Genetic CNN

Lingxi Xie, Alan Yuille
Center for Imaging Science, The Johns Hopkins University, Baltimore, MD, USA
198808xc@gmail.com alan.l.yuille@gmail.com

Abstract

The deep Convolutional Neural Network (CNN) is the state-of-the-art solution for large-scale visual recognition. Following basic principles such as increasing the depth and constructing highway connections, researchers have manually designed a lot of fixed network structures and verified their effectiveness.

In this paper, we discuss the possibility of learning deep network structures automatically. Note that the number of possible network structures increases exponentially with the number of layers in the network, which inspires us to adopt the genetic algorithm to efficiently traverse this large search space. We first propose an encoding method to represent each network structure in a fixed-length binary string, and initialize the genetic algorithm by generating a set of randomized individuals. In each generation, we define standard genetic operations, e.g., selection, mutation and crossover, to eliminate weak individuals and then generate more competitive ones. The competitiveness of each individual is defined as its recognition accuracy, which is obtained via training the network from scratch and evaluating it on a validation set. We run the genetic process on two small datasets, i.e., **MNIST** *and* **CIFAR10***, demonstrating its ability to evolve and find high-quality structures which are little studied before. These structures are also transferrable to the large-scale* **ILSVRC2012** *dataset.*

1. Introduction

Visual recognition is a fundamental task in computer vision, implying a wide range of applications. Recently, the state-of-the-art algorithms on visual recognition are mostly based on the deep Convolutional Neural Network (CNN). Starting from the fundamental network model for large-scale image classification [17], researchers have been increasing the depth of the network [29], as well as designing new inner structures [32][10] to improve recognition accuracy. Although these modern networks have been shown to be efficient, we note that their structures are manually designed, not learned, which limits the flexibility of the approach.

In this paper, we explore the possibility of automatically learning the structure of deep neural networks. We consider a constrained case, in which the network has a limited number of layers, and each layer is designed as a set of predefined building blocks such as convolution and pooling. Even under these limitations, the total number of possible network structures grows exponentially with the number of layers. Therefore, it is impractical to enumerate all the candidates and find the best one. Instead, we formulate this problem as optimization in a large search space, and apply the genetic algorithm to traversing the space efficiently.

The genetic algorithm involves constructing an initial *generation* of *individuals* (candidate solutions), and performing genetic operations to allow them to evolve in a genetic process. To this end, we propose an encoding method to represent each network structure by a fixed-length binary string. After that, we define several standard genetic operations, *i.e.*, selection, mutation and crossover, which eliminate weak individuals of the previous generation and use them to generate competitive ones. The quality of each individual is determined by its recognition accuracy on a reference dataset. **Throughout the genetic process, we evaluate each individual (*i.e.*, network structure) by training it from scratch.** The genetic process comes to an end after a fixed number of generations.

It is worth emphasizing that the genetic algorithm is computationally expensive, because we need to conduct a complete network training process for each generated individual. Therefore, we run the genetic process on two small datasets, *i.e.*, **MNIST** and **CIFAR10**, and demonstrate its ability to find high-quality network structures. **It is interesting to see that the generated structures, most of which have been less studied before, often perform better than the standard manually designed ones.** Finally, we transfer the learned structures to large-scale experiments and verify their effectiveness.

The remainder of this paper is organized as follows. Section 2 briefly introduces related work. Section 3 illustrates the way of using the genetic algorithm to design network structures. Experiments are shown in Section 4,

and conclusions are drawn in Section 5.

2. Related Work

2.1. Convolutional Neural Networks

Image classification is a fundamental problem in computer vision. Recently, researcher have extended conventional classification tasks [18][7] into large-scale environments such as ImageNet [5] and Places [44]. With the availability of powerful computational resources (*e.g.*, GPU), the Convolutional Neural Networks (CNNs) [17][29] have shown superior performance over the conventional Bag-of-Visual-Words models [3][35][26].

CNN is a hierarchical model for large-scale visual recognition. It is based on the observation that a network with enough neurons is able to fit any complicated data distribution. In past years, neural networks were shown effective for simple recognition tasks [20]. More recently, the availability of large-scale training data (*e.g.*, ImageNet [5]) and powerful GPUs make it possible to train deep CNNs [17] which significantly outperform BoVW models. A CNN is composed of several stacked layers. In each of them, responses from the previous layer are convoluted with a filter bank and activated by a differentiable non-linearity. Hence, a CNN can be considered as a composite function, which is trained by back-propagating error signals defined by the difference between the supervision and prediction at the top layer. Recently, several efficient methods were proposed to help CNNs converge faster and prevent over-fitting, such as ReLU activation [17], batch normalization [15], Dropout [11] and DisturbLabel [39]. Features extracted from pre-trained neural networks can be generalized to other recognition tasks [36][40].

Designing powerful CNN structures is an intriguing problem. It is believed that deeper networks produce better recognition results [29][32]. But also, adding highway information has been verified to be useful [10][42]. Efforts are also made to add invariance into the network structure [38]. We find some work which uses stochastic [14] or dense [13] structures, but all these network structures are deterministic (although stochastic operations are used in [14] to accelerate training and prevent over-fitting), which limits the flexibility of the models and thus inspires us to automatically learn network structures.

2.2. Genetic Algorithm

The genetic algorithm is a metaheuristic inspired by the process of natural selection. Genetic algorithms are commonly used to generate high-quality solutions to optimization and search problems [12][27][2][4] by relying on bio-inspired operators such as mutation, crossover and selection.

