Evolving Deep Convolutional Neural Networks for Image Classification

Yanan Sun, *Member, IEEE*, Bing Xue, *Member, IEEE*, Mengjie Zhang, *Fellow, IEEE*, and Gary G. Yen, *Fellow, IEEE*

***Abstract*—Evolutionary paradigms have been successfully applied to neural network designs for two decades. Unfortunately, these methods cannot scale well to the modern deep neural networks due to the complicated architectures and large quantities of connection weights. In this paper, we propose a new method using genetic algorithms for evolving the architectures and connection weight initialization values of a deep convolutional neural network to address image classification problems. In the proposed algorithm, an efficient variable-length gene encoding strategy is designed to represent the different building blocks and the potentially optimal depth in convolutional neural networks. In addition, a new representation scheme is developed for effectively initializing connection weights of deep convolutional neural networks, which is expected to avoid networks getting stuck into local minimum that is typically a major issue in the backward gradient-based optimization. Furthermore, a novel fitness evaluation method is proposed to speed up the heuristic search with substantially less computational resource. The proposed algorithm is examined and compared with 22 existing algorithms on nine widely used image classification tasks, including the state-of-the-art methods. The experimental results demonstrate the remarkable superiority of the proposed algorithm over the state-of-the-art designs in terms of classification error rate and the number of parameters (weights).**

***Index Terms*—Convolutional neural network (CNN), deep learning, genetic algorithms (GAs), image classification.**

I. Introduction

CONVOLUTIONAL neural networks (CNNs) have demonstrated their exceptional superiority in visual recognition tasks, such as traffic sign recognition [1], biological image segmentation [2], and image classification [3]. In addition, CNNs have also improved the development of other machine learning approaches for object detection tasks, such as the transfer learning based on CNNs [4]–[6]. CNNs are originally motivated by the computational model of the cat visual cortex specializing in processing vision and signal related tasks [7]. Since LetNet-5 was proposed in 1989 [8], [9], which is an implementation of CNNs and whose connection weights are optimized by the back-propagation (BP) algorithm [10], various variants of CNNs have been proposed, such as VGGNet [11] and ResNet [12]. These variants significantly improve the classification accuracies compared to the dominated algorithms on image classification tasks. Diverse variants of CNNs differ in their architectures and weight connections.

出色的优势

Mathematically, the overall process of training and recall of a CNN can be formulated as (1) with the data (X, Y)

$$\begin{cases} A &= \mathcal{F}(X, Y) \\ W &= \mathcal{G}(A) \\ Z &= \mathcal{H}(X, W) \end{cases} \tag{1}$$

where X and Y are the input data and the corresponding label, respectively, $\mathcal{F}(\cdot)$ denotes the architecture choosing function for the given data, A is the resulted architecture, $\mathcal{G}(\cdot)$ refers to the initialization method of the connection weights W based on the chosen architecture. $\mathcal{H}$ denotes the sequential operations in the CNN, such as convolutional operations, pooling operations, and nonlinear activation functions. Z denotes the resulted features learned by this CNN based on the input data X and the weights W. In the context of a classification task, a classifier is added to the tail of the CNN by receiving Z. The objective function of training the CNN is determined by the particular classifier based on Y. Typically, the gradient descend (GD)-based approaches, e.g., stochastic GD (SGD), are utilized to optimize the objective function within the given number of epochs, where the connection weight values are iteratively updated. A CNN commonly has a huge number of connection weights. However, $\mathcal{F}(\cdot)$ and $\mathcal{G}(\cdot)$ are countable functions that are discrete and neither convex or concave, and they cannot be well addressed by accurate methods in practice. Furthermore, because the gradient-based optimizers are heavily dependent on the initial values of the weights (including biases), it is essential to choose a suitable $\mathcal{G}(\cdot)$ that can help the employed gradient-based optimizers to escape from local minima. Furthermore, the performance of assigned architectures cannot be evaluated until the optimization of the objective

Manuscript received March 8, 2018; revised September 5, 2018, February 1, 2019, and April 10, 2019; accepted May 6, 2019. Date of publication May 10, 2019; date of current version March 31, 2020. This work was supported in part by the Marsden Fund of New Zealand Government under Contract VUW1209, Contract VUW1509, and Contract VUW1615, in part by the Huawei Industry Fund under Grant E2880/3663, in part by the University Research Fund at Victoria University of Wellington under Grant 209862/3580 and Grant 213150/3662, in part by the National Natural Science Foundation of China for Distinguished Young Scholar under Grant 61625204, and in part by the National Natural Science Foundation of China under Grant 61803277. *(Corresponding author: Gary G. Yen.)*

Y. Sun is with the College of Computer Science, Sichuan University, Chengdu 610065, China, and also with the School of Engineering and Computer Science, Victoria University of Wellington, Wellington 6140, New Zealand.

B. Xue and M. Zhang are with the School of Engineering and Computer Science, Victoria University of Wellington, Wellington 6140, New Zealand (e-mail: bing.xue@ecs.vuw.ac.nz; mengjie.zhang@ecs.vuw.ac.nz).

G. G. Yen is with the School of Electrical and Computer Engineering, Oklahoma State University, Stillwater, OK 74078 USA (e-mail: gyen@okstate.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TEVC.2019.2916183

function is finished, while the optimization involves a progress of multiple iterations, which in turn increases the difficulty of choosing the potential optimal $\mathcal{F}(\cdot)$. Therefore, the architecture design and connection weight initialization strategy should be carefully treated in CNNs.

Typically, most of the architecture design approaches were initially developed for the deep learning algorithms in the early days (e.g., the stacked auto-encoders (SAEs) [13], [14] and the deep belief networks (DBNs) [15]), such as grid search (GS), random search (RS) [16], Bayesian-based Gaussian process (BGP) [17], [18], tree-structured Parzen estimators (TPE) [19], and sequential model-based global optimization (SMBO) [20]. Theoretically, GS exhaustively tests all combinations of the parameters, where the best one is seized. However, GS cannot evaluate all combinations within an acceptable time in reality. Moreover, optimizing continuous-valued parameters is difficult for GS [16]. RS could reduce the cost of GS, but the absolute "random" severely challenges the sampling behavior in the search space [21]–[23]. In addition, BGP incurs extra parameters (i.e., the kernels) that are 艰巨 arduous to tune. TPE treats each parameter independently, while the most key parameters in CNNs are dependent (e.g., the convolutional layer size and its stride). More details can be seen in Section II-A. The methods mentioned above have shown their good performance in most SAEs and DBNs, but are not suitable to CNNs. Their success in SAEs and DBNs is largely due to the architecture construction approaches, which are greedy layer-wised by stacking a group of building blocks with the same structures (i.e., the three-layer neural networks). In each building block, these architecture-search methods are utilized for only optimizing the parameters, such as the number of neurons in the corresponding layer. However, in CNNs, the layer-wised method cannot be applied due to their architecture characteristics of nonstacking routine, and we must determine the entire architectures at a time. Furthermore, multiple different building blocks exist in CNNs, and different orders of them would result in significantly different performance. Therefore, the architecture design in CNNs should be carefully treated. Recently, Saxena and Verbeek [24] proposed a neural fabrics approach named convolutional neural fabrics (CNFs) to design the architectures of CNNs by using GD-based optimizers. However, the major limitations of CNF are the extremely high computational complexity, extra hyper-parameters for guaranteeing the optimal performance of the adopted GD-based optimizer and manual interventions required in the design. Baker *et al.* [25] proposed an architecture design approach for CNNs based on reinforcement learning, named MetaQNN, which employed ten graphic processing unit (GPU) cards for 8–10 days given the CIFAR-10 dataset.

Due to the deficiencies in the existing methods and limited computational resources available to interested researchers, most of these works in CNNs are typically performed by experts with rich domain knowledge [16]. Genetic algorithms (GAs), which are 范例 a paradigm of the evolutionary algorithms that do not require rich domain knowledge [26], [27], adapt the meta-heuristic pattern motivated by the process of natural selection [28] for optimization searches. GAs are preferred in various fields due to their characteristics of gradient-free and insensitivity to local minima [29]. These promising properties are collectively achieved by an iterative process of the selection, mutation, and crossover operators. Therefore, it can be naturally utilized for the optimization of architecture design and the connection weight initialization for CNNs. However, it is not a trivial task. Until very recently in 2017, Google demonstrated their large evolution for image classification (LEIC) method specializing at the architecture optimization of CNNs [30]. LEIC was materialized by GAs without the crossover operator, implemented on 250 high-end computers, and achieved a competitive performance against the state-of-the-art on the CIFAR-10 dataset by training for about 20 days. In addition, Xie and Yuille [31] proposed the genetic CNN method by using a standard GA to evolve CNNs, where the individuals employed the same length of chromosomes, and a full training process on each individual for the fitness evaluation. Actually, by directly using GAs for the architecture design of CNNs, several issues naturally arise.

1) The performance of the architecture encoded by an individual is unknown until it has been evaluated on the corresponding data. However, evaluating the performance of one individual takes a long time, and it becomes even more severely in the population-based algorithms, where multiple individuals exist. This would require much more computational resources for speeding up the evolution.
2) The optimal depth of CNNs for one particular problem is unknown, therefore it is hard to constrain the search space for the architecture optimization. In this regard, a variable-length gene encoding strategy may be the best choice for both 1) and 2), but how to assign the crossover operation for different building blocks is a newly resulted problem.
3) The weight initialization values heavily affect the performance of the resulted architecture, but addressing this problem involves a good gene encoding strategy and the optimization of hundreds and thousands decision variables.

