Autonomous Deep Learning: A Genetic DCNN Designer for Image Classification

Benteng Ma   Yong Xia*

School of Computer Science, Northwestern Polytechnical University

Xian 710072, China

yxia@nwpu.edu.cn

Abstract

Recent years have witnessed the breakthrough success of deep convolutional neural networks (DCNNs) in image classification and other vision applications. Although freeing users from the troublesome handcrafted feature extraction by providing a uniform feature extraction-classification framework, DCNNs still require a handcrafted design of their architectures. In this paper, we propose the genetic DCNN designer, an autonomous learning algorithm can generate a DCNN architecture automatically based on the data available for a specific image classification problem. We first partition a DCNN into multiple stacked meta convolutional blocks and fully connected blocks, each containing the operations of convolution, pooling, fully connection, batch normalization, activation and drop out, and thus convert the architecture into an integer vector. Then, we use refined evolutionary operations, including selection, mutation and crossover to evolve a population of DCNN architectures. Our results on the MNIST, Fashion-MNIST, EMNIST-Digit, EMNIST-Letter, CIFAR10 and CIFAR100 datasets suggest that the proposed genetic DCNN designer is able to produce automatically DCNN architectures, whose performance is comparable to, if not better than, that of state-of-the-art DCNN models.

1. Introduction

Deep convolutional neural networks (DCNNs), such as AlexNet [20], VGGNet [27], GoogLeNet [32], ResNet [13] and DenseNet [15], 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
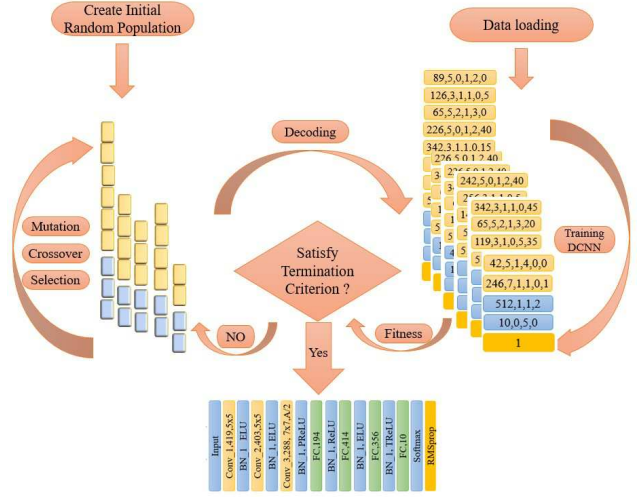

Figure 1: Diagram of the proposed genetic DCNN designer

ambition of artificial intelligence *i.e.* creating autonomous learning systems that require lest human intervention [26].

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 multipoint global search ability, which enable them to quickly locate the areas of high quality solutions even in case of a very complex search space [29]. Despite their success, most evolutionary approaches pose restrictions either on the obtained DCNN architectures, such as the fixed depth, fixed filter size, fixed activation, fixed pooling operation [34] and skipping out the pooling operation [5], or on the employed genetic operations, such as abandoning the crossover [24].

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 [5, 24].

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 [21], EMNIST-Digits [3], EMNIST-Letters [3], Fashion-MNIST [33], CIFAR10 [19] and CIFAR100 [19] 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. [16] 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*. [7] 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*. [11] 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 [4] 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 later applies those filters to the input.

Recently, reinforcement learning has been applied to the design of DCNN architectures. Zoph and Le [36] 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*. [1] 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 $\epsilon - greedy$-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 [29], 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*. [23] 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*. [30] 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) [14] uses heuristics-guided search that simulates the process of natural selection and survival of the fittest. Xie *et al*. [34] 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*. [24] 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 [5] 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 Figure 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 $l_c = 6$ loci in sequence and can be encoded as [NSPBAD]. Similarly, a fully connected block contains $l_f = 4$ 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 , '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 0,1 to indicate whether adopting this operation or not; (5) the value of the activation 'A' ranges from 0 to 5, representing the TReLU [18], ELU [2], PReLU [12], LeakyReLU [22], ReLU [8] and Softmax, respectively; (6)

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

Code	Value scale
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

the dropout operation D takes a value from [0, 0.5] which indicates the probability of the randomly drop; and (7) the value of optimizer 'O' ranges from 0 to 6, representing the SGD [6], RMSprop [35], Adagrad [35], Adadelta [17], Adam [31], Adamax [31] and [9], respectively.