A typical genetic algorithm requires two prerequisites, *i.e.*, a genetic representation of the solution domain, and a fitness function to evaluate each individual. A good example is the travelling-salesman problem (TSP) [9], a famous NP-complete problem which aims at finding the optimal Hamiltonian path in a graph of N nodes. In this situation, each feasible solution is represented as a permutation of $\{1, 2, \ldots, N\}$, and the fitness function is the total cost (distance) of the path. We will show later that deep neural networks can be encoded into a binary string.

The core idea of the genetic algorithm is to allow individuals to evolve via some genetic operations. Popular operations include *selection*, *mutation*, *crossover*, *etc*. The selection process allows us to preserve strong individuals while eliminating weak ones. The ways of performing mutation and crossover vary from case to case, often based on the properties of the specific problem. For example, in the TSP problem with the permutation-based representation, a possible set of mutations is to change the order of two visited nodes. These operations are also used in our work.

There is a lot of research in how to improve the performance of genetic algorithms, including performing local search [34] and generating random keys [30]. In our work, we show that the vanilla genetic algorithm works well enough without these tricks. We also note that some previous work applied the genetic algorithm to exploring efficient neural network architectures [41][31][1][6], but our work aims at learning the architecture of modern CNNs, which is not studied in these prior works.

3. Our Approach

This section presents a genetic algorithm for designing competitive network structures. First, we describe a way of representing the network structure by a fixed-length binary string. Next, several genetic operations are defined, including selection, mutation and crossover, so that we can traverse the search space efficiently and find high-quality solutions.

Throughout this work, the genetic algorithm is only used to propose new network structures, the parameters and classification accuracy of each structure are obtained via standalone training-from-scratch.

3.1. Binary Network Representation

We provide a binary string representation for a network structure in a constrained case. We first note that many state-of-the-art network structures [29][10] can be partitioned into several *stages*. In each stage, the geometric dimensions (width, height and depth) of the layer cube remain unchanged. Neighboring stages are connected via a spatial pooling operation, which may change the spatial resolution. All the convolutional operations within one stage have the same number of filters, or channels.

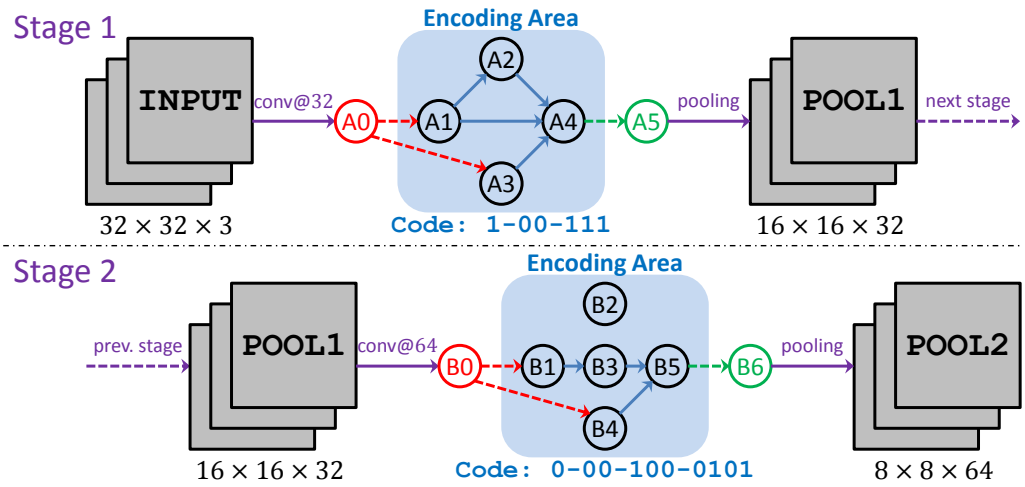

Figure 1. A two-stage network ($S = 2$, $(K_1, K_2) = (4, 5)$) and the encoded binary string (best viewed in color PDF). The default input and output nodes (see Section 3.1.1) and the connections from and to these nodes are marked in red and green, respectively. We only encode the connections in the effective parts (regions with light blue background). Within each stage, the number of convolutional filters is a constant (32 in Stage 1, 64 in Stage 2), and the spatial resolution remains unchanged (32×32 in Stage 1, 16×16 in Stage 2). Each pooling layer down-samples the data by a factor of 2. ReLU and batch normalization are added after each convolution.

We borrow this idea to define a family of networks which can be encoded into fixed-length binary strings. A network is composed of S stages, and the s-th stage, $s = 1, 2, \ldots, S$, contains K_s *nodes*, denoted by v_{s,k_s}, $k_s = 1, 2, \ldots, K_s$. The nodes within each stage are ordered, and we only allow connections from a lower-numbered node to a higher-numbered node. Each node corresponds to a convolutional operation, which takes place after element-wise summing up all its input nodes (lower-numbered nodes that are connected to it). After convolution, batch normalization [15] and ReLU [17] are followed, which are verified efficient in training very deep neural networks [29]. We do not encode the fully-connected part of a network.

In each stage, we use $1 + 2 + \ldots + (K_s - 1) = \frac{1}{2}K_s(K_s - 1)$ bits to encode the inter-node connections. The first bit represents the connection between $(v_{s,1}, v_{s,2})$, then the following two bits represent the connection between $(v_{s,1}, v_{s,3})$ and $(v_{s,2}, v_{s,3})$, *etc*. This process continues until the last $K_s - 1$ bits are used to represent the connection between $v_{s,1}, v_{s,2}, \ldots, v_{s,K_s-1}$ and v_{s,K_s}. For $1 \leqslant i < j \leqslant K_s$, if the code corresponding to $(v_{s,i}, v_{s,j})$ is 1, there is an edge connecting $v_{s,i}$ and $v_{s,j}$, *i.e.*, $v_{s,j}$ takes the output of $v_{s,i}$ as a part of the element-wise summation, and vice versa.

