Genetic CNN

Lingxi Xie, Alan Yuille
Department of Computer 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 some basic principles such as increasing network depth and constructing highway connections, researchers have manually designed a lot of fixed network architectures 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 motivates us to adopt the genetic algorithm to efficiently explore this large search space. The core idea is to propose an encoding method to represent each network structure in a fixed-length binary string. The genetic algorithm is initialized by generating a set of randomized individuals. In each generation, we define standard genetic operations, e.g., selection, mutation and crossover, to generate competitive individuals and eliminate weak ones. The competitiveness of each individual is defined as its recognition accuracy, which is obtained via a standalone training process on a reference dataset. We run the genetic process on CIFAR10, a small-scale dataset, demonstrating its ability to find high-quality structures which are little studied before. The learned powerful structures are also transferrable to the ILSVRC2012 dataset for large-scale visual recognition.

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 chain-styled network models [19], researchers have been increasing the depth of the network [32], as well as designing novel network modules [36][13] 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 reveal 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 stages, and each stage is defined as a set of predefined building blocks such as convolution and pooling layers. Even under these limitations, the total number of possible network structures grows exponentially with the number of layers, making it 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 exploring the space efficiently.

The genetic algorithm involves constructing an initial *population* of *individuals*, and performing genetic operations to allow them to evolve in an iterative process. We propose a novel encoding scheme to represent each network structure as a fixed-length binary string, and define several standard genetic operations, *i.e.*, selection, mutation and crossover, so that new competitive individuals are generated from the previous generation and weak ones are eliminated. The quality (*fitness function*) of each individual is determined by its recognition accuracy on a reference dataset. **To this end, we perform a complete training process for each individual (*i.e.*, network structure) which is independent to the genetic algorithm.** The genetic process comes to an end after a fixed number of generations.

It is worth emphasizing that the genetic algorithm is computationally expensive, as we need to undergo a complete network training process for each generated individual. We adopt the strategy to run the genetic process on a small dataset (**CIFAR10**), in which we observe the ability of the genetic algorithm to find effective network structures, and then transfer the learned top-ranked structures to perform large-scale visual recognition. **The learned structures, most of which have been less studied before, often perform better than the manually designed ones in either small-scale or large-scale experiments.**

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

Recent years have witnessed a revolution in visual recognition. Conventional classification tasks [20][8] are extended into large-scale environments [5][44]. With the availability of powerful computational resources (*e.g.*, GPU), the Convolutional Neural Networks (CNNs) [19][32] have shown superior performance over the conventional Bag-of-Visual-Words [3][38][29] and compositional models [9].

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 [22]. More recently, the availability of large-scale training data (*e.g.*, ImageNet [5]) and powerful GPUs make it possible to train deep CNNs [19] 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 [19], batch normalization [17], Dropout [34] and DisturbLabel [40].

Designing powerful CNN structures is an intriguing problem. It is believed that deeper networks produce better recognition results [32][36]. But also, adding highway information has been verified to be useful [13][42]. We also find some work which uses stochastic [16] or dense [15] structures. All these network structures are deterministic (although a stochastic strategy is used in [16] 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 natural selection process. It is commonly used to generate high-quality solutions to optimization and search problems [14][30][2][4] by performing bio-inspired operators such as mutation, crossover and selection.

A standard genetic algorithm requires two prerequisites, *i.e.*, a genetic representation of the solution domain, and a fitness function to evaluate each individual. A typical example is the travelling-salesman problem (TSP) [11], 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 distance of the path. We will show in Section 3.1 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 are often based on the properties of the specific problem. For example, in the TSP problem with the permutation-based representation, a possible mutation operation is to change the order of two visited nodes.

Researches are conducted to improve the performance of genetic algorithms, including performing local search [37] and generating random keys [33]. 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 learning the structure [35][1] or weights [41][6] of artificial neural networks, but our work aims at learning the architecture of modern CNNs, which is not studied in prior researches.

3. Our Approach

This section presents the genetic algorithm for learning competitive network structures. First, we propose a way of encoding a network structure into a fixed-length binary string. Next, genetic operations are defined, including selection, mutation and crossover, so that we can explore 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 recognition accuracy of each individual are obtained via a standalone training-from-scratch.

