Based Global Optimization (SMBO) [32], which are utilized largely for the neural networks with unified building blocks, such as SAEs and DBNs. The second one covers the algorithms that are designed specifically for CNNs where multiple different building blocks exist. The Meta-modeling algorithm (MetaQNN) [33] and Large Evolution for Image Classification (LEIC) algorithm [34] belong to this category. The third one refers to the NeuroEvolution of Augmenting Topologies (NEAT) [35] algorithm and its diverse variants, such as [36–38]. Experimental results from these algorithms have shown their superiority in exploring the optimal neural network architectures. However, their limitations cause their inapplicability to CAEs or CNNs for general purpose. 1) There are multiple assumptions prior to using the methods from the first category, e.g., the RS method requires the intrinsic parameter search space is a subspace of the entire space, and the TPE method postulates the parameters are independent, which are not constantly true for CAEs. In addition, the BGP method incorporates extra parameters that are difficult to tune. 2) The methods from the second category require intensive computational resources, such as MetaQNN employed 10 Graphics Processing Unit (GPU) cards for 8-10 days, and LSIC employed 250 high-performance computers for 20 days on their investigated problems, respectively. However, sufficient computational resource is not necessarily available to all interested researchers. 3) When using the NEAT-based methods, hybrid connections (i.e., the weight connections between the layers which are not adjacent) would be reached, and the configurations of the input layer and the output layer must be specified in advance, which are not allowed or applicable in CAEs. In addition, there is one method that does not lie on these categories, i.e., the Grid Search method (GS), which tests every combination of the related parameters. In practice, GS cannot give the desirable result within an acceptable time due to its exhaustive search nature. GS method cannot handle well parameters with continuous values. Due to the deficiencies of the existing methods, the de-facto standard in reality for searching for the optimal architectures has been the manual tuning from experts with domain knowledge.

Particle Swarm Optimization (PSO) is a population-based stochastic evolutionary computation algorithm, motivated by the social behavior of fish schooling or bird flocking [39, 40], commonly used for solving optimization problems without rich domain knowledge. Compared with other heuristic algorithms, PSO is enriched with the features of simple concept, easy implementation, and computational efficiency. In PSO, the individuals are called the particles, each particle maintains the best solution (denoted by $pBest$) from the memory of itself, and the population records the best solution (denoted by $pBest$) from the history of all participants. During the process, particles expectedly cooperate and interact with the $pBest$ and $gBest$, enhancing the search ability and pursuing the optimal solutions. Due to the characteristics of no requirements (e.g., convex or differentiable)

imposed on the problems to be optimized, PSO has been widely applied to various real-world applications [41–43], naturally including the architecture design of neural networks, such as [44–49]. In the optimization of neural network architectures, these algorithms employ an implicit method to encode each connection of the neural networks, and take PSO or its variants to search for the optimum. However, they cannot be utilized for CAEs and CNNs, even SAEs and DBNs, which are deep learning algorithms and where tremendous numbers of connection weights exist, causing the unaffordable cost for implementation and effective optimization in these existing PSO-based architecture optimization algorithms [50]. As have discussed that, CAEs without the constraint on the numbers of the convolutional layer and pooling layers would be greatly preferred for stacking the state-of-the-art CNNs. However, the absolute numbers of these layers are unknown before the architecture is confirmed. When PSO is employed for the architecture optimization, particles with variable lengths would be a desirable option. However, the canonical PSO did not provide the way to calculate the velocity for particles with non-identical lengths. In addition, evaluating one particle that represents a deep learning algorithm is time-consuming, and will be more intractable for the population-based updating process. A common way to solve this problem is to employ intensive computational resources, and utilize parallel-computation techniques.

1.1 Goal

The goal of this paper is to design and develop an effective and efficient PSO method to atomically discover the architecture of the flexible convolutional auto-encoder without manual intervention. To achieve this goal, we have specified the objectives as follows:

1. Propose one Flexible CAE (FCAE) where multiple of convolutional layers and pooling layers may exist. The FCAE has no requirement on the particular numbers of the convolutional layers and the pooling layers, and have the potentiality for stacking to different types of CNNs including the state-of-the-art.
2. Design a PSO-based Architecture Optimization (PSOAO) algorithm for the proposed FCAE. In PSOAO, we propose an efficient encoding strategy to encode the FCAE architectures, which involve hundreds of thousands parameters, into each particle, and we also develop one effective velocity calculation mechanism for particles with different lengths.
3. Ingestive the performance of the proposed FCAE when its architecture is optimized by the designed PSOAO on image classification benchmark datasets, compare the classification accuracy to peer competitors, and examine the evolution effectiveness of PSOAO.
4. Investigate the effectiveness of the designed velocity calculation method through quantitative experiments on the comparisons to its opponent.

1.2 Organization

The remainder of this paper is organized as follows. Background of the CAE and PSO are reviewed in Section 2. This is followed by the details of the proposed PSOAO algorithm in Section 3. Then, the experimental design and the result analysis are detailed in Sections 4 and 5, respectively. Finally, the conclusions are drawn in Section 6.

2 Background

This work was built on FCAE and PSOAO. Therefore, their related work, i.e., CAE and PSO, will be detailed in the following subsections.

2.1 Convolutional Auto-Encoder

For convenience of the development, assuming CAEs are utilized for image classification tasks, and each image $x \in \mathbb{R}^{w \times h \times c}$, where w, h, c refer to the image width, height, and number of channels, respectively. Fig. 1 illustrates the architecture of one CAE [16]. The deconvolutional layer is equivalent to the corresponding convolutional operation with inverse parameter settings.

***Convolution*:** Given the input data, convolution operation employs one filter (can be simply viewed as one matrix) to slide from left to right with one defined step size (i.e., the stride width), and

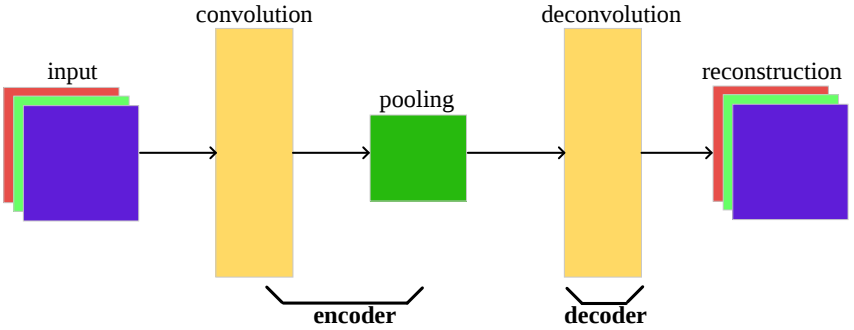


Figure 1: The illustrative architecture of CAE.

then from top to bottom with another defined step size (i.e., the stride height). At each position, convolution operation outputs one element that is the sum of the products of the filter and the input data this filter overlaps. All elements generated by one filter construct one feature map, and multiple feature maps are allowed in convolution operations. By whether to pad zeros to the input data for keeping the identical sizes between the input data and the generated feature map, convolution operation is separated into the SAME type (with padding) and the VALID type (without padding). An example of convolution operation with the SAME type is illustrated in Fig. 2, where the input data and the resulted feature map are with 3×3, the filter is with 2×2, and the stride is with 1×1. As can bee seen that the parameters related the convolutional operation are the *filter size* (width and height), the *stride size* (width and height), the *convolutional type*, and *the number of feature maps*.

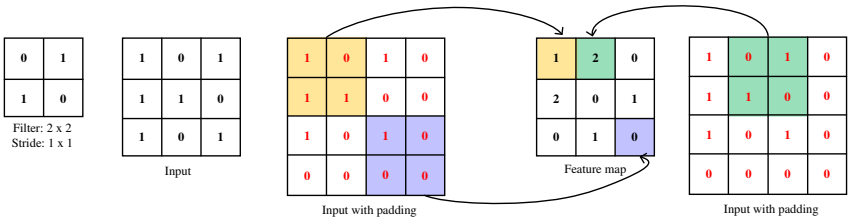


Figure 2: An example of convolution operation with the SAME type. The same colors in the input data (with padding) and the feature map refer to the overlapping area of the filter and the resulted element value of the feature map.