Figure 1 illustrates two examples of network encoding. To summarize, a S-stage network with K_s nodes at the s-th stage is encoded into a binary string with length $L = \frac{1}{2}\sum_s K_s(K_s - 1)$. Equivalently, there are in total 2^L possible network structures. This number may be very large. In the **CIFAR10** experiments (see Section 4.2), we have $S = 3$ and $(K_1, K_2, K_3) = (3, 4, 5)$, therefore $L = 19$ and $2^L = 524{,}288$. It is computationally intractable to enumerate all these structures and find the optimal one(s). In the following parts, we adopt the genetic algorithm to efficiently explore good candidates in this large space.

3.1.1 Technical Details

To make every binary string valid, we define two default nodes in each stage. The default input node, denoted as $v_{s,0}$, receives data from the previous stage, performs convolution, and sends its output to every node without a predecessor, *e.g.*, $v_{s,1}$. The default output node, denoted as v_{s,K_s+1}, receives data from all nodes without a successor, *e.g.*, v_{s,K_s}, sums up them, performs convolution, and sends its output to the pooling layer. Note that the connections between the ordinary nodes and the default nodes are not encoded.

There are two special cases. First, if an ordinary node $v_{s,i}$ is isolated (*i.e.*, it is not connected to any other ordinary nodes $v_{s,j}$, $i \neq j$), then it is simply ignored, *i.e.*, it is not connected to the default input node nor the default output node (see the `B2` node in Figure 1). This is to guarantee that a stage with more nodes can simulate all structures represented by a stage with fewer nodes. Second, if there are no connections at a stage, *i.e.*, all bits in the binary string are 0, then the convolutional operation is performed only once, not twice (one for the default input node and one for the default output node).

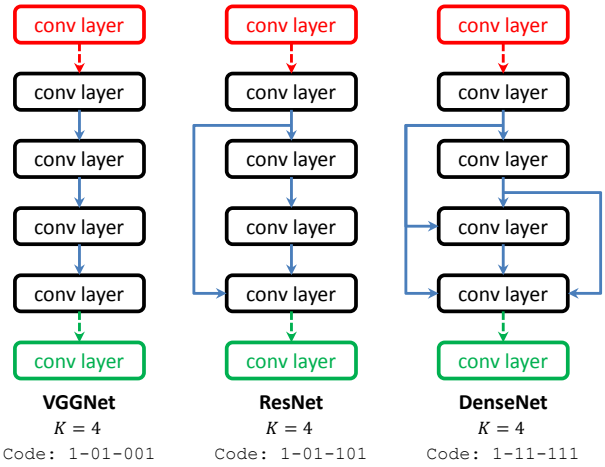

Figure 2. The basic building blocks of **VGGNet** [29] and **ResNet** [10] can be encoded as binary strings defined in Section 3.1.

3.1.2 Examples and Limitations

Many popular network structures can be represented using the proposed encoding scheme. Examples include **VGGNet** [29], **ResNet** [10], and a modified variant of **DenseNet** [13], which are illustrated in Figure 2.

Currently, only convolutional and pooling operations are considered, which makes it impossible to generate some tricky network modules such as Maxout [8]. Also, the size of convolutional filters is fixed within each stage, which limits our network from incorporating multi-scale information as in the inception module [32]. However, we note that all the encoding-based approaches have such limitations. Our approach can be easily modified to include more types of layers and more flexible inter-layer connections. As shown in experiments, we can achieve competitive recognition performance using merely these basic building blocks.

As shown in a recent published work using reinforcement learning to explore neural architecture [45], this type of methods often require heavy computation to traverse the huge solution space. Fortunately, our method can be easily generalized and scaled up, which is done via learning the architecture on a small dataset and transfer the learned information to large-scale datasets. Please refer to the experimental part for details.

3.2. Genetic Operations

The flowchart of the genetic process is shown in Algorithm 1. It starts with an initialized *generation* of N randomized *individuals*. Then, we perform T rounds, or T generations, each of which consists of three operations, *i.e.*, selection, mutation and crossover. The fitness function of each individual is evaluated via training-from-scratch on the reference dataset.

3.2.1 Initialization

We initialize a set of randomized models $\{\mathbb{M}_{0,n}\}_{n=1}^{N}$. Each model is a binary string with L bits, *i.e.*, $\mathbb{M}_{0,n} : \mathbf{b}_{0,n} \in \{0,1\}^L$. Each bit in each individual is independently sampled from a Bernoulli distribution: $b_{0,n}^l \sim \mathcal{B}(0.5)$, $l = 1, 2, \ldots, L$. After this, we evaluate each individual (see Section 3.2.4) to obtain their fitness function values.

As we shall see in Section 4.1.3, different strategies of initialization do not impact the genetic performance too much. Even starting with a naive initialization (all individuals are all-zero strings), the genetic process can discover quite competitive structures with crossover and mutation.

3.2.2 Selection

The selection process is performed at the beginning of every generation. Before the t-th generation, the n-th individual $\mathbb{M}_{t-1,n}$ is assigned a fitness function, which is defined as the recognition rate $r_{t-1,n}$ obtained in the previous generation or initialization. $r_{t-1,n}$ directly impacts the probability that $\mathbb{M}_{t-1,n}$ survives the selection process.

We perform a Russian roulette process to determine which individuals survive. Each individual in the next generation $\mathbb{M}_{t,n}$ is determined independently by a non-uniform sampling over the set $\{\mathbb{M}_{t-1,n}\}_{n=1}^{N}$. The probability of sampling $\mathbb{M}_{t-1,n}$ is proportional to $r_{t-1,n} - r_{t-1,0}$, where $r_{t-1,0} = \min_{n=1}^{N} \{r_{t-1,n}\}$ is the minimal fitness function value in the previous generation. This means that the best individual has the largest probability of being selected, and the worst one is always eliminated. As the number of individuals N remains unchanged, each individual in the previous generation may be selected multiple times.

