Evolving Deep Convolutional Neural Networks by Variable-length Particle Swarm Optimization for Image Classification

Bin Wang, Yanan Sun, Bing Xue and Mengjie Zhang
School of Engineering and Computer Science
Victoria University of Wellington, PO Box 600, Wellington 6140, NEW ZEALAND
Emails: wangbin@myvuw.ac.nz, {yanan.sun, bing.xue, mengjie.zhang}@ecs.vuw.ac.nz

***Abstract*—Convolutional neural networks (CNNs) are one of the most effective deep learning methods to solve image classification problems, but the best architecture of a CNN to solve a specific problem can be extremely complicated and hard to design. This paper focuses on utilising Particle Swarm Optimisation (PSO) to automatically search for the optimal architecture of CNNs without any manual work involved. In order to achieve the goal, three improvements are made based on traditional PSO. First, a novel encoding strategy inspired by computer networks which empowers particle vectors to easily encode CNN layers is proposed; Second, in order to allow the proposed method to learn variable-length CNN architectures, a Disabled layer is designed to hide some dimensions of the particle vector to achieve variable-length particles; Third, since the learning process on large data is slow, partial datasets are randomly picked for the evaluation to dramatically speed it up. The proposed algorithm is examined and compared with 12 existing algorithms including the state-of-art methods on three widely used image classification benchmark datasets. The experimental results show that the proposed algorithm is a strong competitor to the state-of-art algorithms in terms of classification error. This is the first work using PSO for automatically evolving the architectures of CNNs.**

I. Introduction

Convolutional neural networks (CNNs) have demonstrated exceptional superiority in numerous machine learning tasks, such as speech recognition [1], sentence classification [2] and image classification [3]. However, designing the architectures of CNNs for specific tasks can be extremely complex, which can be seen from some existing efforts done by researchers, such as LeNet [4][5], AlexNet [3], VGGNet [6] and GoogLeNet [7]. In addition, one cannot expect to get the optimal performance by applying the same architecture on various tasks, and the CNN architecture needs to be adjusted for each specific task, which will bring tremendous work as there are a large number of types of machine learning tasks in industry.

In order to solve the complex problem of the CNN architecture design, evolutionary computation (EC) has recently been leveraged to automatically design the architecture without any human effort involved. Interested researchers have done excellent work on the automatic design of the CNN architectures by using Genetic Programming (GP) [8] and Genetic Algorithms (GAs) [9][10], such as Large-scale evolution of image classifiers (LEIC) method [11] recently proposed by Google, which have shown that EC can be used in learning CNN architectures that are competitive with the state-of-art algorithms designed by humans. However, the learning process for large data is too slow due to the high computational cost for most of the methods and it might not be practical for industrial use.

A lot of work has been done in order to improve using EC to evolve a CNN architecture, such as the recent proposed EvoCNN using GAs [12]. One of the improvements in EvoCNN is that during the fitness evaluation, instead of training the model for 25,600 steps in LEIC, it only trains each individual by 10 epochs, which dramatically speeds up the learning process. The rationale behind EvoCNN using 10 epochs is that the researchers believe that training 10 epochs can obtain the major trend of the CNN architecture, which would be decisive to the final performance of a model, having been verified by their experiments.

However, not a lot of research has been done by using other EC methods to evolve the architectures of CNNs, so we would like to explore some other major EC methods for evolving the architectures of CNNs without any human interference. Since Particle Swarm Optimisation (PSO) has the advantages of easy implementation, lower computational cost, and fewer parameters to adjust, and it has never been utilised to evolve the architectures of CNNs, PSO is chosen in this paper. However, the fixed-length encoding of the particle in traditional PSO is a big challenge for evolving the architectures of CNN as the optimal CNN architecture varies for different tasks, so a novel flexible encoding scheme is proposed to break the fixed-length constraint, which would be the most fundamental part of the proposed algorithm in this paper.

A. Goal

The overall goal of this paper is to design and develop an effective and efficient PSO method to automatically discover good architectures of CNNs. The specific objectives of this paper are to

1) Design a new particle encoding scheme that has the ability of effectively encoding a CNN architecture, and develop a new PSO algorithm based on the novel encoding strategy.

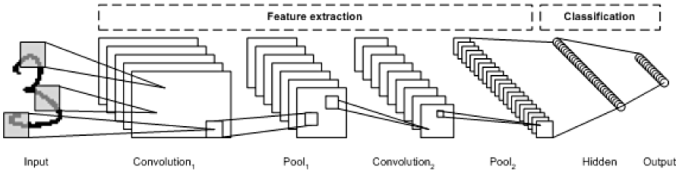


Fig. 1. An general architecture of the Convolutional Neural Network [13]

2) Design a method to break the constraint of the fixed-length encoding of traditional PSO in order to learn variable-length architectures of CNNs. We will introduce a new layer called Disabled layer to attain a variable-length particle.
3) Propose a fitness evaluation method using a partial dataset instead of the whole dataset to significantly speed up the evolutionary process.