***Pooling*:** Pooling operation resembles convolution operation, in addition to the filter and the way to generate the elements of the corresponding feature map. Specifically, the filter in pooling operation is called the kernel, and no value exists in the kernel. Typically, pooling operation does not change the number of feature maps. Each element of the generated feature map in pooling operation is the statistical result of the input data that the kernel envelops when it slides. There are two types statistical indicators in pooling operation: mean and maximum. Typically, the maximal pooling is preferred in CAEs. An example of maximal pooling operation is illustrated in Fig. 3, where the input data is with 4×4, the kernel, the stride, and the resulted feature map are all with 2×2. In summary, the pooling operation requires the parameters: the *kernel size* (width and height), the *stride size* (width and height), and the *pooling type*.

***Learning of CAE*:** The mathematical form of the CAE is represented by Equation (1), where the $conv(\cdot)$, $pool(\cdot)$, and $de\_conv(\cdot)$ denote the convolution, pooling, and deconvolution operations, respectively, $F(\cdot)$ and $G(\cdot)$ refer to the element-wise nonlinear activation functions, b_1 and b_2 are the corresponding bias terms, r and $\hat{x}$ implies the learnt features and reconstruction of x, $l(\cdot)$ measures the differences between x and $\hat{x}$, and Ω is the regularization term to improve the feature quality. By minimizing L, the CAE is trained, parameters in convolution operation, bias terms, and deconvolutional operation are confirmed. Encoders with these parameters from multiple trained CAEs are

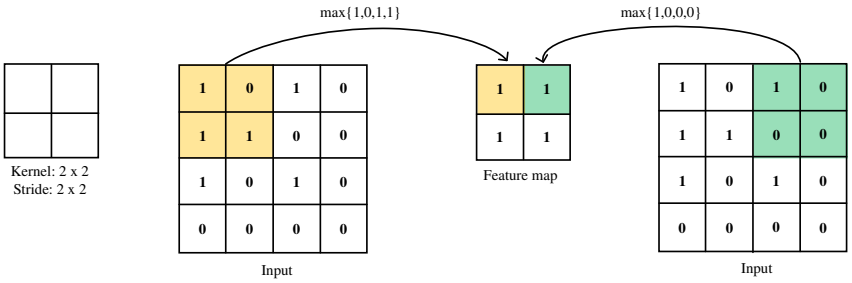


Figure 3: An example of maximal pooling operation. The same colors in the input data and the feature map refer to the enveloped area of the kernel and the resulted element value of the feature map.

composed to be a CNN for learning hierarchical features that benefit the final classification performance [7].

$$\begin{cases} r = & pool\left(F\left(conv\left(x\right)+b_1\right)\right) \\ \hat{x} = & G\left(de\_conv\left(r\right)+b_2\right) \\ minimize & L = l(x,\hat{x}) + \Omega \end{cases} \tag{1}$$

***Motivation of FCAE*:** The architectures of a CNN stacked by CAEs and a state-of-the-art CNN named VGGNet [27] are shown in Fig. 4a and Fig. 4b, respectively. From these two examples, it is evident that CAEs are incapable of stacking to VGGNet. Therefore, the concern is naturally raised that the architecture of CAE should be revised for fitting this situation, which inspires the design of FCAE.

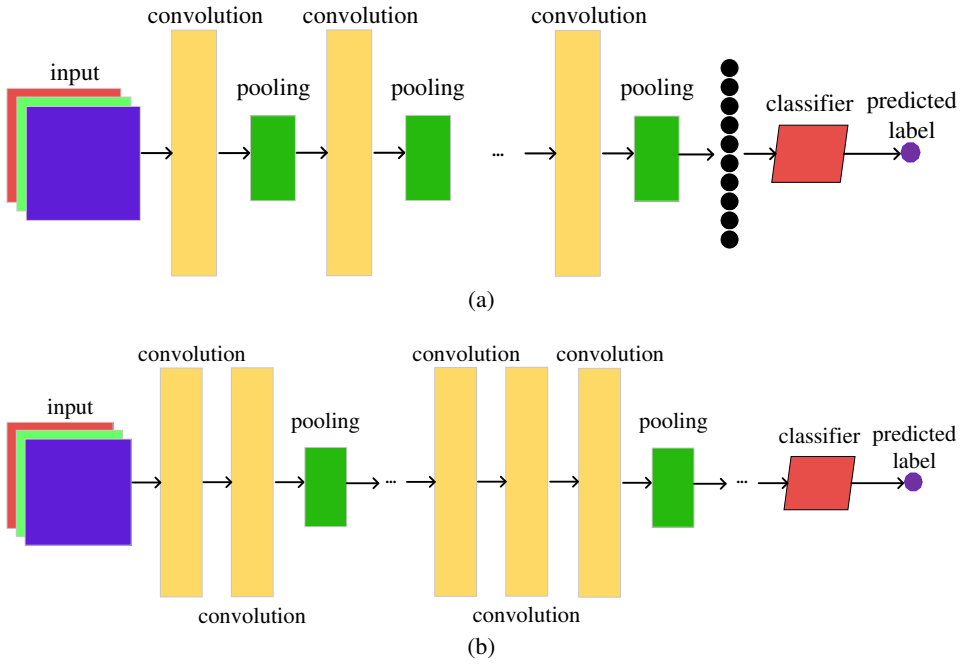


Figure 4: Architectures of the CNN stacked by CAEs (in Fig. 4a) and VGGNet (in Fig. 4b).

2.2 Particle Swarm Optimization

A typical PSO has the procedure as follows:

Step 1): Initialize the particles;

Step 2): Evaluate the fitness of particles;

Step 3): For each particle, choose the best one $pBest$ from its memory;

Step 4): Choose the best particle $gBest$ from the history of all particles;

Step 5): Calculate the velocity v_i of each particle x_i by Equation (2);

Step 6): Update the position p_i of each particle x_i by Equation (3);

Step 7): Repeat Step 2) – Step 6) until it terminates.

$$v_i \leftarrow \overbrace{w \cdot v_i}^{\text{inertia}} + \overbrace{c_1 \cdot r_1 \cdot (p_g - p_i)}^{\text{global search}} + \overbrace{c_2 \cdot r_2 \cdot (p_p - p_i)}^{\text{local search}} \tag{2}$$

$$p_i \leftarrow p_i + v_i \tag{3}$$

In Equation (2), w denotes the inertia weight, c_1 and $c2$ are acceleration constants, r_1 and r_2 are random numbers between 0 and 1, and p_g as well as p_b denotes the positions of $gBest$ as well as $pBest$, respectively. By integrating the "inertia", "global search", and "local search" terms into the velocity updating, the best position is expected to be found by particles.

***Limitation for Architecture Design in FCAE*:** The velocity calculation requires the particle x_i, $gBest$, and $pBest$ have the same length. When PSO is used for the architecture optimization of FCAE, particles represent the potential optimal architectures of FCAE. Because the optimal architecture of FCAE for solving the task at hand is unknown, particles with different lengths will be emerged. Therefore, a novel velocity calculation method need be designed in this regard.

3 The proposed PSOAO Algorithm

In this section, the definition of FACE is proposed, and the details of the designed PSOAO algorithm for FCAE are described in detail.

3.1 Algorithm Overview

Algorithm 1: Framework of the PSOAO Algorithm

```
1  x_0 ←Initialize the particles based on the proposed encoding strategy;
2  t ← 0;
3  while t < the maximal generation number do
4  |  Evaluate the fitness of each particle in x_t;
5  |  Update the pBest and gBest;
6  |  Calculate the velocity of each particle;
7  |  Update the position of each particle;
8  |  t ← t + 1;
9  end
10 Return gBest for deep training.
```

Algorithm 1 outlines the framework of the designed PSOAO algorithm. Firstly, particles are randomly initialized based on the proposed encoding strategy (line 1). Then, particles get start to evolve until the the generation number that it has evolved exceeds the predefined one (lines 3-9). Finally, the $gBest$ particle is picked up for deep training, which is for reaching the final performance to solve tasks at hand (line 10).

During the evolution, the fitness of each particle is evaluated (line 4) first, and then the $pBest$ and $gBest$ is updated based on the fitness (line 5). Next, the velocity of each particle is calculated (line 6) and their positions are updated (line 7) for the next generation of evolution. In the following subsections, keys aspects in PSOAO are detailed.