The aim of this paper is to design an effective and efficient GA method to automatically discover good architectures and corresponding connection weight initialization values of CNNs [i.e., the first two formulas in (1)] without manual intervention. To achieve this goal, the objectives below have been specified.

1) Design a flexible gene encoding scheme of the architecture, which does not constrain the maximal length of the building blocks in CNNs. With this gene encoding scheme, the evolved architecture is expected to benefit CNNs to achieve good performance in solving different tasks at hand.
2) Investigate the connection weight encoding strategy, which is capable of representing tremendous numbers of the connection weights in an efficient way. With this encoding approach, the weight connection initialization problem in CNNs is expected to be effectively optimized by the proposed GA.
3) Develop associated selection (including the environmental selection), crossover, and mutation operators that can

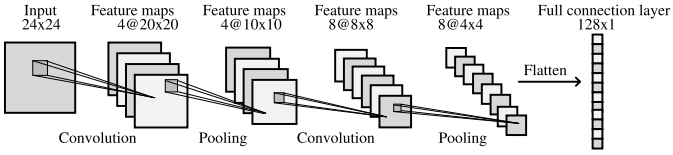

Fig. 1. General architecture of a convolutional neural network.

cope with the designed gene encoding strategies of both architectures and connection weights.

4) Propose an effective fitness measure of the individuals representing different CNNs, which does not require intensive computational resources.
5) Investigate whether the new approach significantly outperform the existing methods in both classification accuracy and number of weights.

The remainder of this paper is organized as follows. The background of the CNNs, the related works on the architecture design and weight initialization approaches of CNNs are reviewed in Section II. The framework and the details of each step in the proposed algorithm are elaborated in Section III. The experimental design and results of the proposed algorithm are shown in Sections IV and V, respectively. Finally, the conclusions and future work are detailed in Section VI.

II. Background and Related Work

A. Architecture of Convolutional Neural Networks

1) Skeleton of CNNs: Fig. 1 exhibits an extensive (广泛的) architecture of one CNN, where there are two convolutional operations, two pooling operations, the resulted four groups of feature maps, and the fully connected layer at the tail. The last layer, which is a fully connected layer, receives the input data by flattening all elements of the fourth group of feature maps. Generally, the convolutional layers and the pooling layers can be mixed to stack together at the top of the architecture, while the fully connected layers are constantly stacked with each other at the tail of the architecture. The numbers in Fig. 1 refer to the sizes of the corresponding layer. Particularly, the input data is with 24×24, the output is with 128×1, and the other numbers denote the feature map configurations. For example, $4@20 \times 20$ implies there are four feature maps, each with the size of 20×20.

2) Convolution: Given an input image with the size of $n \times n$, in order to receive a feature map generated by the convolutional operations, a filter must be defined in advance. Actually, a filter (it can also be simply seen as a matrix) is randomly initialized with a predefined size (i.e., the filter width and the filter height). Then, this filter travels from the leftmost to the rightmost of the input data with the step size equal to a stride (i.e., the stride width), and then travels again after moving downward with the step size equal to a stride (i.e., the stride height), until reaches the bottom-right of the input image. Depending on whether to pad zeros to the input images or not, the convolutional operations are categorized into two types: 1) the VALID (without padding) and 2) the SAME (with padding). Each element in the feature map is the sum of the products of each element from the filter and the corresponding element this filter overlaps. If the input data is with multiple channels, say three, one feature map will also require three different filters, and each filter convolves on each channel, then the results are summed element-wised. The involved parameters in one convolutional operation include the *filter width*, the *filter height*, the *number of feature maps*, the *stride width*, the *stride height*, the *convolutional type*, and the *connection weight* in the filter.

3) Pooling: Intuitively, the pooling operation resembles the convolution operation except for the element-wised product and the resulted values of the corresponding feature map. Briefly, the pooling operation employs a predefined window (i.e., the kernel) to collect the average value or the maximal value of the elements, where it slides, and the slide size is also called "stride" as in the convolutional operation. In the pooling operation, the involved parameters are the *kernel width*, the *kernel height*, the *stride width*, the *stride height*, and the *pooling type*.

B. Connection Weight Initialization

Typically, the initialization methods are classified into three different categories. The first employs constant numbers to initialize all connection weights, such as the zero initializer, one initializer, and other fixed value initializer. The second is the distribution initializer, such as using the Gaussian distribution or uniform distribution to initialize the weights. The third covers the initialization approaches with some prior knowledge, and the famous Xavier initializer [32] belongs to this category. Because of the numerous connection weights in CNNs, it is not necessary that all the connection weights start with the same values, which is the known major deficiency of the initialization methods in the first category. In the second initialization approaches, the deficiency of the first design has been solved, but the major difficulty exists in choosing the parameter of the distribution, such as the range of the uniform distribution, and the mean value as well as the standard derivation of the Gaussian distribution. To solve this problem, the Xavier initializer presents a range for uniform sampling based on the neuron saturation prior using the sigmoid activation function [33]. A couple of major issues remain: 1) if the optimal architectures of the networks are not found, the resulted initialized parameters perform badly as well and 2) the Xavier initializer is presented on the usage of the sigmoid activation, while the widely used activation function in CNNs is the ReLU [3], [12], [34], [35]. To the best of our knowledge, there has not been any evolutionary algorithm developed to search for the connection weight initialization of deep learning algorithms, including CNNs. In the proposed algorithm, we achieve this objective by using GA to evolve the proper mean and standard derivation for the Gaussian distribution.

III. Proposed Algorithm

In this section, the proposed evolving deep convolutional neural networks (EvoCNNs) for image classification is documented in detail.

Algorithm 1: Framework of EvoCNN

1 $P_0 \leftarrow$Initialize the population with ***the proposed gene encoding strategy***;
2 $t \leftarrow 0$;
3 **while** *termination criterion is not satisfied* **do**
4 Evaluate the fitness of individuals in P_t by using ***the proposed fitness evaluation approach***;
5 $S \leftarrow$ Select parent solutions with ***the developed slack binary tournament selection***;
6 $Q_t \leftarrow$ Generate offsprings with ***the designed genetic operators*** from S;
7 $P_{t+1} \leftarrow$Environmental selection from $P_t \cup Q_t$ by using ***the proposed strategy***;
8 $t \leftarrow t + 1$;
9 **end**
10 Select the best individual from P_t and decode it to the corresponding convolutional neural network.

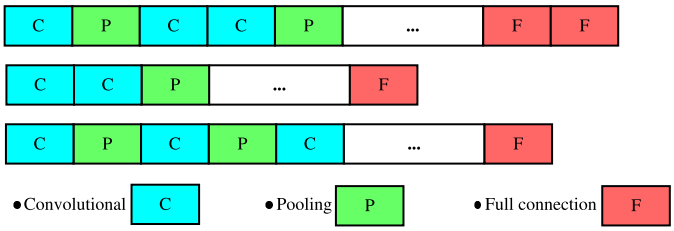

Fig. 2. Example of three chromosomes with different lengths in EvoCNN.

A. Algorithm Overview

Algorithm 1 outlines the framework of the proposed EvoCNN method, where our contributions in this paper are highlighted in bold and italic. First, the population is initialized based on the proposed flexible gene encoding strategy (line 1). Then, the evolution begins to take effect until a predefined termination criterion, such as the maximum number of the generations, has been satisfied (lines 3–9). Finally, the best individual is selected and decoded to the corresponding CNN (line 10) for final training.

During the evolution, all individuals are evaluated first based on the proposed efficient fitness measurement (line 4). After that, parent solutions are selected by our developed slack binary tournament selection (line 5), and new offspring are generated with the designed genetic operators (line 6). Next, representatives are selected from the existing individuals and the newly generated offspring to form the population in the next generation to participate subsequent evolution (line 7). In the following sections, the key steps in Algorithm 1 are narrated in detail.

B. Gene Encoding Strategy

As introduced in Section II-A, three different building blocks, i.e., the *convolutional* layer, the *pooling* layer, and the *fully connected* layer, exist in the architectures of CNNs. Therefore, they should be encoded into one chromosome for evolution. Because the optimal depth of a CNN in solving one particular problem is unknown prior to confirming its architecture, the variable-length gene encoding strategy, which is very suitable for this occasion, is employed in the proposed EvoCNN method. Furthermore, because the performance of CNNs is greatly affected by their depths [11], [36]–[38], this variable-length gene encoding strategy makes the proposed EvoCNN method to have better chances to reach the best result due to no constraint imposed upon the architecture search space.