3.2.3 Mutation and Crossover

The mutation process of an individual $\mathbb{M}_{t,n}$ involves flipping each bit independently with a probability q_{M}. In practice, q_{M} is often small, *e.g.*, 0.05, so that mutation is not likely to change one individual too much. This is to preserve the good properties of a survived individual while providing an opportunity of trying out new possibilities.

The crossover process involves changing two individuals simultaneously. Instead of considering each bit individually, the basic unit in crossover is a stage, which is motivated by the need to retain the local structures within each stage. Similar to mutation, each pair of corresponding stages are exchanged with a small probability q_{C}.

Both mutation and crossover are implemented by an overall flowchart (see Algorithm 1). The probabilities of mutation and crossover for each individual (or pair) are p_{M} and p_{C}, respectively. We understand that there are many different ways of mutation and crossover. As shown in

Algorithm 1 The Genetic Process for Network Design

Input: the reference dataset $\mathcal{D}$, the number of generations T, the number of individuals in each generation N, the mutation and crossover probabilities p_{M} and p_{C}, the mutation parameter q_{M}, and the crossover parameter q_{C}.
Initialization: generating a set of randomized individuals $\{\mathbb{M}_{0,n}\}_{n=1}^{N}$, and computing their recognition accuracies;
for $t = 1, 2, \ldots, T$ **do**
  Selection: producing a new generation $\{\mathbb{M}'_{t,n}\}_{n=1}^{N}$ with a Russian roulette process on $\{\mathbb{M}_{t-1,n}\}_{n=1}^{N}$;
  Crossover: for each pair $\{(\mathbb{M}_{t,2n-1}, \mathbb{M}_{t,2n})\}_{n=1}^{\lfloor N/2 \rfloor}$, performing crossover with probability p_{C} and parameter q_{C};
  Mutation: for each non-crossover individual $\{\mathbb{M}_{t,n}\}_{n=1}^{N}$, doing mutation with probability p_{M} and parameter q_{M};
  Evaluation: computing the recognition accuracy for each new individual $\{\mathbb{M}_{t,n}\}_{n=1}^{N}$;
end for
Output: a set of individuals in the final generation $\{\mathbb{M}_{T,n}\}_{n=1}^{N}$ with their recognition accuracies.

experiments, our simple choice leads to competitive performance.

3.2.4 Evaluation

After the above processes, each individual $\mathbb{M}_{t,n}$ is evaluated to obtain the fitness function value. A reference dataset $\mathcal{D}$ is pre-defined, and we individually train each model $\mathbb{M}_{t,n}$ from scratch. If $\mathbb{M}_{t,n}$ is previously evaluated, we simply evaluate it once again and compute the average accuracy over all its occurrences. This strategy, at least to some extent, alleviates the instability caused by the randomness in the training process.

4. Experiments

The proposed genetic algorithm requires a very large amount of computational resources, which makes it intractable to be directly evaluated on large-scale datasets such as **ILSVRC2012** [28]. Our solution is to explore promising network structures on small datasets such as **MNIST** [19] and **CIFAR10** [16], then transfer these structures to the large-scale recognition tasks.

4.1. MNIST Experiments

The **MNIST** dataset [19] defines a handwritten digit recognition task. There are 60,000 images for training, and 10,000 images for testing, all of them are 28×28 grayscale images. Both training and testing data are uniformly distributed over 10 categories, *i.e.*, digits from 0 to 9. To avoid using the testing data, we leave 10,000 images from the training set for validation.

4.1.1 Settings and Results

We follow the basic **LeNet** for **MNIST** recognition. The original network is abbreviated as:
`C5@20-MP2S2-C5@50-MP2S2-FC500-D0.5-FC10`.
Here, `C5@20` is a convolutional layer with a kernel size 5, a default spatial stride 1 and the number of kernels 20; `MP2S2` is a max-pooling layer with a kernel size 2 and a spatial stride 2, `FC500` is a fully-connected layer with 500 outputs, and `D0.5` is a Dropout layer with a drop ratio 0.5. We apply 20 training epochs with learning rate 10^{-3}, followed by 4 epochs with learning rate 10^{-4}, and another 1 epoch with learning rate 10^{-5}.

We set $S = 2$, $(K_1, K_2) = (3, 5)$, and keep the fully-connected part of **LeNet** unchanged. The first convolutional layer within each stage remains the same as in the original **LeNet**, and other convolutional layers take the kernel size 3×3 and the same channel number. The length L of each binary string is 13, which means that there are $2^{13} = 8{,}192$ possible individuals.

We create an initial generation with $N = 20$ individuals, and run the genetic process for $T = 50$ rounds. Other parameters are set as $p_{\mathrm{M}} = 0.8$, $q_{\mathrm{M}} = 0.1$, $p_{\mathrm{C}} = 0.2$ and $q_{\mathrm{C}} = 0.3$. We set relatively high mutation and crossover probabilities to facilitate new structures to be generated. The maximal number of explored individuals is $20 \times (50 + 1) = 1{,}020 < 8{,}192$. The training phase of each individual takes an average of 2.5 minutes on a modern Titan-X GPU, and the entire genetic process takes about 2 GPU-days, which makes it possible to repeat it with different settings for diagnosis, *e.g.*, to explore different initialization options (see Section 4.1.3).