3.2 Encoding Strategy

For convenience of the development, the definition of FCAE is given by Definition 1 by generalizing the building blocks in all CNNs. Obviously, the CAE is a special form of FCAE when the numbers of convolutional layer and pooling layers are both set to be 1.

Definition 1. *A flexible convolutional auto-encoder encompasses one encoder and one decoder. The encoder is composed of the convolutional layers and pooling layers, where these two types of layers are not mixed and their numbers are arbitrary. The decoder part is the inverse form of the encoder.*

We design an encoding strategy through *variable-length* particles to encode the potential architectures of FCAEs into each particle. Based on the introduction of the convolution operation and pooling operation in Subsection 2.1, all the encoded information of PSOAO for FCAE are summarized in Table 1. Each particle contains different numbers of convolutional layers and pooling layers. Three general encoded particles in PSOAO are illustrated in Fig. 5.

Table 1: Encoded information in the convolutional layers and the pooling layers of FCAE.

Layer Type	Encoded Information
convolutional layer	filter width, filter height, stride width, stride height, convolutional type, number of feature maps, and the coefficient of l_2.
pooling layer	kernel width, kernel height, stride width, stride height, pooling type

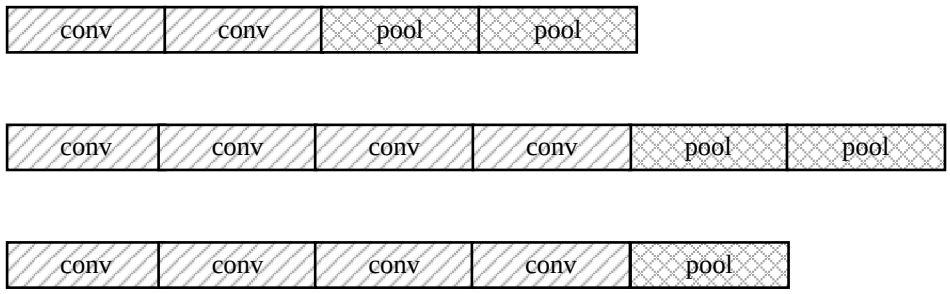


Figure 5: Three particles with different encoded information from PSOAO.

Because deep neural networks are typically with tremendous numbers of parameters that easily cause the overfitting problem, the weight decay regularization term (i.e., the l_2 term) [51] is considered in FCAE. Because only the convolutional layers involve weight parameters, this regularization term is applied only to the convolutional layers. In the following, we will detail the motivation of this encoding strategy.

In the proposed PSOAO algorithm, a variable-length encoding strategy is designed for the particles representing FCAEs with different architectures. The major reason is that the optimal architecture is unknown prior to the optimization is solved and fixed-length encoding strategy is not effective at this occasion. Specifically, If the traditional fixed-length encoding strategy is employed, the maximal length should be specified in advance, and by 0 and 1 indicating whether the corresponding position is available to control the absolute length of each particle. However, the maximal length is not easy to set and needs to carefully tune for the best performance. A smaller number denoting the maximal length would be inefficient for the optimized architecture of FCAE to solve real-world applications. Otherwise, a larger number would take more unnecessary computation, and also results in the worse performance within the same predefined evolution generation number. Furthermore, two types of layers exist in each particle, which increases the difficulty of employing the fixed-length encoding strategy due to the enlarged search space. With the designed variable-length encoding strategy, all the information of potential optimal architecture for FCAE can be flexibly represented for exploitation and exploration without manual intervention, which makes PSOAO be more concise and easy to use.

Algorithm 2: Particle Initialization

Input: The population size n, the maximal number of convolutional layers N_c, and the maximal number of and pooling layers N_p.

Output: Initialized population $\mathbf{x}_0$.

1 $\mathbf{x}_0 \leftarrow \emptyset$;
2 **while** $|\boldsymbol{x}_0| \leq N$ **do**
3 $\quad conv\_list \leftarrow \emptyset$;
4 $\quad n_c \leftarrow$ Uniformaly generate an integer between $[1, N_c]$;
5 $\quad$ **while** $conv\_list \leq n_c$ **do**
6 $\quad\quad conv\_unit \leftarrow$ Initialize a convolutional layer with random settings;
7 $\quad\quad conv\_list \leftarrow conv\_list \cup conv\_unit$;
8 $\quad$ **end**
9 $\quad pool\_list \leftarrow \emptyset$;
10 $\quad n_p \leftarrow$ Uniformaly generate an integer between $[1, N_p]$;
11 $\quad$ **while** $|pool\_list| \leq n_p$ **do**
12 $\quad\quad pool\_unit \leftarrow$ Initialize a pooling layer with random settings;
13 $\quad\quad pool\_list \leftarrow pool\_list \cup pool\_unit$;
14 $\quad$ **end**
15 $\quad \mathbf{x}_0 \leftarrow \mathbf{x}_0 \cup (conv\_list \cup pool\_list)$;
16 **end**
17 **Return** $\mathbf{x}_0$.

3.3 Particle Initialization

Algorithm 2 shows the procedure of the particle initialization within the given population size, maximal numbers of the convolutional layers and the pooling layers. Particularly, lines 3-8 demonstrate the initialization of the convolutional layers, and lines 9-14 imply the initialization of the pooling layers, where the random settings refer to the settings of the information encoded in these two types of layers. Because the decoder part of FACE can be explicitly derived from its encoder part, each particle in the proposed PSOAO algorithm contains only the encoder part for reducing the computational complexity.

3.4 Fitness Evaluation

Algorithm 3 shows the fitness evaluation for the particles in PSOAO. As we have introduced in Subsection 2.1, the reconstruction error added by the loss of the regularization term is always as the objective function for training CAE. However, the loss of the regularization term used in FCAE (i.e., the l_2 loss in line 6) are highly affected by the weights, and different architectures are with different weight numbers and the weight values. In order to investigate whether only the architecture is positive the particle quality, l_2 loss is discarded and only the reconstruction error is employed as the fitness (lines 11-16).

Typically, a deep learning algorithm requires a training epoch number with the magnitude of $10^2 - 10^3$ to train its weight parameters by gradient-based algorithms, which is a lasting period especially in population-based algorithms. In the designed PSOAO algorithm, this number is specified as a smaller number (e.g., 5 or 10). The reason is for largely speeding up PSOAO (running time on the investigated benchmark datasets are shown in Table 2) yet with less computational resources (adopted computational resources for the ingestigated benchmakr datastes are given in Subsection 4.3), while the promising performance of PSOAO is still maintained (experimental results regarding the performance are shown in Subsections 5.1 and 5.3). For example, it will take 2 minutes for training one epoch on the CIFAR10 dataset (with $50,000$ training samples) utilizing one GPU card with the model number of GTX1080. If we train it with 10^2 epochs for each particle with the population size of 50 and 50 generations, it will take about one year, which is not be acceptable for the purpose of general academic research. The widely used solution for easing this adverse is to employ intensive computation resources, such as the LSIC algorithm proposed by Google in very recently of 2017, where 250 computers are employed for about 20 days on the CIFAR-10 dataset using genetic algorithm for the architecture discovering. In fact, it is not necessary to evaluate the final

Algorithm 3: Fitness Evaluation

Input: The population $\mathbf{x}_t$, the training set D_{train}, the number N_{train} of training epoch.
Output: The population $\mathbf{x}_t$ with fitness.

```
for each x in x_t do
    Γ ← Decode x to a FCAE with random weights;
    for i = 1 to N_train do
        for each training batch data in D_train do
            r ← Calculate the reconstruction error;
            l_2 ← Calculate the l_2 loss;
            l ← r + l_2;
            Minimize l and update the weights in Γ;
        end
    end
    error_list ← ∅;
    for each training batch data in D_train do
        r ← Calculate the reconstruction error;
        error_list ← error_list ∪ r;
    end
    Calculate the mean of error_list and set it as the fitness of x;
end
Return x_t.
```

performance of each particle by a large number of training epochs during the architecture discovering, but picking up the promising particle from less training epochs and then deep training once with sufficient training epochs. In the proposed PSOAO algorithm, a small number of training epochs is employed to conduct the fitness evaluation of particles. With the evaluated fitness, the $gBest$ and $pBest$ are selected to guide the search towards the optimum. When the evolution is terminated, the $gBest$ is selected and one-time deep training is performed for reaching the optimal performance.

