Pattern Recognition 109 (2020) 107608

Contents lists available at ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/patcog

# Topological optimization of the DenseNet with pretrained-weights inheritance and genetic channel selection

Zhenyu Fang[a], Jinchang Ren[b,a,*], Stephen Marshall[a], Huimin Zhao[b], Song Wang[c], Xuelong Li[d]

[a] Centre for Signal and Image Processing, University of Strathclyde, Glasgow, UK
[b] School of Computer Sciences, Guangdong Polytechnic Normal University, Guangzhou, China
[c] Department of Computer Science and Engineering, University of South Carolina, Columbia, USA
[d] Center for Optical Imagery Analysis and Learning (OPTIMAL), School of Computer Science, Northwestern Polytechnical University, Xi'an, China

ARTICLE INFO

*Article history:*
Received 6 March 2020
Revised 9 July 2020
Accepted 19 August 2020
Available online 22 August 2020

*Keywords:*
Deep convolutional neural networks
Genetic algorithms
Parameter reduction
Structure optimization
DenseNet

ABSTRACT

Convolutional neural networks (CNNs) have been successfully applied in many computer vision applications [1], especially in image classification tasks, where most of the structures have been designed manually. With the aid of skip connection and dense connection, the depths of the models are becoming "deeper" and the filters of layers are getting "wider" in order to tackle the challenge of large-scale datasets. However, large-scale models in convolutional layers become inefficient due to the redundant channels from input feature maps. In this paper, we aim to automatically optimize the topology of the DenseNet, in which unnecessary convolutional kernels are reduced. To achieve this, we present a training pipeline that generates the network structure using a genetic algorithm. We first propose two encoding methods that can represent the structure of the model using a fixed-length binary string. A three-step based evolutionary process consisting of selection, crossover, and mutation is proposed to optimize the structure. We also present a pretrained weight inheritance method which can largely reduce the total time consumption of the genetic process. Experimental results have demonstrated that our proposed model can achieve comparable accuracy to the state-of-the-art models, across a wide range of image recognition and classification datasets, whilst significantly reducing the number of parameters.



## 1. Introduction

Visual object classification is a fundamental task for a wide range of applications. Traditional algorithms mostly conduct classification tasks in two steps: extraction of hand-crafted features, and classification using a particular classifier [2]. Those manually designed features only work well in specific applications, with relatively small data, and may fail once transferred to a new application, or a much large dataset. In recent years, Convolutional Neural Networks (CNNs) have become the dominant computer vision approach in image classification. Compared with the two-step traditional classification pipeline, CNN completes the task using an end-to-end method. The way of self-feature generation improves the robustness of CNN, producing the state-of-the-art results in many applications [3–5].

To yield higher level feature maps, the number of convolutional layers of the state-of-the-art CNN has increased generation by generation [3,6]. More recently, Residual Network (ResNet) [7] and Densely Connected Convolutional Networks (DenseNet) [8] have expanded this number to more than 100 layers. However, the channels in the inputs of a layer may be redundant. In [9], Veit et al. have found that the deep residual network performs as a combination of several shallow networks. The heat map of DenseNet (shown in Fig. 1) demonstrates the same effect: not all channels in the concatenated input feature maps are essential for the classification tasks. Only a few channels are weighted with a relatively high value and the rest are depressed. The redundant information from these low significance inputs will not only impede the prediction accuracy, but also cause a waste of computational resource.

In this paper, we aim to develop a robust approach to achieve a good trade-off between the efficiency and accuracy by selecting key channels from the inputs. The whole structure is self-generated without manual interference. Generating the model from scratch without constraints [10–12] would involve a huge amount of com-

* Corresponding author.
*E-mail address:* jinchang.ren@strath.ac.uk (J. Ren).

https://doi.org/10.1016/j.patcog.2020.107608
0031-3203/

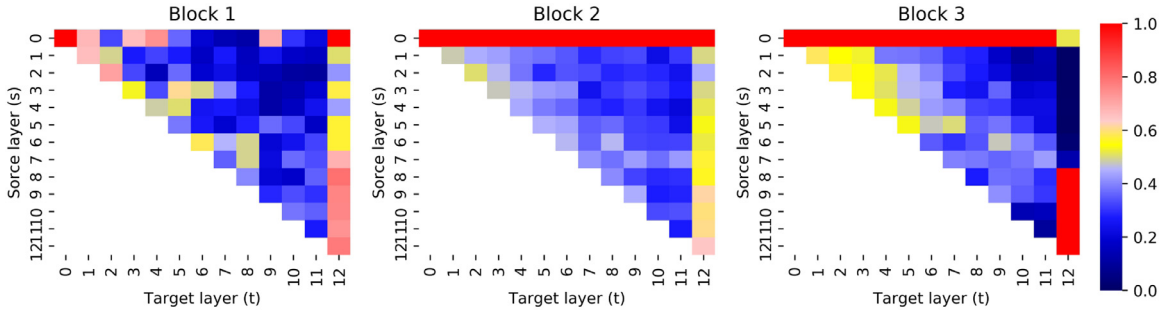



**Fig. 1.** The average absolute filter weights of convolutional layers in DenseNet40 ($G = 12$), trained on the CIFAR-10 dataset. The color patch ($s$, $t$) denotes the average L1 normalized (by number of input depth) weights in the layer "$t$" which takes input from the layer "$s$" in the same block. The last column of each block denotes the transition layer for the first two blocks and the fully connected layer for the last block, respectively. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

putational cost, and the generated model could easily suffer from overfitting. Thus, we constrain the framework of the model, which is learnt using state-of-the-art CNN models, as the "prior knowledge" of the model. Finding the best one by enumerating all the combinations is impractical, because the number of permutations increases exponentially with the number of layers. Rather than a full search of all the combinations, a more effective solution is to optimize the structure using neural architecture search (NAS), which may be based on an evolutionary algorithm (EA) [10,13], reinforcement learning (RL) [12,14], gradient descent [15], or other methods [16]. For the existing approaches, there exist the following drawbacks:

i. **High computational complexity and cost**: For neural evolution methods [10,13,17], denote the number of populations as $I$ and the number of the generation as $N_G$, the whole training procedure will be required to train $I \times N_G$ individuals. In order to achieve a higher performance, the values of both $I$ and $N_G$ should be sufficiently large. In [10], experiments were conducted on about 250 high-end computers. A similar limitation also exists in the reinforcement learning and gradient descent based method. The "discover" process also consumes a huge amount of computational resources. In [11], which adopts reinforcement learning, about 10 Graphic Processing Units (GPUs) are deployed.
ii. **More hyperparameters** are required to control the searching process;
iii. **The learnt architectures are not robust**, i.e. architectures obtained by EA perform well on the datasets they are trained on, but perform poorly on the new datasets.

To tackle these challenges, in this paper, an improved pipeline is proposed for training, in which the Genetic algorithm (GA) is employed to optimize the self-generated model. As we focus mainly on reducing the computational cost, we apply a binary encoding method to represent the structure of the model in a binary string, where "1" and "0" indicates whether the feature is allowed to pass into a layer or not. Three GA operations, selection, crossover and mutation, are employed to evolve the structure. After conducting the selection on each generation, poorly-performing structures are discarded. We measure the performance of the model by evaluating the accuracy on a validation/test dataset.

To improve the training efficiency, a variable-inheritance-fine-tune training method is proposed. Following the experimental settings in [10], instead of training each "individual" from scratch, we apply "variable inheritance" to reduce computational cost on each "individual". This means that a reused kernel will be reinitialized using the values obtained from training on the previous generation rather than randomly generated from a Gaussian distribution. We utilize the structure of the DenseNet as our baseline framework. Compared with the baseline, our model can reduce the number of parameters by up to 30% whilst maintaining the same accuracy. Although the GA procedure is mainly conducted on the CIFAR-10 dataset[18], the model produced also performs well on other datasets and is easily transferred onto large-scale datasets. The experiments can be conducted on a single GTX1080Ti GPU hence the model is very portable and affordable.

**Contributions**. the main contributions of this paper can be summarized as follows:

i. Based on the DenseNet, an effective GA training pipeline is proposed to select the key input channels of the convolution layers automatically;
ii. Under limited computational resources, mechanisms are proposed to optimize the GA process by applying 'weight inheritance' to reduce the total computational cost without degrading the training or testing accuracy of the models;
iii. The self-generated model structure can reduce the number of parameters by up to 30% while achieving a similar accuracy as the baseline with a significantly reduced computation time;
iv. Experiments show that our self-generated model structure performs well not only on the trained dataset, but also on untrained datasets. Even when the structure is generated using a small-size dataset, it was found to work well on large-scale image datasets.

The remaining parts of this paper are organized as follows. Section 2 gives a brief introduction to the related work. The design of the GA training pipeline and the experimental results are detailed in Section 3 and Section 4, respectively. Finally, some concluding remarks are given in Section 5.

## 2. Related work

Balancing the computational efficiency and the accuracy is a critical task in designing a convolutional neural network, as it can help to reduce the computational cost and facilitate more portable implementations i.e. on mobile or low power devices [19–21]. Two possible optimization approaches include: i) Optimizing the connection method between layers [7,8]; and ii) Optimizing the convolutional paradigm, e.g. the kernel size [6], and the activation function [22]. These two categories of methods can be conducted both by manual design and self-generation. Manual-design-based methods are more robust but require several attempts to achieve the best solution. Self-learning-based methods, on the other hand, can optimize the model architecture but may suffer from overfitting.

### 2.1. Manually designed CNN architectures

Manually designed CNN achieved great early success firstly on the MNIST dataset [3]. After that, AlexNet [4] was proposed for

large-scale practical image classification, where the state-of-the-art results were reported on the ILSVRC 2012 classification dataset [5]. However, a large CNN kernel size is inefficient, causing huge computational cost. To tackle this drawback, in VGG [6], only 3-by-3 and 1-by-1 convolutional kernels were applied, producing a deeper network with higher accuracy. Moreover, the computational cost can be largely reduced by using binary kernels [23] and regularization methods [24].

Batch normalization [22] reshapes the distribution of the input, which significantly improves the training efficiency. With increasing numbers of layers, the issue of gradient vanishing or gradient exploding emerges, where the weights of the network struggle to converge. To ease this training difficulty, the Residual Network (ResNet) [7] was proposed to allow groups of layers of the network to learn the difference between the input and the output rather than the input-to-output transformation for each layer. This has significantly reduced the total number of the parameters and allowed the number of layers to be extended to over 100. However, it is found that there are many redundant layers in the ResNet [9,25]. To further improve the efficiency of the information transformation, in Huang et al. [8], a dense connection strategy, DenseNet, is proposed to connect the outputs of all formal layers as the input of the following layer rather than sum all outputs in the ResNet. A comparable accuracy to the ImageNet was achieved while the number of parameters was reduced to 1/3 of the original ResNet. However, there still exist many unnecessary layers in the DenseNet, leaving potential for further optimization.