3.1. Binary Network Representation

We provide a binary string representation for a network structure in a constrained case. We consider those network structures [32][13] which can be organized in several *stages*. In each stage, the geometric dimensions (width, height and depth) of the data 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, *a.k.a.* data channels.

We follow 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 [17]

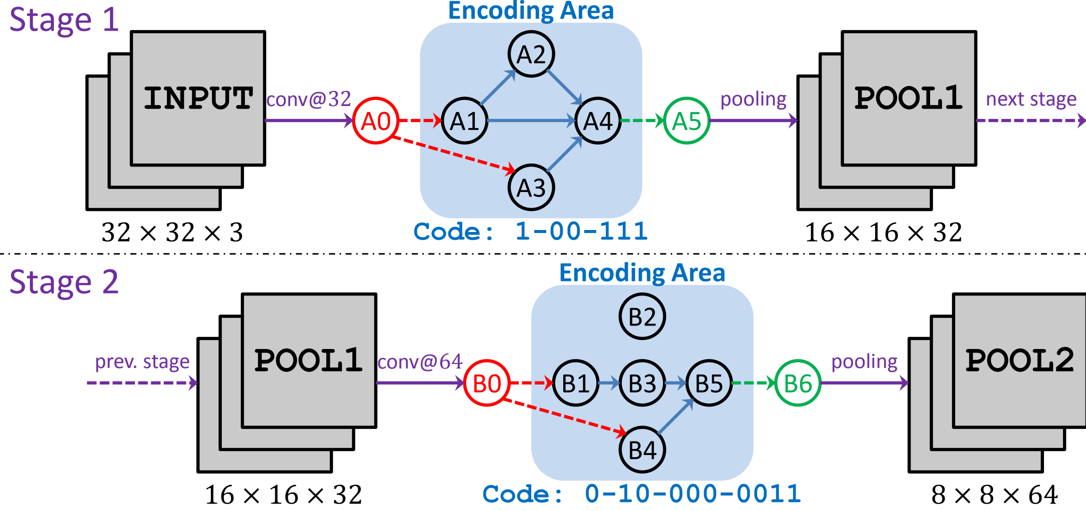

Figure 1. A two-stage network ($S = 2$, $(K_1, K_2) = (4, 5)$) and the encoded binary string (best viewed in color). The default input and output nodes (see Section 3.1.1) and the connections related to these nodes are marked in red and green, respectively. We only encode the connections between the ordinary codes (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.

and ReLU [19] are followed, which are verified efficient in training very deep neural networks [13]. We do not encode the fully-connected layers 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 bit 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. In summary, an S-stage network with K_s nodes at the s-th stage is encoded into a binary string of length $L = \frac{1}{2} \sum_s K_s (K_s - 1)$. Figure 1 illustrates an example of encoding a 2-stage network.

We note that the number of possible network structures (2^L) may be very large. In the **CIFAR10** experiments (see Section 4.1), 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). To this end, we use 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 performed by the default input node and the other by the default output node).

3.1.2 Examples and Limitations

Many popular network structures can be represented using the proposed encoding scheme. Examples include **VGGNet** [32], **ResNet** [13], and a modified variant of **DenseNet** [15], which are illustrated in Figure 2.

Currently, the encoded structures only involve convolutional and pooling operations, which makes it impossible to generate some tricky network modules such as Maxout [10]. Also, the convolutional kernel size and the number of channels are fixed within each stage, which limits the network from incorporating multi-scale information as in

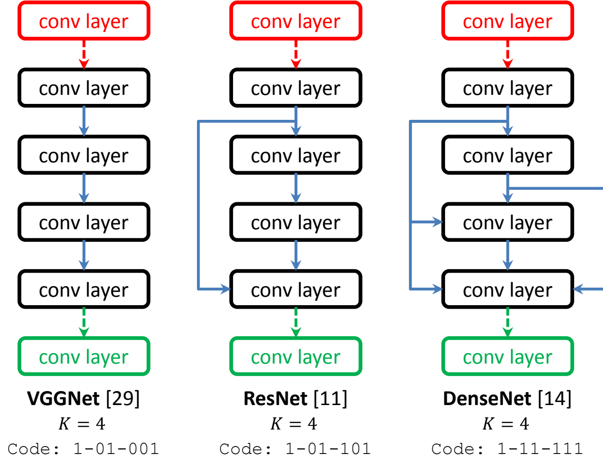

Figure 2. The basic building blocks of **VGGNet** [32], **ResNet** [13] and a variant of **DenseNet** [15] can be encoded as binary strings defined in Section 3.1.

the inception module [36]. We note that all automatically learned network structures have such limitations [45]. 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. We apply a strategy to learn network architectures on a small dataset, and transfer the top-ranked structures to large-scale visual recognition tasks.