II. Background

A. CNN architecture

Fig. 1 exhibits a general architecture of a CNN with two convolutional (Conv) layers, two pooling layers, and two Fully-connected layers - one hidden layer and one output layer at the end [13]. It is well-known that when designing a deep CNN architecture, the number of Conv layers, Pooling layers and Fully-connected layers before the last output layer have to be properly defined along with their positions and configurations. Different types of layers have different configurations as follows: Filter size, stride size and feature maps are the main attributes of the configuration for the Conv layer; Kernel size, stride size and pooling type - max-pooling or average-pooling, are the important parameters for the configuration of the Pooling layer; and the number of neurons is the key attribute of the Fully-connected layer.

B. Particle Swarm Optimisation

Particle Swarm Optimization (PSO) is a population-based algorithm, motivated by the social behaviour of fish schooling or bird flocking [14] [15], commonly used for solving optimization problems without rich domain knowledge [16]. In PSO, the population is composed of a certain number of particles each of which represents a solution, and particles fly in the search space to find the best solution by updating velocity and particle vector according to Equations (1) and (2), respectively, where v_{id} represents the velocity of the particle i in the dth dimension, x_{id} represents the position of particle i in the dth dimension, P_{id} and P_{gd} are the local best and the global best in the dth dimension, r_1, r_2 are random numbers between 0 and 1, w, c_1 and c_2 are PSO parameters used to tweak the performance.

$$v_{id}(t+1) = w * v_{id}(t) + c_1 * r_1 * (P_{id} - x_{id}(t)) + c_2 * r_2 * (P_{gd} - x_{id}(t)) \quad (1)$$

$$x_{id}(t+1) = x_{id}(t) + v_{id}(t+1) \quad (2)$$

C. Internet Protocol address

An Internet Protocol address (IP address) is a numerical label assigned to each device connected to a computer network that uses the Internet Protocol for communication [17]. In order to identify the network of an IP address, the subnet is introduced, which is often described in CIDR (Classless Inter-Domain Routing) style [18] by combining the starting IP address and the length of the subnet mask together. Both the IP address used for identifying the host and its corresponding subnet used for distinguishing different networks are carried by a network interface. For example, a standard IP address of a 32-bit number could be 192.168.1.251, and a standard subnet carries the starting IP address and the length of the subnet mask could be 192.168.1.0/8, which indicates the IP address in the subnet starts from 192.168.1.0 and the length of the subnet mask is 8 defining an IP range from 192.168.1.0 to 192.168.1.255.

Although the outlook of the IP address is a sequence of decimal numbers delimited by full stops, the binary string under the hood is actually used for the network identification, which inspires the new PSO encoding scheme. As there are several attributes in the configuration of each type of CNN layers, each of which is an integer value within a range, each value of the attribute can be smoothly converted to a binary string, and several binary strings, each of which represents the value of an attribute, can be concatenated to a large binary string to represent the whole configuration of a specific layer. It is obvious that the binary string suits the requirement of encoding CNN layers to particles. However, in this way, a huge number converted from the binary string will have to be used as one dimension of the particle vector, which may result in a horrendous searching time in PSO. On the other hand, in the IP structure, instead of utilising one huge integer to mark the identification (ID) of a device in a large network, in order to make the IP address readable and memorable, it divides a huge ID number into several decimal values less than 256, each of which is stored in one byte of the IP address. In this way, the binary string can be divided into several bytes, and each byte comprises one dimension of the particle vector. The convergence of PSO can be facilitated by splitting one dimension of a large number to several dimensions of small numbers because in each round of the particle updates, all of the dimensions can be concurrently learned and the search space of one split dimension is much smaller. In this paper, the new particle encoding scheme will use this idea to gain the flexibility of encoding various types of layers into a particle, and drastically cut down the learning process, which will be described in the next section.

III. The proposed algorithm

In this section, the new IP based PSO (IPPSO) method for evolving deep CNNs will be presented in detail.

A. Algorithm Overview

Algorithm 1 outlines the framework of the proposed algorithm. There are mainly three steps to initialise the population

Algorithm 1: Framework of IPPSO

$P \leftarrow$ Initialize the population with the proposed particle encoding strategy;
$P_{id} \leftarrow empty$;
$P_{gd} \leftarrow empty$;
while termination criterion is not satisfied **do**
  update velocity and position of each particle shown in Algorithm 3;
  evaluate the fitness value of each particle;
  update P_{id} and P_{gd};
end while

by using the particle encoding strategy which will be described in Section III-B, to update the position and velocity, and to check whether the termination criterion is met.