TABLE I
ENCODED INFORMATION IN EVOCNN

Unit Type	Encoded Information
convolutional layer	the filter width, the filter height, the number of feature maps, the stride width, the stride height, the convolutional type, the standard deviation and the mean value of filter elements
pooling layer	the kernel width, the kernel height, the stride width, the stride height, and the pooling type (i.e., the average or the maximal)
fully connected layer	the number of neurons, the standard deviation of connection weights, and the mean value of connection weights

In particular, an example of three chromosomes with different lengths from EvoCNN is illustrated in Fig. 2, and all represented information in these three layers are detailed in Table I. Commonly, hundreds of thousands of connection weights may exist in one convolutional or fully connected layer, which cannot be all explicitly represented by a chromosome and effectively optimized by GAs. Therefore, in EvoCNN, we use only two statistical measures in real numbers (i.e., the standard derivation and mean value of the connection weights) to represent the numerous weight parameters, which can be easily implemented by GAs. When the optimal mean value and the standard derivation are received, the connection weights are sampled from the corresponding Gaussian distribution. The details of population initialization in EvoCNN are given in the next section based on the gene encoding strategy introduced above.

C. Population Initialization

For the convenience of the discussions, each chromosome is separated into two parts. The first part includes the convolutional layers and the pooling layers, and the other part is the fully connected layers. Based on the convention of the CNN architectures, the first part starts with one convolutional layer. The second part can only be added at the tail of the first part. In addition, the length of each part is set by randomly choosing a number within a predefined range.

Algorithm 2 shows the major steps of the population initialization, where $|\cdot|$ is a cardinality operator, lines 3–13 show the generation of the first part of a given chromosome, and lines 14–19 show that of the second part. During the initialization of the first part, a convolutional layer is added first. Then, a convolutional layer or a pooling layer is determined by the once coin tossing probability and then added to the end, which is repeated until the predefined length of this part is reached. For the second part, fully connected layers are chosen and then added. Note here that, convolutional layers, pooling layers, and

Algorithm 2: Population Initialization

Input: population size N, the maximal number of convolutional and pooling layers N_{cp}, and the maximal number of fully connected layers N_f.
Output: Initialized population P_0.

1 $P_0 \leftarrow \emptyset$;
2 **while** $|P_0| \leq N$ **do**
3 $part_1 \leftarrow \emptyset$;
4 $n_{cp} \leftarrow$ Uniformaly generate an integer between $[1, N_{cp}]$;
5 **while** $|part_1| \leq n_{cp}$ **do**
6 $r \leftarrow$ Uniformly generated a number between $[0, 1]$;
7 **if** $r \leq 0.5$ **then**
8 $l \leftarrow$ Initialize a convolutional layer with random settings;
9 **else**
10 $l \leftarrow$ Initialize a pooling layer with random settings;
11 **end**
12 $part_1 \leftarrow part_1 \cup l$;
13 **end**
14 $part_2 \leftarrow \emptyset$;
15 $n_f \leftarrow$ Uniformaly generate an integer between $[1, N_f]$;
16 **while** $|part_2| \leq n_f$ **do**
17 $l \leftarrow$ Initialize a fully connected layer with random settings;
18 $part_2 \leftarrow part_2 \cup l$;
19 **end**
20 $P_0 \leftarrow P_0 \cup (part_1 \cup part_2)$;
21 **end**
22 **Return** P_0.

Algorithm 3: Fitness Evaluation

Input: The population P_t, the training epoch number T for measuring the accuracy tendency, the training set D_{train}, the validation dataset D_{valid}, the batch size *num_of_batch*, the learning rate γ, and the classifer
Output: The population with fitness P_t.

1 **for** *each individual s in* P_t **do**
2 $train\_steps \leftarrow |D_{train}|/num\_of\_batch$;
3 $eval\_steps \leftarrow |D_{valid}|/num\_of\_batch$;
4 Decode the CNN represented by individual s;
5 $W = \{w_1, w_2, \cdots\} \leftarrow$ The weights in the CNN;
6 Initialize $w_1, w_2, \cdots$ based on the Gaussian distribution using the corresponding mean and standard drivation encoded in s;
7 **for** $i \leftarrow 1$ *to* $T - 1$ **do**
8 **for** $j \leftarrow 1$ *to train_steps* **do**
9 $\mathcal{J}(W, D_{train,j}) \leftarrow$ Calculate the objective function of the classifier based on W and the j-th batch data $D_{train,j}$;
10 $W \leftarrow W - \gamma \cdot \nabla_W \mathcal{J}(W, D_{train,j})$;
11 **end**
12 **end**
13 $accy\_list \leftarrow \emptyset$;
14 **for** $j \leftarrow 1$ *to eval_steps* **do**
15 $accy_j \leftarrow$ Evaluate the classification error on the j-th batch data $D_{valid,j}$ based on the classifier;
16 $accy\_list \leftarrow accy\_list \cup accy_j$;
17 **end**
18 $N \leftarrow$ Calculate the number of parameters in s;
19 $\mu \leftarrow \sum_{j=1}^{eval\_steps} accy_j/eval\_steps$;
20 $std \leftarrow \sqrt{\sum_{j=1}^{eval\_steps}(accy_j - \mu)^2/eval\_steps}$;
21 assign N, μ, std to individual s, and update s from P_t;
22 **end**
23 **Return** P_t.

fully connected layers are initialized with the random settings, i.e., the information encoded into them are randomly specified before they are stored into the corresponding part. After finishing these two parts, they are combined and returned as one chromosome. With the same approach, a population of individuals are generated. Noting that the initialized individuals in Algorithm 2 are with the mixed and not minimal lengths that multiple previous works did [30], [39]. The motivations behind this design are: 1) CNNs represented by the individuals with the minimal lengths cannot address a majority of the large datasets, including the benchmark datasets used in this paper; 2) it will take unnecessary time to evolve the individuals from the minimal lengths to longer lengths; and 3) our designed mutation operator (detailed in Section III-F) has the functions to shorten the lengths of the individuals accordingly, if required.

D. Fitness Evaluation

Fitness evaluation aims at giving a quantitative measure determining which individuals qualify for serving as parent solutions. Algorithm 3 manifests the framework of the fitness evaluation in EvoCNN. Specifically, the fitness evaluation of one individual is composed of two parts. The first is to train the CNNs represented by the individuals in the population (lines 4–12), and the second is to calculate their fitness (lines 13–20). During the training, first, each CNN is decoded based on the information represented by the individual (line 4). Second, the weights regarding the convolutional layers and the fully connected layers are initialized based on the Gaussian distribution by using the corresponding mean and standard derivation encoded in the individual (line 6). Third, the CNN is trained with the specified maximal epochs with the batch data in the training set (lines 7–12). In each training step, the objective function in term of the employed classifier is measured based on the current weights and the batch data (line 9), and then the weights are updated by subtracting the product of the learning rate and the gradient (line 10). Note that the process of updating the weights follows the routine of a standard batch SGD, which is commonly used in training deep neural networks. During the calculation of the fitness, the classification error is computed based on the trained weights and the

batch data in the validation dataset (lines 14–17), i.e., each batch data has its own classification error. When all the batch data in the validation dataset has been evaluated, the mean and standard derivations regarding the classification errors of these batch data are calculated. Specifically, lines 19 and 20 mathematically show the calculation of the mean and the standard derivation, respectively. In addition, we also calculate the number of weights (line 18). These three elements collectively are considered as the fitness of the individual in the proposed algorithm.

Because EvoCNN is concerned with solving image classification tasks, the classification error is the best metric to assign their fitness. The number of connection weights is also chosen as an additional indicator to measure the individual's quality based on the principle of Occam's razor [40]. With the conventions, each represented CNN is trained on the training dataset D_{train}, and the fitness is estimated on the validation dataset D_{valid}. Noting that, the test dataset that is disjoint from D_{train} and D_{valid} will not be used during this phase. The reason is that this phase is only for selecting the CNN from multiple candidates not for evaluating the optimal classification accuracy on the task. In addition, we do not need to care about the overfitting problem regarding D_{valid} because we do not use D_{valid} to train the CNNs (we use D_{train} to train the CNNs), we only use D_{valid} to evaluate the fitness of each individual.

CNNs frequently have deep architectures, thus thoroughly training them for receiving the final classification error would take considerable expenditure of computing resource and a very long time due to the large number of training epochs required (>100 epochs are invariably treated in fully training CNNs). This will make it much more impracticable here due to the population-based GAs with multiple generations (i.e., each individual will take a full training in each generation). We have designed an efficient method to address this concern. In this method, each individual is trained with only a small number of epochs, based on their architectures and connection weight initialization values, and then the mean value and the standard derivation of classification error are calculated on each batch of D_{valid} in the last epoch. Both the mean value and the standard derivation of classification errors, are employed as the fitness of one individual. Obviously, the smaller mean value, the better individual it would be regarded. When the compared individuals are with the same mean values, the less standard derivation indicates the better one.

In summary, three indicators are used in the fitness evaluation, which are the mean value, standard derivation, and the number of parameters. There are several motivations behind this fitness evaluation strategy.

1) It is sufficient to investigate only the tendency of the performance. If individuals are with better performance in the first several training epochs of CNNs, they will probably have the better performance in the following training epochs with a greater confidence.
2) The mean value and the standard derivation are statistical significance indicators, thus suitable for investigating this tendency, and the final classification error can be received by optimizing only the best individual evolved by the proposed EvoCNN method.
3) CNN models with less number of connection weights are preferred by manufactures of smart devices (more details are discussed in Section V-C).