### 2.2. Neural evolution methods

To simulate the genotype-based evolution process of the nature, evolutionary algorithms (EAs) are proposed to optimize a model by encoding it to a binary string or even a string of integers or floating-point numbers. In [26], the network architecture evolved using the Neuro Evolution of Augmenting Topologies (NEAT) algorithm, in which the connection or disconnection between nodes was evolved through mutation.

When introducing the backpropagation algorithms for optimizing the weights of the neural networks, it becomes unnecessary to train such weights using an EA. This is because backpropagation algorithms help to converge the weights to the local minimum more easily and quickly. In order to gain the benefit of both the backpropagation and EAs, one strategy is to combine them [27], where they can be used to simultaneously train the weights of the neural network and optimize the architecture. Although these methods achieve comparable results to manually designed CNN models, they can only optimize the coding method and the training efficiency rather than the scale of the whole model. This has led to their models being too small to cope with large-scale image classification tasks. More recently, the scale of the model has been considered in the network architecture [10,13,17] that evolves the process to design the optimized architecture from scratch achieving comparable results on the CIFAR-10 dataset [18].

### 2.3. Neural architecture search (NAS)

Neural Architecture Search (NAS) is a process of automatic architecture engineering. In image classification, NAS methods have outperformed most of manually designed models. In this paper, two widely used NAS approach, **gradient** based NAS and the **reinforcement learning** based NAS, will be briefed. Other methods such as Bayesian Optimization (BO), pruning based, and meta-learning, as introduced in [16], will not be covered here, due to their inefficiency or low performance [15].

#### 2.3.1. Reinforcement learning NAS

In [12], reinforcement learning was used to generate a fixed length structure of a CNN layer by layer. Furthermore, the addition or removal of identical connections is also considered, which helps the model achieve state-of-the-art results. A similar training framework was utilized in [11], where Q-learning was applied to explore the structure of the model in each layer. Instead of training with a fixed length of "Gene string", the number of the layers was determined within the reinforcement learning itself. Mobile NAS (MNAS) [14] proposed a factorized hierarchical search space to balance the flexibility and the size of the search space, which was applied for light-weight network generation. To reduce the scale of the network, Facebook-Berkeley-Nets (FBNets) [28] utilized a layer-wise search space to specify the type of blocks for each layer.

#### 2.3.2. Gradient NAS

Instead of searching the structure via an "agent", gradient NAS optimizes the structure using gradient descent. A differentiable NAS method (DARTS) was proposed in [15], which was applied on both convolutional and recurrent networks. To increase the computational efficiency, stochastic NAS (SNAS) [29] replaced the feedback mechanism with more efficient gradient feedback from generic loss.

### 2.4. Training strategy of evolution based CNN

To optimize the architecture of the CNN, each network model is considered as an "individual" and its fitness value is measured by using the classification accuracy on the validation datasets [10,13,17]. After the mutation and crossover (excluding [10], which replaces the crossover by computing a huge number of combinations), the model is retrained to determine the fitness value individually. Each new individual can be either fully trained i.e. trained 100 epochs on the CIFAR-10 dataset [10,13,17] or partially trained [30] i.e. 5 epochs on the same dataset. In comparison to the fully trained approach, partial training improves the training efficiency but suffers from overestimation or under estimation caused by random initialization from a Gaussian distribution. In Real et al. [10], it is found that an alternative training strategy inherits benefits from both fully and partial training, where each "individual" is partially trained after structure evolution from the full training. If a variable is reused, its value will be taken from the last generation instead of being reinitialized from scratch.

Similar works have been conducted in [10] and [13]. However, a large search space makes both methods hard to be optimized under limited computational resources. The search space of the proposed method is the input channels of each layer, while in [10] and [13] the search space is the connection method between different layers. Even the elements of a layer are also considered in [13]. To improve the training speed and reduce the computational cost, we do not discard the crossover step as used in [10]. Experimental results in Section 4 show that, by using crossover, the total computational cost can be dramatically reduced whilst maintaining the test accuracy. The original weight inheritance method [10] which is optimized for the residual structure, is not fit for the densely connected structure. As a result, we propose a new pretrained weight inheritance method to initialize the weights of the model, which is more suitable for densely connected structures.

## 3. The proposed algorithm

In this section, a detailed description of the proposed approach is presented which discusses how a GA-based method can be used to remove the redundant convolutional kernels. Training from scratch without any constraints is infeasible. Even with very few constraints, the training process can be significantly speeded up

whilst reducing the risk of overfitting to the referenced dataset ("referenced dataset" refers to the dataset where each "individual" is trained). To this end, we borrow from the frameworks of manually designed models to constrain the model architecture.

Specifically, our model is based on two state-of-the-art architectures, the ResNet and the DenseNet styles with skip-connect inputs between different layers with dense connection. More details of the constraints are discussed in Section 3.1. As the GA is applied to evolve the architecture of the model, in Section 3.2, two binary coding methods are proposed based on previous works [10,13]: channel coding and layer coding. In Section 3.3, a simple but efficient genetic process is detailed, which consists of the selection, crossover and mutation operations.

To improve the training efficiency without degrading the model performance, in Section 3.4 we propose: 1). a partial-training and fine-tuning strategy to obtain the fitness value of the evolved structure and 2). a pretrained weight inheritance strategy to initialize the weights of the model, which is more suitable for densely connected structures and easy to implement.

### 3.1. Pre-defined elements of the model

In [10,12,30], it is suggested that the architecture of a model generated from scratch with very few constraints is more efficient than manually designed model, because of the benefit brought by mutation. However, the self-generated structure paradigm may be less robust than manually designed methods [10,18]. On the other hand, generating a model without constraints consumes a huge amount of computational resources [10,12]. As a result, in this paper, to increase the robustness while reducing the computational cost, we pre-define the following constraints by taking some manually-designed structures [7,8,25] as the "prior" of our model:

i. Similar to the ResNet and DenseNet, the network will be split into N blocks, in which the feature sizes of layers within each block are the same. Its width and height will be halved by a "transmission layer" [8] before passing to the next block.
ii. A "transmission layer" consists of 1) a 1-by-1 convolutional layer for fusing all channels and 2) a 2-by-2 mean pooling layer for down-sampling the feature maps. The depth of the output of a transmission layer will either remain the same or be halved depending on the structure of a "convolutional layer". This allows consistent comparisons with other approaches. Therefore, if a convolutional layer contains only one 3-by-3 kernel followed by a batch normalization (BN) layer [22] and a ReLU layer, the depth remains the same. If a convolutional layer contains two convolutional sublayers, one with a 1-by-1 kernel and the other with a 3-by-3 kernel (both with BN-ReLU before each convolution as well), known as the "bottleneck structure", the depth will be halved. We label the "compressed-bottleneck structure" as BC, and only the BC structure will be used after the architecture is generated.
iii. The channel number of the convolution layer is unchanged in each block, denoted as "growth rate" G [8].
iv. The number of layers in each block is fixed during the training.
v. The skip-connection is allowed. Features from different layers will be processed by concatenation [8] instead of addition [7].

### 3.2. Coding method

To improve the training efficiency and attain a comparable performance while reducing the computational cost, binary coding [13] and the GA are used to evolve the architecture of the model, instead of integer/float number coding [10]. For each convolutional layer, we assign a binary string to "gate" the input, which is a concatenation of the outputs of all the previous convolutional layers. Hence, the search space corresponds to a binary string representing the structure of the model. For each bit, "1" and "0" means the associated feature can "pass" into the layer or not. Each bit of the string can represent either a single channel or multiple channels of the input feature map. Here we propose two binary coding methods, i.e. Channel coding and Layer coding.

#### 3.2.1. Channel coding

In channel coding, each bit represents only one channel of the concatenated feature map. Therefore, for a three-convolutional-layer block with the number of initial input channels of $N_{init}$ and a growth rate of $G$, the length of the strings for all three layers are $N_{init}$, $N_{init}+G$ and $N_{init}+2G$, respectively. Thus, the length of the string of the block is $3N_{init}+3G$. As such, the length of binary string $L$ for a single block is

$$\begin{aligned} L_{channel} &= (N_{init}+G)+(N_{init}+2G)+\cdots+(N_{init}+(C-1)G) \\ &= C\times N_{init}+\frac{C(C-1)}{2}G \end{aligned} \tag{1}$$

where $C$ is the number of convolutional layers in the block.

In channel coding, the model can be more flexible than the plain DenseNet for training as it can fit a densely connected structure with an arbitrary number of layers and an arbitrary number of filters in each layer if $G$ is small enough and $N_{init}$ is large enough. An example is illustrated in Fig. 2. Assume a single CNN block has 3 layers (labelled as "0", "1", and "2") with a growth rate of 3, a channel coding string to represent the input (from the layer "0" and layer "1") of the last convolutional layer (layer "2") will be "**101-011**".

#### 3.2.2. Layer coding

Channel coding works well when the scale of the model is small. When the number of the layers and the growth rate are large, the length of the string for the whole model becomes very long and difficult to train. For a model with $N_{init}=24$, $C=12$ and $G=12$, when the number of blocks is 3 ($N=3$), the length of the binary string for the first block, calculated from Eq. (1), will be 1080. An effective way to reduce the length of the binary string is "bit-sharing", which is referred to as "Layer coding" in this paper. In layer coding, each bit represents the status of the channels from the same convolutional layer. The length of the layer coding can be determined by

$$\begin{aligned} L_{layer} &= 1+(1+1)+(1+2)+\cdots+(1+(C-1)) \\ &= \frac{C(C+1)}{2}, \end{aligned} \tag{2}$$

where $C$ is the number of convolutional layers in the block.

The layer coding above can significantly reduce the training difficulty, however, it only fits to a model whose layer depth is the magnitude of the growth rate (ignoring the initial filter size). Taking the same model in Fig. 2 for example, if we apply layer coding to the third layer, the length will become only 1/3 of the channel coding if represented as "**0-1**".

Although channel coding is more flexible, the length of its string is limited by the fixed growth rate and will increase the training difficulty under limited training steps. On the other hand, the model generated by layer coding can support arbitrary growth rates according to the scale of the dataset. Experimental results in Section 4 indicate that channel coding leads to slightly higher accuracy than the layer coding on the MNIST dataset but performs worse when transferred to the CIFAR-10 dataset. Thus, to balance the computational cost and the accuracy, we will apply the layer coding method to code our model.