B. Particle Encoding Strategy

The IPPSO encoding strategy is inspired by how the Network IP address works. Although the CNN architecture is comprised of three types of layers - Convolutional Layer, Pooling Layer, and Fully-Connected Layer, and the encoded information of different types of layers varies in terms of both the number of parameters and the range in each parameter shown in Table I, a Network IP address with a fixed length of enough capacity can be designed to accommodate all the types of CNN layers, and then the Network IP can be divided into numerous subsets, each of which can be used to define a specific type of CNN layers.

First of all, the length of the binary string under the IP-based encoding scheme needs to be designed. With regard to Conv layers, firstly, there are three key parameters - filter size, number of feature maps and stride size listed in the column of Parameter in Table I, which are the fundamental factors affecting the performance of CNNs; Secondly, based on the size of benchmark datasets, the range of the parameters are set to [1,8], [1,128] and [1,4] for the aforementioned three parameters, respectively, shown in the column of Range in table I; Thirdly, taking a CNN architecture with the filter size of 2, number of feature maps of 7 and stride size of 2 as an example, the decimal values can be converted to the binary strings of 001, 000 1111 and 01, where the binary string converted from the decimal value is filled with 0s until the length reaches the corresponding number of bits, illustrated in the column of Example Value in Table I. Lastly, the total number of bits of 12 and the sample binary string of 001 000 1111 01 by concatenating the binary strings of the three parameters are displayed in the summary row of Conv layer in Table I. In terms of Pooling layers and Fully-connected layers, the total number of bits and the sample binary string can be obtained by following the same process of Conv layers, which are listed in the summary rows of Pooling and Fully-connected layers in Table I. As the largest number of bits to represent a layer is 12 as shown in Table I and the unit of an IP address is one byte - 8 bits, there will be 2 bytes required to accommodate the 12 bits IP address.

TABLE I
THE PARAMETERS OF DIFFERENT TYPES OF CNN LAYERS - CONVOLUTIONAL, POOLING, FULLY-CONNECTED AND DISABLED LAYER WITH AN EXAMPLE IN THE EXAMPLE COLUMN

Layer Type	Parameter	Range	# of Bits	Example Value
Conv	Filter size	[1,8]	3	2(001)
	# of feature maps	[1,128]	7	32(000 1111)
	Stride size	[1,4]	2	2(01)
	Summary		12	001 000 1111 01
Pooling	Kernel size	[1,4]	2	2(01)
	Stride size	[1,4]	2	2(01)
	Type: 1(maximal), 2(average)	[1,2]	1	2(1)
	Place holder	[1,128]	6	32(00 1111)
	Summary		11	01 01 0 00 1111
Fully-connected	# of Neurons	[1,2048]	11	1024(011 11111111)
	Summary		11	011 11111111
Disabled	Place holder	[1,2048]	11	1024(011 11111111)
	Summary		11	011 11111111

In addition, the subnets for all types of CNN layers need to be defined according to the number of bits of each layer illustrated in Table I and CIDR style will be used to represent the subnet. As there are three types of CNN layers, we need to define three subnets with enough capacity to represent all the types of layers. Starting with the Conv layer, 0.0 is designed as the starting IP address of the subnet; in addition, the total length of the designed 2-byte IP address is 16 and the total number of bits required by the Conv layer is 12, so the subnet mask length is 4 calculated by subtracting the total number of bits from the length of the IP address, which brings the subnet representation to 0.0/4 with the range from 0.0 to 15.255. Regarding the Pooling layer, the starting IP address is 16.0 obtained by adding 1 to the last IP address of the Conv layer, and the subnet mask length is 5 calculated in the same way as that of the Conv layer, which results in 16.0/5 with the range from 16.0 to 23.255 as the subnet representation of the Pooling layer. Similarly, the subnet 24.0/5 with the range from 24.0 to 31.255 is designed as the subnet of the Fully-connected layer. In order to make the subnets clear, all of the subnets are depicted in Table II.

As the particle length of PSO is fixed after initialisation, in order to cope with the variable-length of the architectures of CNNs, an effective way of disabling some of the layers in the encoded particle vector will be used to achieve this purpose. Therefore, another layer type called the Disabled layer and the corresponding subnet named the Disabled subnet are introduced. To achieve a comparable probability for the Disabled layer, the least total number of bits of 11 among all three types of CNN layers is set as the number of bits of the Disabled layer, so the disabled subnet comes to 32.0/5

TABLE II
FOUR SUBNETS DISTRIBUTED TO THE THREE TYPES OF CNN LAYERS AND THE DISABLED LAYER

Layer type	Subnet(CIDR)	IP Range
Convolutional Layer	0.0/4	0.0-15.255
Fully-Connected Layer	16.0/5	16.0-23.255
Pooling Layer	24.0/5	24.0-31.255
Disabled Layer	32.0/5	32.0-39.255