Results are summarized in Table 1. With the genetic operations, we can find competitive network structures which achieve high recognition accuracy. Although over a short period the recognition rate of the best individual is not improved, the average and medium accuracies generally get higher from generation to generation. This is very important, because it guarantees the genetic algorithm improves the overall quality of the individuals. According to our diagnosis in Section 4.1.2, this is very important for the genetic process, since the quality of a new individual is positively correlated to the quality of its parent(s). After 50 generations, the recognition error rate of the best individual drops from 0.41% to 0.34%.

Gen	Max %	Min %	Avg %	Med %	Std-D
00	99.59	99.38	99.50	99.50	0.06
01	99.61	99.40	99.53	99.54	0.05
02	99.62	99.43	99.55	99.58	0.06
03	99.62	99.40	99.56	99.58	0.06
05	99.62	99.46	99.57	99.57	0.04
08	99.63	99.40	99.57	99.60	0.06
10	99.63	99.50	99.59	99.62	0.05
20	99.63	99.45	99.61	99.63	0.05
30	99.64	99.49	99.61	99.64	0.06
50	99.66	99.51	99.62	99.65	0.06

Table 1. Recognition accuracy (%) on the **MNIST** testing set. The zeroth generation is the initialized generation. We set $S = 2$ and $(K_1, K_2) = (3, 5)$.

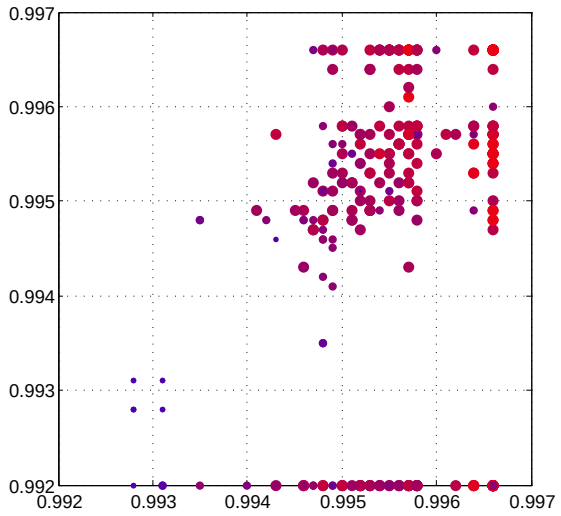

Figure 3. The relationship in accuracy between the parent(s) and the child(ren) (best viewed in color PDF). A point is bigger and close to blue if the recognition error rate is lower, otherwise it is smaller and close to blue. The points on the horizontal axis are from mutation operations, while others are from crossover operations.

4.1.2 Diagnosis

We perform diagnostic experiments to verify the hypothesis, that a better individual is more likely to generate a good individual via mutation or crossover. For this, we randomly select several occurrences of mutation and crossover in the **CIFAR10** genetic process, and observe the relationship between an individual and its parent(s). Figure 3 supports our point. We argue that the genetic operations tend to preserve a fraction of the good local properties, so that the excellent "genes" from the parent(s) are more likely to be preserved.

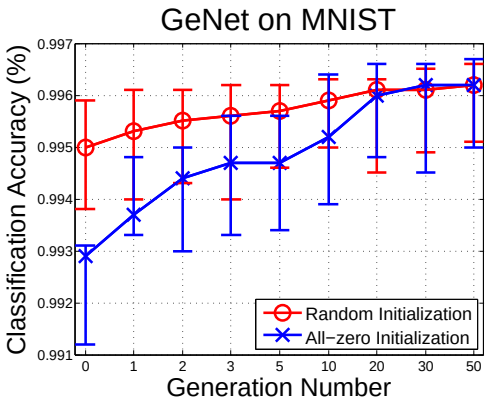

Figure 4. The average recognition accuracy over all individuals with respect to the generation number. The bars indicate the highest and lowest accuracies in the corresponding generation.

4.1.3 Initialization Issues

Finally, we observe the impact of different initialized networks. For this, we start a naive population with $N = 20$ all-zero individuals, and use the same parameters for a complete genetic process. Results are shown in Figure 4. We find that, although the all-zero string corresponds to a very simple and less competitive network structure, the genetic algorithm is able to generate strong individuals after several generations. This naive initialization achieves the initial performance of randomized individuals with about 10 generations. After about 30 generations, there is almost no difference, by statistics, between these two populations.

4.2. CIFAR10 Experiments

The **CIFAR10** dataset [16] is a subset of the 80-million tiny image database [33]. There are 50,000 images for training, and 10,000 images for testing, all of them are 32×32 RGB images. **CIFAR10** contains 10 basic categories, and both training and testing data are uniformly distributed over these categories. To avoid using the testing data, we leave 10,000 images from the training set for validation.

4.2.1 Settings and Results

We follow a revised **LeNet** for **CIFAR10** recognition. The original network is abbreviated as:

`C5(P2)@8-MP3(S2)-C5(P2)@16-MP3(S2)-C5(P2)@32-MP3(S2)-FC128-D0.5-FC10.`

Note that we significantly reduce the filter numbers at each stage to accelerate the training phase. We will show later that this does not prevent the genetic process from learning promising network structures. We apply 120 training epochs with learning rate 10^{-2}, followed by 60 epochs with learning rate 10^{-3}, 40 epoch with learning rate 10^{-4} and another 20 epoch with learning rate 10^{-5}.

We keep the fully-connected part of the above network unchanged, and set $S = 3$ and $(K_1, K_2, K_3) = (3, 4, 5)$. Similarly, the first convolutional layer within each stage remains the same as in the original **LeNet**, and other convolutional layers take the kernel size 3×3 and the same channel number. The length L of each binary string is 19, which means that there are $2^{19} = 524{,}288$ possible individuals.