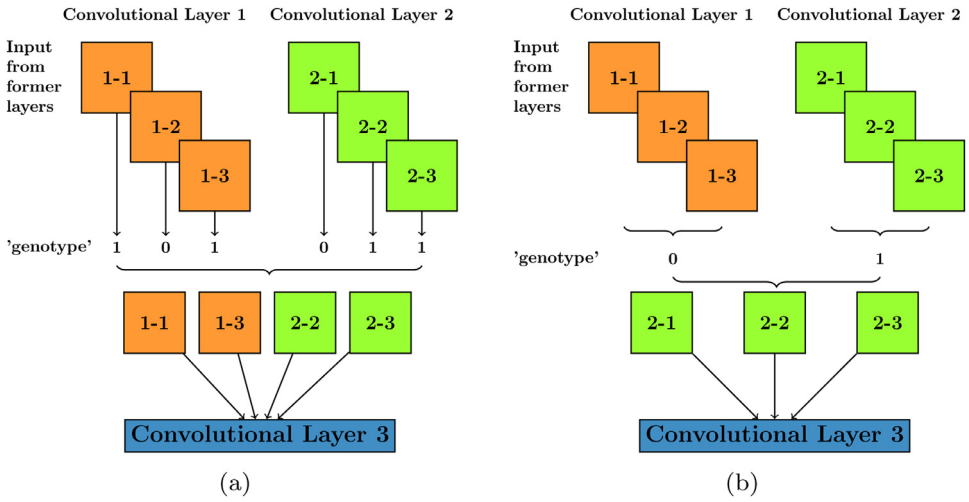



**Fig. 2.** An example to show how the channel coding (a) and the layer coding (b) filter the input of a convolutional layer: The layer has two preceding layers with $G = 3$. Features from layers 1–2 are highlighted by orange and green, respectively. A channel coding string to represent the input (from the layers 1–2) of the last convolutional layer (layer 3), calculated using Eq. (1), will be "**101-011**". For the channel coding, the length of this binary string is 6, where each bit represents a single channel of the input. While for layer coding, each bit represents channels from a layer. Therefore, the binary string is represented as "**0-1**" in Eq. (2). Compared with channel coding, the length of layer coding will reduce to only 1/3. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

**Algorithm 1.** A Python style pseudocode of the genetic process in our proposed method. The corresponding flowchart is shown in Figure 3.

**Genetic process of structure evolution**

**I**: initial population
**T**: The number of generations to conduct evolving process
**P**: parent population is denoted

**Weight Initialization**
Fully train the baseline (i.e. DenseNet) on the reference dataset **D**
**Individual Initialization**
(1) Generate individuals in **I** via **B(0.5)**
(2) Partially train and evaluate the individuals on **D**
for **i** in range(**T**):
  **P** = [ ]
  for **ii** in range(length(**I**)):
    Random select **S** individuals from **I**
    Select the best individual, and save it into **P**
  **Crossover**
  for **iii** in range(length(**I**)/2):
    Conduct crossover for **P**[*iii*] and **P**[*iii* + 1] with $\mathbf{P_c}$ and $\mathbf{Pb_c}$
  **Mutation**
  for **iiii** in range(length(**I**)):
    Conduct mutation for **P**[*iiii*] with $\mathbf{P_m}$ and $\mathbf{Pb_m}$
  **Evaluation**
  Evaluate **P** on **D**
  **Best Selection**
  Save and store the best individual in the generation **i**

### 3.3. GA-based structure evolution

The genetic process for evolving the architecture of the model is given in Algorithm 1. In the Weight initialization step, after pretraining the baseline model, weights will be initialized using the values derived from the baseline model as detailed in Section 3.4. As the possible values of each bit in the binary string are '0' and '1', we randomly initialize each bit using the Bernoulli distribution with a probability of 0.5, i.e. B(0.5). The fitness value for each "individual" in the first generation is the classification accuracy of the model on the validation dataset D after fine-tuning of training. The number of individuals, i.e. "population size" $I$, in the first generation will remain the same during the following iterations. The evolving process will conduct $T$ generations. Following the suggestions in [13], we assign three evolutionary operations to evolve the architecture of the model: selection, crossover and mutation.

For the selection operation, we use tournament selection [31] as it is commonly used in similar applications [10,13]. After that, in the crossover part, bits from different individuals but at the same position in the string are randomly "swapped". The probability of crossover for each pair is $P_C$ and the probability of crossover of a bit is denoted as $Pb_C$. During crossover, the structures of a pair are modified simultaneously, and we conduct mutation to bring more variance to the structure. The probability of mutation for each pair is $P_M$ and the probability of mutation of a bit is $Pb_M$. To avoid overfitting, we automatically flip the first and the last bits to "1" rather than discarding the layer as did in [10,13]. Experimental results in [10] indicates that, without this constraint, the model generated on the CIFAR-10 has fewer layers. However, when applying on the CIFAR-100 dataset, the generated structure becomes suboptimal. As a result, the number of layers should not be stabilized during the evolution process.

### 3.4. Pretrained weight-inheritance based individual training strategy

After the structure evolution, some "individuals" need to be retrained to determine the fitness value. Using the approach, which is partially trained from scratch [30], is hard to converge the weights to the global optimal, causing underestimation or overestimation of the model. Meanwhile, the fully trained method [13] has an extremely high computational cost and takes too long to converge. In the original weight inheritance from partially training method [10,30], weights inherit their values from the last update and are reused during the training instead of initializing from scratch, generating a well-performed model structure. The original weight inheritance method [10] guarantees that reused weights are actually fully trained, without being constrained by the limited training steps of each generation. However, this strategy assumes that the model takes the residual structure as the baseline without implementing a concatenation operation. In a residual structure, the output can be interpreted by

$$X_{n+1} = F(X_n)_n + X_n, \tag{3}$$

where $X_n$ is the input of the $n^{th}$ layer, $F(*)_n$ is the convolution operation of layer $n$ which may contain two convolution sublayers with a batch normalization and ReLU connected after each of those [7], or three convolutional sublayers known as the "bottleneck" structure [7]. In a densely connected structure, the output can be given

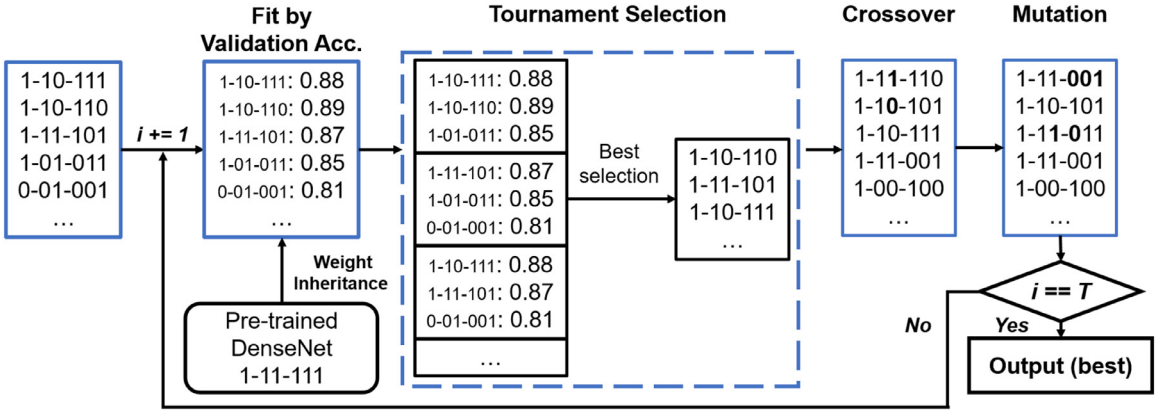



**Fig. 3.** Flowchart of the architecture optimization process using genetic algorithm. The number of layers in this example is 3. As a result, the length of genotype, encoded using layer coding, is 6. Bits presenting different layers are sequentially separated by "-". Take the network encoded with "1-10-111" for example, the first two layers use the initial feature map as the input; both the initial feature maps and the outputs of the first two layers are used as the input for the third layer. The genetic algorithm consists of three operators: tournament selection ($S = 3$), crossover and mutation. Bits updated by crossover and mutation are highlighted by bold in the figure. At the end of the process, the architecture with the best performance is chosen as the final output.

as

$$X_{n+1} = F([X_n, X_{n-1}, \ldots, X_1])_n, \tag{4}$$

where $[X_n, X_{n-1}, \ldots, X_1]$ is the concatenated inputs from all former layers of the layer $n$.

As seen in ResNet, the convolutional operation minimizes the residual error between the input and the output, hence its magnitude is usually very small. As suggested in Veit et al. [9], the performance of the ResNet will not be significantly affected when removing several layers. Thus, the modification of the weights in a layer only slightly affects the outputs of the following layers and can be easily fine-tuned by a few training steps. However, in a densely connected structure, the output of a layer will be connected as part of the input to all the following layers, which has a direct effect on. The modification of a layer in a densely connected structure will strongly affect the structure of the model, leading to possible oscillation of the optimization process. Experimental results in Section 4 shows that the performance of the densely-connected model trained using the original weight inheritance strategy is almost equivalent to that trained from scratch. The fitness values of individuals oscillate with a small margin between generations. On the other hand, the original weight inheritance method must record all the latest trained variables regardless of whether they will be reused or not, which is flexible for models without a predefined framework. However, in our training pipeline, the total number of variables is fixed, and the framework is predefined. Recalling the weights and reformatting the structure each time when the training starts is inefficient and unnecessary.

To tackle this issue, we propose a pretrained weight inheritance strategy to train each "individual". Before the evolutionary procedure starts, we first set all bits on the genotype string to one as the baseline and fully train the baseline. The length of the model is fixed, and its growth rate remains the same during the procedure, the structure of which is the same as the plain DenseNet. When the evolutionary procedure starts, the weights from each "individual" will be initialized using the values from the well-trained baseline instead of from scratch. Each "individual" will be fine-tuned for several epochs to optimize the fitness value, i.e. each "individual" is partially trained. All fine-tuned weights will not be inherited by the next generation, and the weights of the next generation will be initialized using the values from the baseline as well. To achieve this, during the fine-tuning procedure, the binary string will be partitioned back onto each layer (when conducting evolutionary steps, binary strings from different layers are concatenated together as discussed in Section 3.2.2) and act as a binary mask for each channel of the input.