TABLE III
AN EXAMPLE OF IP ADDRESSES - ONE FOR EACH TYPE OF CNN LAYERS

Layer type	Binary (filled to 2 bytes)	IP address
Convolutional Layer	(0000)001 000 1111 01	2.61
Pooling Layer	(00000)01 01 0 00 1111	18.143
Fully-Connected Layer	(00000)011 11111111	27.255
Disabled Layer	(00000)01111111111	35.255

with the range from 32.0 to 39.255, shown in Table II, where each layer will be encoded into an IP address of 2 bytes. Table III shows how the example in Table I is encoded into IP addresses by combining all the binary string of each parameter of a specific layer into one binary string, filling the combined binary string with zeros until reaching the length of 2 bytes, applying the subnet mask on the binary string, and converting the final binary string to an IP address with one byte as a unit delimited by full stops. For instance, the sample binary string of the Conv layer in Table I is 001 000 1111 01, which is filled to 0000 001 000 1111 01 to reach the length of 2 bytes; then 2-byte binary string - 0000 0010 and 0011 1101, can be obtained by applying the subnet mask, in which the starting IP address of the subnet is added to the binary string; Finally, the IP address of 2.61 is achieved by converting the first byte to the decimal value of 2 and the second byte to 61.

After converting each layer into a 2-byte IP address, the position and velocity of PSO can be defined. However, there are a few parameters that need to be mentioned first - max_length(maximum number of CNN layers), max_fully_connected(maximum Fully-connected layers with the constraint of at least one Fully-connected layer) listed in Table IV. The encoded data type of the position and the velocity will be a byte array with a fixed length of maximum_length * 2 and each byte will be deemed as one dimension of the particle.

Here is an example of a particle vector to explain how the CNN architecture is encoded and how it copes with variable-length of CNN architecture. Assume max_length is 5, a sequence of IP addresses representing a CNN architecture with the maximum number of 5 layers can be encoded into 5 IP addresses in Fig. 2 by using the sample IP addresses in Table III, where C represents a Conv layer, P represents a Pooling layer, F represents a Fully-connected layer, and D represents a Disabled layer. The corresponding particle vector with the dimension of 10 is shown in Fig. 3. Since there is one Disabled layer in the example, the actual number of layers is 4.

2.61(C)	18.143(P)	2.61(C)	35.255(D)	27.255(F)

Fig. 2. An example of IP addresses in a particle containing 5 CNN layers

2	61	18	143	2	61	35	255	27	255

Fig. 3. An example of a particle vector with 5 CNN layers encoded

However, after a few PSO updates, the seventh dimension and the eighth dimension of the particle vector may become 18 and 143, respectively, which turns the third IP address representing a Disabled layers to a Pooling layer, so the updated particle carries a CNN architecture of 5 layers; Conversely, after a few updates, the fifth dimension and the sixth dimension of the particle vector may become 35 and 255, respectively, which makes the third IP address fall into the disabled subnet, so the actual number of layers is 3. To conclude, as shown in this example, the particle with IPPSO encoding scheme is capable of representing variable-length architectures of CNNs - 3, 4 and 5 in this example.

C. Population Initialisation

In terms of the population initialisation, after the size of the population is set up, individuals are randomly created until reaching the population size. For each individual, an empty vector is initialised first, and each element in it will be used to store a Network Interface containing the IP address and subnet information. The first element will always be a Conv layer; From the second to $(max\_length - max\_fully\_connected)$ layer, each element can be filled with a Conv layer, Pooling layer or Disabled layer; From $(max\_length - max\_fully\_connected)$ to $(max\_length - 1)$ layer, it can be filled with any of the four types of layers until the first Fully-connected is added, and after that only Fully-connected layers or Disabled layers are allowed; The last element will always be a Fully-connected layer with the size the same as the number of classes. In addition, each layer will be generated with the random settings - a random IP address in a valid subnet.

D. Fitness Evaluation

Before performing the fitness evaluation, a proper weight initialisation method has to be chosen, and Xavier weight initialisation [19] is chosen as it has been proved as an effective way, and has been implemented in most of Deep Learning frameworks. With regard to the fitness evaluation (shown in Algorithm 2), each individual is decoded to a CNN architecture with its settings, which will be trained for k epochs on the first part of the training dataset. Then the partially trained CNN will be batch-evaluated on the second part of the training dataset, which will produce a series of accuracies. Finally, we calculate the mean value of the accuracies for each individual, which will be stored as the individual fitness.