3.5 Velocity Calculation and Position Update

In PSOAO, the particles are with different lengths, and Equation (2) cannot be directly used. To solve this concern, a method, named "x-reference", is designed to calculate the velocity. In "x-reference", $gBest$ and $pBest$ refer to the length of the current particle x. Because the $pBest$ is selected from the memory of each particle, and the $gBest$ is chosen from all particles, the current particle x is always with the same length of $pBest$ and the "x-reference" is applied only to the "global search" part of Equation (2). Algorithm 4 shows the details of the "x-reference" method.

Specifically, this "x-reference" method is applied twice with the same manner in the velocity calculation for the "global search" part of Equation (2). The first is on the convolutional layer part of $gBest$ and x (lines 2-14), and the other is on the pooling layer part (line 15). For the convolutional layer part, if the number of convolutional layers cg from $gBest$ is smaller than that from x, new convolutional layers initialized with zero values are padded to the tail of cg. Otherwise, extra convolutional layers are truncated from the tail of cg. After the "global search" part is derived by Algorithm 4, the "inertia" and "local search" parts in Equation (2) are calculated as normal, then the complete velocity is calculated and the particle position is updated by Equation (3).

For intuitive understanding the proposed x-reference velocity calculation method, an example is provided in Fig. 6. Specifically, Fig. 6a denotes the $gBest$ and x that are used to do the "global search" part velocity calculation. In Fig. 6b, the convolutional layers and pooling layers are collected from $gBest$ and x. Because the lengths of convolutional layers and pooling layers from x are 2 and 4, and those from $gBest$ are 3 and 2, the last convolutional layer form the convolutional layer part from $gBest$ is truncated, and other two pooling layers are padded to the tail of the pooling layer part of $gBest$. In particular, the padded pooling layers are with white, which means they are created with the values of encoded information equal to 0. Fig 6c demonstrates the calculation between the convolutional layer part and pooling layer part from $gBest$ and x. Fig 6a shows the results of this calculation.

Algorithm 4: The x-reference Velocity Calculation Method

Input: The partilce x, the $gBest$, the acceleration constant c_1.
Output: The gloabl search part of velocity calculation.

$r_1 \leftarrow$ Randomly sample a number from $[0, 1]$;
$cg \leftarrow$ Extract the convolutional layers from $gBest$;
$cx \leftarrow$ Extract the convolutional layers from x;
$pos\_c \leftarrow \emptyset$;
if $|cg| < |cx|$ **then**
    $c \leftarrow$ Initialize $|cx| - |cg|$ convolutional layers with encoded information of zeros;
    $cg \leftarrow$ Pad c to the tail of cg;
else
    $cg \leftarrow$ Truncate the last $|cg| - |cx|$ convolutional layers from cg;
end
for $i = 1$ *to* $|cx|$ **do**
    $p_{cg_i}, p_{x_i} \leftarrow$ Extract the position of the i-th convolution layer from cg and cx;
    $pos\_c \leftarrow pos\_c \cup c_1 \cdot r_1 \cdot (p_{cg_i} - p_{x_i})$;
end
$pos\_p \leftarrow$ Analogy the operations in lines 2-14 on the pooling layers of $gBest$ and x;
Return $pos\_c \cup pos\_p$.

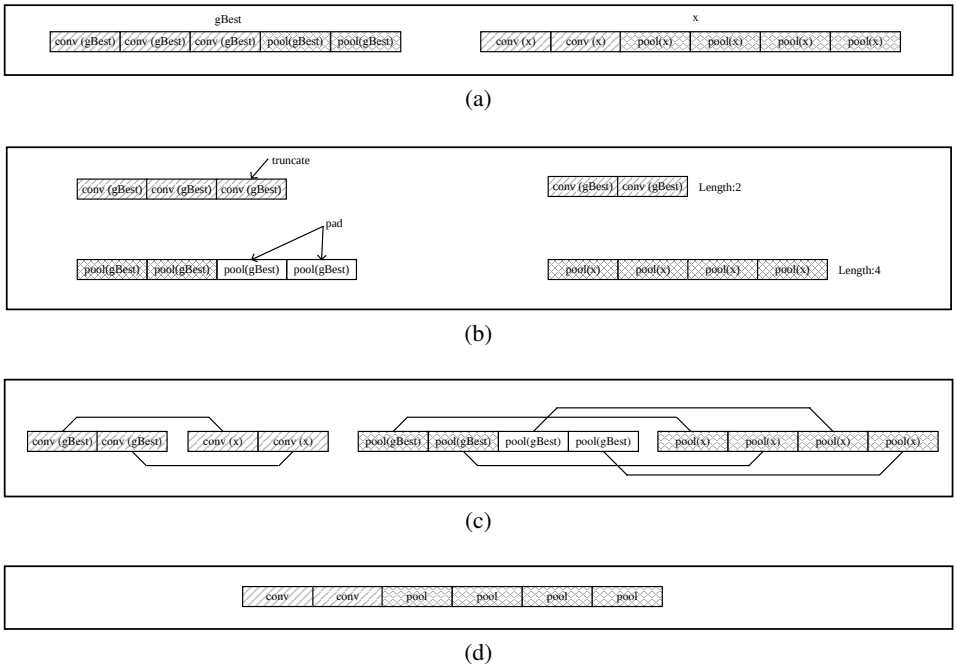


Figure 6: An example to illustrate the proposed x-reference velocity calculation method.

No matter padding or truncating is operated in the designed x-reference velocity calculation method, the goal is to maintain the same length of convolutional layers and pooling layers in $gBest$ to that of the convolutional layers and pooling layers in x, respectively. The mechanism behind this design is discussed as follows. In PSOAO, there are a group of particles with different lengths in the population, with the same goal of searching for the optimal architectures of FCAEs for solving image classification tasks. If we have each particle follow the length of $gBest$ (i.e., the $gBest$-reference velocity calculation method), all the particles will have the same lengths to that of $gBest$ from the second generation. Because the $pBest$ is chosen from the memory of each particle, the $gBest$, $pBest$, and current particle x are all have the same length may from the third generation. Consequently, all particles anticipate in the optimization with one particular depth of FCAE and only change the encoded information. Indeed, the variation of length regarding $gBest$ can be seen as an exploration search behaviour, and that of the encoded information is viewed as an exploitation search behaviour. When all particles are with the same lengths, the length of $gBest$ will be constant until it terminates. In this regard, the exploration search ability is lost if we employ the $gBest$-reference velocity calculation method. In addition, keeping the length of x equal to $gBest$ can also be viewed as the loss of diversity, which would easily cause the premature phenomenon in population-based algorithms. Both the loss of exploration search and premature phenomenon will result in the bad performance. An experiment is conducted in Subsection 5.4 to further quantitatively investigate this velocity calculation design.

3.6 Deep Training on $gBest$

When the evolution of PSOAO is finished, the best particle $gBest$ is picked for deep training. As we have stated in Subsection 3.4 that each particle is trained with only several epochs, which cannot indicate the final performance for solving real-world applications. Therefore, this deep training is provided.

4 Experiment Design

In this section, the benchmark datasets, peer competitors, and the parameter settings are detailed for the experiments investigating the performance of the proposed FCAE of which architecture is optimized by the designed PSOAO algorithm.

4.1 Benchmark Datasets

The experiments are conducted on four image classification benchmark datasets, which are widely used and specifically for investigating the performance of AEs. They are the CIFAR10 dataset [52], the MNIST dataset [9], the STL-10 dataset [53], and the Caltech-101 dataset [54]. Fig. 7 shows examples from these benchmark datasets. In the following, briefly details about these chosen datasets are briefly introduced.

4.1.1 CIFAR-10 dataset

It contains $50,000$ training images and $10,000$ test images, each one is a 3-channel RGB image with the size of 32×32, and contains 10 categories of natural objectives (i.e., truck, ship, horse, frog, dog, deer, cat, bird, automobile, and airplane) with roughly the same number in each category. In addition, different objects occupy different areas of the images.

4.1.2 MNIST dataset

It is a handwritten digit recognition dataset to classify the numbers of $0-9$, including $60,000$ training images and $10,000$ test images. Each one is a 1-channel gray image with the size of 28×28, and the samples in each category are with different variations, such as rotation and so on. Each category is composed about the same numbers of image samples.