For each filter, the forward-propagation process and the back-propagation process are mathematically interpreted as follows:

**Forward** :

$$\begin{aligned} X^m_{n+1} &= \textstyle\sum_{i=1}^{D} bin_i \times F(X_i, K^m_i)_n \\ &= \begin{cases} \sum_{i=1}^{D} F(X_i, K^m{}_i)_n, & if\ bin_i = 1 \\ 0, & if\ bin_i = 0; \end{cases} \end{aligned} \tag{5}$$

**Backward** :

$$\begin{aligned} \tfrac{\partial Loss}{\partial K^m_i} &= \tfrac{\partial Loss}{\partial X_{n+1}} \times \tfrac{\partial X_{n+1}}{\partial K^m_i} \\ &= \tfrac{\partial Loss}{\partial X_{n+1}} \times bin_i \times \partial F(X_i, K^m_i)_n \\ &= \begin{cases} \tfrac{\partial Loss}{\partial X_{n+1}} \times \partial F(X_i, K^m_i)_n, & if\ bin_i = 1 \\ 0, & if\ bin_i = 0; \end{cases} \end{aligned} \tag{6}$$

where $X^m_{n+1}$ denotes the $m_{th}$ channel of the input for the $(n+1)^{th}$ layer; $D$ is the depth of the input $X$; $bin_i$ denotes the binary bit of the $i_{th}$ channel of $X$ (channels generated by the same layer will have the same bit value, as described in Section 3.2.2); $K^m{}_i$ denotes the $m_{th}$ channel of the kernel and $F(*)_n$ denotes the convolutional operation of the layer $n$.

Let the size of the input $X$ of a layer be $32 \times 32 \times 3$, it can be generated by three convolutional layers beforehand, and the size of the baseline kernel (denoted as $K$) of the layer is $3 \times 3 \times 3 \times 24$. The length of the corresponding binary string will be 3. If the string is "$1-1-0$", it is obvious that the first two channels of each filter can be trained using the back-propagation algorithm. However, the last channel of each filter is excluded from both the forward propagation and the backpropagation as its input is an all-zero feature map.

## 4. Experimental results and discussions

### 4.1. Datasets and pre-processing

#### 4.1.1. MNIST

The MNIST [3] is a handwritten digit dataset for recognition tasks of digits from 0 to 9, which contains 60,000 images for training and 10,000 for testing. As the number of the epochs for fully training is small, we do not split a validation dataset during the training. We do not apply any data augmentation or pre-processing

during the training of this dataset in order to reproduce consistent conditions to other approaches.

#### 4.1.2. CIFAR-10 and CIFAR-100

The CIFAR-10 and CIFAR-100 datasets [18] contain colored natural images in 10 classes and 100 classes, respectively. Both datasets have 50,000 images for training and 10,000 for testing, and each image has 32 × 32 pixels. We split 5000 images from the 50,000 training images as a validation dataset and keep the remaining 45,000 images for training. The data augmentation method for the two CIFAR datasets is the same as used in [7,8,32], where 4 pixels are padded on each side of the original image or its horizontal flip. A 32 × 32 image is cropped randomly from the padded image. When testing, the input images remain the same as the original without padding and randomly cropping. For pre-processing, pixel-based normalization is used to normalize the image using the channel means and the channel standard deviations. For a fair comparison with other methods [8,13,17,30], we use the CIFAR-10 training dataset as the reference dataset (in the genetic process) to generate the structure of model and the benchmark dataset (in the model evaluation process). After that, we fix the structure and use the CIFAR-10 and CIFAR-100 datasets as benchmark datasets to evaluate the performance of the learned model as in [8,25,32]. For the CIFAR-10 dataset, we do not train the model on the validation dataset when it acts as a reference dataset. When it acts as a benchmark dataset, we train our model on the validation dataset only at the final epoch as in [8].

#### 4.1.3. Street view house numbers (SVHN)

Similar to the CIFAR-10 dataset, the SVHN dataset [33] is also an image dataset of $0-9$ digits with a size of 32 × 32 pixels each. In addition to 73,257 training images and 26,032 testing images, there are 531,131 images for extra training. We only use the SVHN dataset as the benchmark dataset. For a fair comparison with other models, we do not apply any data augmentation step and train the model using all images from the training dataset and the extra training dataset except for 6000 images used as a validation dataset as did in [8,25]. We only normalize the range of the images from [0, 255] to [0, 1] as data pre-processing [8,34]. We load the weights of the model with the lowest validation error during training and evaluate it on the test dataset.

#### 4.1.4. ImageNet

The ILSVRC 2012 classification dataset [5] is a large-scale image dataset. which consists of 1.2 million training images and 50,000 validation images, uniformly distributed in 1000 classes. We adopt the augmentation method as used in [7,8], where the per-pixel normalized image is resized without modifying the *width*/*height* ratio as scale augmentation. A 224 × 224 sub-image is randomly cropped from the scaled image or its horizontal flip with color augmentation. We only apply the data augmentation during training. As most of methods evaluate their results on the validation set [6–8,15,35,36], we also test our model on the validation set in the same way, and report the single-crop classification errors.

### 4.2. Channel coding vs layer coding

To fairly compare the performances between the channel coding and the layer coding methods, we generate the structure of the model using the MNIST dataset and evaluate the results on both the MNIST and CIFAR10 datasets. We assign 3 blocks in the model, i.e. $N = 3$, and 8 convolutional layers without a bottleneck structure in each block. The number of outputs of each convolutional layer is 8, i.e. $G = 8$. The depth of the input image is expanded to 16 using a 3-by-3 convolutional layer before entering the first block. The model is completed with a global average pooling, a 10-output fully-connected layer, and the softmax output. As the MNIST is easy to train, we did not apply any weight inheritance strategy, and every individual is fully trained from scratch.

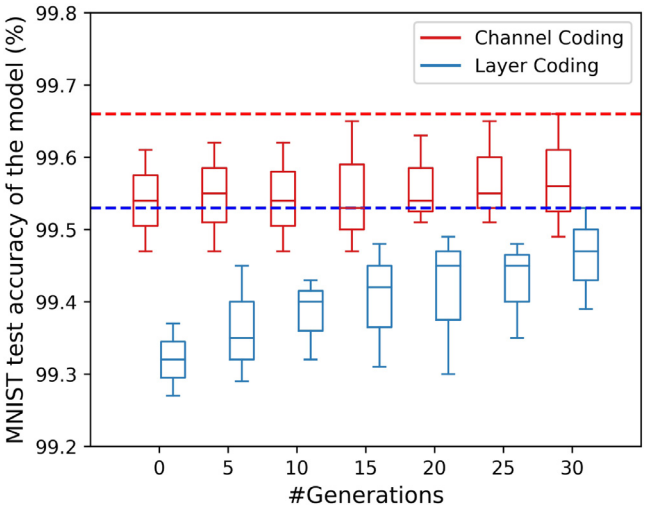



**Fig. 4.** Test accuracy (%) of the model on the MNIST dataset using the two coding methods. The genetic process conducts 30 generations for both methods separately and records the maximum, minimum and average test accuracy of each generation. It is obvious to see that the results trained using channel coding method (denoted by dash red) outperforms the layer coding method-based training (denoted by dash blue) by **0.13%**. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

#### 4.2.1. Training implementation

We use a weight decay of 0.0001 and optimize the model using Stochastic Gradient Descent (SGD) with a momentum of 0.9. Weights are initialized by using the method in [8]. These models are trained with a batch size of 64 on a single GTX1080Ti GPU, where each "individual" is trained by 20 epochs on the MNIST dataset. We start with a learning rate of 0.1, then divide it by 10 at 5 epochs, 10 epochs and 15 epochs. When the generated architecture of the model is transferred to the CIFAR-10 dataset, we fix the structure and fully train the model from scratch. Following the settings on [8], the model is trained by 300 epochs on the CIFAR10 dataset and the learning rate is initialized as 0.1, divided by 10 at 150 and 225 epochs with the same weight decay and batch size as used on the MNIST dataset.

#### 4.2.2. Genetic hyperparameters

Following the parameter setting used in [10,13], we set the population size as $I = 20$ and the number of generations $T = 30$. The sample size of tournament selection is $S = 3$. The probability of the pair-wise crossover and the bit-wise crossover for each "individual" are $P_C = 0.2$ and $Pb_C = 0.2$, respectively. The probability of the individual mutation and the bit mutation are $P_M = 0.8$ and $Pb_M = 0.05$, respectively. It takes about 3.2 GPU days to conduct each genetic process, and the experimental results are given in Fig. 4.

A similar setting for the number of generations and the population size can be found in [10] and [13]. In [13], the number of population is 20 and the number of generations is 50. The generated model has no specific gains after 30 generations. In [10], the method with a population size of 2 has the best result at an early stage. For our model, when the number of generations is set to 50, the best model is found from the 24th generation as shown in Fig. 5.

This is because: 1). Both MNIST and CIFAR are small datasets, for which it is easy to find the optimal models; 2). Compared with manually designed models such as the DenseNet and ResNet with over 100 layers, the model sizes of self-structure generation from

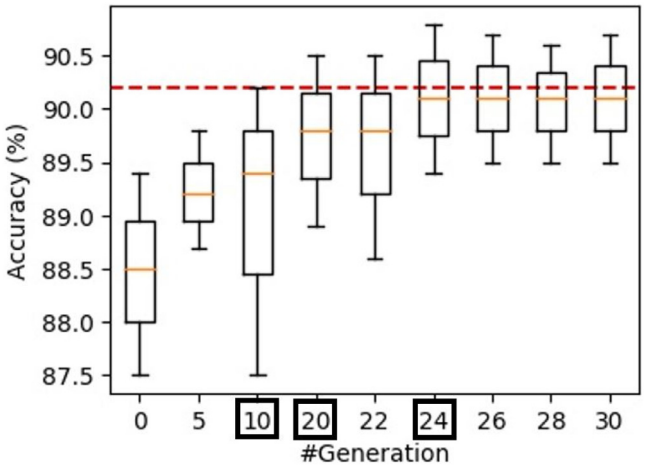



**Fig. 5.** Test accuracy (%) of the model on the CIFAR-10 dataset using our training pipeline given in Section 3 with the genetic process conducted over 30 generations. We denote the baseline (**Generation 0: 90.2%**) using the red dash line and highlight the key generation IDs using a black box. With the performance comparable to the baseline at **Generation 10** (**90.2%**), the model outperforms the baseline starting from **Generation 20** (**90.5%**) and achieves the best among all individuals at **Generation 24** (**90.8%**). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

**Table 1**
Relationship between the Number of Generation and the Test Accuracy on CIFAR-10.

| Test accuracy of the fully trained best structure(%) | | |
|---|---|---|
| #Generations | Channel coding | Layer coding |
| 10 | 90.40 | 90.30 |
| 20 | 90.87 | 91.10 |
| 30 | 91.23 | **92.63** |
| 50 | **92.69** | 92.60 |

[10,13] and our proposed approach are relatively small, i.e. around 20–30.

We fully train the model with the best structure on the CIFAR-10 dataset, where an accuracy of **88.3%** and **92.3%** are achieved from the channel-coding based model and layer-coding based model, respectively. When tested on the reference dataset (MNIST), channel coding slightly outperforms layer coding but underperforms the layer coding by a large margin on transfer learning. We also conduct a similar experiment on the CIFAR-10 dataset for both methods, and the results are shown in Table 1. It turns out that the channel coding method requires **66%** more generations than the layer coding to reach the same test accuracy (**92.6%**).

