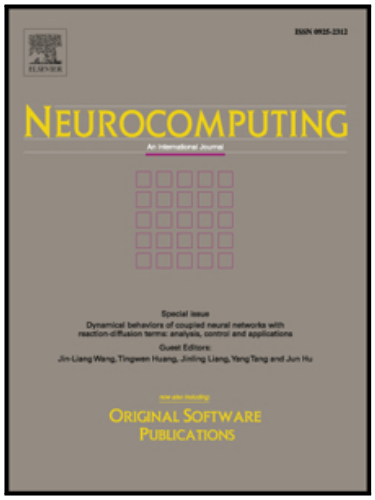
Journal Pre-proof



Autonomous Deep Learning: A Genetic DCNN Designer for Image Classification

Benteng Ma, Xiang Li, Yong Xia, Yanning Zhang

PII: S0925-2312(19)31379-7
DOI: https://doi.org/10.1016/j.neucom.2019.10.007
Reference: NEUCOM 21348

To appear in: *Neurocomputing*

Received date: 15 March 2019
Revised date: 26 June 2019
Accepted date: 5 October 2019

Please cite this article as: Benteng Ma, Xiang Li, Yong Xia, Yanning Zhang, Autonomous Deep Learning: A Genetic DCNN Designer for Image Classification, *Neurocomputing* (2019), doi: https://doi.org/10.1016/j.neucom.2019.10.007

Highlights

- Search deep convolutional neural networks automatically for image classification.
- Encode a deep convolutional neural networks architecture into an integer string.
- Evolve a population of DCNN architectures using genetic evolutionary operations.
- Obtained DCNNs achieved satisfying results with less layers than pre-trained models.
- Image classification using neither handcrafted features nor handcrafted networks.

Autonomous Deep Learning: A Genetic DCNN Designer for Image Classification

Benteng Ma[a,b], Xiang Li[b], Yong Xia[a,b], Yanning Zhang[b]

[a]*Research & Development Institute of Northwestern Polytechnical University in Shenzhen, Shenzhen 518057, China*
[b]*National Engineering Laboratory for Integrated Aero-Space-Ground-Ocean Big Data Application Technology, School of Computer Science and Engineering, Northwestern Polytechnical University, Xian 710072, China*

Abstract

Recent years have witnessed the breakthrough success of deep convolutional neural networks (DCNNs) in image classification and other vision applications. DCNNs have distinct advantages over traditional solutions in providing a uniform feature extraction-classification framework to free users from troublesome handcrafted feature extraction. However, DCNNs are far from autonomous, since their performance relies heavily on the handcrafted architectures, which also requires a lot expertise and experience to design, and cannot be continuously improved once the tuning of hyper-parameters converges. In this paper, we propose an autonomous and continuous learning (ACL) algorithm to generate automatically a DCNN architecture for each given vision task. We first partition a DCNN into multiple stacked meta convolutional blocks and fully connected blocks, each of which may contain the operations of convolution, pooling, fully connection, batch normalization, activation and drop out, and thus convert the architecture into an integer code. Then, we use genetic evolutionary operations, including selection, mutation and crossover to evolve a population of DCNN architectures. We have evaluated this algorithm on six image classification tasks, i.e., MNIST, Fashion-MNIST, EMNIST-Letters, EMNIST-Digits, CIFAR10 and CIFAR100. Our results indicate that the proposed ACL algorithm is able to

Email address: yxia@nwpu.edu.cn (Yanning Zhang)

evolve the DCNN architecture continuously if more time cost is allowed and can find a suboptimal DCNN architecture, whose performance is comparable to the state of the art.

Keywords: deep convolutional neural networks (DCNNs), neural Architecture Search, genetic algorithm (GA), image classification

1. Introduction

Deep convolutional neural networks (DCNNs), such as AlexNet [1], VGGNet [2], GoogLeNet [3], ResNet [4] and DenseNet [5], have significantly improved the baselines of most computer version tasks. Despite their distinct advantages over traditional approaches, DCNNs are still specialist systems that leverage a myriad amount of human expertise and data. They provide a uniform feature extraction-classification framework to free human from troublesome handcrafted feature extraction at the expense of handcrafted network design. Designing the architecture of DCNN automatically can not only bypass this issue but also take a fundamental step towards the long-standing ambition of artificial intelligence i.e. creating autonomous learning systems that require lest human intervention [6]. Automated design of DCNN architectures has drawn more and more research attentions in recent years, resulting in a number of algorithms in the literature, which can be roughly divided into four categories: (1) DCNN architecture selection from a group of candidates, (2) DCNN architecture optimization using deep learning, (3) reinforcement learning-based DCNN architecture optimization, and (4) evolutionary optimization of DCNN architectures. Among them, evolutionary optimization approaches have a long history and seem to be very promising due to their multi-point global search ability, which enable them to quickly locate the areas of high quality solutions even in case of a very complex search space [7]. Despite their success, most evolutionary approaches pose restrictions either on the obtained DCNN architectures, such as the fixed depth, fixed filter size, fixed activation function, fixed pooling operation [8] and skipping out the pooling operation [9], or on the employed

genetic operations, such as abandoning the crossover [10]. These restrictions may reduce the computational complexity, but also lead to lower performance. Alternatively, other evolutionary approaches may require hundreds even thousands of computers to perform parallel optimization [9, 10]. In this paper, we propose a genetic DCNN designer to automatically generate the architecture of a DCNN for each given image classification problem. To reduce the complexity of representing a DCNN architecture, we develop a simple but effective encoding scheme, in which almost all the operations in a DCNN, such as the convolution, pooling, batch normalization, activation, fully connection, drop out and optimizer, are encoded as an integer vector. We start with a population of randomly initialized DCNN architectures and iteratively evolve this population on a generation-by-generation basis to create better architectures using the redefined genetic operations, including selection, crossover and mutation. We have evaluated our approach on six image classification tasks using the MNIST [11], EMNIST-Digits [12], EMNIST-Letters [12], Fashion-MNIST [13], CIFAR10 [14] and CIFAR100 [14] datasets. Our results indicate that the proposed genetic DCNN designer is able to generate automatically a DCNN architecture for each given image classification task, whose performance is comparable to the state of the art.