Algorithm 4: Slack Binary Tournament Selection

Input: Two compared individuals, the mean value threshold α, and the parameter number threshold β.
Output: The selected individual.

```
1  s1 ← The individual with larger mean value;
2  s2 ← The other individual;
3  μ1, μ2 ← The mean values of s1, s2;
4  std1, std2 ← The standard derivations of s1, s2;
5  c1, c2 ← The parameter numbers of s1, s2;
6  if μ1 − μ2 > α then
7  |  Return s1.
8  else
9  |  if c1 − c2 > β then
10 |  |  Return s2.
11 |  else
12 |  |  if std1 < std2 then
13 |  |  |  Return s1.
14 |  |  else if std1 > std2 then
15 |  |  |  Return s2.
16 |  |  else
17 |  |  |  Return randon one from {s1, s2}.
18 |  |  end
19 |  end
20 end
```

E. Slack Binary Tournament Selection

We develop one slack version of the standard binary tournament selection, which are documented in Algorithm 4, to select parent solutions for the crossover operations in the proposed EvoCNN method. Briefly, two sets of comparisons are employed. The comparisons between the mean values of individuals involves a threshold α, and that comparisons between the parameter numbers involves another threshold β. If the parent solution cannot be selected with these comparisons, the individual with smaller standard derivation is chosen.

In practice, tremendous number of parameters exist in deep CNNs, which would easily cause the overfitting problem. Therefore, when the difference between the mean values of two individuals is smaller than the threshold α, we further consider the number of connection weights. The slight change of the parameter numbers will not highly affect the performance of CNNs. Consequently, β is also introduced.

By iteratively performing this selection, parent solutions are selected and stored into a mating pool. In the proposed EvoCNN method, the size of the mating pool is set to be equal to the population size.

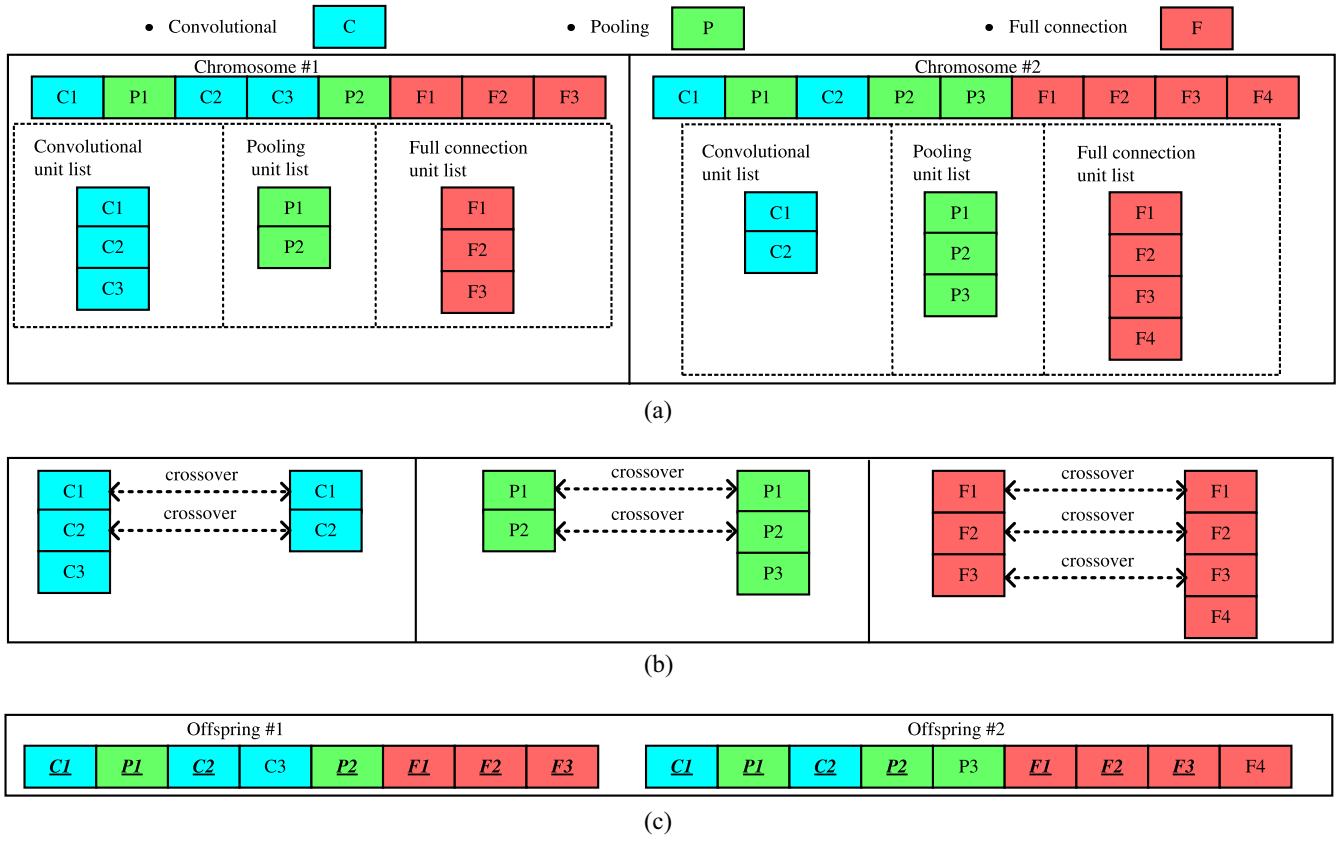

Fig. 3. Example to illustrate the entire crossover process. In this example, the first chromosome is with length 8, including three convolutional layers, two pooling layers, and three fully connected layers; the other one is with length 9, including two convolutional layers, three pooling layers, and four fully connected layers. In the first step of crossover (i.e., the unit collection), the convolutional layers, the pooling layers, and the fully connected layers are collected from each chromosome and stacked with the same orders to them in each chromosome (a). In the second step, the unit lists with the same unit types are aligned at the top, i.e., the two convolutional layer lists are picked and aligned, and the same operations on the other two lists. When these unit lists finish the alignment, units at the same positions from the each two lists are paired and performed crossover operation, which can be shown in (b). At last, units from these unit lists are restored based on the positions, where they are from (b). Note in (b), the units that have experienced crossover operations are highlighted with italics and underline fonts, while the units that do not perform the crossover operations remain the same.

F. Offspring Generation

In GA, offspring are generated by the crossover and mutation operators. The crossover operator performs the local search, while the mutation exercises the global search. Both of them collectively give rise to the promising performance of GAs. In the proposed algorithm, the steps for generating offspring are given as follows.

Step 1: Randomly select two parent solutions from the mating pool.
Step 2: Use crossover operator on the selected solutions to generate offspring.
Step 3: Use mutation operator on the generated offspring.
Step 4: Store offspring, remove the parent solutions from the mating pool, and perform steps 1–3 until the mating pool is empty.

The proposed crossover operation can be seen in Fig. 3. To achieve crossover, we design a method called unit alignment (UA) for recombining two individuals with different chromosome lengths. During the crossover operation, three different units, i.e., the convolutional layer, the pooling layer, and the fully connected layer, are first collected into three different lists based on their orders in the corresponding chromosome, which refers to the unit collection (UC) phase. Then, these three lists are aligned at the top, and units at the same positions are performed the crossover. This phase is named the UA and crossover phase. Finally, the unit restore (UR) phase is employed, i.e., when the crossover operation is completed, the units in these lists are restored to their original positions of the associated chromosomes. With these three sequential phases (i.e., the UC, the UA and crossover, and the UR), two chromosomes with different lengths could easily exchange their gene information for crossover. Because the crossover operation is performed on the unit lists, where only the units with the same types are loaded, this proposed UA crossover operation is reasonable (because they have the same origins). For the remaining units, which do not perform crossover operations due to no paired ones, are kept at the same position.

Mutation operations may perform on each position of the units from one chromosome. For a selected mutation point, a unit could be added, deleted, or modified, each of which is with an identical probability. In the case of unit addition, a convolutional layer, a pooling layer, or a fully connected layer could be added with an equal probability of 1/3. If the mutation is to modify an existing unit, the particular modification is dependent on the unit type, and all the encoded information

Algorithm 5: Environmental Selection

Input: The elitism fraction γ, and the current population $P_t \cup Q_t$.
Output: The selected population P_{t+1}.

1 $a \leftarrow$ Calculate the number of elites based on γ and the predefined population size N from Algorithm 2;
2 $P_{t+1} \leftarrow$ Select a individuals that have the best mean values from $P_t \cup Q_t$;
3 $P_t \cup Q_t \leftarrow P_t \cup Q_t - P_{t+1}$;
4 **while** $|P_{t+1}| < N$ **do**
5 $\quad s_1, s_2 \leftarrow$ Randomly select two individuals from $P_t \cup Q_t$;
6 $\quad s \leftarrow$ Employe Algorithm 4 to select one individual from s_1 and s_2;
7 $\quad P_{t+1} \leftarrow P_{t+1} \cup s$;
8 **end**
9 **Return** P_{t+1}.

in the unit would be changed (encoded information on each unit type can be seen in Table I). Note that all the encoded formation is denoted by real numbers, therefore the simulated binary crossover (SBX) [41] and the polynomial mutation [42] are employed in the proposed EvoCNN method due to their promising performance in real number gene representations.