In summary, the channel coding method is capable of generating a more competitive model than the layer coding as long as the number of generations is sufficiently large, while layer coding performs better at transfer learning. To balance the training cost and the performance of transfer learning, **layer coding is applied in all the following experiments**.

### *4.3. Weight inheritance methods*

In this section, we compare the difference between the original weight inheritance in [10] and our proposed pretrained weight inheritance strategy. We use the layer coding method to derive the architecture of the model during the genetic process. The baseline structure of the model is the same as discussed in Section 4.2 with the same weight decay. We use the CIFAR-10 dataset as the reference dataset $D$ and implement the Adam optimizer [37] in a weighted training pipeline. For the original method, each "individual" is trained around 30 epochs with a fixed learning rate of

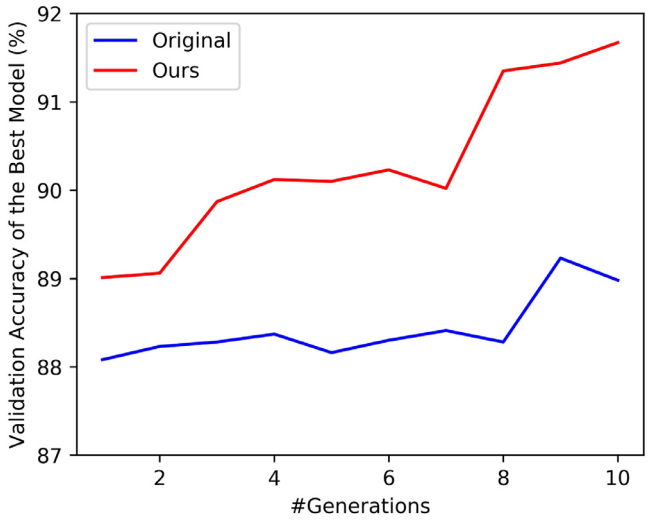



**Fig. 6.** Comparison of test accuracy on the CIFAR-10 dataset using the original (blue line) and our (red line) weight inheritance methods. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

0.01 and evaluated on the validation dataset. After the evolutionary steps, each model inherits the weights trained from the last generation. For our pretrained weight inheritance method, we first fully train the model with all bits set to 1 in the binary string under the same learning rate fixed at 0.01, which makes it similar to the DenseNet. All weights are stored in a "checkpoint model". During the fine-tuning procedure, each "individual" is trained for 30 epochs, which is the same as the original method. We also divide the learning rate by 10, i.e. 0.001, to avoid the weights oscillating between the local minima and the global minima. The genetic hyperparameters are the same as in Section 4.2 except that the number of generations is reduced to 10, i.e. $T = 10$. Each of the genetic training procedure takes about 3.2 days on a single GTX1080Ti.

Experimental results are summarized in Fig. 6. As seen, the increment of the test accuracy from the validation dataset is insignificant during the training process when using the original weights inheritance method in [10]. The weights of the model can inherit very limited information of the last generation. The reason behind is the structural difference of the models, where the original weight inheritance method is designed for the residual structure while our model is for the densely connected structures. In addition, our pretrained weight inheritance method is more efficient. The weights can therefore inherit more information from the last generation using our method, and the model can even reach the same test accuracy as the baseline. The experiments in Section 4.4 indicate that if the model is evolved by more generations, the self-structure-generating model can even outperform the baseline but using fewer parameters.

### *4.4. Generating the model architecture on the CIFAR-10 dataset*

In the previous two sections, we discussed the coding methods and the individual weight optimization strategies. In this section we will detail the final training pipeline of the proposed GA based self-generating structure method. Following the work in [10,12,13], we assign the CIFAR-10 dataset as the reference dataset D. For the baseline design, we slightly upscale the number of layers of the model for transfer learning on different scales of the datasets while downscaling the number of the growth rate to improve the training speed. As a result, we assign 10 convolutional layers in each block without the bottleneck structure and set the growth rate to $G = 4$. Thus, the total length of the binary string is 162. The depth

**Table 2**
Architecture of the Model.

| Block | Layer ID | Output points to |
|---|---|---|
| 1 | 0 | 1, 2, 3, 6, 8, 9 |
| | 1 | 2, 3, 4, 5, 7, 9, 10 |
| | 2 | 6, 8, 9 |
| | 3 | 5, 6, 8, 9 |
| | 4 | 6, 8, 10 |
| | 5 | 8, 9, 10 |
| | 6 | 10 |
| | 7 | 10 |
| | 8 | – |
| | 9 | – |
| | **Transition 1** | |
| 2 | 0 | 1, 2, 6, 7, 8, 10 |
| | 1 | 2, 5, 6, 9 |
| | 2 | – |
| | 3 | 7 |
| | 4 | 7, 10 |
| | 5 | 6, 9, 10 |
| | 6 | 7, 9 |
| | 7 | – |
| | 8 | – |
| | 9 | – |
| | **Transition 2** | |
| 3 | 0 | 1, 3, 4, 5, 6, 7, 8, 9, 10 |
| | 1 | 4, 7 |
| | 2 | 3, 4, 5, 7, 8, 9, 10 |
| | 3 | 5, 7, 9, 10 |
| | 4 | 5, 6, 7 |
| | 5 | 6, 10 |
| | 6 | 7 |
| | 7 | – |
| | 8 | 9 |
| | 9 | 10 |
| Global Average Pooling | | |
| FC-Softmax | | |

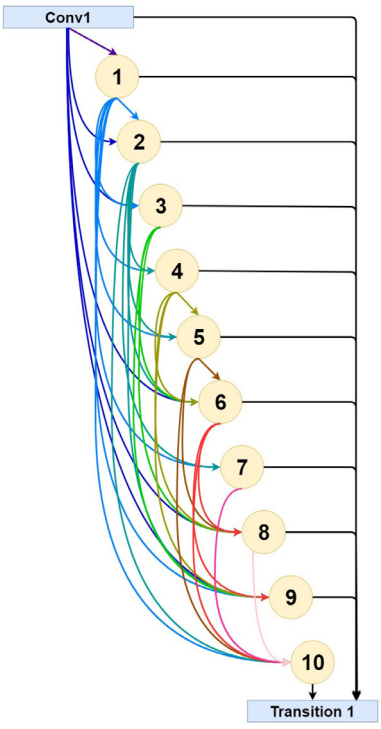



**Fig. 7.** Visualization of the first block. Each numbered circle denotes a convolutional layer. Connections between layers are expressed by color arrows. "Conv1" denotes the first convolutional layer before the first block. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

of the initial feature map is 32. Other settings are the same as that in the per-training weight inheritance strategy as discussed in Section 4.3.

Evolutionary results of each generation are presented in Fig. 5, where the test accuracy of the baseline is 90.23%. After 10 generations, our derived model reaches a similar accuracy as the baseline. After 20 generations, our model outperforms the baseline, achieving an accuracy of 90.8% on the validation dataset at the 24th generation. Individuals in the later generations perform worse than the former generations due to the high individual mutation rate $P_M$. A high mutation rate brings benefits of generating a high-performance model but fails to guarantee the mutation variance. We display the details of the best individual in Table 2, which summarizes the structure of the model and layer connections in each block. The model generated using the GA consists of three blocks with 10 convolutional layers in each block. At the end of the first two blocks, a transition layer is connected to fuse the feature map. The third block consists of a global average pooling layer and a fully connected layer to make the final prediction. Layer id "0" indicates the input of the block and the "Output points to" column denotes the layers to which the output of the current layer will be fed. "-" means that the output of the current layer will only be fed to the transition layer. As the output of the last convolutional layer can only be fed to the transition layer, we omit the last layer in the table.

Due to limited page space, we only present the first block, i.e. the most densely connected one in Fig. 7. As seen, the connection path of the model is sparser than the baseline (DenseNet). The feature maps from the first three layers are frequently reused in each block, and the reuse frequency of each layer's output is reduced with the increasing layer id. This validates the importance of the shallow feature map in the final classification. This phenomenon is also found in many manually designed CNN models [7,8].

### 4.5. Effect of GA

In this subsection, an ablation study is conducted to verify the effect of the evolutionary operators, i.e. tournament selection, crossover and mutation by respectively disabling these three operators individually. When the tournament selection is disabled, the mechanism will randomly select samples from the parent generation as suggested in large-scale evolution [10] and the experimental results are reported in Fig. 8 (a). As seen, with the disabled tournament selection, the best mean accuracy of generations is decreased by 0.65%, because the sub-optimal models are not excluded at an early stage. The similar finding is also reported in [10,38], where the overall performance degrades when the tournament selection is skipped. When the crossover is disabled, the proposed model is still able to achieve a comparable accuracy, however, with more generations being searched. This indicates that the crossover brings benefits of reducing the searching cost for efficiency. Without the mutation method, the classification accuracy drops by 0.22%, due to limited exploration of the architecture, i.e. variety of the generated model space. Moreover, we notice that the mean accuracy of the generation is found decreasing, when all three operators are disabled. We deduce that this is caused by the reduction of generation variety, because the well-performed individuals may be randomly discarded during random selection. This further validates the effectiveness of mutation.