2. Related Work

Many research efforts for automated design of DCNN architectures have been devoted to searching and selecting the most effective DCNN architecture from a group of candidates. Jin et al. [15] introduced the sub-modular and super-modular optimization to the construction of DCNNs and established the guidelines to set the width and depth of DCNNs. Fernando et al. [16] proposed the PathNet algorithm, which samples a sub-network from a super DCNN that has a much larger architecture, and demonstrated that this algorithm is capable of sustaining transfer learning on multiple tasks in both the supervised and reinforcement learning settings. Alternatively, the architecture and / or weights

of a DCNN can be optimized by using another deep neural network. Ha et al. [17] used a hyper network and the discrete cosine transform (DCT) to evolve the weights of a DCNN, which has a fixed architecture. To generate the filters in a DCNN, Brabandere [18] proposed a dynamic filter network, which can be separated into a filter-generating network and a dynamic filtering layer. The former generates dynamically sample-specific filter parameters conditioned on the input, and the latter applies those filters to the input. Recently, reinforcement learning has been applied to the design of DCNN architectures. Zoph and Le [19] employed a recurrent neural network trained by reinforcement learning to maximize the expected accuracy of the generated DCNN architecture on a validation set of images. This method, however, creates a DCNN with fixed-depth on a layer-by-layer basis, where each layer has a pre-determined number of filters and a pre-determined filter size. This method used distributed training and asynchronous parameters updated with 800 graphs processing units (GPUs). Barker et al. [20] proposed the MetaQNN method to generate DCNN architectures based on reinforcement learning. In this method, a Q-learning agent explores and exploits the space of model architectures with the -greedy strategy and experience replay. Various evolutionary algorithms have long been applied to the design of neural networks even before they became deep. Evolutionary optimization of neural network architectures can be traced back to NEAT, a seminal work done by Stanley and Miikkulainen [7], which stores every neuron and connection in the DNA and generates architectures by mutation which can be divided into three kinds: (1) modifying a weight, (2) adding a connection between existing connections, and (3) inserting a neuron while splitting an existing connection. Based on this, Miikkulainen et al. [21] proposed the CoDeepNEAT algorithm, in which a population of chromosomes with minimal complexity is created and the structure is added to the graph incrementally through mutation over generations. Suganuma et al. [22] proposed an approach to design DCNN architecture based on genetic programming. The DCNN architecture is encoded by Cartesian genetic programming as directed acyclic graphs with a two-dimensional grid defined on the computational neurons. As one of

the most prevalent evolutionary algorithms, the genetic algorithm (GA) [23] uses heuristics-guided search that simulates the process of natural selection and survival of the fittest. Xie et al. [8] encoded the architecture of each DCNN into a fixed-length binary string, and thus proposed a genetic DCNN (GDCNN) method to generate DCNN architectures automatically. In this method, the coding of DCNN architecture only contains the operation between pooling layers and the fully connected layers are not encoded. The DCNNs generated by this method, however, have a limited number of layers, fixed filter size and fixed filter number. Real et al. [10] evolved DCNN architectures on the CIFAR-10 and CIFAR-100 datasets using GA with the DNA based coding scheme, which is same as that in NEAT. DCNN architectures evolve through mutation operations and the conflicts in filter sizes are handled by reshaping non-primary edges with zeroth order interpolation. All DCNN architectures generated by this method are fully trained. This method uses a distributed algorithm with more than 250 computers. Desell [9] proposed the EXACT method, which can evolve DCNNs with flexible connection and filter size based on GA with an asynchronous evolution strategy. The EXACT method used over 4,500 volunteered computers and trained over 120,000 DCNNs on the MNIST dataset.

3. Method

The proposed genetic DCNN designer evolves and improves a population of individuals, each encoding an admissible DCNN architecture. The population is randomly initialized. The fitness of each individual is evaluated as the performance of the DCNN encoded by that individual in a specific image classification problem. Based on the current generation, a new generation is produced by performing a combination of redefined genetic operators, including selection, crossover and mutation, to improve the overall fitness of individuals. The evolution is performed iteratively on a generation-by-generation basis until meting a stopping criterion is fulfilled or the number of generations attains a pre-defined number. The diagram that summarizes this algorithm is shown in

Fig. 1.

3.1. Encoding scheme

Our DCNN architecture encoding scheme is inspired by the representation of locus on a chromosome. A chromosome can be divided into two components: p-arm and q-arm. The ordered list of loci known for a particular genome is called a gene map. Intuitively, various operations in a DCNN can be viewed as the loci on a chromosome, and thus the architecture of a DCNN can be encoded as a gene map, where convolutional blocks compose a convolutional arm and fully connected blocks compose a fully connected arm. A convolutional block contains five operations: convolution, batch normalization, pooling, activation and drop out. The convolutional operation has two parameters: the number of filters N and the size of filters S. The operations of batch normalization, pooling, activation and dropout are denoted by B, P, A and D, respectively. Thus, a convolutional block contains loci in sequence and can be encoded as $[NSPBAD]$. Similarly, a fully connected block contains loci in sequence and can be encoded as $[NBAD]$, where N is the number of neurons in a fully connected layer, B represents the operation of batch normalization, A represents the activation function, D represents the operation of drop out. Besides, the optimizer also has an impact on the DCNN architecture, and is represented as O. Table 1 shows the value range at each locus of a network code: (1) the number of filters N ranges from 16 to 512; (2) the alternative sizes of filters are 3x3, 5x5 or 7x7; (3) the pooling operation P may take a value of 0 (without pooling), 1 (max pooling) or 2 (average pooling), all using the same stride of 2; (4) the batch normalization operation B takes a value from to indicate whether adopting this operation or not; (5) the value of the activation function A ranges from 0 to 5, representing the TReLU [24], ELU [25], PReLU [26], LeakyReLU [27], ReLU [28] and Softmax functions, respectively; (6) the dropout operation D also takes a value from [0, 0.5] ; and (7) the value of optimizer O ranges from 0 to 6, representing the SGD [29], RMSprop [30], Adagrad [30], Adadelta [31], Adam [32], Adamax [32] and Nadam [33], respectively.