We create an initial generation with $N = 20$ individuals, and run the genetic process for $T = 50$ rounds. Other parameters are set to be $p_{\mathrm{M}} = 0.8$, $q_{\mathrm{M}} = 0.05$, $p_{\mathrm{C}} = 0.2$ and $q_{\mathrm{C}} = 0.2$. The mutation and crossover parameters q_{M} and q_{C} are set to be smaller because the strings become longer. The maximal number of explored individuals is $20 \times (50 + 1) = 1{,}020 \ll 524{,}288$. The training phase of each individual takes an average of 0.4 hour, and the entire genetic process takes about 17 GPU-days.

We perform two individual genetic processes. The results of one process are summarized in Table 2. As in the **MNIST** experiments, all the important statistics (*e.g.*, average and median accuracies) grow from generation to generation. We also report the best network structures in the table, and visualize the best structures throughout these two processes in Figure 5.

4.2.2 Comparison to Densely-Connected Nets

Under our encoding scheme, a straightforward way of network construction is to set all bits to be 1, which leads to a network in which any two layers within the same stage are connected. This network produces a 76.84% recognition rate, which is a little bit lower than those reported in Table 2. Considering that the densely-connected network requires heavier computational overheads, we conclude that the genetic algorithm helps to find more effective and efficient structures than the dense connections.

4.2.3 Observation

In Figure 5, we plot the the network structures learned from two individual genetic processes. The structures learned by the genetic algorithm are quite different from the manually designed ones, although some manually designed local structures are observed, like the chain-shaped networks, multi-path networks and highway networks. We emphasize that these two networks are obtained by independent genetic processes, which demonstrates that our genetic process generally converges to similar network structures.

4.3. CIFAR and SVHN Experiments

We apply the networks learned from the **CIFAR10** experiments to more small-scale datasets. We test three datasets, *i.e.*, **CIFAR10**, **CIFAR100** and **SVHN**. **CIFAR100** is an extension to **CIFAR10** containing 100 finer-grained categories. It has the same numbers of training and testing images as **CIFAR10**, and these images are also uniformly distributed over 100 categories.

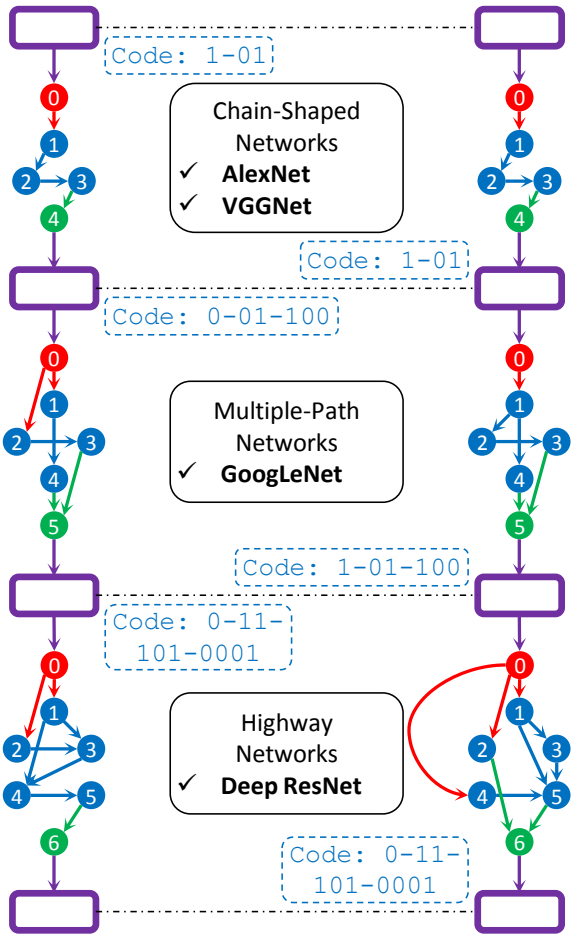

Figure 5. Two network structures learned from the two independent genetic processes (best viewed in color PDF).

SVHN (Street View House Numbers) [25] is a large collection of 32×32 RGB images, *i.e.*, 73,257 training samples, 26,032 testing samples, and 531,131 extra training samples. We preprocess the data as in the previous work [25], *i.e.*, selecting 400 samples per category from the training set as well as 200 samples per category from the extra set, using these 6,000 images for validation, and the remaining 598,388 images as training samples. We also use Local Contrast Normalization (LCN) for preprocessing [8].

We evaluate the best network structure in each generation of the genetic process. We resume using a large number of filters at each stage, *i.e.*, the three stages and the first fully-connected layer are equipped with 64, 128, 256 and 1024 filters, respectively. The training strategy, include the numbers of epochs and learning rates, remains the same as in the previous experiments.

Gen	Max %	Min %	Avg %	Med %	Std-D	Best Network Structure
00	75.96	71.81	74.39	74.53	0.91	`0-01|0-01-111|0-11-010-0111`
01	75.96	73.93	75.01	75.17	0.57	`0-01|0-01-111|0-11-010-0111`
02	75.96	73.95	75.32	75.48	0.57	`0-01|0-01-111|0-11-010-0111`
03	76.06	73.47	75.37	75.62	0.70	`1-01|0-01-111|0-11-010-0111`
05	76.24	72.60	75.32	75.65	0.89	`1-01|0-01-111|0-11-010-0011`
08	76.59	74.75	75.77	75.86	0.53	`1-01|0-01-111|0-11-010-1011`
10	76.72	73.92	75.68	75.80	0.88	`1-01|0-01-110|0-11-111-0001`
20	76.83	74.91	76.45	76.79	0.61	`1-01|1-01-110|0-11-111-0001`
30	76.95	74.38	76.42	76.53	0.46	`1-01|0-01-100|0-11-111-0001`
50	**77.06**	**75.34**	**76.58**	**76.81**	0.55	`1-01|0-01-100|0-11-101-0001`

Table 2. Recognition accuracy (%) on the **CIFAR10** testing set. The zeroth generation is the initialized generation. We set $S = 3$ and $(K_1, K_2, K_3) = (3, 4, 5)$.