Algorithm 2: Fitness Evaluation

Input: The population P, the training epoch number k, the training set D_{train}, the fitness evaluation dataset $D_{fitness}$, the batch size $batch\_size$;
Output: The population with fitness P;
for individual s **in** P **do**
  $i \leftarrow 1$;
  while $i <= k$ **do**
    Train the connection weights of the CNN represented by individual s;
  end while
  $accy\_list \leftarrow$ Batch-evaluate the trained model on the dataset $D_{fitness}$ with the batch size $batch\_size$ and store the accuracy for each batch;
  $mean \leftarrow$ Calculate the mean value of acc_list
  $fitness \leftarrow mean$;
  $P \leftarrow$ Update the fitness of the individual ind in the population P;
end for
return P

Algorithm 3: Update Particle with Velocity Clamping

Input: particle individual vector ind, acceleration coefficient array for P_{id} c_1, acceleration coefficient array for P_{gd} c_2, inertia weight w, max velocity array v_{max};
Output: updated individual vector ind;
for element $interface$ **in** ind **do**
  $i \leftarrow 0$;
  for $i <$ number of bytes of IP address in $interface$ **do**
    $x \leftarrow$ the ith byte of the IP address in the $interface$;
    $(r_1, r_2)) \leftarrow$ uniformly generate r_1, r_2 between [0, 1];
    $v_{new} \leftarrow$ Update velocity based on Equation 3;
    $v_{new} \leftarrow$ Apply velocity clamping using v_{max};
    $x_{new} \leftarrow x + v_{new}$
    if $x_{new} > 255$ **then**
      $x_{new} \leftarrow x_{new} - 255$;
    end if
  end for
end for
$fitness \leftarrow$ evaluate the updated individual ind;
$(P_{id}, P_{gd}) \leftarrow$ Update $pbest$ and $gbest$ by comparing their $fitness$;
return ind

E. Update Particle with Velocity Clamping

In Algorithm 3, as each layer is encoded into an interface with 2 bytes in the particle vector, and we want to control the acceleration coefficients for each byte, the two acceleration coefficients implemented as two float arrays with the size of 2 are required shown in Equation 3. v and x are decimal values of the ith byte of the 2-byte IP address and its corresponding velocity, P_{id} and P_{gd} are decimal values of the ith byte of the IP address of the local best and global bet, respectively, and w, r_1, r_2 are the same as traditional PSO in Equation 1. The major difference is how the acceleration coefficients are implemented - $c_1[i]$ and $c_2[i]$ are the acceleration coefficients for the ith byte of the IP address, where i is 1 or 2 in the case of 2-byte IP encoding, comparing to a singular value for each of the acceleration coefficients in traditional PSO. The reason of separating the acceleration coefficients for each byte of the IP address is that different parameters may fall into different bytes of the IP address, and the ability to explore a specific parameter more than others may be needed when fine-tuning the learning process.

After the coefficients defined, we go through each byte in the particle and update the velocity and position by using the corresponding coefficients for that byte. Since there are some constraints for each interface in the particle vector according to its position in the particle vector, e.g. the second interface can only be a Conv layer, Pooling layer or Disabled layer, the new interface needs to be replaced by an interface with a random IP address in a valid subnet if the new interface does not fall in a valid subnet. After all the bytes being updated, the new particle is evaluated, and the fitness value is compared with the local best and global best in order to update the two bests if needed.

$$v_{new} = w * v + c_1[i] * r_1 * (P_{id} - x) + c_2[i] * r_2 * (P_{gd} - x) \quad (3)$$

F. Best Individual Selection and Decoding

The global best of PSO will be reported as the best individual. In terms of the decoding, a list of network interfaces - stored in every 2 bytes from left to right in the particle vector of the global best, can be extracted from a particle vector. According to the subnets in Table II the type of layer can be distinguished, and then based on Table I the IP address can be decoded into different sets of binary string, which indicate the parameter values of the layer. After decoding all the interfaces in the global best, the final CNN architecture can be attained by connecting all of the decoded layers in the same order as that of the interfaces in the particle vector.

IV. Experiment design

In this section, the benchmark datasets, peer competitors and parameter settings of the proposed IPPSO algorithm will be described.

A. Benchmark Datasets

In these experiments, three datasets are chosen from the widely used image classification benchmark datasets to examine the performance of the proposed IPPSO method. They are the MNIST Basic (MB) [20], the MNIST with Rotated Digits plus Background Images (MRDBI) [20] and the Convex Sets (CS) [20].

The first two benchmark datasets are two of the MNIST [5] variants for classifying 10 hand-written digits (i.e., 0-9). There are a couple of reasons for using MNIST variants instead of MNIST. Firstly, as the classification accuracy of MNIST

Fig. 4. Examples of the three datasets. From left to right, each two images as a group are from one benchmark, and each group is from MB, MRDBI, and CS, respectively

has achieved 97%, in order to challenge the algorithm, different noises (e.g., random backgrounds, rotations) are added into these MNIST variants from the MNIST to improve the complexity of the dataset. Secondly, there are 12,000 training images and 50,000 test images in these variants, which further challenges the classification algorithms due to the much less training data but more test data. The third benchmark dataset is for recognizing the shapes of objects (i.e., convex or not), which contains 8,000 training images and 50,000 test images. Since it is a two-class classification problem comparing to 10 classes of MNIST dataset, and the images contain shapes rather than digits, it is chosen as a supplement benchmark to the two MNIST variants in order to thoroughly test the performance of the proposed IPPSO method.