3.2. Genetic Operations

The flowchart of the genetic process is shown in Algorithm 1. It starts with an initialized population 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 4.1.2, 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 via 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 performed in an overall flowchart (see Algorithm 1). The probabilities of mutation and crossover for each individual (or pair) are p_{M} and p_{C}, respectively. Of course, there are many different ways of performing mutation and crossover. In 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.

Algorithm 1 The Genetic Process for Network Design

1: **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}.
2: **Initialization:** generating a set of randomized individuals $\{\mathbb{M}_{0,n}\}_{n=1}^{N}$, and computing their recognition accuracies;
3: **for** $t = 1, 2, \ldots, T$ **do**
4: **Selection:** producing a new generation $\{\mathbb{M}'_{t,n}\}_{n=1}^{N}$ with a Russian roulette process on $\{\mathbb{M}_{t-1,n}\}_{n=1}^{N}$;
5: **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};
6: **Mutation:** for each non-crossover individual $\{\mathbb{M}_{t,n}\}_{n=1}^{N}$, doing mutation with probability p_{M} and parameter q_{M};
7: **Evaluation:** computing the recognition accuracy for each new individual $\{\mathbb{M}_{t,n}\}_{n=1}^{N}$;
8: **end for**
9: **Output:** a set of individuals in the final generation $\{\mathbb{M}_{T,n}\}_{n=1}^{N}$ with their recognition accuracies.

4. Experiments

Like other methods to learn network structures [45], our genetic algorithm requires a very large amount of computational resources, which makes it intractable to be directly evaluated a large-scale dataset such as **ILSVRC2012** [31]. Our strategy is to explore promising network structures on a small dataset, namely **CIFAR10** [18], then transfer these structures to the large-scale environment.

4.1. CIFAR10 Experiments

The **CIFAR10** dataset [18] contains 10 basic categories of 32×32 RGB images. There are 50,000 images for training, and 10,000 images for testing. To avoid seeing the testing data in the genetic process, we leave out 10,000 images from the training set for validation.

4.1.1 Settings and Results

The basic configuration follows a revised version of **LeNet** [21], and the network structure abbreviated as:
`C3(P1)@8-MP3(S2)-C3(P1)@8-MP3(S2)-C3(P1)@16-MP3(S2)-FC32-D0.5-FC10.`
Here, `C3(P1)@8` is a convolutional layer with a kernel size of 3×3, a default spatial stride of 1, a padding width of 1 and the number of kernels of 8. `MP3(S2)` is a max-pooling layer with a kernel size of 3 and a spatial stride of 2, `FC32` is a fully-connected layer with 32 outputs, and `D0.5` is a Dropout layer with a drop ratio of 0.5. Please note that we significantly reduce the number of filters at each stage to accelerate the training process. We apply 120 training epochs with a learning rate of 10^{-2}, followed by 60 epochs with a learning rate of 10^{-3}, 40 epochs with a learning rate of 10^{-4} and another 20 epochs with a learning rate of 10^{-5}.

We keep the fully-connected part of the above network unchanged, and set $S = 3$ and $(K_1, K_2, K_3) = (3, 4, 5)$. Within each stage, the first convolutional layer 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 population 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 $20 \times (50 + 1) = 1{,}020 \ll 524{,}288$. Training each individual takes an average of 0.4 hour, and the entire genetic process takes about 17 GPU-days. 10 GPUs are used, and each of them trains 2 networks in each generation. As a result, we can finish the entire genetic process in 2 days. We note that [45] trained $10\times$ more networks and each one is much more complicated, resulting in at least $100\times$ more computational overheads than our work.