	SVHN	**CF10**	**CF100**
Zeiler *et.al* [43]	2.80	15.13	42.51
Goodfellow *et.al* [8]	2.47	9.38	38.57
Lin *et.al* [24]	2.35	8.81	35.68
Lee *et.al* [22]	1.92	7.97	34.57
Liang *et.al* [23]	1.77	7.09	31.75
Lee *et.al* [21]	1.69	6.05	32.37
Zagoruyko *et.al* [42]	1.85	5.37	24.53
Xie *et.al* [37]	1.67	5.31	25.01
Huang *et.al* [14]	1.75	5.25	24.98
Huang *et.al* [13]	**1.59**	**3.74**	**19.25**
GeNet after G-00	2.25	8.18	31.46
GeNet after G-05	2.15	7.67	30.17
GeNet after G-20	2.05	7.36	29.63
GeNet #1 (G-50)	1.99	7.19	**29.03**
GeNet #2 (G-50)	**1.97**	**7.10**	29.05

Table 3. Comparison of the recognition error rate (%) with the state-of-the-arts. We apply data augmentation on all these datasets. **GeNet** #1 and **GeNet** #2 are the structures shown in Figure 5.

	Top-1	Top-5	Depth
AlexNet [17]	42.6	19.6	8
GoogLeNet [32]	34.2	12.9	22
VGGNet-16 [29]	28.5	9.9	16
VGGNet-19 [29]	28.7	9.9	19
ResNet-50 [10]	24.6	7.7	50
ResNet-101 [10]	23.4	7.0	101
ResNet-152 [10]	23.0	6.7	152
GeNet #1	28.12	9.95	22
GeNet #2	27.87	9.74	22

Table 4. Top-1 and top-5 recognition error rates (%) on the **ILSVRC2012** dataset. For all competitors, we report the single-model performance without using any complicated data augmentation in *testing*. These numbers are copied from this page: `http://www.vlfeat.org/matconvnet/pretrained/`. We use the networks shown in Figure 5, and name them as **GeNet** #1 and #2, respectively.

We compare our results with some state-of-the-art methods in Table 3. Among these competitors, we note that the densely-connected network [13] is closely related to our work. Although **GeNet** (17 layers) produces lower recognition accuracy, we note that the structures used in [42][37][14][13] are much deeper (*e.g.*, 40–100 layers). Since dense connection is often not the best option (see Section 4.2.2), we believe that it is possible to use the genetic algorithm to optimize the connections used in [13].

4.4. ILSVRC2012 Experiments

We evaluate the top-2 networks on the **ILSVRC2012** classification task [28], a subset of the **ImageNet** database [5] which contains 1,000 object categories. The training set, validation set and testing set contain 1.3M, 50K and 150K images, respectively. The input images are of $224 \times 224 \times 3$ pixels. We first apply the first two stages in the **VGGNet** (4 convolutional layers and two pooling layers) to change the data dimension to $56 \times 56 \times 128$. Then, we apply the two networks shown in Figure 5, and adjust the numbers of filters at three stages to 256, 512 and 512 (following **VGGNet**), respectively. After these stages, we obtain a $7 \times 7 \times 512$ data cube. We preserve the fully-connected layers in **VGGNet** with the dropout rate 0.5. We apply the training strategy as in **VGGNet**. The entire training process of each network takes around 20 GPU-days.

Results are summarized in Table 4. We can see that, in general, structures learned from a small dataset (**CIFAR10**) can be transferred to large-scale visual recognition (**ILSVRC2012**). Our model achieves better performance than **VGGNet**, because the original three chain-shaped stages are replaced by the automatically learned structures.

5. Conclusions

This paper applies the genetic algorithm to designing the structure of deep neural networks. We first propose an encoding method to represent each network structure with a fixed-length binary string, then uses some popular genetic operations such as mutation and crossover to explore the search space efficiently. Different initialization strategies do not make much difference on the genetic process. We conduct the genetic algorithm with a relatively small reference dataset (**CIFAR10**), and find that the generated structures transfer well to other tasks, including the large-scale **ILSVRC2012** dataset.

Despite the interesting results we have obtained, our algorithm suffers from several drawbacks. First, a large fraction of network structures are still unexplored, including those with non-convolutional modules like Maxout [8], and the multi-scale strategy used in the inception module [32]. Second, in the current work, the genetic algorithm is only used to explore the network structure, whereas the network training process is performed separately. It would be very interesting to incorporate the genetic algorithm to training the network structure and weights simultaneously. These directions are left for future work.

Acknowledgements

We thank John Flynn, Wei Shen, Chenxi Liu and Siyuan Qiao for instructive discussions.

References

[1] J. Bayer, D. Wierstra, J. Togelius, and J. Schmidhuber. Evolving Memory Cell Structures for Sequence Learning. *International Conference on Artificial Neural Networks*, 2009.

[2] J. Beasley and P. Chu. A Genetic Algorithm for the Set Covering Problem. *European Journal of Operational Research*, 94(2):392–404, 1996.

[3] G. Csurka, C. Dance, L. Fan, J. Willamowski, and C. Bray. Visual Categorization with Bags of Keypoints. *Workshop on Statistical Learning in Computer Vision, European Conference on Computer Vision*, 1(22):1–2, 2004.

[4] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[5] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. *Computer Vision and Pattern Recognition*, 2009.

[6] S. Ding, H. Li, C. Su, J. Yu, and F. Jin. Evolutionary Artificial Neural Networks: A Review. *Artificial Intelligence Review*, 39(3):251–260, 2013.