With this coding scheme, a DCNN can be decomposed into a convolutional arm that contains a sequence of convolutional blocks [NSPBAD] and a fully connected arm that contains a sequence of fully connected blocks [NBAD]. Taking the VGG-19 model shown in Figure 2 as the case study, it can be presented as $\{\{[NSPBAD]_i\}_{i=1}^{16}, \{[NBAD]_j\}_{j=1}^{3}, O\}$ 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

$$S_n = \{\{[NSPBAD]_i\}_{i=1}^{N_n^C}, \{[NBAD]_j\}_{j=1}^{N_n^F}, O\} \quad (1)$$

whose code-length is

$$L_n = N_n^C * l_c + N_n^F * l_f \quad (2)$$

At the initialization step, we set $N_n^C \epsilon [1, 20]$ and $N_n^F \epsilon [1, 3]$ and randomly sampled a population of DCNN architectures, denoted by $\{S_n\}_{n=1}^T$.

3.3. Selection

Before producing the next generation, we evaluate each individuals fitness. Based on the fitness ranking, we use the elitism roulette wheel selection scheme [9] to select 0.1T of top ranking individuals (elitist) from the current generation to carry over to the next, unaltered, and select 0.9T of individuals for subsequent genetic operations. This selection strategy guarantees that the highest fitness obtained by the GA will not decrease from one generation to the next.
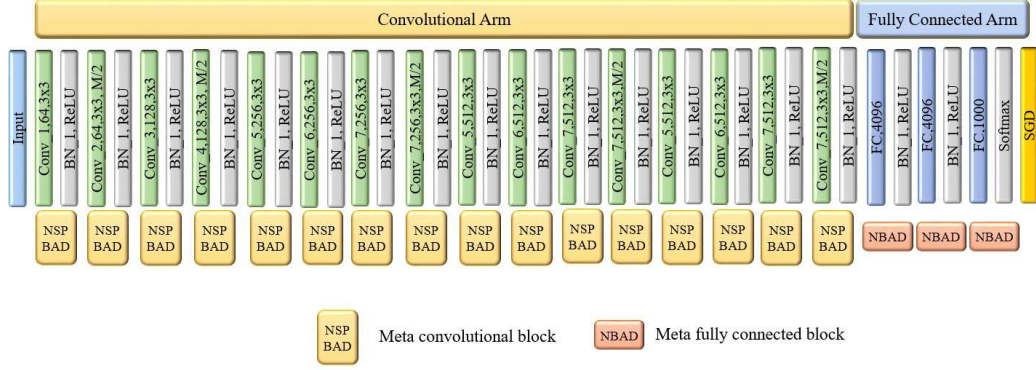
Figure 2: The blocks of VGGNet

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 k_j is 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

$$\begin{cases} L_i' = L_i + (m_i - m_j)*l_c \\ L_j' = L_j + (m_j - m_i)*l_c \end{cases} \quad (3)$$

A typical example is shown Figure 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 $[NBAD]_{m_i}$ on the fully connected arm of the DCNN S_i, its location can be expressed as $N_n^C * l_c + (m_i - 1) * l_c + x$. Then, we require that the other cross point k_j also locates on the fully connected arm of the DCNN S_j and its location can be expressed as $N_n^C * l_c + (m_j - 1) * l_c + x$. After the crossover, the code-length of two newly generated DCNNs are