Each image in these benchmarks is with the size 28×28, and examples from these three datasets are displayed in Fig. 4. Another reason for choosing these three benchmark datasets is that different algorithms have reported their promising results, so it is convenient for comparing the performance of the proposed IPPSO method with these existing algorithms.

B. Peer Competitors

In the experiments, state-of-the-art algorithms, that have reported promising classification errors on the chosen benchmarks, are collected as the peer competitors of the proposed IPPSO method. To be specific, the peer competitors on the three benchmarks are CAE-2 [21], TIRBM [22], PGBM+DN1 [23], ScatNet-2 [25], RandNet-2 [24], PCANet-2 (softmax) [24], LDANet-2 [24], SVM+RBF [20], SVM+Poly [20], NNet [20], SAA-3 [20] and DBN-3 [20], which are from the literature [24] recently published and the provider of the benchmarks[1].

C. Parameter Settings

All the parameter settings are set based on the conventions in the communities of PSO [26] and deep learning [27] which are listed in Table IV.

The proposed IPPSO method is implemented in Tensorflow [28], and each copy of the code runs on a computer equipped with two GPU cards with the identical model number GTX1080. Due to the stochastic nature of the proposed IPPSO method, 30 independent runs are performed on each benchmark dataset, and the mean results are used for the comparisons unless otherwise specified. The experiments take two and a half hours approximately on each run of the benchmark dataset. The evolved CNN architectures on each benchmark are illustrated in Section V-B.

[1]http://www.iro.umontreal.ca/~lisa/twiki/bin/view.cgi/Public/DeepVsShallowComparisonICML2007

TABLE IV
PARAMETER LIST

Parameter Name	Parameter Meaning	Value
max_length	maximum length of CNN layers	9
max_fully_connected	maximum fully-connected layers given at least there is one fully-connected layer	3
N	population size	30
k	the training epoch number before evaluating the trained CNN	10
num_of_batch	the batch size for evaluating the CNN	200
c1	acceleration coefficient array for P_{id}	[1.49618,1.49618]
c2	acceleration coefficient array for P_{gd}	[1.49618,1.49618]
w	inertia weight for updating velocity	0.7298
v_{max}	maximum velocity	4,25.6(0.1*search space)

V. Results and Analysis

In this section, the classification performance along with the analysis against peer competitors, the CNN architectures learned by the proposed IPPSO method, and the related visualisation will be reported.

A. Overall performance

Experimental results on all the three benchmark datasets are shown in Table V where the last three rows denote the mean classification errors, the best classification errors and the standard deviations of the classification errors obtained by the proposed IPPSO method from the 30 runs, and the other rows show the best classification errors reported by peer competitors[2]. In order to conveniently investigate the comparisons, the terms "(+)" and "(-)" are provided to indicate whether the result generated by the proposed IPPSO method is better or worse than the best result obtained by the corresponding peer competitor. The term "–" means there is no available result reported from the provider or cannot be counted.

It is clearly shown in Table V that by comparing the mean classification errors of the proposed IPPSO method with the best performance of the peer competitors, IPPSO performs the second best on the MB dataset, which is only a little bit worse than LDANet-2. IPPSO is the best on the MDRBI dataset, which is the most complicated dataset among these three, and the fifth best on the CS dataset, which is not ideal but very competitive.

B. Evolved CNN Architectures

Although the proposed IPPSO method is performed on each benchmark with 30 independent runs, only one is chosen on

[2]It is a convention in the deep learning community that only the best result is reported.

TABLE V
THE CLASSIFICATION ERRORS OF THE PROPOSED IPPSO METHOD AGAINST THE PEER COMPETITORS ON THE MB, MDRBI AND CS BENCHMARK DATASETS

classifier	MB	MDRBI	CS
CAE-2	2.48(+)	45.23(+)	–
TIRBM	–	35.50(+)	–
PGBM+DN-1	–	36.76(+)	–
ScatNet-2	1.27(+)	50.48(+)	6.50(-)
RandNet-2	1.25(+)	43.69(+)	5.45(-)
PCANet-2 (softmax)	1.40(+)	35.86(+)	4.19(-)
LDANet-2	1.05(-)	38.54(+)	7.22(-)
SVM+RBF	3.03(+)	55.18(+)	19.13(+)
SVM+Poly	3.69(+)	56.41(+)	19.82(+)
NNet	4.69(+)	62.16(+)	32.25(+)
SAA-3	3.46(+)	51.93(+)	18.41(+)
DBN-3	3.11(+)	47.39(+)	18.63(+)
IPPSO(mean)	1.21	34.50	12.06
IPPSO(best)	1.13	33	8.48
IPPSO(standard deviation)	0.103	2.96	2.25