4.1.3 STL-10 dataset

It is a widely used dataset for unsupervised learning, containing $100,000$ unlabeled images, $5,000$ training images, and $8,000$ test images for 10-category natural image object recognition (i.e., air-

(a) Examples from the CIFAR-10 dataset. From left to right, they are from the categories of truck, ship, horse, frog, dog, deer, cat, bird, automobile, and airplane, respectively.

(b) Examples from the Mnist dataset. From left to right, they are from the categories of $0-9$, respectively.

(c) Examples from the STL-10 dataset. From left to right, they are from categories of airplane, bird, car, cat, deer, dog, horse, monkey, ship, and truck, respectively.

(d) Examples from the Caltech-101 dataset. From left to right, they are from categories of accordio, bass, camera, dolphin, elephant, ferry, gramophone, headphone, lamp, and sunflower, respectively.

Figure 7: Examples from the chosen benchmark datasets.

plane, bird, car, cat, deer, dog, horse, monkey, ship, truck). Each one is a 3-channel RGB image with the size of 96×96. In addition, the unlabeled images contain images that are beyond the 10 categories. Due to the less number of training images, this dataset challenges the feature learning ability of CAEs/AEs.

4.1.4 Caltech-101 dataset

It is a 101-category image classification dataset where the weights and heights of images vary from 80 to 708 pixels. Most images are 3-channel RGB while occasionally gray, and with different numbers in each category from 31 to 800. In addition, most images only display a small part in the image, and other areas are occupied by noises for increasing the difficulty of classification accuracy. Due to the quite less number of images, and the non-identical numbers of images in each category, this dataset also challenges the feature learning algorithms.

4.2 Peer Competitors

Peer competitors of FCAE, which have been introduced in Section 1, are employed for performing the comparisons on the chosen image classification benchmark datasets. They are the CAE [16], Convolutional RBM (CRBM) [19], and the state-of-the-art Convolutional Denoising AE (CDAE) [24]. In addition, two widely used variants of AEs are also employed as the peer competitors for a comprehensive comparison. They are the Sparse AE (SAE) [55] and Denosing AE (DAE) [25].

Because this paper aims at proposing a FCAE that could be stacked to the state-of-the-art CNNs. Peer competitors for comparisons here should be the stacked forms of these CAEs/AEs, i.e., the Stacked CAE (SCAE), the Stacked CRBM (SCRBM), the Stacked CDAE (SCDAE), the Stacked SAE (SSAE), and the stacked DAE (SDAE). The Stacked form of the proposed FCAE is SFCAE.

4.3 Parameter Settings

The peer competitors SCAE, SRBM, SSAE, and SDAE have been investigated very recently on the chosen benchmark datasets with a wide range of parameter tunings in [24]. Their classification results are directly referred in the comparisons, thus none of their parameter settings needs to be specified. In addition, the state-of-the-art SCDAE provided the classification results with the usage of only one and two building blocks on the chosen benchmark datasets. In order to do a fair comparison, we also perform experiments on SFCAE with at most two building blocks. Noting that the SFCAE is tested on the chosen benchmark datasets without the preprocessing of data augmentation for keeping consistency to its peer competitors. In the following, the parameter settings of PSOAO are provided in detail.

In the designed PSOAO algorithm, PSO related parameters are specified based on their conventions [56], i.e., the inertia weight w is set to be 0.72984, the acceleration constants c_1 and c_2 are both set to be 1.496172, and the initialize velocity is set to be 0. For the fitness evaluation of PSOAO, the training set of MNIST and CIFAR-10, the unlabeled data of STL-10 are natuarally used. Due to the non-identical and quite less numbers of images in each category of the Caltech-101 dataset, 30 images randomly selected from each category are used for the fitness evaluation and aso as the training set based on the suggestions in [24]. Because the inapproaciate settings of convolutional operations and pooling operations would cause the unaffordable computational cost and make FCAE no sence, in the exploration of each particle, the number of feature maps is set to be $[20, 100]$, the kernel with the same size of width and height is set to be $[2, 5]$, the maximal number of pooling layers is set to be 1, and that of convolutional layer is set to be 5. In addition, the coefficient of l_2 term is set to be $[0.0001, 0.01]$, which is a commonly utilized range for training neural networks in practice.

In the deep training phase, the FCAE with the architecture confirmed by PSOAO and the weights initialized by the widey used Xavier method [57], by adding one full connection layer with 512 units and 50% Dropout [58] from the conventions of deep learning comunity. We investigate the classification results by feeding the trained model with the corresponding test set[1], employing the widely used rectifier linear unit [59] as the activation function, the Adam [60] optimizer with its default settings as the training algorithm for weight optimization, and the BatchNorm [61] technique for speeding up the training. For keeping consistency to the results to be compared, the experiments with the trained model on each benchmark dataset are also independently performed 5 runs. Due to the extreme unbalance training data exist in the Caltech-101 dataset, we investigate this dataset based on its convention [19], i.e., investigating the classification accuracy on each category of the images, and then reporting the mean and standard derivations over the benchmark dataset.

The proposed PSOAO algorithm is implemented by Tensorflow [62], and each copy of the source code runs on one GPU card with the same model number of GTX1080. In addition, the architecture configurarions of FCAEs optimized by PSOAO for the chosen benchmark datasets in these experiments are provided in Appendix 6. Training time of the proposed PSOAO algorithm on investigated benchmark datasets are shown in Table 2.

Table 2: Consumed time (hours) of the proposed PSOAO algorithm for different benchmark datasets.

CIFAR-10	MNIST	STL-10	Caltech-101
81.5	118	230	22.5

5 Experimental Results and Analysis

5.1 Overview Performance

Table 3 shows the mean and standard derivations of the classification accuracies of the proposed FCAE method of which the architecture is optimized by the designed PSOAO algorithm against the peer competitors on the chosen benchmark datasets. Because literatures do not the provide the standard derivations of SCAE on CIFAR-10 and MNIST, SCRBM on CIFAR-10, and SDAE on

[1]The test data of Caltech-101 dataset is the entire dataset excluding from the training set.

MNIST, only their mean classification accuracies are shown. The references in Table 3 denote the sources of the corresponding mean classification accuracy, and the best mean classification results are highlighted with bold. The terms "SFCAE-1" and "SFCAE-2" refer to the SFCAE with one and two building blocks, respectively, which is the same meaning to the terms "SCDAE-1" and "SCDAE-2".

Table 3: The classification accuracy of the proposed FCAE method against its peer competitors on the chosen benchmark datasets.

Algorithm	CIFAR-10	MNIST	STL-10	Caltech-101
SSAE	74.0 (0.9)	96.29 (0.12)	55.5 (1.2)	66.2 (1.2)
SDAE	70.1 (1.0)	99.06 [25]	53.5 (1.5)	59.5 (0.3)
SCAE	78.2 [16]	99.29 [16]	40.0 (3.1)	58.0 (2.0)
SCRBM	78.9 [63]	99.18 [19]	43.5 (2.3)	65.4 (0.5) [19]
SCDAE-1	75.0 (1.2)	99.17 (0.10)	56.6 (0.8)	71.5 (1.6)
SCDAE-2	80.4 (1.1)	99.38 (0.05)	60.5 (0.9)	78.6 (1.2)
SFCAE-1	78.9 (0.3)	99.30 (0.03)	**61.2 (1.2)**	**79.8 (0.0)**
SFCAE-2	**83.5 (0.5)**	**99.51 (0.09)**	56.8 (0.2)	79.6 (0.0)

It is clearly shown in Table 3 that FCAE outperforms the traditional AEs (i.e., the SSAE and the SDAE) and traditional CAEs (i.e., the SCAE and the SCRBM) on all the chosen benchmark datasets. In addition, FCAE also wins the state-of-the-art CAEs (i.e, the SCDAE-1 and the SCDAE-2) on these benchmark datasets. Furthermore, the best results on CIFAR-10 and MNIST are reached by SFCAE-2, and those on STL-10 and Caltech-101 are by SFCAE-1. Noting that SFCAE-2 performs worse on STL-10 and Caltech-101 than SFCAE-1, which is caused by the much less smaller of training data in these two benchmark datasets, and deeper architectures are suffered from the over-fitting problem. Because CIFAR-10 and MNIST are with much more training samples ($50,000$ in CIFAR-10 and $60,000$ in MNIST), a deeper architecture naturally results in the promising classification accuracy. In summary, when the architecture of the proposed FCAE method is optimized by the designed PSOAO algorithm, FCAE shows superiority performance among its peer competitors on the four image classification benchmark datasets.