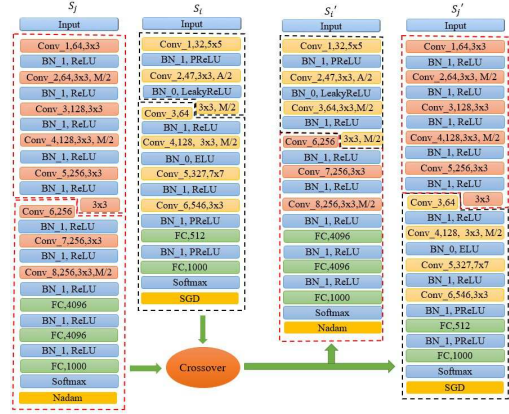$$\begin{cases} L_i' = L_i + (m_i - m_j)*l_f \\ L_j' = L_j + (m_j - m_i)*l_f \end{cases} \quad (4)$$

Figure 3: An example of crossover on the convolutional arm

3.4.1 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 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 $\left[\frac{8}{L_n}, 0.5\right]$ for each individual S_n. An illustrative example of mutation is shown in Figure 4, where six loci in the code of S_i were mutated. After the mutation, the activation 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.
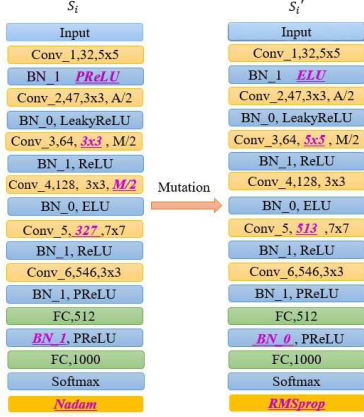
Figure 4: An example of mutation

Table 2: The summary of the datasets

	# Training	# 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	40,000	10	28*28	Grayscale
Fashion-MNIST	60,000	10,000	10	28*28	Grayscale
CIFAR10	50,000	10,000	10	32*32	RGB
CIFAR100	50,000	10,000	100	32*32	RGB

4. Experiments and Results

4.1. Datasets

The datasets used for this study include the MNIST [21], EMNIST-Digits [3], EMNIST-Letters [3], Fashion-MNIST [33], CIFAR10 [19] and CIFAR100 [19]. 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. 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,In each experiment, 10% of training images were used as a validation set, and others training images were used for training. To enlarge the training dataset, we ap-

Table 3: Classification accuracy of seven DCNNs on MNIST(%)

Method	Accuracy
AlexNet [20]	98.81
VGGNet [27]	99.32
ResNet [13]	99.37
Capsule Net [25]	99.57
GDCNN [34]	99.65
EXACT [5]	98.32
Proposed	99.72

plied the simple data augmentation method [13] to each training image. During the training of each DCNN, we use the cross-entropy loss 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.

4.2. 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 Figure 5(a)). The first convolutional block contains 419 filters of size 5x5, no pooling layer, batch normalization, ELU activation and 20% dropout. The second convolutional block contains 403 filters of size 5x5, no pooling layer, batch normalization, ELU activation and 0% dropout. The third convolutional block contains 288 filters of size 7x7, an average-pooling with stride of 2, batch normalization, PReLU activation and 0% dropout. The first fully connected block contains 194 neurons with the ReLU activation , batch normalization and 30% dropout. The second fully connected block contains 414 neurons with the ELU activation , batch normalization and 45% dropout. The third fully connected block contains 356 neurons with the TReLU activation , batch normalization and 5% dropout. The last fully connected block is the output layer, which has 10 neurons with the softmax activation . The optimizer is Adamax.

We compared the classification accuracy achieved by this architecture to the accuracy of AlexNet [20], VGGNet [27], ResNet [13] and Capsule Net [25]] and the DCNNs generated by **GDCNN** [34] and EXACT [5] in Table 3. It shows that the DCNN created by our genetic DCNN designer achieved the highest accuracy of 99.72%. The plot of highest classification accuracy achieved in each generation (see Figure 6(a)) shows that the performance of best DCNN architecture generated by our genetic DCNN designer has become stable.