As shown in some previous works [10,13,38], the performance of these searching methods is insensitive to the population size, the crossover rate and the probability of crossover in terms of each bit ($Pb_C$). As a different mutation operation is applied in this paper, we specifically investigate the impact of the mutation rate ($Pb_M$)

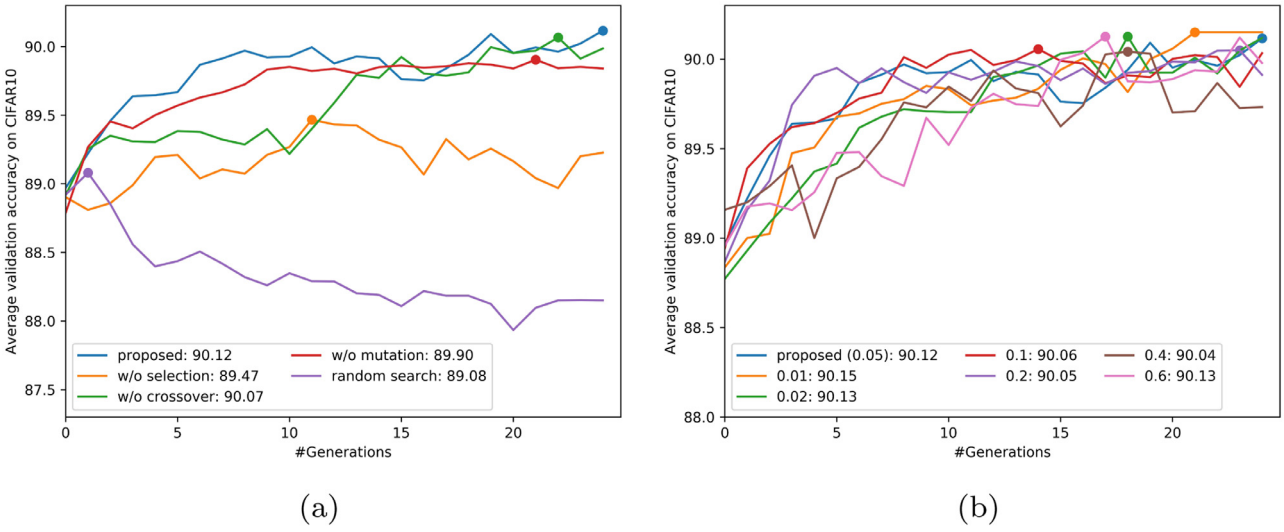



**Fig. 8.** Ablation studies of evolutionary operators (a) and the probability of mutation in terms of each bit ($Pb_M$). The best performance with respect to each test is highlighted by dot, with the corresponding value shown in the legend.

to our proposed method below. We vary the $Pb_M$ and keep other evolutionary hyperparameters the same, and the results are shown in Fig. 8 (b). At first, the mean accuracy drops as the mutation rate increases. When the mutation rate exceeds 0.5, however, the mean accuracy starts to increase. Specifically, when $Pb_M < 0.1$, the maximum decrease is only 0.03, which verifies that the proposed method is insensitive to the mutation rate, especially when it is below 0.1. Across this ablation study, it takes more than 20 GPU days to search on a single evolutionary hyperparameter, which is not computational efficiency. One future work is to further improve the efficiency of the evolutionary hyperparameter searching.

### 4.6. Classification test on multiple datasets

With the selected best structure, we also test it on multiple benchmark datasets. Apart from structure with a the single convolutional layer, we also consider the "bottleneck" structure in our model, where the number of layers is either 34 (without bottleneck) or 64 (with bottleneck). We denote our genetically generated densely connected model as "GADNet" for the non-bottleneck structure and "GADNet-BC" for the bottleneck structure. We will compare our results with other manually designed models, especially the variance of the ResNet and the DenseNet, as well as some self-structure-generating approaches.

#### 4.6.1. Test on the CIFAR-10, CIFAR-100 and SVHN datasets

Experimental results, shown in Table 3, indicate that our model has advantages over manually designed and other self-structure-generating models in terms of the accuracy, transfer capability, parameter saving and efficiency of feature reuse as explained below.

**Accuracy**. Our best model with 64 layers only lags the state-of-the-art method (DenseNet, L = 169, G = 40) by no more than 0.6% on the CIFAR-10 and SVHN datasets and no more than 2% on the CIFAR-100 dataset. This is due to the large margin of the length between models (169 layers for the best DenseNet, which is 2.6 times longer than our model), as our model surpasses the baseline by a slight margin. It is conceivable that our model can reach a more comparable result by extending the length of the model. Apart from the DenseNet, our results on the CIFAR-10 dataset surpass the FractalNet with drop-path regularization [32] and wide ResNet by 15% and 5% lower of accuracy, respectively. On the CIFAR-100 and SVHN datasets, our model produces similar results to the wide ResNet.

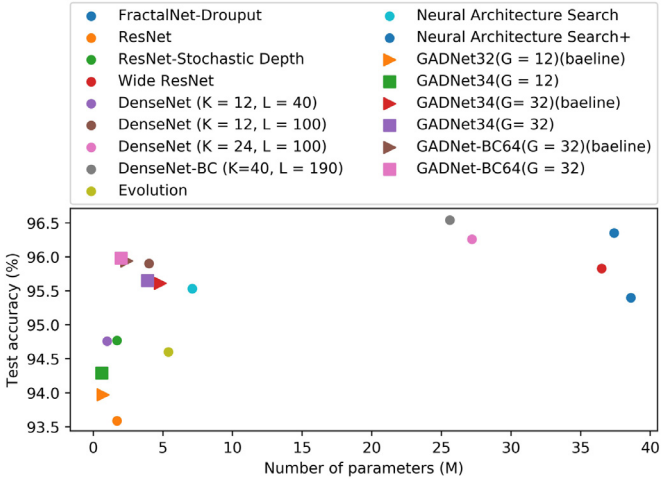



**Fig. 9.** The number of parameters (M) and the corresponding test accuracy (%) of each model on the CIFAR-10 dataset. Our models are labelled using large squares, while the baseline models are marked as large triangular. Other models are denoted by dots.

**Transfer capability**. Most of the self-structure-generating methods cannot generate robust models that can yield comparable results on different recognition tasks or different scales of tasks, i.e. poor transfer capability. Large scale evolution methods [10] and GeNet [13] can generate comparable results in multiple datasets but still lag the manually designed methods by a small margin. However, our proposed method has outperformed most of the manually designed models and can be easily implemented on different tasks. The experiments described in Section 4.6.2 have also validated that our model has a high flexibility and can cope with a large-scale dataset such as the ImageNet.

**Saving of parameters and computational cost**. We plot the relationship between the number of the parameters and the test accuracy on the CIFAR-10 dataset in Fig. 9. As seen, our method outperforms state-of-the-art manually designed models, while using significantly fewer parameters and less computation time to achieve the comparable results. When comparing the parameter requirement with the densely connection and the genetic connection, they show a similar trend. For instance, our best model lags

**Table 3**
Evaluation Results on Multiple Datasets.

| Model | #Params (M) | #Layers | C10 | C100 | SVHN |
|---|---|---|---|---|---|
| **Manually designed methods** | | | | | |
| FractalNet [32] | 38.6 | 21 | 4.60 | 23.73 | 1.87 |
| ResNet by [25] | 1.7 | 110 | 6.41 | 27.22 | 2.01 |
| with Stochastic Depth | 1.7 | 110 | 5.23 | 24.58 | 1.75 |
| Wide ResNet [34] | 36.5 | 28 | 4.17 | 20.50 | – |
| With Drouput | 2.7 | 16 | – | – | 1.64 |
| DenseNet [8] | 1.0 | 40 | 5.24 | 24.42 | 1.79 |
| DenseNet (K=12) | 4.0 | 100 | 4.10 | 20.20 | 1.67 |
| DenseNet (K=24) | 27.2 | 100 | 3.74 | 19.25 | **1.59** |
| DenseNet-BC (K=40) | 25.6 | 190 | **3.46** | **17.18** | – |
| **Evolutionary algorithm methods** | | | | | |
| Large-scale Evolution-C10 [10] | 5.4 | – | 5.40 | – | – |
| Large-scale Evolution-C100 | 40.4 | – | – | 23.00 | – |
| CGP-CNN [27] | 3.9 | – | 23.48 | – | – |
| CGP-CNN (ResSet) | 0.8 | – | 23.47 | – | – |
| GeNet#1 [13] | – | 12 | 7.19 | 29.03 | 1.99 |
| GeNet#2 | – | 12 | 7.10 | 29.05 | 1.97 |
| **Our methods** | | | | | |
| GADNet (G=12)* | 0.7 | 34 | 6.03 | 26.00 | 1.81 |
| GADNet (G=12) | **0.6** | 34 | 5.71 | 25.50 | 1.74 |
| GADNet (G=32)* | 4.8 | 34 | 4.39 | 21.83 | 1.65 |
| GADNet (G=32) | **3.9** | 34 | 4.35 | 21.10 | **1.61** |
| GADNet-BC (G=32)* | 2.4 | 64 | 4.06 | 21.00 | 1.71 |
| GADNet-BC (G=32) | **2.0** | 64 | **4.02** | **20.50** | 1.70 |

("*" indicates the baseline of the model).

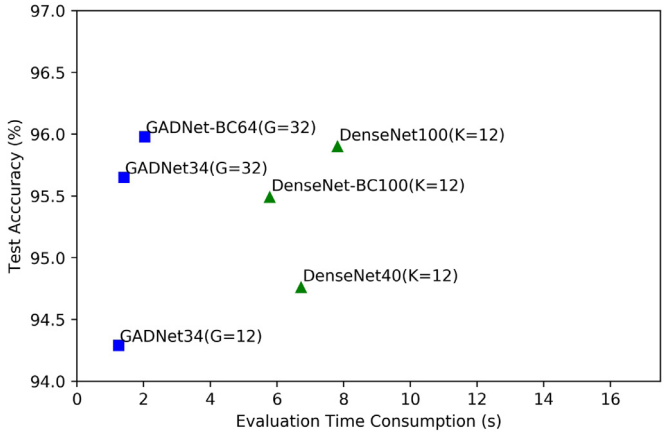



**Fig. 10.** Comparison of the evaluation times (S) of our method (marked by squares) and the DenseNet (labelled by triangles). Evaluation times are measured on the CIFAR-10 test dataset, where only forward propagation is conducted. We run all models on a single GTX1080ti during the time measurement.

the best DenseNet by 0.5% but with 92.2% fewer parameters. As the number of FLOPs (floating point operations) required is in proportion to the number of parameters, models generated using the GAs can, as a result, significantly reduce the computational cost, while achieving comparable classification accuracy. We also measure the evaluation times (forward propagation only) on the CIFAR-10 dataset for both the DenseNet and our method, as shown in Fig. 10. As seen, under a similar test accuracy, our method only requires 1/3 of the computational time than the DenseNet. Compared with the best model of DenseNet, i.e. DenseNet-BC190 (K=40), our model GADNet-BC64(G=32) lags 0.5% on accuracy but improves the evaluation speed by about 6 times, directly benefitting from the fewer parameters of the model.

**High efficiency of reused features**. Following the measurement method in [8], we measure the efficiency of reused features in each convolutional layer using the absolute average weights of the input channels. A higher value in the layer $l$ whose input is from the layer $s$ denotes that the feature map from the layer $s$ is strongly used in layer $l$. In Fig. 11, we plot the heat maps of a 34-layer model using both densely connected (baseline, shown in (11 a)) and the genetic connected (our method, shown in (11 b)) methods. Compared with the densely connected method, our method maintains almost all the strong features whilst discarding the weakly used features from the input of each convolutional layer. This has indicated that the GA improves the efficiency of feature reuse significantly. There are also few "cold zones" in the heat map of our method, possibly due to the hyperparameter settings of the GA, which is left for future investigation.