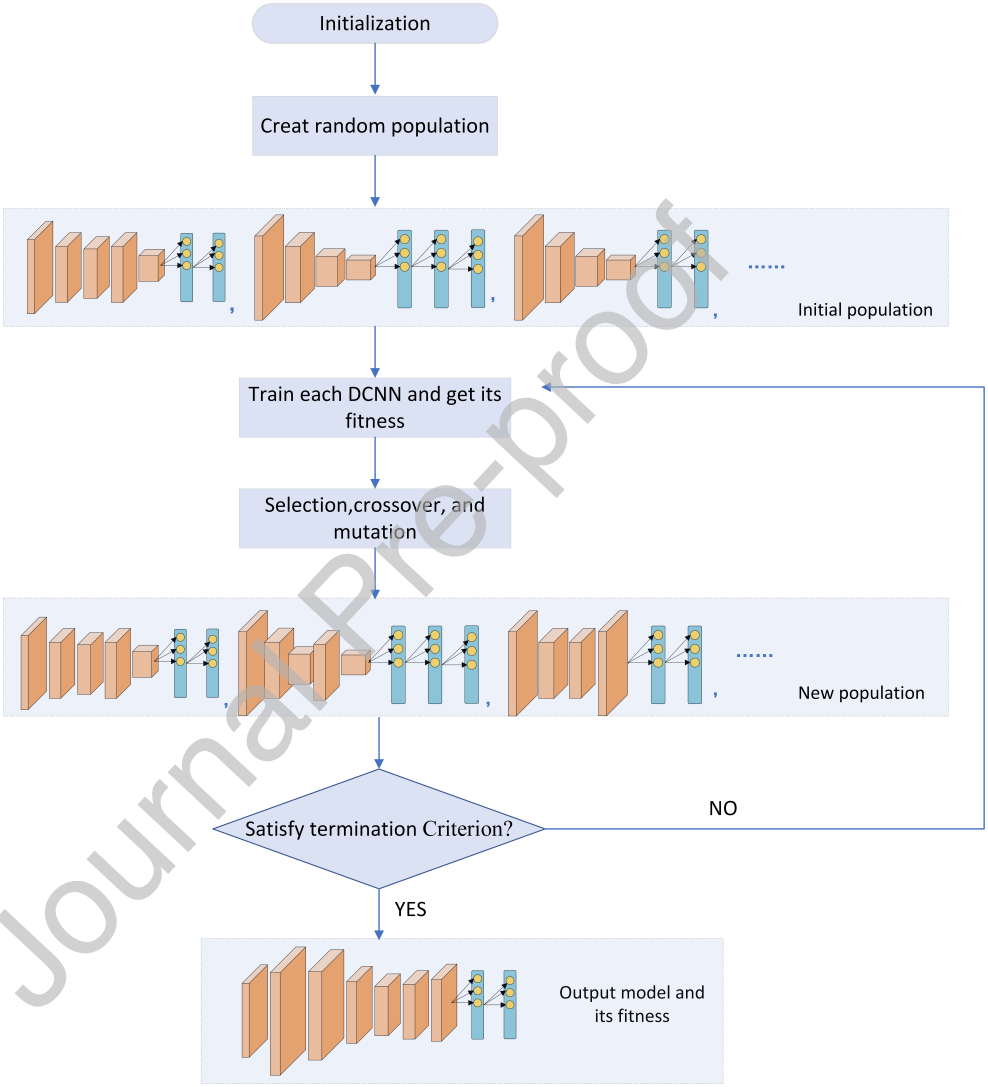


Fig. 1. Diagram of the proposed genetic DCNN designer

Table 1. Value range of each at each locus of a network code

Code	*Valuescale*
N	$[16, 512]$
S	$3, 5, 7$
P	$0, 1, 2$
B	$0, 1$
A	$0, 1, 2, 3, 4$
D	$[0, 0.5]$
O	$0, 1, 2, 3, 4, 5, 6$

With this coding scheme, a DCNN can be decomposed into a convolutional arm that contains a sequence of convolutional blocks and a fully connected arm that contains a sequence of fully connected blocks . Taking the VGG-19 model shown in Fig. 1 as the case study, it can be presented as $\left\{\{[NSPBAD]_i\}_{i=1}^{16}, \{[NBAD]_j\}_{j=1}^{3}, 0\right\}$ and be decoded as {[64, 3, 0, 1, 4, 0], [64, 3, 2, 1, 4, 0], [128, 3, 0, 1, 4, 0], [128, 3, 2, 1, 4, 0], [256, 3, 0, 1, 4, 0], [256, 3, 0, 1, 4, 0], [256, 3, 0, 1, 4, 0], [256, 3, 2, 1, 4, 0], [512, 3, 0, 1, 4, 0], [512, 3, 2, 1, 4, 0], [512, 3, 0, 1, 4, 0], [512, 3, 2, 1, 4, 0], [512, 3, 0, 1, 4, 0], [512, 3, 2, 1, 4, 0], [512, 3, 0, 1, 4, 0], [512, 3, 2, 1, 4, 0], [4096, 1, 4, 0], [4096, 1, 4, 0], [1000, 0, 5 , 0], 0 }.

3.2. Initialization

A DCNN with N_n^C convolutional blocks and N_n^F fully connected blocks can be presented as

$$S_n = \left\{\{[NSPBAD]_i\}_{i=1}^{N_n^C}, \{[NBAD]_j\}_{j=1}^{N_n^F}, 0\right\} \tag{1}$$

whose code-length is

$$L_n = N_n^C l_c + N_n^F l_f \tag{2}$$

At the initialization step, we set $N_{c_n} \in [1, 20]$ and $N_{f_n} \in [1, 3]$, and randomly sampled a population of DCNN architectures, denoted by $\{S_n\}_{n=1}^{T}$ where T

represent the size of population and we set it as 50 and the max generations is 40. There are two main reasons for these setting numbers. First of all, since our limited computing resource , we need to balance the epochs we run for each model and the new models we explore. In addition, we found 40 generations were enough for our experiments and the best performance we obtained changes little after 10 generations.

3.3. Selection

Before producing the next generation, each individual's fitness is evaluated. Based on the fitness ranking, we use the elitism roulette wheel selection scheme [34] to select individuals for producing the next generation. Specifically, we first select 10% of top ranking individuals as an elitist set and directly copy them to the next generation. Then we select each individual according to the following probability

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j} \tag{3}$$

where f_i is the fitness of individual i, and N is the number of individuals in the population. Each pair of selected individuals are then processed by crossover and mutation to generate new individuals for the next generation. The roulette wheel selection will not stop until the size of the next generation is equal to that of the current one.

3.4. Crossover

For a pair of selected DCNNs S_i and S_j, we randomly locate a cross point on each of them, which breaks the DCNN architecture into two segments. By swapping the segments of those two DCNN architectures, two new DCNNs $S_i^{'}$ and $S_j^{'}$ are generated, whose depths may be different from the depths of their parents. If the cross point k_i is located within the m_i-th convolutional blocks $[NSPBAD]_{m_i}$ on the convolutional arm of the DCNN S_i, its location can be expressed as $(m_i - 1)l_c + x$. Then, we require that the other cross point also locates on the convolutional arm of the DCNN S_j and its location can be

expressed as $(m_j - 1)l_c + x$. After the crossover, the code-length of two newly generated DCNNs are

$$\begin{cases} L_i^{'} = L_i + (m_i - m_j)l_c \\ L_j^{'} = L_j + (m_j - m_i)l_c \end{cases} \tag{4}$$