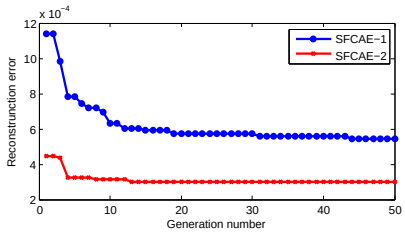
5.2 Evolution Trajectory of PSOAO

In order to intuitively investigate the efficacy of the designed PSOAO algorithm in optimizing the architectures of the proposed FCAE method, its evolution trajectories on the chosen benchmark datasets during the training phases are plotted in Fig. 8.

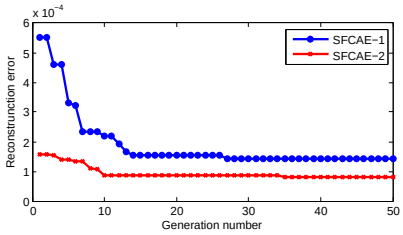
As can be seen from Figs. 8a, 8b, 8c, and 8d, PSOAO has converged within the specified maximal generation number. Specifically, it has converged since about the 15-th generation on all benchmark datasets for both SFCAE-1 and SFCAE-2, and about the 5-th generation on the CIFAR-10 and Caltech-101 datasets for SFCAE-2. Noting that the reconstruction error of SFACE-2 is smaller than that of SFCAE-1 on STL-10 dataset (shown in Fig. 8c), which is caused by the different input data for them.

5.3 Performance on Different Numbers of Training Examples

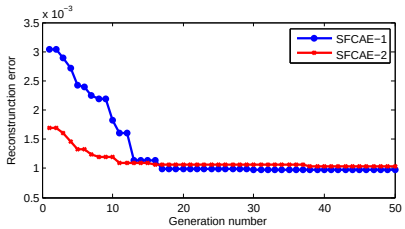
In this subsection, we investigate the classification performance of the proposed FCAE method whose architecture optimized by the designed PSOAO algorithm on different numbers of training samples. The peer competitors on the MNIST dataset are SCRBM [19], ULIFH [64], SSE [65], and SCAE [16], those for the CIFAR-10 dataset are SCAE [16], Mean-cov, RBM [63], and K-means (4k feat) [53]. The reason of choosing these benchmark datasets and peer competitors is that literatures have provided their corresponding information that are usually used by the comparisons between various variants of CAEs.



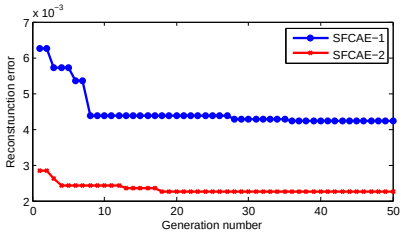
(a) The evolution trajectories on the CIFAR-10 dataset.



(b) The evolution trajectories on the MNIST dataset.



(c) The evolution trajectories on the STL-10 dataset.



(d) The evolution trajectories on the Caltech-101 dataset.

Figure 8: Trajectories of the PSOAO algorithm in automatically discovering the architectures of FCAE on the chosen benchmark datasets.

Table 4: The classification accuracy of FCAE-2 against peer competitors on the different numbers of training samples from MNIST.

# samples	1K	2K	3K	5K	10K	60K
SCRBM	97.38	97.87	98.09	98.41	—	99.18
ULIFH	96.79	97.47	—	98.49	—	99.36
SSAE	97.27	—	98.17	—	—	98.50
SCAE	92.77	—	—	—	98.12	99.29
SFCAE	**97.49**	**98.45**	**98.80**	**99.09**	**99.16**	**99.51**

Table 5: The classification accuracy of FCAE-2 against peer competitors on the different numbers of training samples from CIFAR-10.

# samples	1K	10K	50K
SCAE	47.70	65.65	78.20
Mean-cov. RBM	—	—	71.00
K-means (4k feat)	—	—	79.60
SFCAE	**53.79**	**73.96**	**83.47**

Tables 4 and 5 show the experimental results of FCAE-2 with the architecture confirmed in Subsection 5.1 on different numbers of training samples of the MNIST and CIFAR-10 benchmark datasets. The term "—" denotes their is no result reported in the corresponding literature. The best classification accuracy is highlighted with bold.

As can be seen from Tables 4 and 5, SFCAE-2 surpasses all peer competitors on this experiment. Especially, with much less number (1K) of training examples, SFCAE-2 is better than the seminal work of CAE (SCAE) with 4.72% classification accuracy improvement on MNIST and 6.09% on CIFAR-10. These experimental implies the promising scalability of SFCAE-2 in dealing with different numbers of training samples.

5.4 Investigation on x-reference Velocity Calculation

To further investigate the superiority of the designed x-reference velocity calculation method, we replace it with its opponent, i.e., the $gBest$-reference velocity calculation method, to compare the performance still on the chosen benchmark datasets introduced in Subsection 4.1. To achieve this, we first let the length of $pBest$ equal to that of $gBest$. If the length of the convolutional layers in $pBest$ is less than that of $gBest$, zeros are padded. Otherwise, truncating corresponding parts from $pBest$. Using this method to the pooling layers of $pBest$ and these two types of layers in x. Then, we use Equations3 and 2 to update the position of each particle. The experimental results are shown in Fig. 9, where Fig. 9a shows the results of FSCAE-1 and Fig. 9b shows the those of FSCAE-2.

As can be seen from Fig. 9a, with the x-reference velocity calculation method, PSOAO achieves the classification accuracy improvements of 5.7%, 8.8%, 5.9%, and 7.9% on the CIFAR-10, MNIST, STL-10, and Caltech-101 benchmark datasets, respectively. The same promising performance of SFCAE-2 can also be observed with the classification accuracy improvements of 7.7%, 10.6%, 5.6%, and 9.6% on these chosen benchmark datasets.

In summary, the analysis and experimental results collectively justify the effectiveness of the designed x-reference velocity calculation method in the proposed PSOAO algorithm.