G. Environmental Selection

The environmental selection is shown in Algorithm 5. During the environmental selection, the elitism and the diversity are addressed. To be specific, a fraction of individuals with promising mean values are chosen at first, and then the remaining individuals are selected by the modified binary tournament selection demonstrated in Section III-F. By these two strategies, the elitism and the diversity are considered simultaneously, which are expected to jointly improve the performance of the proposed EvoCNN method.

Note here that, the selected elites are removed before the tournament selection gets to work (line 3 of Algorithm 5), while the individuals selected for the purpose of diversity are kept in the current population for the next round of tournament selection (lines 4–8 of Algorithm 5) based on the convention in the community.

H. Best Individual Selection and Decoding

At the end of evolution, multiple individuals could have promising mean values but different architectures and connection weight initialization values. In this regard, there will be multiple choices to select the "Best Individual." For example, if we are only concerned with the best performance, we could neglect their architecture configurations and consider only the classification accuracy. Otherwise, if we emphasize the smaller number of parameters, corresponding decision could be made. Once the best individual is confirmed, the corresponding CNN is decoded based on the encoded architecture and connection weight initialization information, and then the decoded CNN will be deeply trained with a larger number of epochs by SGD for future usage. Note that an ReLU nonlinear layer is added to each convolutional layer and fully connection layer during decoding the individual to the corresponding CNN.

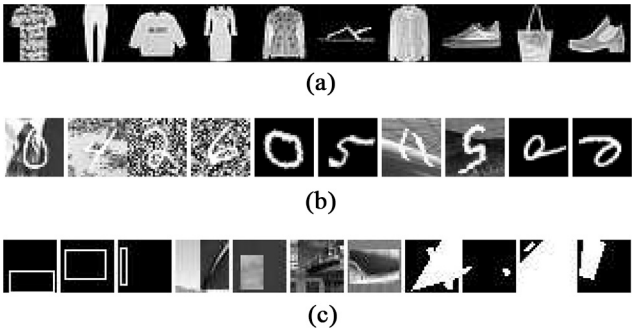

Fig. 4. Examples from the benchmarks chosen. (a) Examples from the first category. From left to right, they are T-shirt, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag, and ankle boot. (b) Examples from the second category. From left to right, each two images are from one group, and each group is from MBi, MRB, MRD, MRDBI, and MB, respectively. These images refer to the hand-written digits 0, 4, 2, 6, 0, 5, 7, 5, 9, and 6, respectively. (c) Examples from the thrid category, from left to right, the first three images are from the rectangle benchmark, the following four ones are from the RI benchmark, and the remainings are from the convex benchmark. Specifically, these examples with the index 1, 2, 6, 7, and 11 are positive samples.

IV. Experimental Design

In order to quantify the performance of the proposed EvoCNN, a series of experiments are designed and performed on the chosen image classification benchmark datasets, which are further compared to some chosen state-of-the-art peer competitors. In the following, these benchmark datasets are briefly introduced at first. Then, the peer competitors are discussed. Finally, parameter settings of the proposed EvoCNN method are documented.

A. Benchmark Datasets

In these experiments, nine widely used image classification benchmark datasets are used to examine the performance of the proposed EvoCNN method. They are the Fashion [43], the Rectangle [44], the Rectangle Images (RI) [44], the Convex Sets (CS) [44], the MNIST Basic (MB) [44], the MNIST with background images (MBI) [44], the MNIST with Random Background (MRB) [44], the MNIST with Rotated Digits (MRD) [44], and the MNIST with RD plus Background Images (MRDBI) [44] benchmarks.

Based on the types of the classification objects, these benchmarks are classified into three different categories. The first category includes only Fashion dataset, which is to recognize ten fashion objects (e.g., trousers, coats, and so on) in the 50 000 training images and 10 000 test images. The second one is composed of the MNIST [9] variants, including the MB, MBI, MRB, MRD, and the MRDBI benchmarks for classifying ten hand-written digits (i.e., 0–9). Because 97% accuracy has been easily achieved on the MNIST dataset, these MNIST variants introduced different barriers (e.g., random backgrounds and rotations) imposed upon the MNIST to significantly increase the complexity of classification tasks at hand. Furthermore, these variants have 12 000 training images

and 50 000 test images, which further challenges the classification algorithms due to the imbalance distribution of training and testing data. The third category is for recognizing the shapes of objects (i.e., the rectangle or not for the Rectangle and RI benchmarks, and the convex or not for the Convex benchmark). Obviously, this category covers the Rectangle, the RI, and the CS benchmarks that contain 1200, 12 000, and 8000 training images, respectively, and all of them include 50 000 test images. Compared to the Rectangle benchmark, the RI is generated by adding randomly sampled backgrounds into the MNIST to increase the difficulties in classification.

In addition, each image in these benchmarks is with the size 28×28, and examples from these benchmarks are shown in Fig. 4 for reference. Furthermore, another reason for using these benchmark datasets is that different algorithms have reported their promising results, which is convenient for the comparisons on the performance of the proposed EvoCNN method and the algorithms achieving the best performance on them (details can be seen Section IV-B).

B. Peer Competitors

Ideally, the algorithms for neural network architecture design discussed in Section I should be considered as the peer competitors. However, because MetaQNN [25] and LEIC [30] highly rely on the computational resources, it is impossible to reproduce the experimental results in an academic environment. Furthermore, the benchmark dataset investigated in these two algorithms employed different data preprocessing and augmentation techniques, which will highly enhance the final classification accuracy. As for other architecture design approaches introduced in Section I, such as GS, RS, among others, they are not scalable to CNNs and not suitable to be directly compared as well.

Note that LEIC is built on more complex CNN architectures, such as CNNs with the skip connections and CNNs with dense connections, which can be seen in state-of-the-art CNNs, such as the ResNet [12] and the DenseNet [45]. In this paper, as we have introduced in Section II-A, the proposed algorithm is based on the traditional CNNs. We all know that there is no single CNN which achieves the promising performance on every dataset, i.e., the CNNs designed by LEIC and Genetic CNN are only suitable to their investigated datasets. To this end, it is unfair if we use the proposed algorithm to perform the comparison on the datasets which have been investigated by LEIC and Genetic CNN, and vice versa.

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 EvoCNN method. To be specific, the peer competitors on the Fashion benchmark are collected from the dataset homepage.[1] They are 2C1P2F+Dropout, 2C1P, 3C2F, 3C1P2F+Dropout, GRU+SVM+Dropout, GoogleNet [35], AlexNet [3], SqueezeNet-200 [46], MLP 256-128-64, and VGG16 [11], which perform the experiments on the raw dataset without any preprocessing. The peer competitors on other benchmarks are CAE-2 [47], TIRBM [48], PGBM+DN-1 [49], ScatNet-2 [50], RandNet-2 [51], PCANet-2 (softmax) [51], LDANet-2 [51], SVM+RBF [44], SVM+Poly [44], NNet [44], SAA-3 [44], and DBN-3 [44], which are from [51] recently published and the provider of the benchmarks.[2]

C. Parameter Settings

All the parameter settings are chosen based on the conventions in the communities of evolutionary algorithms [52] and deep learning [53], in addition to the maximal length of each basic layer. Specifically, the population size and the total generation number are set to be 100. The distribution index of SBX and Polynomial mutation are both set to be 20, and their associated probabilities are specified as 0.9 and 0.1, respectively. The maximum lengths of the convolutional layers, the pooling layers, and the fully connected layers are set to be the same as 5 (i.e., the maximal depths of CNNs in these experiments are 15), which is believed sufficient to solve the benchmark datasets employed. Moreover, the proportion of the elitism is set to be 20% based on the Pareto principle. In addition, 20% data are randomly selected from the training images as the validation dataset based on the conventions from machine learning community. For the implementation of the proposed EvoCNN method, we constrain the same values of the width and height for filter, stride, and kernel. The width and height for stride in the convolutional layer are set to be 1, those in the pooling layer are specified at the same value to its kernel, and the convolutional type is fixed to "SAME." These settings are very common in the deep learning community. In addition, the architectures of the evolved CNNs in EvoCNN will strictly conform the standard architectures of CNNs (the standard architectures have been introduced in Section II-A) but the mixed ones, although the mixed ones may have better performance, which is not in the scope of this paper. For all the benchmark dataset, the deep training epoch is set to 100, and the epoch for catching the tendency of the performance is set to 10 which is specified based on our experiences on these benchmark datasets.

Here, we will introduce a method to set the proper epoch number for catching the tendency of performance when using the proposed algorithm for unfamiliar datasets: 1) randomly initializing multiple individuals and then deeply training them on the corresponding dataset; 2) recording their classification errors at different epochs; 3) finding the appropriate epoch number from which the tendency in term of the classification errors of each individual can be evidently observed; and 4) using the chosen epoch number to catch the tendency of the performance for experimental analysis when using the proposed algorithm.

The proposed EvoCNN method is implemented by Tensorflow [54], and each copy of the code runs in a computer equipped with two GPU cards with the identical model number GTX1080. During the final training phase, each individual is subjected to the BatchNorm [55] for speeding up and the weight decay with an unified number for preventing from

[1] https://github.com/zalandoresearch/fashion-mnist

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

the overfitting. Due to the heuristic nature of the proposed EvoCNN method, 30 independent runs are performed on each benchmark dataset, and the mean results are calculated for the comparisons unless otherwise specified. Furthermore, the experiments take four days on the Fashion benchmark for each run, 2–3 days on other benchmarks. The source code is available at https://github.com/sunkevin1214/codes for reproducing the experimental results of the proposed EvoCNN method reported in this paper.