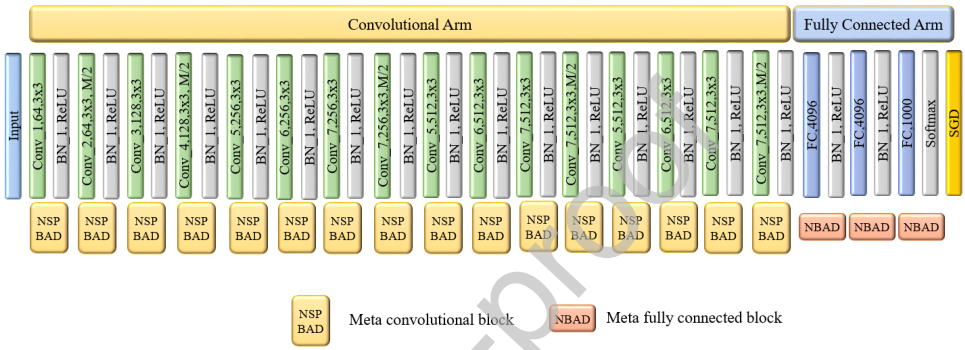


Fig. 2. The blocks of VGGNet

A typical example is shown Fig. 3, where one cross point k_i is located at $3l_c+1$ in a DCNN with 8 learnable layers and the other cross point k_j is located at $5l_c+1$ in a DCNN with 11 learnable layers. After the crossover, we obtained a DCNN with 9 learnable layers and a DCNN with 10 learnable layers. If the cross point k_i is located within the m_i-th fully connected blocks $[NSPBAD]_{m_i}$ on the fully connected arm of the DCNN S_i, its location can be expressed as . Then, we require that the location of the other cross point is also on the fully connected arm of the DCNN and can be expressed as . After the crossover, the code-length of two newly generated DCNNs are

$$\begin{cases} L_i^{'} = L_i + (m_i - m_j)l_f \\ L_j^{'} = L_j + (m_j - m_i)l_f \end{cases} \tag{5}$$

3.5. Mutation

To maintain genetic diversity from one generation to the next, the mutation operation is applied to each individual, which alters an DCNN architecture S_n

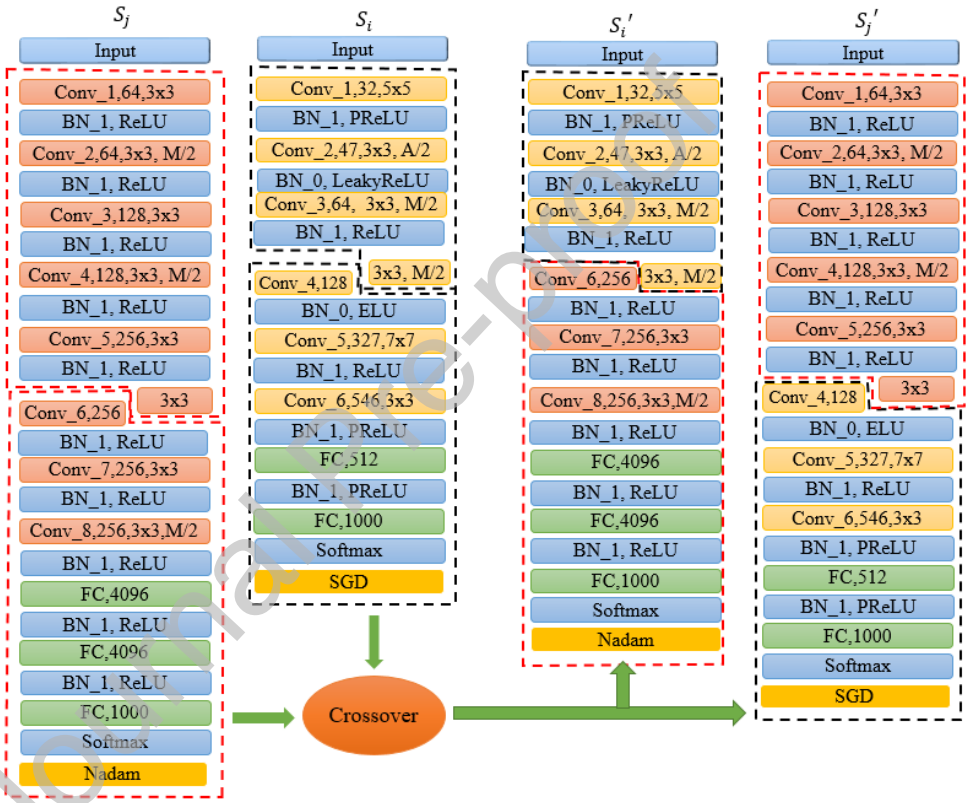


Fig. 3. An example of crossover on the convolutional arm

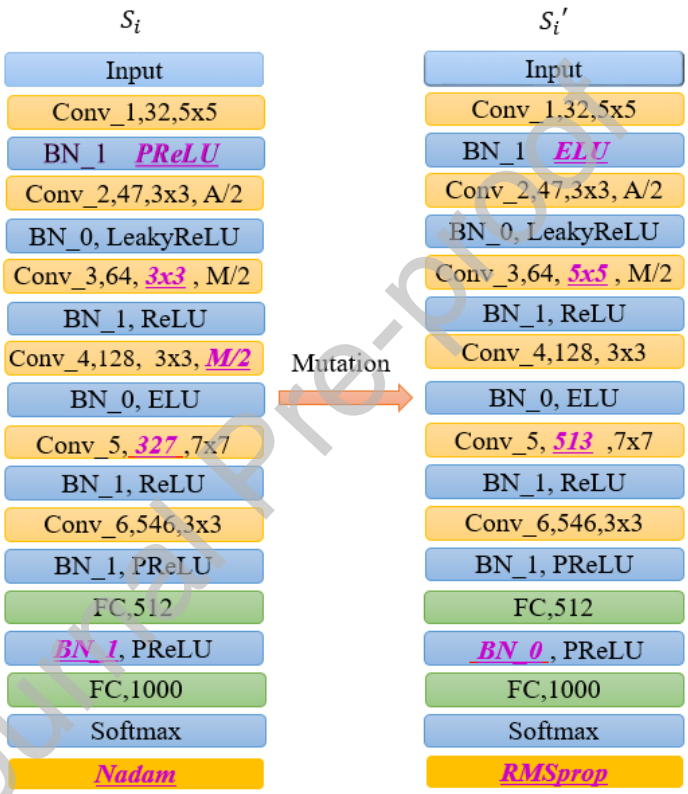


Fig. 4. An example of mutation

by resampling each locus evenly and independently from the value range with a probability of q_m. To accelerate the generation of new architectures, the value of q_m is evenly sampled from the range $[\frac{8}{L_n}, 0.5]$ for each individual S_n. An illustrative example of mutation is shown in Fig. 4, where six loci in the code of S_i were mutated. After the mutation, the activation function in the first convolutional block was changed from PReLU to ELU, the kernel size in the third convolutional block was changed from 3x3 to 5x5, the max-pooling layer in the fourth convolutional block was removed, the number of kernels in the fifth convolutional block was changed from 327 to 513, the batch normalization in the first fully connected block was removed, and the optimizer was changed from Nadam to RMSprop.

4. Dataset

The datasets used for this study include the MNIST [11], EMNIST-Digits [12], EMNIST-Letters [12], Fashion-MNIST [13], CIFAR10 [14] and CIFAR100 [14]. The MNIST dataset defines a handwritten digit recognition task. The EMNIST-Letters and EMNIST-Digits datasets were derived from the NIST Special Database 19, which represents the final collection of handwritten characters, containing additional handwritten digits and an extensive collection of uppercase and lowercase handwritten letters. The EMNIST-Letters dataset merges all the uppercase and lowercase classes and gets a dataset with 26 classes comprising [a-z]. The CIFAR10 dataset is a subset of the 80-million tiny image database. In this dataset, both training and testing images are uniformly distributed over 10 categories. CIFAR100 is an extension to CIFAR10, and it contains 100 categories. Table 2 provides a brief summary of these six datasets, including the number of training images, testing images and classes, image size and color.