[7] L. Fei-Fei, R. Fergus, and P. Perona. Learning Generative Visual Models from Few Training Examples: An Incremental Bayesian Approach Tested on 101 Object Categories. *Computer Vision and Image Understanding*, 106(1):59–70, 2007.

[8] I. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. Maxout networks. *International Conference on Machine Learning*, 2013.

[9] J. Grefenstette, R. Gopal, B. Rosmaita, and D. Van Gucht. Genetic Algorithms for the Traveling Salesman Problem. *International Conference on Genetic Algorithms and their Applications*, 1985.

[10] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. *Computer Vision and Pattern Recognition*, 2016.

[11] G. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Improving Neural Networks by Preventing Co-adaptation of Feature Detectors. *arXiv preprint, arXiv: 1207.0580*, 2012.

[12] C. Houck, J. Joines, and M. Kay. A Genetic Algorithm for Function Optimization: A Matlab Implementation. *Technical Report, North Carolina State University*, 2009.

[13] G. Huang, Z. Liu, and K. Weinberger. Densely Connected Convolutional Networks. *arXiv preprint, arXiv: 1608.06993*, 2016.

[14] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Weinberger. Deep Networks with Stochastic Depth. *European Conference on Computer Vision*, 2016.

[15] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *International Conference on Machine Learning*, 2015.

[16] A. Krizhevsky and G. Hinton. Learning Multiple Layers of Features from Tiny Images. *Technical Report, University of Toronto*, 1(4):7, 2009.

[17] A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems*, 2012.

[18] S. Lazebnik, C. Schmid, and J. Ponce. Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories. *Computer Vision and Pattern Recognition*, 2006.

[19] 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.

[20] Y. LeCun, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel. Handwritten Digit Recognition with a Back-Propagation Network. *Advances in Neural Information Processing Systems*, 1990.

[21] C. Lee, P. Gallagher, and Z. Tu. Generalizing Pooling Functions in Convolutional Neural Networks: Mixed, Gated, and Tree. *International Conference on Artificial Intelligence and Statistics*, 2016.

[22] C. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu. Deeply-Supervised Nets. *International Conference on Artificial Intelligence and Statistics*, 2015.

[23] M. Liang and X. Hu. Recurrent Convolutional Neural Network for Object Recognition. *Computer Vision and Pattern Recognition*, 2015.

[24] M. Lin, Q. Chen, and S. Yan. Network in Network. *International Conference on Learning Representations*, 2014.

[25] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Ng. Reading Digits in Natural Images with Unsupervised Feature Learning. *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

[26] F. Perronnin, J. Sanchez, and T. Mensink. Improving the Fisher Kernel for Large-scale Image Classification. *European Conference on Computer Vision*, 2010.

[27] C. Reeves. A Genetic Algorithm for Flowshop Sequencing. *Computers & Operations Research*, 22(1):5–13, 1995.

[28] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, pages 1–42, 2015.

[29] K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *International Conference on Learning Representations*, 2014.

[30] L. Snyder and M. Daskin. A Random-Key Genetic Algorithm for the Generalized Traveling Salesman Problem. *European Journal of Operational Research*, 174(1):38–53, 2006.

[31] K. Stanley and R. Miikkulainen. Evolving Neural Networks through Augmenting Topologies. *Evolutionary Computation*, 10(2):99–127, 2002.

[32] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going Deeper with Convolutions. *Computer Vision and Pattern Recognition*, 2015.

[33] A. Torralba, R. Fergus, and W. Freeman. 80 Million Tiny Images: A Large Data Set for Nonparametric Object and Scene Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(11):1958–1970, 2008.

[34] N. Ulder, E. Aarts, H. Bandelt, P. van Laarhoven, and E. Pesch. Genetic Local Search Algorithms for the Traveling Salesman Problem. *International Conference on Parallel Problem Solving from Nature*, 1990.

[35] J. Wang, J. Yang, K. Yu, F. Lv, T. Huang, and Y. Gong. Locality-Constrained Linear Coding for Image Classification. *Computer Vision and Pattern Recognition*, 2010.

[36] L. Xie, R. Hong, B. Zhang, and Q. Tian. Image Classification and Retrieval are ONE. *International Conference on Multimedia Retrieval*, 2015.

[37] L. Xie, Q. Tian, J. Flynn, J. Wang, and A. Yuille. Geometric Neural Phrase Pooling: Modeling the Spatial Co-occurrence of Neurons. *European Conference on Computer Vision*, 2016.

[38] L. Xie, J. Wang, W. Lin, B. Zhang, and Q. Tian. Towards Reversal-Invariant Image Representation. *International Journal on Computer Vision*, 2016.

[39] L. Xie, J. Wang, Z. Wei, M. Wang, and Q. Tian. DisturbLabel: Regularizing CNN on the Loss Layer. *Computer Vision and Patter Recognition*, 2016.

[40] L. Xie, L. Zheng, J. Wang, A. Yuille, and Q. Tian. InterActive: Inter-Layer Activeness Propagation. *Computer Vision and Patter Recognition*, 2016.

[41] X. Yao. Evolving Artificial Neural Networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999.

[42] S. Zagoruyko and N. Komodakis. Wide Residual Networks. *arXiv preprint, arXiv: 1605.07146*, 2016.

[43] M. Zeiler and R. Fergus. Stochastic Pooling for Regularization of Deep Convolutional Neural Networks. *International Conference on Learning Representations*, 2013.

[44] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva. Learning Deep Features for Scene Recognition Using Places Database. *Advances in Neural Information Processing Systems*, 2014.

[45] B. Zoph and Q. Le. Neural architecture search with reinforcement learning. *International Conference on Learning Representations*, 2017.