V. Experimental Results and Analysis

In this section, the experimental results and analysis of the proposed EvoCNN method against peer competitors are shown in Section V-A. Then, the weight initialization method in the proposed EvoCNN method are specifically investigated in Section V-B.

A. Overall Results

The experimental results on the Fashion benchmark dataset are shown in Table II, while those on the MB, MRD, MRB, MBI, MRDBI, Rectangle, RI, and Convex benchmark datasets are shown in Table III. In Tables II and III, the last two rows denote the *best* and *mean* classification errors received from the proposed EvoCNN method, respectively, and other rows show the *best* classification errors reported by peer competitors. The numbers following the mean classification error in the last row of Tables II and III are the corresponding standard derivations of these errors out of 30 independent runs. Again, it is a convention in evolutionary computation community that the statistical results should be used to do the comparison. However, the results of the peer competitors are extracted from the original publications, and the statistical results are not provided. Indeed, it is a convention in deep learning community that only the *best* result is reported. In order to conveniently investigate the comparisons, the terms "(+)" and "(-)" are provided to denote whether the best result generated by the proposed EvoCNN method is better or worse that the best result obtained by the corresponding peer competitor. The term "—" means there is no available result reported from the provider. Most information on the number of parameters and number of training epoch from the peer competitors on the Fashion benchmark dataset is available, therefore, such information from EvoCNN is also shown in Table II for comparisons. However, for the benchmarks in Table III, such information is not presented because they are not available from the peer competitors. Note that all the results from the peer competitors and the proposed EvoCNN method are without any data augmentation preprocessing on the benchmarks for a fair comparison.

It is clearly shown in Table II that by comparing the best performance, the proposed EvoCNN method outperforms *all* ten peer competitors. The two state-of-the-art algorithms, GoogleNet and VGG16, achieve, respectively, 6.5% and 6.3% classification error rates, where the difference is only 0.2%. The proposed EvoCNN method further reduces the error rate by 0.83%–5.47%. Furthermore, the mean performance of EvoCNN is even better than the best performance of eight competitors, and only slightly worse than the best in GoogleNet and VGG16. In addition, EvoCNN has a much smaller number of connection weights—EvoCNN uses 6.52 million weights while GoogleNet uses 101 million and VGG uses 26 million weights. EvoCNN also employs only half of the numbers of training epochs used in the VGG16. The results show that the proposed EvoCNN method obtains much better performance in the architecture design and connection weight initialization of CNNs on the Fashion benchmark.

TABLE II
Classification Errors of the Proposed EvoCNN Method Against the Peer Competitors on the Fashion Benchmark Dataset

classifier	error(%)	# parameters	# epochs
2C1P2F+Drouout	8.40(+)	3.27M	300
2C1P	7.50(+)	100K	30
3C2F	9.30(+)	—	—
3C1P2F+Dropout	7.40(+)	7.14M	150
GRU+SVM+Dropout	10.30(+)	—	100
GoogleNet [35]	6.30(+)	101M	—
AlexNet [3]	10.10(+)	60M	—
SqueezeNet-200 [46]	10.00(+)	500K	200
MLP 256-128-64	10.00(+)	41K	25
VGG16 [11]	6.50(+)	26M	200
EvoCNN (best)	5.47	6.68M	100
EvoCNN (mean)	7.28 (1.69)	6.52M	100

According to Table III, EvoCNN is the *best* one among all 13 different methods. Specifically, EvoCNN achieves the best performance on five of the eight datasets, and the second best on the other three datasets—the MB, MRD, and Convex datasets, where LDANet-2, TIRBM, and PCANet-2(softmax) achieve the lowest classification error rates, respectively. Furthermore, comparing the *mean* performance of EvoCNN with the *best* of the other 12 methods, EvoCNN is the best on four datasets (MRB, MBI, Rectangle, and RI), second best on two (MRD and Convex), third best and fourth best on the other two datasets (MRDBI and MB). Particularly on MRB and MBI, the lowest classification error rates of the others are 6.08% and 11.5%, and the mean error rates of EvoCNN are 3.59% and 4.62%, respectively. In summary, the best classification error of the proposed EvoCNN method wins 80 out of 84 comparisons against the best results from the 12 peer competitors. Also the mean classification error of EvoCNN is better than the best error of the 12 methods on 75 out of the 84 comparisons.

B. Performance Regarding Weight Initialization

Further experiments are performed to examine the effectiveness of the connection weight initialization method in the proposed EvoCNN approach. By comparing different weight initializers, we would also investigate whether the architecture or the connection weight initialization values will affect the classification performance. To achieve this, we initialize another group of CNNs with the same architectures as that of the evolved EvoCNN, but their weights are initialized with the widely used Xavier initializer [32] (details in Section II-B).

TABLE III
CLASSIFICATION ERRORS OF THE PROPOSED EVOCNN METHOD AGAINST THE PEER COMPETITORS ON THE MB, MRD, MRB, MBI, MRDBI, RECTANGLE, RI, AND CONVEX BENCHMARK DATASETS

classifier	MB	MRD	MRB	MBI	MRDBI	Rectangle	RI	Convex
CAE-2 [47]	2.48(+)	9.66(+)	10.90(+)	15.50(+)	45.23(+)	1.21(+)	21.54(+)	—
TIRBM [48]	—	4.20(-)	—	—	35.50(+)	—	—	—
PGBM+DN-1 [49]	—	—	6.08(+)	12.25(+)	36.76(+)	—	—	—
ScatNet-2 [50]	1.27(+)	7.48(+)	12.30(+)	18.40(+)	50.48(+)	0.01(=)	8.02(+)	6.50(+)
RandNet-2 [51]	1.25(+)	8.47(+)	13.47(+)	11.65(+)	43.69(+)	0.09(+)	17.00(+)	5.45(+)
PCANet-2 (softmax) [51]	1.40(+)	8.52(+)	6.85(+)	11.55(+)	35.86(+)	0.49(+)	13.39(+)	4.19(-)
LDANet-2 [51]	1.05(-)	7.52(+)	6.81(+)	12.42(+)	38.54(+)	0.14(+)	16.20(+)	7.22(+)
SVM+RBF [44]	3.03(+)	11.11(+)	14.58(+)	22.61(+)	55.18(+)	2.15(+)	24.04(+)	19.13(+)
SVM+Poly [44]	3.69(+)	15.42(+)	16.62(+)	24.01(+)	56.41(+)	2.15(+)	24.05(+)	19.82(+)
NNet [44]	4.69(+)	18.11(+)	20.04(+)	27.41(+)	62.16(+)	7.16(+)	33.20(+)	32.25(+)
SAA-3 [44]	3.46(+)	10.30(+)	11.28(+)	23.00(+)	51.93(+)	2.41(+)	24.05(+)	18.41(+)
DBN-3 [44]	3.11(+)	10.30(+)	6.73(+)	16.31(+)	47.39(+)	2.61(+)	22.50(+)	18.63(+)
EvoCNN (best)	1.18	5.22	2.80	4.53	35.03	0.01	5.03	4.82
EvoCNN (mean)	1.28 (0.15)	5.46 (0.25)	3.59 (0.89)	4.62 (0.12)	37.38 (1.75)	0.01 (0.00)	5.97 (0.79)	5.39 (0.60)

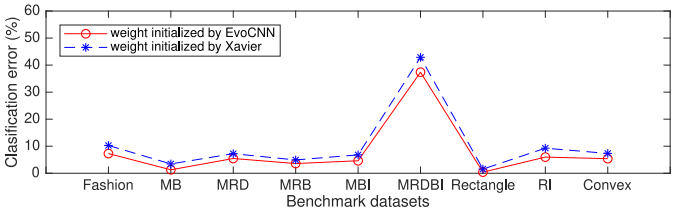

Fig. 5. Performance comparison between the proposed EvoCNN method and the CNN using the evolved architecture and the Xavier weight initializer.

The comparisons are illustrated in Fig. 5, where the horizon axis denotes the benchmark datasets, and vertical axis denotes the classification error rates. Fig. 5 clearly shows that the proposed connection weight initialization strategy in EvoCNN improves the classification performance on *all* the benchmarks over the popularly used Xavier initializer. To be specific, the proposed weight initialization method reaches ≈1.5% classification accuracy improvement on the Fashion, MB, MRD, MBI, RI, and Convex benchmarks, and 4.5% on the MRDBI benchmark. By comparing with the results in Tables II and III, it also can be concluded that the architectures of CNNs contribute to the classification performance more than that of the connection weight initialization. The proposed EvoCNN method gained promising performance by automatically evolving both the initial connection weights and the architectures of CNNs.

C. Further Discussion

In this section, we will further discuss the encoding strategy of the architectures and the weights related parameters, and the fitness evaluation in the proposed EvoCNN method. Findings from the experimental results are discussed as well, which could provide useful insights on the applications of the proposed EvoCNN method.