Table 2. The summary of the datasets

Infomation / Class / Name	*Trainning*	*Testing*	*Classes*	*Size*	*Color*
MNIST	60,000	10,000	10	$28 * 28$	*grayscale*
$EMNIST - Letters$	124,800	20,800	26	$28 * 28$	*grayscale*
$EMNIST - Digits$	240,000	4,000	10	$28 * 28$	*grayscale*
$Fashion - MNIST$	60,000	10,000	10	$28 * 28$	*grayscale*
*CIFAR*10	50,000	10,000	10	$32 * 32$	*grayscale*
*CIFAR*100	50,000	10,000	100	$32 * 32$	*grayscale*

5. Experiments and Results

For each dataset, the proposed genetic DCNN designer was used to generate a DCNN that can solve the corresponding image classification problem with a satisfying accuracy. In each experiment,We randomly split training data of every dataset into two part: 10% data form the validation set and the others were used for training. To enlarge the training dataset, we employed the ImgeDataAugmentation tools provided by Keras[37], which include cropping by no more than 12.5% of width or height, flipping vertically or horizontally, and random rotation by no more than 5 degrees, to generate augmented copies for training images in each batch.

During the training of each DCNN, we use the cross-entropy loss function for all the DCNN architectures, fixed the maximum iteration number to 100, chose the min-batch stochastic gradient decent with a batch size of 256, set the learning rate as small as 0.0001 and further reduce it exponentially.

5.1. DCNN Designed for MNIST

The best DCNN architecture generated by the proposed genetic DCNN designer in ten generations for the MNIST dataset consists of three convolutional blocks and four fully connected blocks (see Fig. 5). The first convolutional block

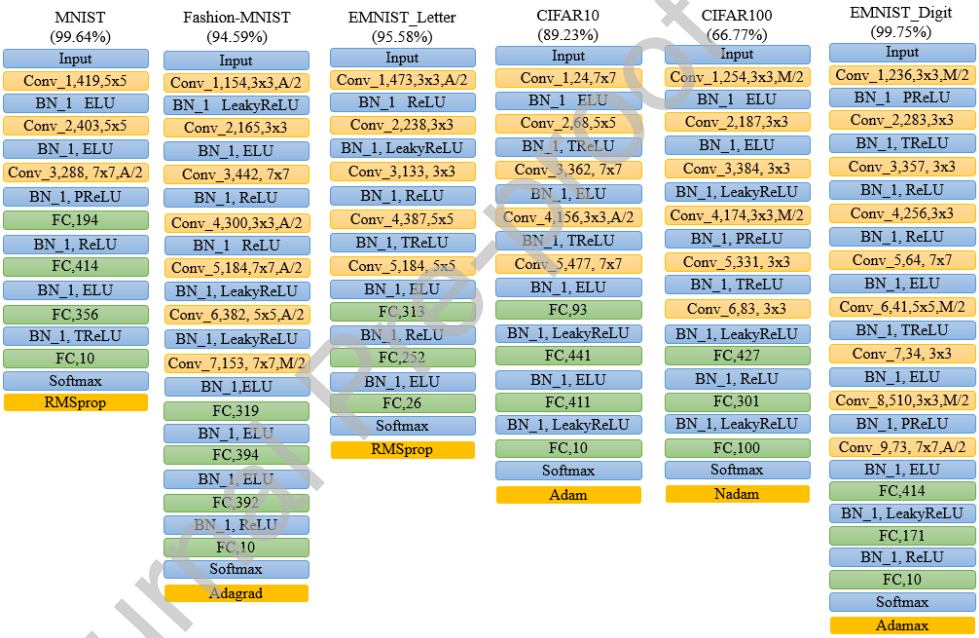


Fig. 5. The DCNN architectures generated by proposed algorithm

Table 3. Classification accuracy of different DCNNs on the six datasets (%)

DCNN Models	MNIST	Fashion-MNIST	EMNIST-Letter	EMNIST-Digit	CIFAR10	CIFAR100
AlexNet[1]	98.81	86.43	89.36	99.21	82.53	60.53
VGGNet[2]	99.32	90.45	94.62	99.62	84.62	64.37
ResNet[4]	99.37	94.39	94.44	99.63	**90.61**	**69.59**
CapsuleNet[35]	99.57	90.03	91.58	99.37	89.40	*
EXACT[9]	98.32	*	*	*	*	*
HighwayNet[5]	*	*	*	*	89.16	67.60
Maxout[36]	*	*	*	*	*	61.40
Designed by our DCNN designer	**99.64**	**94.60**	**95.58**	**99.75**	89.32	66.77

contains 419 filters of size 5x5, no pooling layer, batch normalization, ELU activation function and 20% dropout. The second convolutional block contains 403 filters of size 5x5, no pooling layer, batch normalization, ELU activation function and 0% dropout. The third convolutional block contains 288 filters of size 7x7, an average-pooling layer with stride of 2, batch normalization, PReLU activation function and 0% dropout. The first fully connected block contains 194 neurons with the ReLU activation function, batch normalization and 30% dropout. The second fully connected block contains 414 neurons with the ELU activation function, batch normalization and 45% dropout. The third fully connected block contains 356 neurons with the TReLU activation function, batch normalization and 5% dropout. The last fully connected block is the output layer, which has 10 neurons with the softmax activation function. The optimizer is Adamax. We compared the classification accuracy achieved by this architecture to the accuracy of AlexNet [1], VGGNet [2], ResNet [4] and Capsule Net [35] and the DCNNs generated by GCNN [8] and EXACT [9] in Table 3. It shows that the DCNN created by our genetic DCNN designer achieved the highest accuracy of 99.64%, higher than the accuracy (98.32%) of the DCNN designed by EXACT. The plot of highest classification accuracy achieved in each generation (see Fig. 6) shows that the performance of best DCNN architecture generated by our genetic DCNN designer has become stable.