We perform two individual genetic processes. The results of one of them are summarized in Table 1. With the genetic operations, we can find competitive network structures with improved recognition performance. Although over a short period the best individual may not be updated, 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 4.1.3, this facilitates strong individuals to be created, 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 24.04% to 22.81%. We also visualize the best structures found by these two processes in Figure 5.

4.1.2 Initialization Issues

We observe the impact of different initializations. 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 3. We find that, although the all-zero string corresponds to a very simple and

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	**77.19**	**75.34**	**76.58**	**76.81**	0.55	`1-01|0-01-100|0-11-101-0001`

Table 1. Recognition accuracy (%) on the **CIFAR10** testing set. The zeroth generation is the initial population. We set $S = 3$ and $(K_1, K_2, K_3) = (3, 4, 5)$. The best individual in each generation is also shown in binary codes.

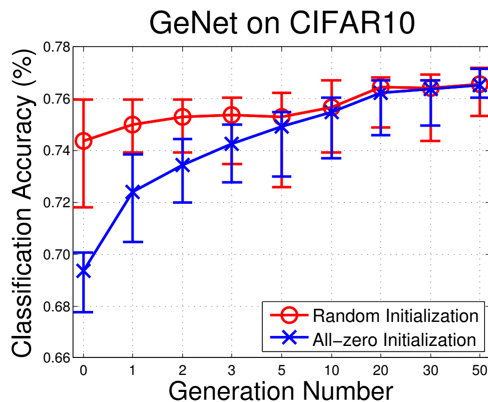

Figure 3. The average recognition accuracy over all individuals with respect to the generation number. The bars indicate the highest and lowest accuracies in the corresponding generation.

less competitive network structure, the genetic algorithm is able to find strong individuals after several generations. This naive initialization achieves the initial performance of randomized individuals with about 5 generations. After about 30 generations, there is almost no difference, by statistics, between these two populations.

4.1.3 Reasonability and Efficiency

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 genetic process, and observe the relationship between an individual and its parent(s). Figure 4 shows the results. We argue that the genetic operations tend to preserve the excellent "genes" from the parent(s), making it possible for the population to evolve after some generations.

We also investigate the efficiency of the genetic algorithm. To this respect, we randomly generate $20 \times (50 + 1) = 1020$ network architectures and evaluate each of them. The best individual in these 1020 candidates reports 76.94% accuracy, which is lower than the number (77.19%) obtained after the entire genetic process. From Table 1, we find that after 30 rounds, the genetic process is able to find an individual generating 76.95% accuracy, which suggests that the genetic process is much more efficient than random search in the large solution space.

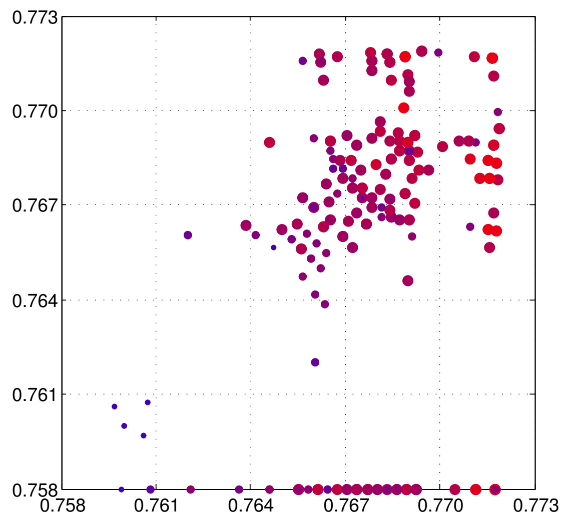

Figure 4. The relationship in accuracy between the parent(s) and the child(ren) (best viewed in color). A dot is bigger and close to red if the recognition rate is higher, otherwise it is smaller and close to blue. The dots on the horizontal axis are from mutation operations, while others are from crossover operations.

4.1.4 Parameters and Complexity

We note that the number of learnable weights of a network is related to the number of non-isolated nodes, since each of them contributes the same number of weights regardless of the number of lower-numbered nodes that are connected to it. In experiments, isolation rarely happens, and thus all the

individuals have a very similar number of parameters.