*4.6.2. Test on large-scale dataset of the imagenet*

We also evaluate our model on the ImageNet by upscaling its depth and width. As the size of the images in ImageNet is far larger than the CIFAR and SVHN datasets, we down sample the feature maps of the model as in [4,6–8,13] by connecting a densely connected block with 6 bottleneck layers [8] at the input end of our model. As a result, the first block is densely connected and the following three are genetically connected. We assign different growth rates in each block, which are 32, 32, 64 and 128 for each of the four blocks, respectively. Experimental results are given in Table 4. As seen, our genetic connection model, denoted as "GADNet-expand", can yield comparable results to manually designed methods as well as other self-structure-generating approaches. This has validated that our model is robust even with the scale variance, while most of the self-generating structure methods may fail on large scale classification tasks such as ImageNet.

As shown in Table 4, the proposed method has remarkably reduced the floating-point operations per second (FLOPs), when compared with state-of-the-art methods. However, as presented in [20], the computational cost from the ordinary convolution layer is still too large for mobile and embedded vision applications. To validate the performance for those resource-constrained environments, following the works in [15,20,36,39], we replace the convolution layers in the proposed GADNet by separable convolution layers (with a bottleneck structure), which is verified as a low-cost operation [20]. Meanwhile, we discard the last 5 layers for each GA-searched block to further reduce the computational cost. We denote the low-cost implementation of the proposed method as "GADNet-mobile". As a result, the number of parameters is further

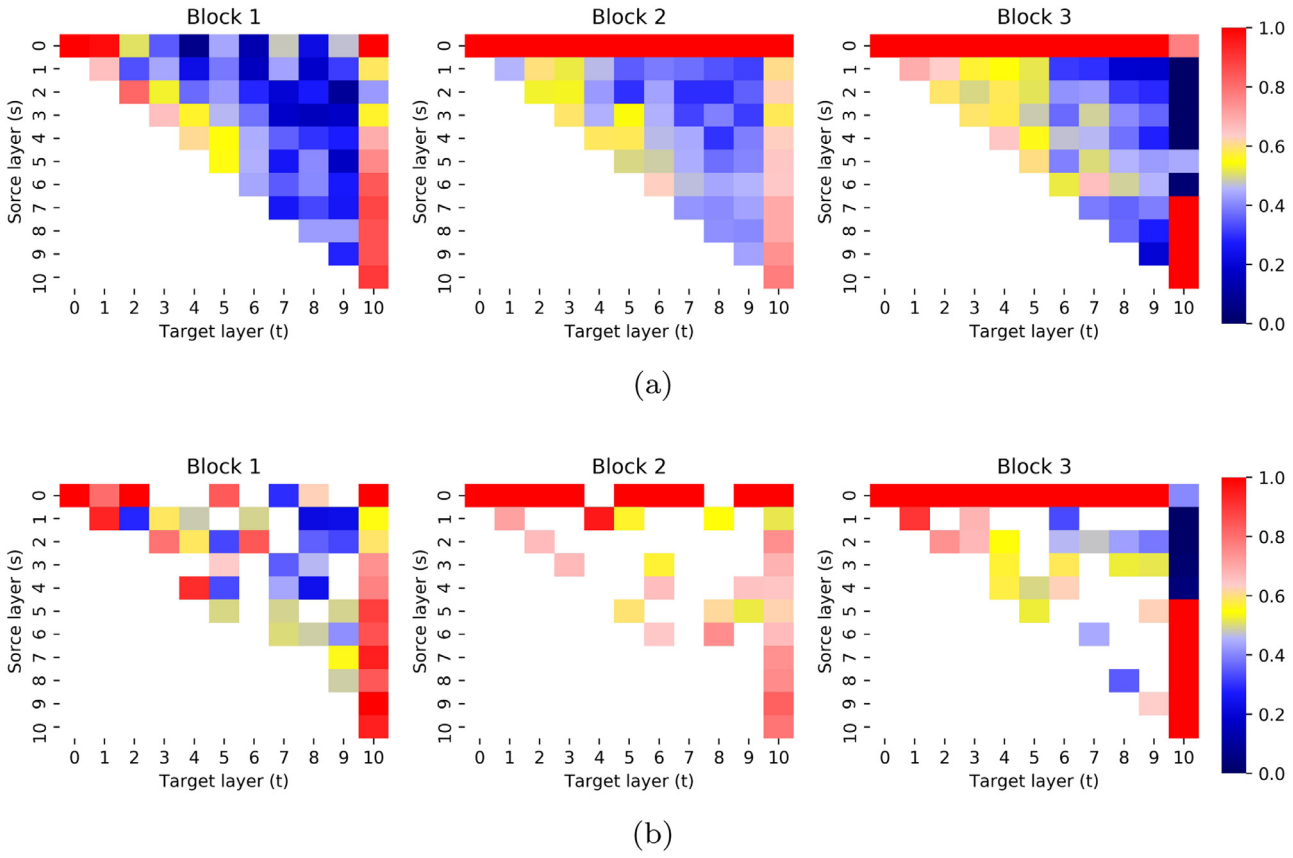



**Fig. 11.** The average absolute filter weights of convolutional layers in (a) DenseNet34 (G = 12) and (b) GADNet34 (G = 12), trained on the CIFAR-10 dataset. The color patch (*s*, *t*) denotes the average L1 normalized (by number of input depth) weights in the layer "t" which takes input from the layer "s" in the same block. The last column of each block denotes the transition layer for the first two blocks and the fully connected layer for the last block respectively. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

**Table 4**
Validation Error Rate on ImageNet.

| Model | #Params (M) | FLOPs (G) | Top-1 Err. | Top-5 Err. |
|---|---|---|---|---|
| **State-of-the-art methods** | | | | |
| AlexNet [4] | 60 | 0.77 | 43.45 | 20.91 |
| VGG-19 [6] | 144 | 19.77 | 27.62 | 9.12 |
| ResNet-18 [7] | 11.7 | 1.82 | 30.24 | 10.92 |
| ResNet-34 [7] | 21.8 | 3.68 | 26.70 | 8.58 |
| ResNet-152 [7] | 60.2 | 11.61 | 21.69 | 5.94 |
| DenseNet-121 [8] | 7.2 | 2.90 | 25.35 | 7.83 |
| DenseNet-169 [8] | 13.0 | 3.44 | 24.00 | 7.00 |
| **Manually designed light-weight methods** | | | | |
| SqueezeNet [19] | 1.2 | 0.35 | 41.80 | 19.30 |
| MobileNet v2 [20] | 3.4 | 0.30 | 28.12 | 9.71 |
| ShuffleNet v2 [21] | 2.3 | 0.15 | 30.64 | 11.68 |
| **Self-structure-generating methods** | | | | |
| NASNet-A (4@1056) [39] | 5.3 | 0.56 | 26.0 | 8.4 |
| GeNet [13] | 30.6 | – | 27.87 | 9.74 |
| LEMONADE [36] | 6.0 | 0.60 | 28.3 | 9.6 |
| SNAS [29] | 4.3 | 0.52 | 27.3 | 9.2 |
| DARTS [15] | 4.7 | 0.57 | 26.7 | 8.7 |
| FBNet-C [28] | 5.5 | 0.37 | 25.1 | – |
| MnasNet-A3 [14] | 5.2 | 0.43 | 24.3 | 6.7 |
| AmoebaNet-A3 [35] | 469 | 104.00 | **17.1** | **3.4** |
| **Proposed methods** | | | | |
| GADNet-expand | 5.0 | 2.70 | 27.35 | 8.91 |
| GADNet-mobile | 4.3 | 0.42 | 28.03 | 9.51 |

**Table 5**
Searching cost of NAS and the proposed method.

| Model | #Params (M) | Searching Cost (GPU days) | C10 Err. (%) |
|---|---|---|---|
| **NAS methods** | | | |
| MetaQNN [11] | – | 80-100 | 7.02 |
| NAS v3 [12] | 7.1 | – | 4.47 |
| Progressive NAS [40] | 3.2 | 2000 | 3.63 |
| NASNet-A [39] | 3.3 | 2000 | 3.41 |
| AmoebaNet [35] | 3.2 | 3150 | 3.34 |
| SNAS * [29] | 2.9 | 2.7 | 3.70 |
| DARTS ** [15] | 3.9 | 5 | 2.95 |
| **The Proposed Method** | | | |
| GADNet-EXT | 2.0 | 3.2 | 3.34 |
| GADNet-EXT+ | 5.3 | | 2.91 |

* Result is generated using: "https://github.com/xdhhh/Stochastic_neural_architecture_search ".
**Result is generated using the official code: "https://github.com/quark0/darts".

reduced by 14%, whilst the FLOPs have been significantly reduced by 84%. Meanwhile, the top-1 error is only slightly degraded from 27.35% to 28.03%.

We also compared our proposed approach with other manually designed light-weight models, such as SqueezeNet [19], MobileNet v2 [20] and ShuffleNet [21]. With additional 0.07-0.27 million FLOPs, the proposed GADNet-mobile leads by 4.37% on average. The existing manually designed methods optimize the deep learning structures by either optimizing the convolutional operation (MobileNet, ShuffleNet) or connecting additional short-cut (SqueezeNet), which are all local optimization methods. On the contrary, our method is a form of global optimization as it optimizes the connection routine across the whole network. The results in Table 4 have clearly validated that the global optimization is more efficient than the local optimization. Nevertheless, our method did not optimize the convolutional structure. We deduce that a combination of both global and local optimization methods can further improve the optimization performance, and this will also be part of our future work for investigation.

*4.6.3. NAS vs. GA*

Specifically, we compare our GA based optimization method with Neural Architecture Search (NAS). Most existing NAS methods [15,29,35,39] report their results based on additional enhancements such as cutout [42], path dropout [39] and auxiliary towers [15]. For a fair comparison, we also adopt these enhancements on the "GADNet-BC (G=32)"and the enhanced model is denoted as "GADNet-EXT" in Table 5. Results in Table 5 indicate that the model derived by the proposed method reaches comparable results as those from NAS, but with fewer parameters. For example, the proposed method achieves a comparable error rate as the AmoebaNet, but with a reduction of 37.5% on parameters. A similar conclusion can be seen when compared on the ImageNet, as shown in Table 4. On the other hand, the proposed method aims to improve the computational efficiency of the DenseNet, in which NAS is designed for improving the accuracy. We have also compared, as shown in Table 5, the searching cost required by NAS and the proposed GA method. As can be seen, some NAS methods need an RNN to optimize the structure of CNN, which naturally requires more GPUs than GA. Some of the NAS methods even require more than 1000 GPU days [39,40], raising the impractical difficulty of implementation, particularly in resource-constrained applications and research labs. Although DARTS outperforms the proposed method by 0.39%, we have significantly reduced the searching cost by at least 36%. As the performance of deep learning models is closely related to its scale [6–8], to validate the robustness under a larger scale, we upscale the proposed model by adding 10 densely connected layers at the end of the third block. At the same time, we set the growth rate to 40, which is denoted as "GADNet-EXT+" in Table 5. The results indicate that by using more layers, the proposed method can achieve a similar performance as DARTS, without increasing the searching cost. This again has validated the superior performance of the proposed method in optimizing the structure of the DenseNet.