In GAs, crossover operators play the role of exploitation search (i.e., the local search), and mutation operators act as the exploration search (i.e., the global search). Because the local search and the global search should complement to each other, only well designing both of them could effectively improve the performance. Generally, the crossover operators operate on the chromosomes with the same lengths, while the proposed EvoCNN method employs the variable-length ones. Furthermore, multiple different basic layers exist in CNNs, which improve the difficulties of designing crossover operation in this context (because crossover operations cannot be easily performed between the genes from different origins). Therefore, a new crossover operation is introduced to the proposed EvoCNN method. UC, UA, as well as UR are designed to complete the crossover operation, which enhances the communications between the encoded information, and expectedly enhances the performance in searching for the promising architectures of CNNs.

The typically used approaches in optimizing the weights of CNNs are based on the gradient information. It is well known that the gradient-based optimizers are sensitive to the starting position of the parameters to be optimized. Without a better starting position, the gradient-based algorithms are easy to be trapped into local minima. Intuitively, finding a better starting position of the connection weights by GAs is intractable due to the huge numbers of parameters. As we have elaborated, a huge number of parameters cannot be efficiently encoded into the chromosomes, nor effectively optimized. In the proposed EvoCNN method, an indirect encoding approach is employed, which only encodes the means and standard derivations of the weights in each layer. In fact, employing the standard derivation and the mean to initialize the starting position of connection weights is a compromised approach and can be seen from deep learning libraries [54], [56], [57], which is the motivation of this design in the proposed EvoCNN method. With this strategy, only two real numbers are used to denote the hundreds of thousands of parameters, which could save much computational resource for encoding and optimization.

Existing techniques for searching for the architectures of CNNs typically take the final classification accuracy as the

TABLE IV
CLASSIFICATION ACCURACY AND THE PARAMETER NUMBERS FROM THE INDIVIDUALS ON THE FASHION BENCHMARK DATASET

classification accuracy	99%	98%	97%	96%	95%
# parameters	569,250	98,588	7,035	3,205	955

fitness of individuals. A final classification accuracy typically requires many more epochs of the training, which is a very time-consuming process. In order to complete the architecture design with this fitness evaluation approach, it is natural to employ a lot of computational resource to perform this task in addition to speed up the design. However, computational resource is not necessarily available to all interested researchers. In fact, a tendency of each individual that could predict the future quality would be sufficient, and it is not necessary to check the final classification accuracy. Therefore, we only employ a small number of epochs to train these individuals. With this kind of fitness measurement, the proposed EvoCNN method does not highly rely on the computational resource. In summary, with the well-designed yet simple encoding strategies and this fitness evaluation method, researchers without rich domain knowledge could also design the promising CNN models for addressing specific tasks in their (academic) environment, where the computational resources are typically limited.

Moreover, interesting findings have also been discovered when the proposed EvoCNN method terminates, i.e., multiple individuals are with the similar performance but with significantly different numbers of connection weights (see Table IV), largely due to population-based nature of the proposed EvoCNN method. Recently, various applications have been developed, such as the auto-driving car and some interesting real-time mobile applications. Due to the limited processing capacity and battery power in these devices, trained CNNs with similar performance but fewer parameters are much more preferred because they require less computational resource and the energy consumption. In addition, similar performance of the individuals is also found with different lengths of the basic layers, and there are multiple pieces of hardware that have been specifically designed to speed up the calculations in CNNs, such as the specialized hardware for convolutional operations. In this regard, the proposed EvoCNN method could give manufacturers of such devices more choices to make a decision based on their own preferences. If the traditional approaches are employed here, it is difficult to find out such models at a single run.

VI. CONCLUSION

The objective of this paper is to develop a new evolutionary approach to automatically evolve the architectures and weights of CNNs for image classification problems. This goal has been successfully achieved by proposing a new representation for weight initialization strategy, a new encoding scheme of variable-length chromosomes, a new genetic operator to chromosomes with different lengths, a slacked binary tournament selection for selecting promising individuals, and an efficient fitness evaluation method to speed up the evolution. This approach was examined and compared with 22 peer competitors, including most state-of-the-are algorithms on nine benchmark datasets commonly used in deep learning. The experimental results show that the proposed EvoCNN method significantly outperforms all of these existing algorithms on almost all these datasets in terms of their best classification performance. Furthermore, the mean classification error rate of the proposed algorithm is even better than the best of others in many cases. In addition, the model optimized by EvoCNN is with a much smaller number of parameters yet promising best classification performance. Specifically, the model optimized by EvoCNN employs 100 epochs to reach the lowest classification error rate of 5.47% on the Fasion dataset, while the state-of-the-art VGG16 employs 200 epochs achieving 6.50% classification error rate. Findings from the experimental results also suggest that the proposed EvoCNN method could provide more options for the manufacturers of smart devices that are interested in integrating CNNs into their products with limited computational capacity and battery power.

In this paper, we investigate the proposed EvoCNN method only on the commonly used middle-scale benchmarks. However, large-scale data also widely exist in the current era of big data. Because evaluating only one epoch on these large-scale data would require significant computational resource and take a long period, the proposed fitness evaluation method is not suitable for them unless a huge amount of computational resources are made available. In the future, we will invest efforts on efficient fitness evaluation techniques. In addition, we will also investigate evolutionary algorithms for recurrent neural networks, which are powerful tools for addressing time-dependent data, such as the voice and video data.

REFERENCES

[1] D. Cireşan, U. Meier, J. Masci, and J. Schmidhuber, "Multi-column deep neural network for traffic sign classification," *Neural Netw.*, vol. 32, pp. 333–338, Aug. 2012.
[2] F. Ning, D. Delhomme, Y. LeCun, F. Piano, L. Bottou, and P. E. Barbano, "Toward automatic phenotyping of developing embryos from videos," *IEEE Trans. Image Process.*, vol. 14, no. 9, pp. 1360–1371, Sep. 2005.
[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.
[4] M. Oquab, L. Bottou, I. Laptev, and J. Sivic, "Learning and transferring mid-level image representations using convolutional neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Columbus, OH, USA, 2014, pp. 1717–1724.
[5] H.-C. Shin *et al.*, "Deep convolutional neural networks for computer-aided detection: CNN architectures, dataset characteristics and transfer learning," *IEEE Trans. Med. Imag.*, vol. 35, no. 5, pp. 1285–1298, May 2016.
[6] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" in *Proc. Adv. Neural Inf. Process. Syst.*, Montreal, QC, Canada, 2014, pp. 3320–3328.
[7] D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *J. Physiol.*, vol. 160, no. 1, pp. 106–154, 1962.
[8] Y. LeCun *et al.*, "Backpropagation applied to handwritten zip code recognition," *Neural Comput.*, vol. 1, no. 4, pp. 541–551, Dec. 1989.
[9] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.
[10] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Cogn. Model.*, vol. 5, no. 3, p. 1, 1988.