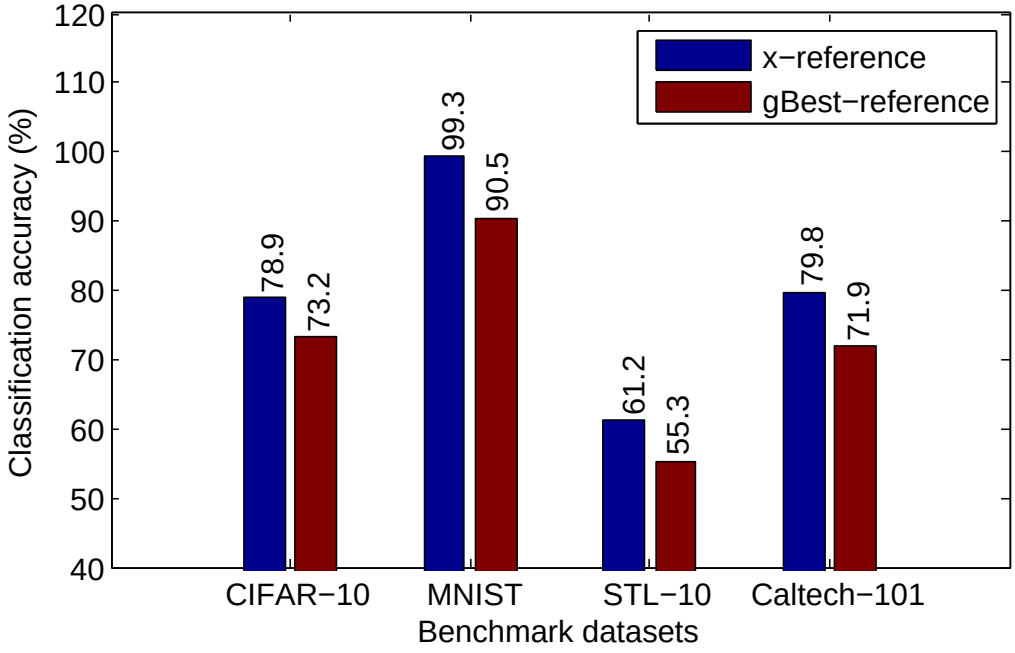
6 Conclusions

The goal of the paper was to develop a new particle swarm optimization algorithm (named PSOAO) to automatically discover the optimal architecture of the flexible convolutional auto-encoder (named FCAE) for image classification problems without manual intervention. This goal has been successfully achieved by defining the FCAE that has the potential to construct the state-of-the-art deep convolutional neural networks, an efficient encoding strategy that is capable of representing particles with non-identical lengths in PSOAO, and an effective velocity calculation mechanism for these particles. The FCAE with the optimal architecture was achieved by PSOAO, and compared with five peer competitors including the most state-of-the-art algorithm on four benchmark datasets specifically used by auto-encoders for image classification task. The experimental results indicate that FCAE remarkably outperforms all compared algorithms on all adopted benchmark datasets in term of their classification accuracies. Furthermore, FCAE with only one building block can surpasses the state-of-the-art with two building blocks on the STL-10 and Caltech-101 benchmark datasets. In addition, FCAE reaches the best classification accuracies compared with four peer competitors when only 1K, 2K, 3K, 5K, and 10K training images of the MNIST benchmark dataset are used, and significantly wins three peer competitors when only 1K and 10K training images of the CIFAR-10 benchmark dataset are used. Moreover, the designed PSOAO algorithm also shows the excellent characteristic of fast convergence by investigating its evolution trajectories, and the effective velocity calculation mechanism through the quantitative comparison to its component.

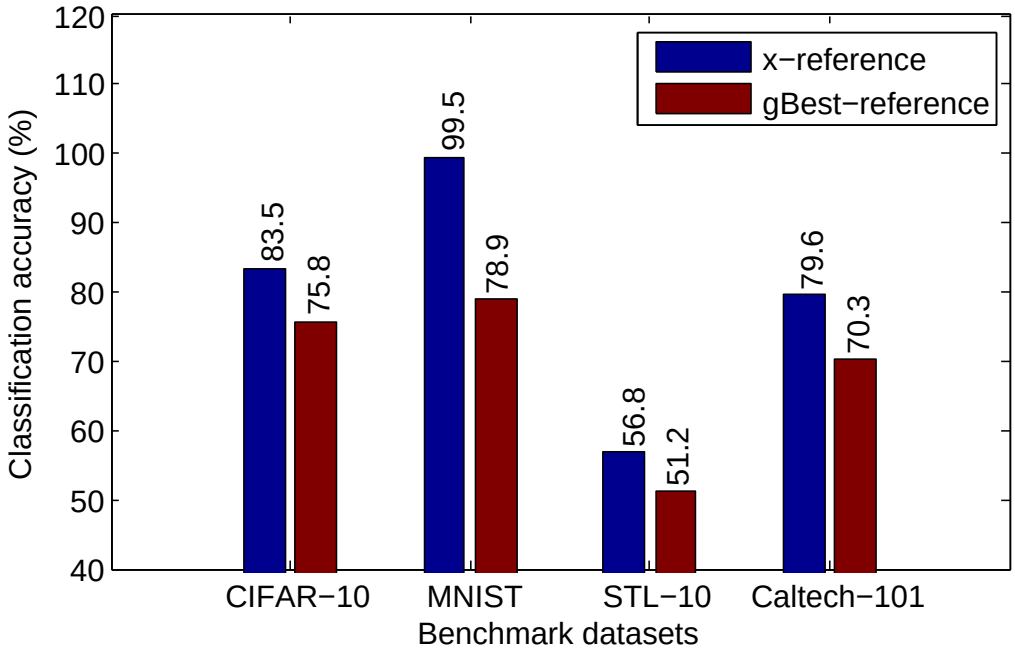
References

[1] D. E. Rumelhart, G. E. Hinton, R. J. Williams *et al.*, "Learning representations by back-propagating errors," *Cognitive modeling*, vol. 5, no. 3, p. 1, 1988.

[2] H. Bourlard and Y. Kamp, "Auto-association by multilayer perceptrons and singular value decomposition," *Biological cybernetics*, vol. 59, no. 4, pp. 291–294, 1988.



(a) Classification accuracy of FSCAE-1 by truncating $gBest$ and x.



(b) Classification accuracy of FSCAE-2 by truncating $gBest$ and x.

Figure 9: Classification accuracy comparisons between the x-reference and $gBest$-reference velocity calculation strategy in the designed PSOAO method.

[3] G. E. Hinton and R. S. Zemel, "Autoencoders, minimum description length and helmholtz free energy," in *Advances in neural information processing systems*, 1994, pp. 3–10.

[4] H. Schwenk and M. Milgram, "Transformation invariant autoassociation with application to handwritten character recognition," in *Advances in neural information processing systems*, 1995, pp. 992–998.

[5] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *science*, vol. 313, no. 5786, pp. 504–507, 2006.

[6] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Advances in neural information processing systems*, 2007, pp. 153–160.

[7] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[8] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[9] 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, 1998.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[11] C. Farabet, C. Couprie, L. Najman, and Y. LeCun, "Learning hierarchical features for scene labeling," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1915–1929, 2013.

[12] 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*, 2015, pp. 1–9.

[13] J. P. Pluim, J. A. Maintz, and M. A. Viergever, "Mutual-information-based registration of medical images: a survey," *IEEE transactions on medical imaging*, vol. 22, no. 8, pp. 986–1004, 2003.

[14] G. Pass, R. Zabih, and J. Miller, "Comparing images using color coherence vectors," in *Proceedings of the fourth ACM international conference on Multimedia*. ACM, 1997, pp. 65–73.

[15] X. Peng, B. Zhao, R. Yan, H. Tang, and Z. Yi, "Bag of events: An efficient probability-based feature extraction method for aer image sensors," *IEEE transactions on neural networks and learning systems*, vol. 28, no. 4, pp. 791–803, 2017.

[16] J. Masci, U. Meier, D. Cireşan, and J. Schmidhuber, "Stacked convolutional auto-encoders for hierarchical feature extraction," *Artificial Neural Networks and Machine Learning–ICANN 2011*, pp. 52–59, 2011.

[17] M. Norouzi, M. Ranjbar, and G. Mori, "Stacks of convolutional restricted boltzmann machines for shift-invariant feature learning," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009, pp. 2735–2742.

[18] P. Smolensky, "Information processing in dynamical systems: Foundations of harmony theory," COLORADO UNIV AT BOULDER DEPT OF COMPUTER SCIENCE, Tech. Rep., 1986.

[19] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations," in *Proceedings of the 26th annual international conference on machine learning*. ACM, 2009, pp. 609–616.

[20] M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus, "Deconvolutional networks," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. IEEE, 2010, pp. 2528–2535.

[21] M. D. Zeiler, G. W. Taylor, and R. Fergus, "Adaptive deconvolutional networks for mid and high level feature learning," in *Computer Vision (ICCV), 2011 IEEE International Conference on*. IEEE, 2011, pp. 2018–2025.

[22] B. A. Olshausen and D. J. Field, "Sparse coding with an overcomplete basis set: A strategy employed by v1?" *Vision research*, vol. 37, no. 23, pp. 3311–3325, 1997.

[23] K. Kavukcuoglu, P. Sermanet, Y.-L. Boureau, K. Gregor, M. Mathieu, and Y. L. Cun, "Learning convolutional feature hierarchies for visual recognition," in *Advances in neural information processing systems*, 2010, pp. 1090–1098.

[24] B. Du, W. Xiong, J. Wu, L. Zhang, L. Zhang, and D. Tao, "Stacked convolutional denoising auto-encoders for feature representation," *IEEE transactions on cybernetics*, vol. 47, no. 4, pp. 1017–1027, 2017.

[25] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *Journal of Machine Learning Research*, vol. 11, no. Dec, pp. 3371–3408, 2010.

[26] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[27] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[28] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, no. Feb, pp. 281–305, 2012.

[29] C. E. Rasmussen and C. K. Williams, *Gaussian processes for machine learning*. MIT press Cambridge, 2006, vol. 1.

[30] J. Močkus, "On bayesian methods for seeking the extremum," in *Optimization Techniques IFIP Technical Conference*. Springer, 1975, pp. 400–404.

[31] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Advances in Neural Information Processing Systems*, 2011, pp. 2546–2554.

[32] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration." *LION*, vol. 5, pp. 507–523, 2011.

[33] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," *International Conference on Learning Representations*, 2017.