4.3. DCNN Designed for Fashion-MNIST

The best DCNN architecture generated by the proposed genetic DCNN designer in ten generations for the Fashion-
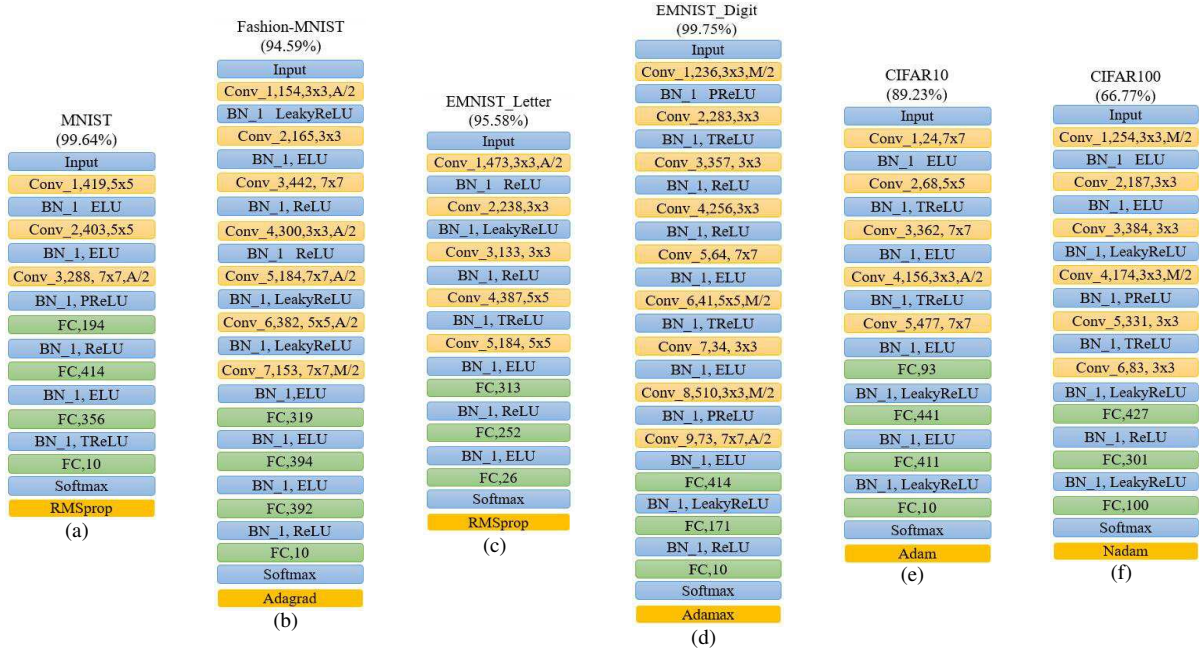
Figure 5: The DCNN architectures generated by proposed genetic DCNN designer

MNIST dataset consists of seven convolutional blocks and four fully connected blocks (see Figure 5(b)). The first convolutional block contains 154 filters of size 5x5, average pooling, batch normalization, LeakyReLU activation and 0% drop out. The second convolutional block contains 165 filters of size 3x3, no pooling , batch normalization, ELU activation and 50% drop out. The third convolutional block contains 442 filters of size 7x7, no pooling , batch normalization, ReLU activation and 40% drop out. The fourth convolutional block contains 300 filters of size 3x3, average pooling , batch normalization, ReLU activation and 5% drop out. The fifth convolutional block contains 184 filters of size 7x7, average pooling , batch normalization, LeakyReLU activation and 25% drop out. The sixth convolutional block contains 382 filters of size 5x5, average pooling , LeakyReLU activation and 40% drop out. The seventh convolutional block contains 153 filters of size 7x7, max pooling , batch normalization, ELU activation and 40% drop out. The first fully connected block contains 319 neurons with the ELU activation and the 5% drop out. The second fully connected block contains 394 neurons with the ELU activation and 45% drop out. The third fully connected block contains 392 neurons with ReLU activation and 50% drop out. The last fully connected block is the output layer, which has 10 neurons with the softmax activation . The optimizer is Adagrad.