[11] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. 3rd Int. Conf. Learn. Represent.*, San Diego, CA, USA, May 2015. [Online]. Available: http://arxiv.org/abs/1409.1556
[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Las Vegas, NV, USA, 2016, pp. 770–778.
[13] H. Bourlard and Y. Kamp, "Auto-association by multilayer perceptrons and singular value decomposition," *Biol. Cybern.*, vol. 59, nos. 4–5, pp. 291–294, 1988.
[14] G. E. Hinton and R. S. Zemel, "Autoencoders, minimum description length, and Helmholtz free energy," in *Proc. Adv. Neural Inf. Process. Syst.*, 1994, p. 3.
[15] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.
[16] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, pp. 281–305, Feb. 2012.
[17] C. E. Rasmussen and C. K. Williams, *Gaussian Processes for Machine Learning*, vol. 1. Cambridge, U.K.: MIT Press, 2006.
[18] J. Močkus, "On Bayesian methods for seeking the extremum," in *Proc. Optim. Techn. IFIP Tech. Conf.*, 1975, pp. 400–404.
[19] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Proc. Adv. Neural Inf. Process. Syst.*, 2011, pp. 2546–2554.
[20] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *Proc. LION*, vol. 5, 2011, pp. 507–523.
[21] J. Beck and T. Fiala, "'Integer-making' theorems," *Discr. Appl. Math.*, vol. 3, no. 1, pp. 1–8, 1981.
[22] J. Halton and G. Smith, "Radical inverse quasi-random point sequence, algorithm 247," *Commun. ACM*, vol. 7, no. 12, p. 701, 1964.
[23] E. I. Atanassov, A. Karaivanova, and S. Ivanovska, "Tuning the generation of Sobol sequence with Owen scrambling," in *Large-Scale Scientific Computing*. Heidelberg, Germany: Springer, 2009, pp. 459–466.
[24] S. Saxena and J. Verbeek, "Convolutional neural fabrics," in *Advances in Neural Information Processing Systems*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds. Red Hook, NY, USA: Curran Assoc., 2016, pp. 4053–4061. [Online]. Available: http://papers.nips.cc/paper/6304-convolutional-neural-fabrics.pdf
[25] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," in *Proc. Int. Conf. Learn. Represent.*, 2017, pp. 1–18.
[26] D. Ashlock, *Evolutionary Computation for Modeling and Optimization*. New York, NY, USA: Springer, 2006.
[27] T. Back, *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. New York, NY, USA: Oxford Univ. Press, 1996.
[28] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press, 1998.
[29] X. Yao, "Evolving artificial neural networks," *Proc. IEEE*, vol. 87, no. 9, pp. 1423–1447, Sep. 1999.
[30] E. Real *et al.*, "Large-scale evolution of image classifiers," in *Proc. 34th Int. Conf. Mach. Learn.*, vol. 70, 2017, pp. 2902–2911.
[31] L. Xie and A. L. Yuille, "Genetic CNN," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Venice, Italy, 2017, Oct. 2017, pp. 1388–1397. [Online]. Available: https://doi.org/10.1109/ICCV.2017.154
[32] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proc. 13th Int. Conf. Artif. Intell. Stat.*, Sardinia, Italy, 2010, pp. 249–256.
[33] A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *J. Physiol.*, vol. 117, no. 4, pp. 500–544, 1952.
[34] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Proc. Eur. Conf. Comput. Vis.*, Zürich, Switzerland, 2014, pp. 818–833.
[35] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Boston, MA, USA, 2015, pp. 1–9.
[36] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 8, pp. 1798–1828, Aug. 2013.
[37] Y. Bengio and O. Delalleau, "On the expressive power of deep architectures," in *Proc. Int. Conf. Algorithmic Learn. Theory*, 2011, pp. 18–36.
[38] O. Delalleau and Y. Bengio, "Shallow vs. deep sum-product networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2011, pp. 666–674.
[39] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evol. Comput.*, vol. 10, no. 2, pp. 99–127, 2002.
[40] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth, "Occam's razor," *Inf. Process. Lett.*, vol. 24, no. 6, pp. 377–380, 1987.
[41] K. Deb and R. B. Agrawal, "Simulated binary crossover for continuous search space," *Complex Syst.*, vol. 9, no. 3, pp. 1–15, 1994.
[42] K. Deb, *Multi-Objective Optimization Using Evolutionary Algorithms*, vol. 16. New York, NY, USA: Wiley, 2001.
[43] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms," *CoRR*, vol. abs/1708.07747, 2017. [Online]. Available: http://arxiv.org/abs/1708.07747
[44] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio, "An empirical evaluation of deep architectures on problems with many factors of variation," in *Proc. 24th Int. Conf. Mach. Learn.*, 2007, pp. 473–480.
[45] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Honolulu, HI, USA, 2017, pp. 2261–2269.
[46] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size," *CoRR*, vol. abs/1602.07360, 2016. [Online]. Available: http://arxiv.org/abs/1602.07360
[47] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio, "Contractive auto-encoders: Explicit invariance during feature extraction," in *Proc. 28th Int. Conf. Mach. Learn.*, 2011, pp. 833–840.
[48] K. Sohn and H. Lee, "Learning invariant representations with local transformations," in *Proc. 29th Int. Conf. Mach. Learn. (ICML)*, 2012, pp. 1339–1346.
[49] K. Sohn, G. Zhou, C. Lee, and H. Lee, "Learning and selecting features jointly with point-wise gated Boltzmann machines," in *Proc. Int. Conf. Mach. Learn.*, 2013, pp. 217–225.
[50] J. Bruna and S. Mallat, "Invariant scattering convolution networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 8, pp. 1872–1886, Aug. 2013.
[51] T.-H. Chan, K. Jia, S. Gao, J. Lu, Z. Zeng, and Y. Ma, "PCANet: A simple deep learning baseline for image classification?" *IEEE Trans. Image Process.*, vol. 24, no. 12, pp. 5017–5032, Dec. 2015.
[52] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002.
[53] G. E. Hinton, "A practical guide to training restricted Boltzmann machines," in *Neural Networks: Tricks of the Trade*. Heidelberg, Germany: Springer, 2012, pp. 599–619.
[54] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous distributed systems," *CoRR*, vol. abs/1603.04467, 2016. [Online]. Available: http://arxiv.org/abs/1603.04467
[55] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 448–456.
[56] Y. Jia *et al.*, "Caffe: Convolutional architecture for fast feature embedding," in *Proc. 22nd ACM Int. Conf. Multimedia*, 2014, pp. 675–678.
[57] T. Chen *et al.*, "MXnet: A flexible and efficient machine learning library for heterogeneous distributed systems," *CoRR*, vol. abs/1512.01274, 2015. [Online]. Available: http://arxiv.org/abs/1512.01274

Yanan Sun (S'15–M'18) received the Ph.D. degree in engineering from Sichuan University, Chengdu, China, in 2017.

He was a Research Fellow with the School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand. He is currently a Professor (research) with the College of Computer Science, Sichuan University. His current research interests include evolutionary algorithms, deep learning, and evolutionary deep learning.

Dr. Sun is the Leading Organizer of the First Workshop on Evolutionary Deep Learning, the Leading Organizer of the Special Session on Evolutionary Deep Learning and Applications in CEC19, and the Founding Chair of the IEEE CIS Task Force on Evolutionary Deep Learning and Applications.

Bing Xue (M'10) received the B.Sc. degree from the Henan University of Economics and Law, Zhengzhou, China, in 2007, the M.Sc. degree in management from Shenzhen University, Shenzhen, China, in 2010, and the Ph.D. degree in computer science with Victoria University of Wellington, New Zealand, in 2014.

She is currently an Associate Professor with the School of Engineering and Computer Science, Victoria University of Wellington. She has over 100 papers published in fully refereed international journals and conferences and most of them are on evolutionary feature selection and construction. Her current research interests include evolutionary computation, feature selection, feature construction, multiobjective optimization, image analysis, transfer learning, data mining, and machine learning.

Dr. Xue is currently the Chair of the IEEE Task Force on Evolutionary Feature Selection and Construction, the IEEE Computational Intelligence Society (CIS), the Vice-Chair of the IEEE CIS Data Mining and Big Data Analytics Technical Committee, and the Vice-Chair of IEEE CIS Task Force on Transfer Learning and Transfer Optimization. She is also an Associate Editor/Editorial Board Member of five international journals and a Reviewer of over 50 international journals. She is the Finance Chair of IEEE Congress on Evolutionary Computation 2019.

Mengjie Zhang (M'04–SM'10–F'18) received the B.E. and M.E. degrees from Artificial Intelligence Research Center, Agricultural University of Hebei, Hebei, China, and the Ph.D. degree in computer science from RMIT University, Melbourne, VIC, Australia, in 1989, 1992, and 2000, respectively.

He is currently a Professor of computer science, the Head of the Evolutionary Computation Research Group, and the Associate Dean (Research and Innovation) in the Faculty of Engineering. He has published over 350 research papers in refereed international journals and conferences. His current research interests include evolutionary computation, particularly genetic programming, particle swarm optimization, and learning classifier systems with application areas of image analysis, multiobjective optimization, feature selection and reduction, job shop scheduling, and transfer learning.

Prof. Zhang is currently chairing the IEEE CIS Intelligent Systems and Applications Technical Committee, and the immediate Past Chair for the IEEE CIS Emergent Technologies Technical Committee and the Evolutionary Computation Technical Committee, and a member of the IEEE CIS Award Committee. He is the Vice-Chair of the IEEE CIS Task Force on Evolutionary Feature Selection and Construction, the Vice-Chair of the Task Force on Evolutionary Computer Vision and Image Processing, and the Founding Chair of the IEEE Computational Intelligence Chapter in New Zealand. He is also a Committee Member of the IEEE NZ Central Section. He is a fellow of Royal Society of New Zealand and have been a Panel member of the Marsden Fund (New Zealand Government Funding). He is also a member of ACM.

Gary G. Yen (S'87–M'88–SM'97–F'09) received the Ph.D. degree in electrical and computer engineering from the University of Notre Dame, Notre Dame, IN, USA, in 1992.

In 1997, he was with the Structure Control Division, U.S. Air Force Research Laboratory, Albuquerque, NM, USA. He is currently a Regents Professor with the School of Electrical and Computer Engineering, Oklahoma State University, Stillwater, OK, USA. His current research interest includes intelligent control, computational intelligence, conditional health monitoring, and signal processing and their industrial/defense applications.

Dr. Yen was a recipient of the Andrew P Sage Best Transactions Paper Award from the IEEE Systems, Man and Cybernetics Society in 2011 and the Meritorious Service Award from IEEE Computational Intelligence Society in 2014. He was an Associate Editor of the *IEEE Control Systems Magazine*, the IEEE TRANSACTIONS ON CONTROL SYSTEMS TECHNOLOGY, *Automatica*, *Mechantronics*, the IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS—PART A: SYSTEMS AND HUMANS, the IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS—PART B: CYBERNETICS, the IEEE TRANSACTIONS ON NEURAL NETWORKS. He is currently serving as an Associate Editor for the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION and the IEEE TRANSACTIONS ON CYBERNETICS. He served as the General Chair for the 2003 IEEE International Symposium on Intelligent Control in Houston, TX, USA, and the 2006 IEEE World Congress on Computational Intelligence in Vancouver, BC, Canada. He served as the Vice President for the Technical Activities from 2005 to 2006 and then President from 2010 to 2011 of the IEEE Computational intelligence Society. He was the Founding Editor-in-Chief of the *IEEE Computational Intelligence Magazine* from 2006 to 2009.