5.2. DCNN Designed for Fashion-MNIST

The best DCNN architecture generated by the proposed genetic DCNN designer in ten generations for the Fashion-MNIST dataset consists of seven convolutional blocks and four fully connected blocks (see Fig. 5). The first convolutional block contains 154 filters of size 5x5, average pooling, batch normalization, LeakyReLU activation function and 0% drop out. The second convolutional block contains 165 filters of size 3x3, no pooling layer, batch normalization, ELU activation function and 50% drop out. The third convolutional block contains 442 filters of size 7x7, no pooling layer, batch normalization, ReLU activation function and 40% drop out. The forth convolutional block contains 300 filters of size 3x3, average pooling layer, batch normalization, ReLU activation function and 5% drop out. The fifth convolutional block contains 184 filters of size 7x7, average pooling layer, batch normalization, LeakyReLU activation function and 25% drop out. The sixth convolutional block contains 382 filters of size 5x5, average pooling layer, LeakyReLU activation function and 40% drop out. The seventh convolutional block contains 153 filters of size 7x7, max pooling layer, batch normalization, ELU activation function and 40% drop out. The first fully connected block contains 319 neurons with the ELU activation function and the 5% drop out. The second fully connected block contains 394 neurons with the ELU activation function and 45% drop out. The third fully connected block contains 392 neurons with ReLU activation function and 50% drop out. The last fully connected block is the output layer, which has 10 neurons with the softmax activation function. The optimizer is Adagrad. We compared the classification accuracy achieved by this architecture to the accuracy of AlexNet [1], VGGNet [2], ResNet [4] and Capsule Net [35] in the second column of Table 3. It shows that the DCNN generated by our genetic DCNN designer achieved the highest accuracy of 94.60%. The plot of highest classification accuracy achieved in each generation (see Fig. 6) shows that best DCNN architecture generated by our genetic DCNN designer is still improving.

5.3. DCNN Designed for EMNIST-Letters

The best DCNN architecture generated by the proposed genetic DCNN designer in ten generations for the EMNIST-Letters dataset consists of five convolutional blocks and three fully connected blocks (see Fig. 5). The first convolutional block contains 473 filters of size 3x3, average pooling layer, batch normalization, ReLU activation function and 15% drop out. The second convolutional block contains 238 filters of size 3x3, no pooling layer, batch normalization, LeakyReLU activation function and 20% drop out. The third convolutional block contains 133 filters of size 3x3, no pooling layer, batch normalization, ReLU activation function and 10% drop out. The forth convolutional block contains 387 filters of size 3x3, no pooling layer, batch normalization, TReLU activation function and 10% drop out. The fifth convolutional block contains 187 filters of size 5x5, no pooling layer, batch normalization, ELU activation function and 50% drop out. The first fully connected block contains 313 neurons with ReLU activation function, batch normalization and 20% dropout. The second block contains 252 neurons with ELU activation function, batch normalization and 20% drop out. The last fully connected block is the output layer, which has 26 neurons with the softmax activation function. The optimizer is RMSprop. We compared the classification accuracy achieved by this architecture to the accuracy of AlexNet [1], VGGNet [2], ResNet [4] and Capsule Net [35] in the third column of Table 3. It reveals that the DCNN generated by our genetic DCNN designer achieved the highest accuracy of 95.58%. However, the plot of highest classification accuracy achieved in each generation (see Fig. 6) shows that best DCNN architecture generated by our genetic DCNN designer is still improving. The plot of highest classification accuracy achieved in each generation (see Fig. 6) shows that best DCNN architecture generated by our genetic DCNN designer is still improving.

5.4. DCNN Designed for EMNIST-Digits

The best DCNN architecture generated by the proposed genetic DCNN designer in ten generations for the EMNIST-Digits dataset consists of nine convo-

lutional blocks and three fully connected blocks (see Fig. 5). The first convolutional block contains 236 filters of size 3x3, max pooling layer, batch normalization, PReLU activation function and 0% drop out. The second convolutional block contains 283 filters of size 5x5, no pooling layer, batch normalization, TReLU activation function and 30% drop out. The third convolutional block contains 357 filters of size 3x3, no pooling layer, batch normalization, ReLU activation function and 35% drop out. The forth convolutional block contains 256 filters of size 3x3, no pooling layer, batch normalization, ReLU activation function and 15% drop out. The fifth convolutional block contains 64 filters of size 7x7, no pooling layer, batch normalization, LeakyReLU activation function and 30% drop out. The sixth convolutional block contains 41 filters of size 5x5, max pooling layer, batch normalization, TReLU activation function and 20% drop out. The seventh convolutional block contains 34 filters of size 3x3, no pooling layer, batch normalization, ELU activation function and 25% drop out is 0.25. The eighth convolutional block contains 510 filters of size 3x3, max pooling layer, batch normalization, PReLU activation function and 45% drop out. The ninth convolutional block contains 73 filters of size 7x7, average pooling layer, batch normalization, ELU activation function and 5% drop out. The first fully connected block contains 414 neurons with batch normalization, PReLU activation function and 20% dropout. The second block contains 171 neurons with batch normalization, ReLU activation function and 40% drop out. The last fully connected block is the output layer, which has 10 neurons with the softmax activation function. The optimizer is Adamax. We compared the classification accuracy achieved by this architecture to the accuracy of AlexNet [1], VGGNet [2], ResNet [4] and Capsule Net [35] in the fourth column of Table 3. It reveals that the DCNN generated by our genetic DCNN designer achieved the highest accuracy of 99.75%. The plot of highest classification accuracy achieved in each generation (see Fig. 6) shows that the performance of best DCNN architecture generated by our genetic DCNN designer has become stable.

5.5. DCNN Designed for CIFAR10

The best DCNN architecture generated by the proposed genetic DCNN designer in ten generations for the CIFAR10 dataset consists of five convolutional blocks and four fully connected blocks (see Fig. 5). The first convolutional block contains 24 filters of size 7x7, no pooling layer, batch normalization, ELU activation function and 15% drop out. The second convolutional block contains 68 filters of size 5x5, no pooling layer, TReLU activation function and 10% drop out. The third convolutional block contains 362 filters of size 7x7, no pooling layer, batch normalization, ELU activation function and 0% drop out. The forth convolutional block contains 156 filters of size 3x3, average pooling, batch normalization, TReLU activation function and 0% drop out. The fifth convolutional block contains 477 filters of size 7x7, average pooling, batch normalization, ELU activation function and 0% drop out. The first fully connected block contains 93 neurons with batch normalization, LeakyReLU activation function and the 5% of dropout. The second block contains 441 neurons with batch normalization, ELU activation function and 50% drop out. The third block contains 411 neurons with batch normalization, LeakyReLU activation function and the 45% drop out. The last fully connected block is the output layer, which has 10 neurons with the softmax activation function. The optimizer is Adam. We compared the classification accuracy achieved by this architecture to the accuracy of AlexNet [1], VGGNet [2], ResNet [4], HIGHWAY-32 [34] and Capsule Net [35] in Table 3. It reveals that the DCNN generated by our genetic DCNN designer achieved an image classification accuracy of 89.32%, which ranks the third. The ResNet with 20 residual blocks and the Capsule Net with an ensemble of seven capsule models obtained the top two accuracies. However, the plot of highest classification accuracy achieved in each generation (see Fig. 6) shows that best DCNN architecture generated by our genetic DCNN designer is still improving.