TABLE VI
AN EVOLVED ARCHITECTURE FOR THE MB BENCHMARK

Layer type	Configuration
conv	Filter size: 2, Stride size: 1, feature maps: 26
conv	Filter size: 6, Stride size: 3, feature maps: 82
conv	Filter size: 8, Stride size: 4, feature maps: 114
conv	Filter size: 7, Stride size: 4, feature maps: 107
full	Neurons: 1686
full	Neurons: 10

TABLE VII
AN EVOLVED ARCHITECTURE FOR THE MDRBI BENCHMARK

Layer type	Configuration
conv	Filter size: 2, Stride size: 1, feature maps: 32
conv	Filter size: 6, Stride size: 3, feature maps: 90
conv	Filter size: 7, Stride size: 4, feature maps: 101
conv	Filter size: 7, Stride size: 4, feature maps: 97
pool	Kernel size: 4, Stride size: 4, Type: Average
conv	Filter size: 5, Stride size: 3, feature maps: 68
full	Neurons: 1577
full	Neurons: 10

each benchmark for this description purpose shown from Table VI to Table VIII. Since Disabled layers have been removed during the decoding process, they do not show up in the learned CNN architectures. Therefore, it turns out that IPPSO is able to learn a variable-length CNN architecture, which can be obviously seen from the listed architectures - 6 CNN layers for the MB and CS benchmark and 8 CNN layers for the MDRBI benchmark.

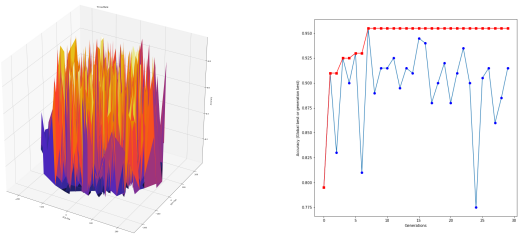
TABLE VIII
AN EVOLVED ARCHITECTURE FOR THE CS BENCHMARK

Layer type	Configuration
conv	Filter size: 1, Stride size: 1, feature maps: 11
conv	Filter size:7, Stride size: 4, feature maps: 108
conv	Filter size: 1, Stride size: 1, feature maps: 8
conv	Filter size: 6, Stride size: 3, feature maps: 92
full	Neurons: 906
full	Neurons: 2

C. Visualisation

In order to achieve a better understanding of the proposed IPPSO method, we visualise two parts of the evolutionary process - the accuracy distribution of the PSO vectors, where the architectures of CNNs are encoded, and the PSO trajectory of the evolving process.

In terms of the accuracy distribution, first of all, we obtained the PSO vectors and their corresponding accuracies from 10 runs of the experiments; in addition, the first two principal components from Principal Component Analysis (PCA) are extracted for the usage of visualisation. A 3-D triangulated surface with the data containing the first two components and the corresponding accuracy is plotted shown in Fig. 5a. It is observed that there are a lot of steep hills on the surface whose summits are at the similar level, so it means that there are quite a number of local optima, but most of them are very close to each other, which means that those local optima are acceptable as a good solution of the task.



(a) Surface (b) Trajectory

Fig. 5. (5a): The surface of CNN accuracies after training 10 epochs with IPPSO encoding; (5b): PSO Trajectory.

Regarding the trajectory, the best result of the particles of each generation and the global best in each generation from one run of the experiments are obtained and plotted in blue colour and red colour, respectively, in Fig. 5b. It can be seen that after only a few generations, the global best is found, after which the particles are still flying in the search space, but none of them can obtain a better accuracy, which means the optimum has been reached by PSO after only a few steps. Even though the surface of the optimisation task shown in Fig. 5a is extremely complicated, the PSO method with only 30 particles can climb up to the optimum very quickly, which proves the effectiveness and efficiency of PSO on optimisation tasks.

VI. Conclusions

The goal of this paper was to develop a new PSO approach with variable length to automatically evolve the architectures of CNNs for image classification problems. This goal has been successfully achieved by proposing a new encoding scheme of using a network interface containing an IP address and its corresponding subnet to carry the configurations of a CNN layers, the design of four subnets including a disabled subnet in order to simulate a variable-length PSO, and an efficient fitness evaluation method by using partial dataset. This approach was examined and compared with 12 peer competitors including the most state-of-the-art algorithms on three benchmark datasets commonly used in deep learning and the experimental results show that the proposed IPPSO method can achieve a very competitive accuracy by outperforming all others on the MDRBI benchmark dataset, being the second-best on the MNIST benchmark dataset and ranking above the middle line on the CS benchmark dataset.