The number of 1-bits in network encoding (inter-layer connections) is the main factor of network complexity. However, we point out that a network with more 1-bits does not mean to dominate another with fewer 1-bits. As a direct evidence, we investigate the individual with all bits set 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 significantly lower than the number (77.19%) reported in Table 1. Considering that the densely-connected network requires heavier computational overheads, we conclude that the structures learned by the genetic algorithm are more effective and efficient than using dense connections.

4.1.5 Visualization

In Figure 5, we visualize the the network structures learned from two individual genetic processes. The structures learned by the genetic algorithm are somewhat 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, though obtained by independent genetic processes, are somewhat similar, which demonstrates that the genetic process generally converges to similar network structures.

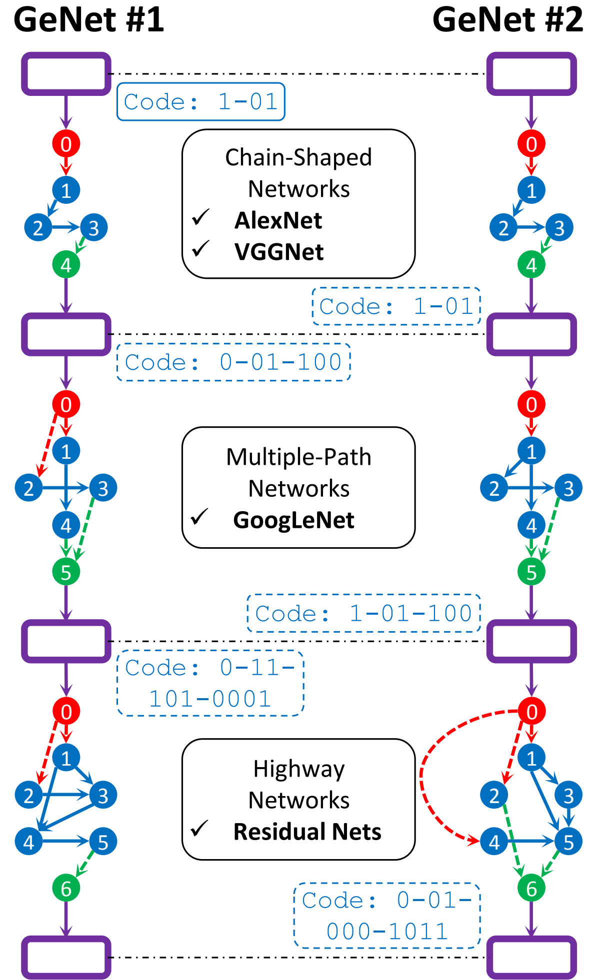

Figure 5. Two network structures learned from the two independent genetic processes on the **CIFAR10** dataset (best viewed in color). These are three-stage networks ($S = 3$) with $(K_1, K_2, K_3) = (3, 4, 5)$.

4.2. Small-Scale Transfer 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** which contains 100 categories at a finer level. It has the same numbers of training and testing images as **CIFAR10**, and these images are also uniformly distributed over 100 categories.

SVHN (Street View House Numbers) [28] 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 [28], *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 [10].

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.

We compare our results with some state-of-the-art methods in Table 2. First we note that the recognition accuracy goes up through the genetic process, which verifies the transfer ability of the learned network structures. Although these accuracies are lower than some state-of-the-art candidates [42][16][15], we note that these networks are much deeper (*e.g.*, 40–100 layers, compared to the 17-layer **GeNet** #1 and #2). For fair comparison, we start from the 40-layer wide residual network [42]. We create a population of 10 identical individuals, and perform genetic operation for 5 rounds. Each of the initialized individuals is a variant of the 40-layer network with a few bits randomly reversed. Using 10 GPUs to train these networks simultaneously, this process takes around 10 days. As a result, we find a better individual different from the original network structure, and the error rates on **CIFAR10**, **CIFAR100** and **SVHN** are 5.39%, 25.12% and 1.71%, respectively. This provides

	SVHN	CF10	CF100
Zeiler *et.al* [43]	2.80	15.13	42.51
Goodfellow *et.al* [10]	2.47	9.38	38.57
Lin *et.al* [26]	2.35	8.81	35.68
Lee *et.al* [24]	1.92	7.97	34.57
Liang *et.al* [25]	1.77	7.09	31.75
Lee *et.al* [23]	1.69	6.05	32.37
Zagoruyko *et.al* [42]	1.77	5.54	25.52
Xie *et.al* [39]	1.67	5.31	25.01
Huang *et.al* [16]	1.75	5.25	24.98
Huang *et.al* [15]	**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
GeNet from WRN [42]	**1.71**	**5.39**	**25.12**

Table 2. 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.

an alternative strategy to generate better architectures from existing manually designed ones.