The second column of Table 3 shows that, when compared to AlexNet [20], VGGNet [27], ResNet ResNet [13], Capsule Net [25], the architecture we generated

Method	MNIST	Fashion-MNIST	EMNIST-Letters	EMNIST-Digits
AlexNet [20]	98.81	86.43	89.36	99.21
VGGNet [27]	99.32	90.45	94.62	99.62
ResNet [13]	99.37	94.39	94.44	99.63
Capsule Net [25]	99.57	90.03	91.58	99.37
Proposed	**99.72**	**94.60**	**95.58**	**99.75**

achieved the highest accuracy of 94.60%. The plot of highest classification accuracy obtained in each generation (see Figure 6(b)) shows that best DCNN architecture we created is still improving after 10 generations.

4.4. 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 Figure 5(c)). The first convolutional block contains 473 filters of size 3x3, average pooling , batch normalization, ReLU activation and 15% drop out. The second convolutional block contains 238 filters of size 3x3, no pooling , batch normalization, LeakyReLU activation and 20% drop out. The third convolutional block contains 133 filters of size 3x3, no pooling , batch normalization, ReLU activation and 10% drop out. The fourth convolutional block contains 387 filters of size 3x3, no pooling , batch normalization, TReLU activation and 10% drop out. The fifth convolutional block contains 187 filters of size 5x5, no pooling , batch normalization,

ELU activation and 50% drop out. The first fully connected block contains 313 neurons with ReLU activation , batch normalization and 20% dropout. The second block contains 252 neurons with ELU activation , batch normalization and 20% drop out. The last fully connected block is the output layer, which has 26 neurons with the softmax activation . The optimizer is RMSprop.

The third column of Table 3 shows that, when compared to AlexNet [20], VGGNet [27], ResNet ResNet [13], Capsule Net [25], the architecture we generated achieved the highest accuracy of 95.58%. The plot of highest classification accuracy obtained in each generation (see Figure 6(c)) shows that best DCNN architecture we created is still improving after 10 generations.

4.5. 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 convolutional blocks and three fully connected blocks (see Figure 5(d)). The first convolutional block contains 236 filters of size 3x3, max pooling , batch normalization, PReLU activation and 0% drop out. The second convolutional block contains 283 filters of size 5x5, no pooling layer, batch normalization, TReLU activation and 30% drop out. The third convolutional block contains 357 filters of size 3x3, no pooling , batch normalization, ReLU activation and 35% drop out. The fourth convolutional block contains 256 filters of size 3x3, no pooling , batch normalization, ReLU activation and 15% drop out. The fifth convolutional block contains 64 filters of size 7x7, no pooling , batch normalization, LeakyReLU activation and 30% drop out. The sixth convolutional block contains 41 filters of size 5x5, max pooling , batch normalization, TReLU activation and 20% drop out. The seventh convolutional block contains 34 filters of size 3x3, no pooling , batch normalization, ELU activation and 25% drop out is 0.25. The eighth convolutional block contains 510 filters of size 3x3, max pooling , batch normalization, PReLU activation and 45% drop out. The ninth convolutional block contains 73 filters of size 7x7, average pooling , batch normalization, ELU activation and 5% drop out. The first fully connected block contains 414 neurons with batch normalization, PReLU activation and 20% dropout. The second block contains 171 neurons with batch normalization, ReLU activation and 40% drop out. The last fully connected block is the output layer, which has 10 neurons with the softmax activation . The optimizer is Adamax.

The fourth column of Table 3 shows that, when compared to AlexNet [20], VGGNet [27], ResNet ResNet [13], Capsule Net [25], the architecture we generated achieved the highest accuracy of 99.75%. The plot of highest classification accuracy obtained in each generation (see Figure 6(d)) shows that best DCNN architecture we created

Table 5: Classification accuracy on CIFAR10 and CIFAR100 (%)

Method	CIFAR10	CIFAR100
AlexNet [20]	82.53	60.53
VGGNet [27]	84.62	64.37
ResNet [13]	**90.61**	67.61
Highway Net [28]	89.18	67.60
Capsule Net [25]	89.40	-
Proposed	89.23	**66.70**

has become stable after 10 generations.