The most important improvement from the traditional PSO to the novel IPPSO proposed in the paper is to invent the new encoding strategy of using network interface. Since the subnet in the interface can distinguish any type of layers, any layer configurations can be encoded into the IP address and the length of the IP address can be easily extended to 4 bytes (the length of real IP addresses) or even more, the IPPSO method has the ability of encoding any type of Deep Neural Network layers. In addition, the particle length of the IPPSO method can be easily made variable by simply introducing a disabled layer which could be deemed as another major improvement as it breaks the obstacle of traditional PSO being fixed-length.

In this paper, we have investigated the proposed IPPSO method for evolving deep CNN and it is proved of obtaining promising results. Based on this research, there are a couple of further researches that are worth doing. Firstly, as this paper is mainly to propose the novel encoding strategy of the IPPSO, it will be interesting to see how different PSO topologies[3] will affect the performance of the IPPSO and design the best topology for it. Secondly, we will also investigate how the proposed IPPSO algorithm performs for evolving recurrent neural networks, which are powerful tools for addressing sequential data tasks, such as language processing problems.

References

[1] Ossama Abdel-Hamid, Li Deng and Dong Yu, *Exploring Convolutional Neural Network Structures and Optimization Techniques for Speech Recognition*. INTERSPEECH 2013, 5 - 29 August 2013, Lyon, France

[2] Yoon Kim, *Convolutional Neural Networks for Sentence Classification*, arXiv:1408.5882v2 [cs.CL] 3 Sep 2014

[3] Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton, *ImageNet Classification with Deep Convolutional Neural Networks*, Advances in Neural Information Processing Systems 25 (NIPS 2012), 2012

[4] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, *Backpropagation applied to handwritten zip code recognition*, Neural Computation, vol. 1, no. 4, pp. 541–551, 1989.

[5] 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.

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

[7] 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.

[8] Masanori Suganuma, Shinichi Shirakawa, and Tomoharu Nagao, *A Genetic Programming Approach to Designing Convolutional Neural Network Architectures*, arXiv:1704.00764v2 [cs.NE] 11 Aug 2017

[9] K. O. Stanley and R. Miikkulainen, *Evolving neural networks through augmenting topologies*, Evolutionary computation, vol. 10, no. 2, pp. 99–127, 2002.

[10] Yanan Sun, Gary G. Yen, Zhang Yi, *"Evolving Unsupervised Deep Neural Networks for Learning Meaningful Representations"*. IEEE Transactions on Evolutionary Computation. DOI:10.1109/TEVC.2018.2808689.

[11] 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.

[12] Yanan Sun, Bing Xue, Mengjie Zhang, *Evolving Deep Convolutional Neural Networks for Image Classification[J]*. arXiv preprint arXiv:1710.10741, 2017.

[13] M. Tim Jones, *Deep learning architectures and The rise of artificial intelligence*. IBM DeveloperWorks, September 08, 2017.

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

[15] . 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.

[16] Yanan Sun, Bing Xue, Mengjie Zhang, *A Particle Swarm Optimization-based Flexible Convolutional Auto-Encoder for Image Classification[J]*. arXiv preprint arXiv:11712.05042, 2017.

[17] Postel, J., *DoD standard Internet Protocol*, RFC 760, DOI 10.17487/RFC0760, January 1980, ¡https://www.rfc-editor.org/info/rfc760¿.

[18] Fuller, V., Li, T., Yu, J., and K. Varadhan, *Classless Inter-Domain Routing (CIDR): an Address Assignment and Aggregation Strategy*, RFC 1519, DOI 10.17487/RFC1519, September 1993, ¡https://www.rfc-editor.org/info/rfc1519¿.

[19] 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.

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

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

[22] K. Sohn and H. Lee, *Learning invariant representations with local transformations*, arXiv preprint arXiv:1206.6418, 2012.

[23] K. Sohn, G. Zhou, C. Lee, and H. Lee, *Learning and selecting features jointly with point-wise gated boltzmann machines*, in International Conference on Machine Learning, 2013, pp. 217–225.

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

[25] J. Bruna and S. Mallat, *Invariant scattering convolution networks*, IEEE transactions on Pattern Analysis and Machine Intelligence, vol. 35, no. 8, pp. 1872–1886, 2013.

[26] F. van den Bergh, A.P. Engelbrecht, *A study of particle swarm optimization particle trajectories*, doi:10.1016/j.ins.2005.02.003

[27] G. E. Hinton, *A practical guide to training restricted boltzmann machines*, in Neural Networks: Tricks of the Trade. Springer, 2012, pp. 599–619.

[28] 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.

[3]Fully-connected topology is used in this paper as it is easy to be implemented.