5.6. DCNN Designed for CIFAR100

The best DCNN architecture generated by the proposed genetic DCNN designer in ten generations for the CIFAR100 dataset consists of six convolutional

blocks and three fully connected blocks (see Fig. 5). The first convolutional block contains 254 filters of size 3x3, max pooling layer, batch normalization, ELU activation function and 0% drop out. The second convolutional block contains 187 filters of size 3x3, no pooling layer, batch normalization, ELU activation function and 0% drop out. The third convolutional block contains 384 filters of size 3x3, no pooling layer, batch normalization, LeakyReLU activation function and 30% drop out. The forth convolutional block contains 174 filters of size 3x3, max pooling, batch normalization, PReLU activation function and 20% drop out. The fifth convolutional block contains 331 filters of size 3x3, no pooling layer, batch normalization, LeakyReLU activation function and 40% drop out. The sixth convolutional block contains 83 filters of size 3x3, no pooling layer, batch normalization, LeakyReLU activation function and 35% drop out. The first fully connected block contains 427 neurons, batch normalization, ReLU activation function and 35% drop out. The second fully connected block contains 301 neurons with batch normalization, LeakyReLU activation function and 15% drop out. The last fully connected block is the output layer, which has 10 neurons with the softmax activation function. The optimizer is Nadam. We compared the classification accuracy achieved by this architecture to the accuracy of AlexNet [1], VGGNet [2], ResNet [4], Highway Net [34] and Maxout [35] in Table 3. It reveals that the DCNN generated by our genetic DCNN designer achieved an image classification accuracy of 66.7%, which ranks the third. The ResNet with 20 residual blocks and the Highway Net produced much more accurate classification. However, the plot of highest classification accuracy achieved in each generation (see Fig. 6) shows that best DCNN architecture generated by our genetic DCNN designer is still fast improving.

In summary, after evolving ten generations, the proposed genetic DCNN designer generated DCNNs, which performed better than several state-of-the-art DCNNs on three out of six datasets and achieved comparable performance on other datasets. It should be note that our genetic DCNN designer is prone to produce DCNNs with less layers.

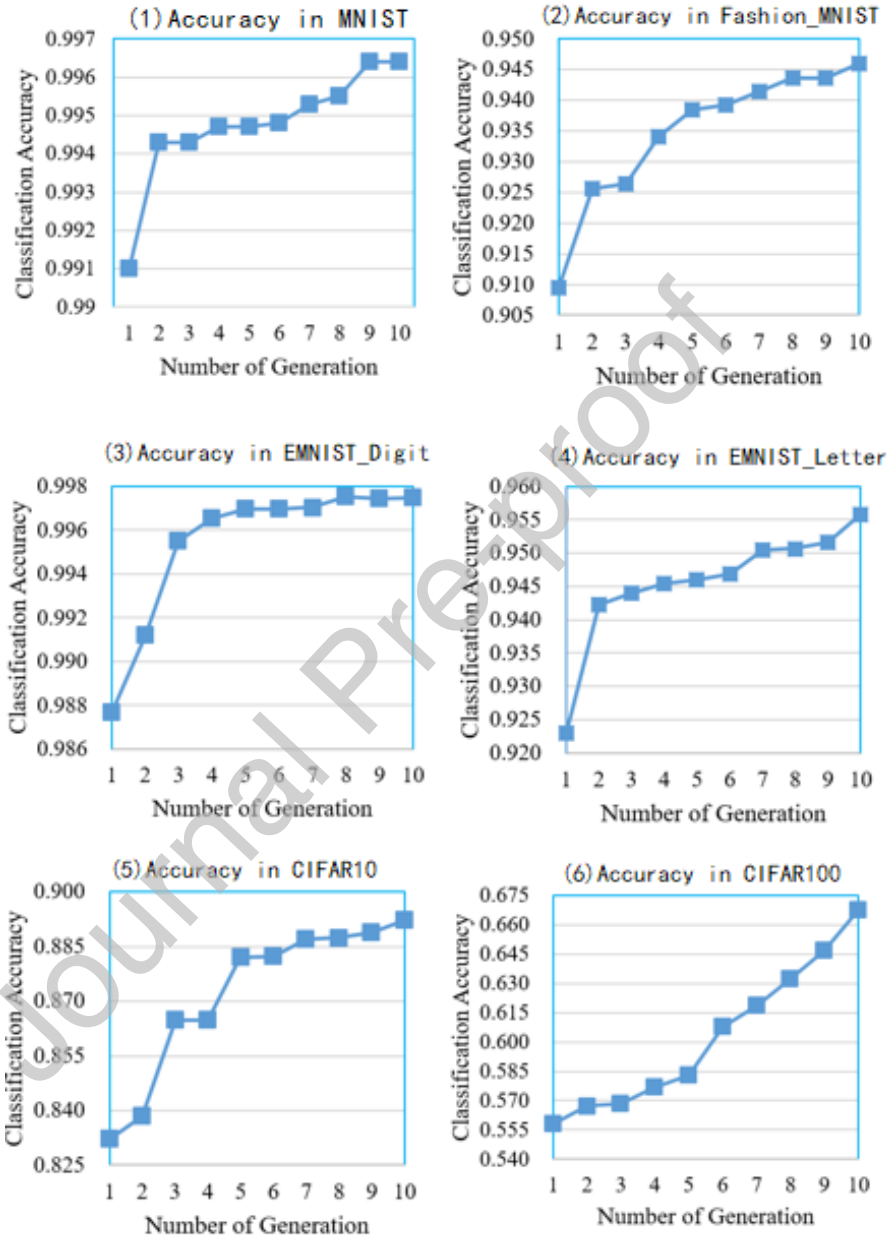


Fig. 6. The DCNN architectures generated by proposed algorithm

6. Discussion on Complexity

Although the training of each generated DCNN is limited to 100 epochs, the proposed genetic DCNN designer still has an extremely high computational and space complexity, due to storing and evaluating a large number of DCNN structures. Table 4 shows that it takes about 3, 5, 18, 12, 11 and 11 GPU-days on average to evolve the DCNN structure for one generation on the MNIST, Fashion-MNIST, EMNIST-Digits, EMNIST-Letters, CIFAR10 and CIFAR100 dataset, respectively. The experiments were conducted using a server with 2 Intel Xeon E5-2678 V3 2.50 GHz and 1 NVIDIA Tesla TiTanXp GPU, 512 GB Memory, 240 GB SSD and Matlab 2017a.

Table 4. Average time cost for running the genetic DCNN designer one generation

Data set	*MNIST*	*Fashion MNIST*	*EMNIST Digit*	*EMNIST Letter*	*CIFAR 10*	*CIAFR 100*
GPUday	3	5	18	12	12	11

7. Conclusion

In this paper, we propose the genetic DCNN designer, an autonomous learning algorithm that can generate DCNN architecture automatically based on the genetic algorithm and data available for the specific image classification problem. This designer is prone to creating DCNNs with less layers. The experimental results on the MNIST, Fashion_MNIST, EMNIST_Digit, EMNIST_Letter, CIFAR10 and CIFAR100 datasets suggest that the proposed algorithm is able to produce automatically DCNN architectures, which are comparative or even outperform the state-of-the-art DCNN models.