4.6. 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 Figure 5(e)). The first convolutional block contains 24 filters of size 7x7, no pooling , batch normalization, ELU activation and 15% drop out. The second convolutional block contains 68 filters of size 5x5, no pooling , TReLU activation and 10% drop out. The third convolutional block contains 362 filters of size 7x7, no pooling , batch normalization, ELU activation and 0% drop out. The fourth convolutional block contains 156 filters of size 3x3, average pooling, batch normalization, TReLU activation and 0% drop out. The fifth convolutional block contains 477 filters of size 7x7, average pooling, batch normalization, ELU activation and 0% drop out. The first fully connected block contains 93 neurons with batch normalization, LeakyReLU activation and the 5% of dropout. The second block contains 441 neurons with batch normalization, ELU activation and 50% drop out. The third block contains 411 neurons with batch normalization, LeakyReLU activation and the 45% drop out. The last fully connected block is the output , which has 10 neurons with the softmax activation . The optimizer is Adam.

The second column of Table 4 shows that, when compared to AlexNet [20], VGGNet [27], ResNet ResNet [13], [28] and Capsule Net [25],, the architecture we generated achieved the third highest accuracy of 89.32%, lower than the accuracy of Capsule Net (89.40%) and ResNet (90.61%). However, the plot of highest classification accuracy obtained in each generation (see Figure 6(e)) shows that the performance of best DCNN architecture we generated has become stable after 10 generations.

4.7. 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 Figure 5(f)). The first convolutional block contains 254 filters of size 3x3, max
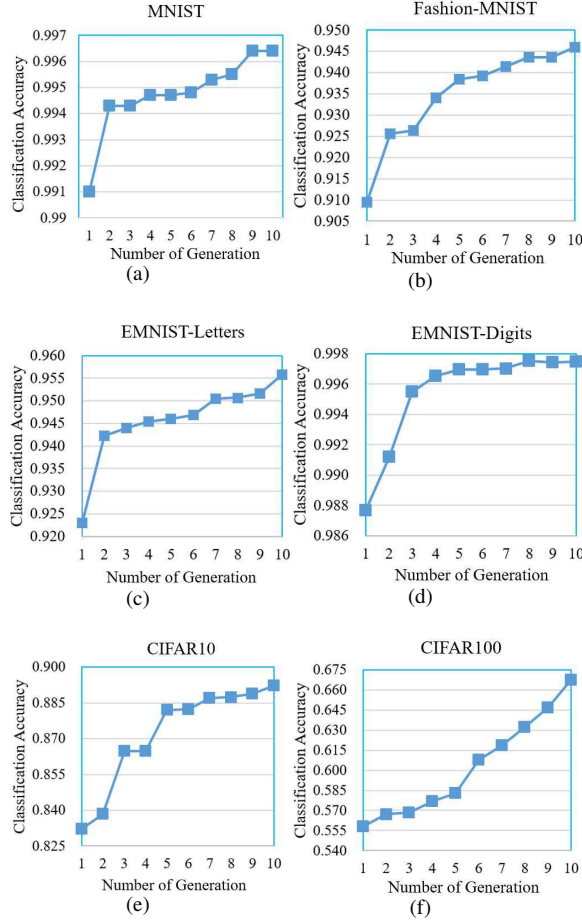
Figure 6: Highest classification accuracy achieved in each generation on each dataset

Nadam.

The third column of Table 4 shows that, when compared to AlexNet [20], VGGNet [27], and ResNet [13], the architecture we generated achieved the third highest accuracy of 66.7%, lower than the accuracy of Highway Net (67.60%) and ResNet (69.59%). However, the plot of highest classification accuracy obtained in each generation (see Figure 6(f)) shows that the performance of best DCNN architecture we generated has become stable after 10 generations.

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.

5. 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. 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 8 NVIDIA Tesla TiTanXp GPU, 512 GB Memory, 240 GB SSD and Matlab 2017a.