[34] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, Q. Le, and A. Kurakin, "Large-scale evolution of image classifiers," *arXiv preprint arXiv:1703.01041*, 2017.

[35] K. O. Stanley, D. B. D'Ambrosio, and J. Gauci, "A hypercube-based encoding for evolving large-scale neural networks," *Artificial life*, vol. 15, no. 2, pp. 185–212, 2009.

[36] J. K. Pugh and K. O. Stanley, "Evolving multimodal controllers with hyperneat," in *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*. ACM, 2013, pp. 735–742.

[37] M. Kim and L. Rigazio, "Deep clustered convolutional kernels," in *Feature Extraction: Modern Questions and Challenges*, 2015, pp. 160–172.

[38] C. Fernando, D. Banarse, M. Reynolds, F. Besse, D. Pfau, M. Jaderberg, M. Lanctot, and D. Wierstra, "Convolution by evolution: Differentiable pattern producing networks," in *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference*. ACM, 2016, pp. 109–116.

[39] J. Kennedy and R. E. P. S. Optimization, "Ieee int," in *Conf. on Neural Networks*, vol. 4, 1995.

[40] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *Micro Machine and Human Science, 1995. MHS'95., Proceedings of the Sixth International Symposium on*. IEEE, 1995, pp. 39–43.

[41] B. Xue, M. Zhang, and W. N. Browne, "Particle swarm optimization for feature selection in classification: A multi-objective approach," *IEEE transactions on cybernetics*, vol. 43, no. 6, pp. 1656–1671, 2013.

[42] A. W. Mohemmed, M. Zhang, and M. Johnston, "Particle swarm optimization based adaboost for face detection," in *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*. IEEE, 2009, pp. 2494–2501.

[43] M. Setayesh, M. Zhang, and M. Johnston, "A novel particle swarm optimisation approach to detecting continuous, thin and smooth edges in noisy images," *Information Sciences*, vol. 246, pp. 28–51, 2013.

[44] J. Yu, S. Wang, and L. Xi, "Evolving artificial neural networks using an improved pso and dpso," *Neurocomputing*, vol. 71, no. 4, pp. 1054–1060, 2008.

[45] M. Settles, B. Rodebaugh, and T. Soule, "Comparison of genetic algorithm and particle swarm optimizer when evolving a recurrent neural network," in *Genetic and Evolutionary Computation—GECCO 2003*. Springer, 2003, pp. 200–200.

[46] Y. Da and G. Xiurun, "An improved pso-based ann with simulated annealing technique," *Neurocomputing*, vol. 63, pp. 527–533, 2005.

[47] C.-F. Juang, "A hybrid of genetic algorithm and particle swarm optimization for recurrent network design," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 34, no. 2, pp. 997–1006, 2004.

[48] W. Lu, H. Fan, and S. Lo, "Application of evolutionary neural network method in predicting pollutant levels in downtown area of hong kong," *Neurocomputing*, vol. 51, pp. 387–400, 2003.

[49] J. Salerno, "Using the particle swarm optimization technique to train a recurrent neural model," in *Tools with Artificial Intelligence, 1997. Proceedings., Ninth IEEE International Conference on*. IEEE, 1997, pp. 45–49.

[50] M. N. Omidvar, X. Li, Y. Mei, and X. Yao, "Cooperative co-evolution with differential grouping for large scale optimization," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 3, pp. 378–393, 2014.

[51] A. Krogh and J. A. Hertz, "A simple weight decay can improve generalization," in *Advances in neural information processing systems*, 1992, pp. 950–957.

[52] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009.

[53] A. Coates, A. Ng, and H. Lee, "An analysis of single-layer networks in unsupervised feature learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, pp. 215–223.

[54] L. Fei-Fei, R. Fergus, and P. Perona, "Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories," *Computer vision and Image understanding*, vol. 106, no. 1, pp. 59–70, 2007.

[55] H.-C. Shin, M. R. Orton, D. J. Collins, S. J. Doran, and M. O. Leach, "Stacked autoencoders for unsupervised feature learning and multiple organ detection in a pilot study using 4d patient data," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1930–1943, 2013.

[56] D. Bratton and J. Kennedy, "Defining a standard for particle swarm optimization," in *Swarm Intelligence Symposium, 2007. SIS 2007. IEEE*. IEEE, 2007, pp. 120–127.

[57] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*, 2010, pp. 249–256.

[58] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting." *Journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[59] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, 2011, pp. 315–323.

[60] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[61] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning*, 2015, pp. 448–456.

[62] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[63] A. Krizhevsky and G. Hinton, "Convolutional deep belief networks on cifar-10," *Unpublished manuscript*, vol. 40, 2010.

[64] F. J. Huang, Y.-L. Boureau, Y. LeCun *et al.*, "Unsupervised learning of invariant feature hierarchies with applications to object recognition," in *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*. IEEE, 2007, pp. 1–8.

[65] J. Weston, F. Ratle, H. Mobahi, and R. Collobert, "Deep learning via semi-supervised embedding," in *Neural Networks: Tricks of the Trade*. Springer, 2012, pp. 639–655.

Appendix

Tables 6, 8, 10, and 12 show the architecture configurations of SFCAE-1 for the CIFAR-10, MNIST, STL-10 and Caltech-101 datasets, respectively. Tables 7, 9, 11, and 13 show the architecture configurations of SFCAE-2 for the CIFAR-10, MNIST, STL-10 and Caltech-101 datasets, respectively.

Table 6: The architecture configuration of SFCAE-1 on the CIFAR-10 dataset.

layer type	**configuration**
conv	Filter: 2×2, # Feature map: 24, L_2: 0.0016
conv	Filter: 3×3, # Feature map: 57, L_2: 0.0001
conv	Filter: 5×5, # Feature map: 63, L_2: 0.0096
conv	Filter: 5×5, # Feature map: 35, L_2: 0.0071
conv	Filter: 3×3, # Feature map: 76, L_2: 0.0015
pooling	Kernel: 2×2, Stride: 2×2

Table 7: The architecture configuration of SFCAE-2 on the CIFAR-10 dataset.

layer type	**configuration**
conv	Filter: 4×4, # Feature map: 36, L_2: 0.0001
pooling	Kernel: 2×2, Stride: 2×2

Table 8: The architecture configuration of SFCAE-1 on the MNIST dataset.

layer type	**configuration**
conv	Filter: 2×2, # Feature map: 100, L_2: 0.0010
conv	Filter: 2×2, # Feature map: 82, L_2: 0.0018
conv	Filter: 2×2, # Feature map: 100, L_2: 0.0001
conv	Filter: 2×2, # Feature map: 100, L_2: 0.0001
pooling	Kernel: 2×2, Stride: 2×2

Table 9: The architecture configuration of SFCAE-2 on the MNIST dataset.

layer type	**configuration**
conv	Filter: 3×3, # Feature map: 89, L_2: 0.0001
conv	Filter: 3×3, # Feature map: 90, L_2: 0.0001
conv	Filter: 3×3, # Feature map: 93, L_2: 0.0005
pooling	Kernel: 2×2, Stride: 2×2

Table 10: The architecture configuration of SFCAE-1 on the STL-10 dataset.

layer type	**configuration**
conv	Filter: 2×2, # Feature map: 85, L_2: 0.0096
conv	Filter: 3×3, # Feature map: 77, L_2: 0.0086
conv	Filter: 3×3, # Feature map: 83, L_2: 0.0025
pooling	Kernel: 2×2, Stride: 2×2

Table 11: The architecture configuration of SFCAE-2 on the STL-10 dataset.

layer type	**configuration**
conv	Filter: 2×2, # Feature map: 83, L_2: 0.0094
conv	Filter: 4×4, # Feature map: 49, L_2: 0.0087
pooling	Kernel: 2×2, Stride: 2×2

Table 12: The architecture configuration of SFCAE-1 on the Caltech-101 dataset.

layer type	**configuration**
conv	Filter: 2×2, # Feature map: 14, L_2: 0.0084
conv	Filter: 5×5, # Feature map: 8, L_2: 0.0002
pooling	Kernel: 2×2, Stride: 2×2

Table 13: The architecture configuration of SFCAE-2 on the Caltech-101 dataset.

layer type	**configuration**
conv	Filter: 5×5, # Feature map: 15, L_2: 0.0051
pooling	Kernel: 2×2, Stride: 2×2