Acknowledgment

This work was supported in part by the Science and Technology Innovation Committee of Shenzhen Municipality, China, under Grants JCYJ20180306171334997,

in part by the National Natural Science Foundation of China under Grants 61771397, and in part by the Innovation Foundation for Doctor Dissertation of Northwestern Polytechnical University under Grants CX201835.

Conflict of Interest Statement

The authors declare that they do not have any financial or nonfinancial conflict of interests.

References

[1] A. Krizhevsky, I. Sutskever,G. E. Hinton, Imagenet classification with deep convolutional neural networks, Advances in neural information processing systems (2012) 1097-1105.

[2] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, arXiv:1409.1556 (2014).

[3] C. Szegedy et al., Going deeper with convolutions, Proceedings of the IEEE conference on computer vision and pattern recognition (2015) 1-9.

[4] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, Proceedings of the IEEE conference on computer vision and pattern recognition(2016) 770-778.

[5] G. Huang, Z. Liu, K. Q. Weinberger, L. van der Maaten, Densely connected convolutional networks, arXiv preprint arXiv:1608.06993, 2016.

[6] D. Silver et al., Mastering the game of Go without human knowledge, Nature 550 (7676) (2017) 354-359.

[7] K. O. Stanley, R. Miikkulainen, Evolving neural networks through augmenting topologies, Evolutionary computation 10(2)(2002) 99-127.

[8] L. Xie, A. Yuille, Genetic CNN.,arXiv preprint arXiv:1703.01513 (2017).

[9] T. Desell, Large Scale Evolution of Convolutional Neural Networks Using Volunteer Computing, arXiv preprint arXiv:1703.05422 (2017).

[10] E. Real et al., Large-scale evolution of image classifiers, arXiv:1703.01041 (2017).

[11] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE 86 (11) (1998) 2278-2324.

[12] G. Cohen, S. Afshar, J. Tapson, A. van Schaik, EMNIST: an extension of MNIST to handwritten letters, arXiv:1702.05373 (2017).

[13] H. Xiao, K. Rasul, R. Vollgraf, Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, arXiv:1708.07747 (2017).

[14] A. Krizhevsky, G. Hinton, Learning multiple layers of features from tiny images, (2009).

[15] J. Jin, Z. Yan, K. Fu, N. Jiang, C. Zhang, Neural Network Architecture Optimization through Submodularity and Supermodularity, arXiv:1609.00074 (2017).

[16] C. Fernando et al., Pathnet: Evolution channels gradient descent in super neural networks, arXiv:1701.08734 (2017).

[17] D. Ha, A. Dai, and Q. V. Le, HyperNetworks, arXiv:1609.09106 (2016).

[18] B. De Brabandere, X. Jia, T. Tuytelaars, L. Van Gool, Dynamic filter networks, Neural Information Processing Systems (NIPS) (2016).

[19] B. Zoph, Q. V. Le, Neural architecture search with reinforcement learning, arXiv:1611.01578 (2016).

[20] B. Baker, O. Gupta, N. Naik, R. Raskar, Designing neural network architectures using reinforcement learning, arXiv:1611.02167 (2016).

[21] R. Miikkulainen et al., Evolving Deep Neural Networks, arXiv:1703.00548 (2017).

[22] M. Suganuma, S. Shirakawa, T. Nagao, A Genetic Programming Approach to Designing Convolutional Neural Network Architectures,arXiv:1704.00764 (2017).

[23] J. Horn, N. Nafpliotis, D. E. Goldberg, A niched Pareto genetic algorithm for multiobjective optimization, Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence 1 (1994) 82-87.

[24] R. Memisevic and D. Krueger, Zero-bias autoencoders and the benefits of co-adapting features, stat 1050 (2014) 13.

[25] D.-A. Clevert, T. Unterthiner, S. Hochreiter, Fast and accurate deep network learning by exponential linear units (elus), arXiv:1511.07289 (2015).

[26] K. He, X. Zhang, S. Ren, J. Sun, Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, Proceedings of the IEEE international conference on computer vision (2015) 1026-1034.

[27] A. L. Maas, A. Y. Hannun, A. Y. Ng, Rectifier nonlinearities improve neural network acoustic models, Proc. ICML 30 (1) (2013).

[28] X. Glorot, A. Bordes, and Y. Bengio, Deep sparse rectifier neural networks, Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (2011) 315-323.

[29] L. Bottou, Large-scale machine learning with stochastic gradient descent, Proceedings of COMPSTAT'2010: Springer (2010) 177-186.

[30] J. Duchi, E. Hazan, Y. Singer, Adaptive subgradient methods for online learning and stochastic optimization,Journal of Machine Learning Research 12 (Jul) (2011) 2121-2159.

[31] M. D. Zeiler, ADADELTA: an adaptive learning rate method,arXiv:1212.5701 (2012).

[32] D. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv:1412.6980 (2014).

[33] I. Sutskever, J. Martens, G. Dahl, G. Hinton, On the importance of initialization and momentum in deep learning, International conference on machine learning (2013) 1139-1147.

[34] D. E. Goldberg K. Deb, A Comparative Analysis of Selection Schemes Used in Genetic Algorithms, Foundations of Genetic Algorithms 1 (1991) 69-93.

[35] S. Sabour, N. Frosst, G. E. Hinton, Dynamic Routing Between Capsules, arXiv:1710.09829 (2017).

[36] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, Y. Bengio, Maxout networks, arXiv:1302.4389 (2013).

[37] Chollet, F., et al., Keras. GitHub repository (2015). https://keras.io/zh/

Benteng Ma received her B.E. in Computer Science from Northwestern Polytechnical University (NPU), Xian, China, in 2016. He is currently working toward the Ph.D. degree at the School of Computer Science and Engineering, NPU. His research areas include computer vision, deep learning, and neural architecture search.

Xiang Li received his B.E. in Computer Science from Northwestern Polytechnical University (NPU), Xian, China, in 2018. He is currently a Master student at the School of Computer Science and Engineering, NPU. His research areas include medical image analysis, deep learning, and neural architecture search.

Prof. Yong Xia received the B.E., M.E., and Ph.D. degrees in computer science and technology from Northwestern Polytechnical University (NPU), Xian, China, in 2001, 2004, and 2007, respectively. He is currently a Professor at the School of Computer Science and Engineering, NPU. His research interests include medical image analysis, computer-aided diagnosis, pattern recognition, machine learning, and data mining.

Prof. Yanning Zhang received her B.S. Degree from Dalian University of Science and Engineering in 1988, M.S. and Ph.D. degrees from Northwestern

Polytechnical University (NPU) in 1993 and 1996, respectively. She is presently a Professor of School of Computer Science and Engineering, NPU. Her research work focuses on signal and image processing, computer vision and pattern recognition. She has published over 200 papers in leading international journals and conferences.