4.3. Large-Scale Transfer Experiments

We evaluate the learned network structures on the **ILSVRC2012** classification task [31]. This is 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**. Training each network takes around 20 GPU-days.

Results are summarized in Table 3. 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-16** and **VGGNet-19**, because the original chain-styled stages are replaced by the automatically learned structures which are verified more effective.

Finally, we evaluate the transfer ability of the **GeNet**s on the **Caltech256** dataset [12]. We use **VGGNet-16**, **VGGNet-19** and **GeNet**s to extract 4,096-dimensional features on the first fully-connected layer, perform ReLU activation [27] followed by square-root normalization and ℓ_2 normalization, and feed the feature vectors to a linear SVM classifier [7]. With 60 training samples per category, the classification accuracy with **VGGNet-16** and **VGGNet-19** features are 82.69% and 83.51%, respectively. The **GeNet**s #1 and #2 produce 83.59% and 83.78% accuracies, which is slightly higher. This verifies that the benefits of **GeNet**s are generally transferrable to other visual recognition tasks.

	Top-1	Top-5	# Paras
AlexNet [19]	42.6	19.6	62M
GoogLeNet [36]	34.2	12.9	13M
VGGNet-16 [32]	28.5	9.9	138M
VGGNet-19 [32]	28.7	9.9	144M
GeNet #1	28.12	9.95	156M
GeNet #2	**27.87**	**9.74**	156M

Table 3. 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/`. **GeNet** #1 and **GeNet** #2 are the structures shown in Figure 5.

5. Conclusions

This paper applies the genetic algorithm to automatically learning the structure of deep convolutional neural networks. Our main idea is to use an encoding scheme to represent each network structure as a fixed-length binary string, and evaluate each generated individual via a standalone training process on a reference dataset. Based on this framework, we design some genetic operations, such as mutation and crossover, to explore the search space efficiently. We perform the genetic algorithm on a small reference dataset (**CIFAR10**), and find that the generated structures are able to transfer to the **ILSVRC2012** dataset and extracting deep features for other visual recognition tasks.

Despite the interesting results we have obtained, our algorithm suffers from several drawbacks. First, a large fraction of network structures are still unexplored, including some novel modules like Maxout [10], channel concatenation [36][15], and introducing multi-scale into convolutions [36]. In addition, the recurrent structure is also worth exploring [45]. 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. This work was supported by NSF CCF-1231216 and ONR N00014-15-1-2356. We thank Dr. Wei Shen, Yuyin Zhou and Siyuan Qiao for 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] R. Fan, K. Chang, C. Hsieh, X. Wang, and C. Lin. LIBLINEAR: A Library for Large Linear Classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

[8] 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.

[9] P. Felzenszwalb, R. Girshick, D. McAllester, and D. Ramanan. Object Detection with Discriminatively Trained Part-Based Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9):1627–1645, 2010.

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

[11] 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.

[12] G. Griffin, A. Holub, and P. Perona. Caltech-256 Object Category Dataset. *Technical Report: CNS-TR-2007-001*, 2007.

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

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

[15] G. Huang, Z. Liu, and K. Weinberger. Densely Connected Convolutional Networks. *Computer Vision and Patter Recognition*, 2017.

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

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

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

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

[20] 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.

[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.

[22] 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.

[23] 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.

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

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

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

[27] V. Nair and G. Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. *International Conference on Machine Learning*, 2010.

[28] 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.

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

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

[31] 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.

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

[33] 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.

[34] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A Simple Way to Prevent Neural

Networks from Overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

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

[36] 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.

[37] 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.

[38] 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.

[39] 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.

[40] L. Xie, J. Wang, Z. Wei, M. Wang, and Q. Tian. DisturbLabel: Regularizing CNN on the Loss Layer. *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.