6. 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.

pooling , batch normalization, ELU activation and 0% drop out. The second convolutional block contains 187 filters of size 3x3, no pooling , batch normalization, ELU activation and 0% drop out. The third convolutional block contains 384 filters of size 3x3, no pooling , batch normalization, LeakyReLU activation and 30% drop out. The fourth convolutional block contains 174 filters of size 3x3, max pooling, batch normalization, PReLU activation and 20% drop out. The fifth convolutional block contains 331 filters of size 3x3, no pooling , batch normalization, LeakyReLU activation and 40% drop out. The sixth convolutional block contains 83 filters of size 3x3, no pooling , batch normalization, LeakyReLU activation and 35% drop out. The first fully connected block contains 427 neurons, batch normalization, ReLU activation and 35% drop out. The second fully connected block contains 301 neurons with batch normalization, LeakyReLU activation and 15% drop out. The last fully connected block is the output layer, which has 10 neurons with the softmax activation . The optimizer is

References

[1] B. Baker, O. Gupta, N. Naik, and R. Raskar. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*, 2016. 2

[2] D.-A. Clevert, T. Unterthiner, S. Hochreiter, D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). In *ICLR*, 2016. 3

[3] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik. Emnist: an extension of mnist to handwritten letters. *arXiv preprint arXiv:1702.05373*, 2017. 2, 5

[4] B. De Brabandere, X. Jia, T. Tuytelaars, and L. Van Gool. Dynamic filter networks. In *Neural Information Processing Systems (NIPS)*, 2016. 2

[5] T. Desell. Large scale evolution of convolutional neural networks using volunteer computing. *arXiv preprint arXiv:1703.05422*, 2017. 1, 2, 5

[6] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011. 3

[7] C. Fernando, D. Banarse, C. Blundell, Y. Zwols, D. Ha, A. A. Rusu, A. Pritzel, and D. Wierstra. Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*, 2017. 2

[8] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 315–323, 2011. 3

[9] D. E. Goldberg and K. Deb. A comparative analysis of selection schemes used in genetic algorithms. *Foundations of genetic algorithms*, 1:69–93, 1991. 3

[10] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. Maxout networks. *arXiv preprint arXiv:1302.4389*, 2013.

[11] D. Ha, A. Dai, and Q. V. Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016. 2

[12] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015. 3

[13] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 1, 5, 6, 7, 8

[14] J. Horn, N. Nafpliotis, and D. E. Goldberg. *A Niched Pareto Genetic Algorithm for Multiobjective Optimization*. 1994. 2

[15] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*, 2016. 1

[16] J. Jin, Z. Yan, K. Fu, N. Jiang, and C. Zhang. Neural network architecture optimization through submodularity and supermodularity. *arXiv preprint arXiv:1609.00074*, 2016. 2

[17] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 3

[18] K. Konda, R. Memisevic, and D. Krueger. Zero-bias autoencoders and the benefits of co-adapting features. *arXiv preprint arXiv:1402.3337*, 2014. 3

[19] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. 2009. 2, 5

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 1, 5, 6, 7, 8

[21] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 2, 5

[22] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, volume 30, 2013. 3

[23] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, A. Navruzyan, N. Duffy, and B. Hodjat. Evolving deep neural networks. *arXiv preprint arXiv:1703.00548*, 2017. 2

[24] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, Q. Le, and A. Kurakin. Large-scale evolution of image classifiers. *arXiv preprint arXiv:1703.01041*, 2017. 1, 2

[25] S. Sabour, N. Frosst, and G. E. Hinton. Dynamic routing between capsules. *arXiv preprint arXiv:1710.09829*, 2017. 5, 6, 7

[26] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017. 1

[27] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 1, 5, 6, 7, 8

[28] R. K. Srivastava, K. Greff, and Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015. 7

[29] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002. 1, 2

[30] M. Suganuma, S. Shirakawa, and T. Nagao. A genetic programming approach to designing convolutional neural network architectures. *arXiv preprint arXiv:1704.00764*, 2017. 2

[31] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013. 3

[32] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015. 1

[33] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017. 2, 5

[34] L. Xie and A. Yuille. Genetic cnn. arxiv preprint. *arXiv preprint arXiv:1703.01513*, 2017. 1, 2, 5

[35] M. D. Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012. 3

[36] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016. 2