On the other hand, NAS based approaches need adaptation and additional training for testing on a large dataset such as the ImageNet. In contrast, our proposed method does not need such adaptation and additional training when migrating from a small dataset to larger ones, where the scalability can help to significantly save the computational cost.

*4.6.4. Comparison with other neuroevolution-based methods*

Neuroevolution algorithms have been widely applied in a variety of fields, such as accelerating neural evolution,[1] architecture design,[2] improved training,[3] parameter optimization,[4] and hyperparameter selection.[5]vvvvv In Table 6, some typical neuroevolution methods for image classification are compared. As can be seen, the proposed method outperforms most of existing neuroevolution methods on the CIFAR-10, except for LEMONADE. The proposed GADNet lags LEMONADE by 0.33% on the error rate. However, our method only consumes 4% of the searching cost as LEMONADE does. This has verified that the proposed method can achieve a promising result with a high searching efficiency. We deduce this is due to the application of the bit crossover, as well as the well-designed search space. As presented in Section 4.5, the bit crossover can help to reduce the number of generations, hence reducing the searching cost. Meanwhile, the proposed method searches the network architectures based on the well performing manually designed model, which has further reduced the searching cost.

However, our method still has some limitations. First, it evolves the structure of the model with a fixed length. Although the computational cost is significantly reduced compared with those without predefined lengths, the fixed length also constrains the flexibility of the evolution, e.g. when being evaluated under resource-limited condition, we have to manually assign the number of layers on each searched block.

*4.6.5. Further implementation details*

We set the batch size to 64 for all the datasets (CIFAR, SVHN and ImageNet) and train each model using the stochastic gradient descent (SGD) optimizer with a Nesterov momentum [43] of 0.9 without dampening. We fixed the weight decay to 0.0001 on each

[1] http://www.jmlr.org/papers/v9/gomez08a.html.
[2] https://www.nature.com/articles/s42256-018-0006-z.
[3] https://dl.acm.org/doi/abs/10.1145/3205651.3208763.
[4] https://dl.acm.org/doi/abs/10.1145/2834892.2834896.
[5] https://dl.acm.org/doi/abs/10.1145/3071178.3071208.

**Table 6**
Comparison with other neuroevolution methods.

| Methods | Searching Spaces | Evolutionary Operators | Searching Cost (GPU days) | C10 Err. | Pros/Cons | |
|---|---|---|---|---|---|---|
| Genetic CNN [13] | Layer connections (add.) | 1. Fitness proportionate selection 2. Bit crossover 3. Bit mutation | 17 | 7.10 | Pros | 1. One search for multiple datasets 2. Easy to implement |
| | | | | | Cons | Constrained performance as the network connection only relies on addition |
| Large-scale evolution [10] | | 1. Tournament selection 2. Mutation from predefined set | 250 | 5.40 | Pros | Both network architecture and the cell architectures are optimized |
| | | | | | Cons | The large searching space increases the searching cost |
| CoDeepNEAT [17] | 1. Layer connection(add. & concat.) 2. Cell Structure * (layer type, kernel parameters) 3. Hyperparameters | 1. Crossover 2. Mutation | - | 7.30 | Pros | Reaching competitive error rate with existing ones, while using low searching cost (as claimed) |
| | | | | | Cons | Fixed network architecture and the cell structure at the same time cause difficulty of the application on large-scale datasets |
| Hierarchical architecture search [38] | 1. Layer connection (add. & concat.) 2. Cell Structure (layer type) | 1. Tournament selection 2. Mutation from predefined set | 300 | 3.75 | Pros | Promising results |
| | | | | | Cons | The cell searching with 6 different types of layers remarkably increases the cost |
| Memetic Evolution [41] | 1. Layer connections (add.) | 1. Crossover 2. Gaussian mutation | 0.08 | 27.3 | Pros | Extremely low searching cost |
| | | | | | Cons | 1. Low performance of searched model 2. Weights inherited from the last generation may be not the best architecture but the best-trained model. |
| LEMONADE [36] | 1. Layer connection (add. & concat.) 2. Cell Structure | 1. Tournament selection 2. Network morphism | 80 | 2.58 | Pros | Promising performance on both speed and accuracy |
| | | | | | Cons | The layer connections searching is asynchronous with cell searching, causing additional searching cost. |
| GADNet (proposed) | Layer connection (concat.) | 1. Tournament selection 2. Bit crossover 3. Bit mutation | **3.2** | **2.91** | Pros | 1. Only one search for multiple datasets 2. Low searching costend |
| | | | | | Cons | 1. Only the best model is selected. 2. The length of the architecture requires manual input |

* "Cell" refers to the structure of subnet, which may be used by multiples to build the network, e.g. bottleneck is a cell for ResNet.

dataset. The learning rate is initialized as 0.1. When training on the CIFAR and SVHN datasets, the learning rate is divided by 10 on 150 and 225 epochs and the training terminates at 300 epochs. When training on the ImageNet, the learning rate is divided by 10 on 30 and 60 epochs and the training stops at 90 epochs. When training on the SVHN dataset, we add a dropout layer as in [8,32,34] and set the dropout rate to 0.2. All experimental tests on the evaluation section are conducted using Pytorch [44].

## 5. Conclusion

We have applied the GA to optimize the structure of the DenseNet. The GA assigns a binary bit to each channel of the input to remove the redundant features from the feature map. We first propose two encoding methods which enable the GA to evolve the structure of the model. Then we apply a simple but effective genetic process containing selection, crossover and mutation. Conventional evaluation methods, in which each "individual" is fully trained, consumes extremely high computational resources and takes a very long time for convergence. Previous work that optimizes the individual training by using weight inheritance between generations, is restricted by the residual structure. Thus, in our proposed approach, a pretrained weight inheritance method can not only reduce the individual training time, but also release the constraints of the residual structure. Experimental results indicate that our model can reach competitive classification accuracy and requires significantly fewer parameters.

Rather than generating structures from scratch, our method is built based on predefined constraints, which has significantly reduced the computational cost and is capable of generating a well performing structure under limited training resources. When tested on the CIFAR-10 dataset, our model saves more than 90%

of the parameters in comparison to the state-of-the-art models. The comparable results from the ImageNet has validated that the proposed model is robust in both the variance and the scale of the classification tasks. At the current stage, we only validate the proposed approach on CNN. In the future, we will evaluate our model using other popular deep learning networks, such as RNN and LSTM.

However, our method still has a few limitations. First, our method evolves the structure of the model with a fixed length, of which the computational cost is significantly reduced compared with those without predefined lengths. However, the fixed length also constrains the flexibility of the evolution. Another drawback is that the searching time on the evolutionary hyperparameters is still too long, although the searching method is not sensitive to most of them, e.g. population size and crossover rate. In the future works, we will further optimize the searching pipeline in two aspects, i) Reducing the total searching time by fusing the evolutionary hyperparameters into the searching procedure and ii) Generating the structure of the model without the predefined length while preserving the robustness of the model.

## Declaration of Competing Interest

Regarding our paper "Structral Optimizing the DenseNet with Pre-Trained Weight Inheritance and Genetic Selection of Input Channels" authored by Zhenyu Fang, Jinchang Ren, Stephen Marshall, et al., on behalf of all co-authors, herein we confirm there is no conflict of interest in submitting the paper to Pattern Recognition for considering of publication.

## Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.patcog.2020.107608.

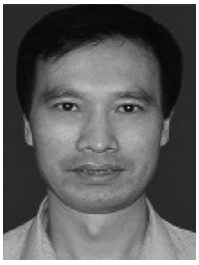
## References


[1] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai, et al., Recent advances in convolutional neural networks, Pattern Recognit. 77 (2018) 354–377.
[2] M.A. Hearst, S.T. Dumais, E. Osuna, J. Platt, B. Scholkopf, Support vector machines, IEEE Intell. Syst. Appl. 13 (4) (1998) 18–28.
[3] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, et al., Gradient-based learning applied to document recognition, Proc. IEEE 86 (11) (1998) 2278–2324.
[4] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, in: Advances in Neural Information Processing Systems (ANIPS), 2012, pp. 1097–1105.
[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, L. Fei-Fei, Imagenet: a large-scale hierarchical image database, in: Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), 2009, pp. 248–255.
[6] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, in: Int. Conf. on Learning Representations (ICLR), 2015.
[7] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proc. the IEEE Conf. CVPR, 2016, pp. 770–778.
[8] G. Huang, Z. Liu, L. Van Der Maaten, K.Q. Weinberger, Densely connected convolutional networks, in: Proc. the IEEE Conf. CVPR, 2017, pp. 4700–4708.
[9] A. Veit, M.J. Wilber, S. Belongie, Residual networks behave like ensembles of relatively shallow networks, in: ANIPS, 2016, pp. 550–558.
[10] E. Real, S. Moore, A. Selle, S. Saxena, Y.L. Suematsu, J. Tan, Q.V. Le, A. Kurakin, Large-scale evolution of image classifiers, in: Proc. the 34th Int. Conf. on Machine Learning (ICML), 70, 2017, pp. 2902–2911.
[11] B. Baker, O. Gupta, N. Naik, R. Raskar, Designing neural network architectures using reinforcement learning, in: 5th Int. Conf. ICLR, Toulon, France, April 24–26, 2017.
[12] B. Zoph, Q.V. Le, Neural architecture search with reinforcement learning, in: 5th Int. Conf. ICLR, Toulon, France, April 24–26, 2017.
[13] L. Xie, A. Yuille, Genetic cnn, in: Proc. the IEEE Int. Conf. on Computer Vision (ICCV), 2017, pp. 1379–1388.
[14] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, Q.V. Le, Mnasnet: Platform-aware neural architecture search for mobile, in: Proc. the IEEE Conf. CVPR, 2019, pp. 2820–2828.
[15] H. Liu, K. Simonyan, Y. Yang, DARTS: differentiable architecture search, in: 7th Int. Conf. ICLR, New Orleans, LA, USA, May 6–9, 2019.
[16] T. Elsken, J.H. Metzen, F. Hutter, Neural architecture search: a survey, J. Mach. Learn. Res. 20 (55) (2019) 1–21.
[17] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy, et al., Evolving deep neural networks, in: Artificial Intelligence in the Age of Neural Networks and Brain Computing, Elsevier, 2019, pp. 293–312.
[18] A. Krizhevsky, G. Hinton, Learning Multiple Layers of Features from Tiny Images, Technical Report, Citeseer, 2009.
[19] F.N. Iandola, S. Han, M.W. Moskewicz, K. Ashraf, W.J. Dally, K. Keutzer, Squeezenet: alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size, arXiv:1602.07360(2016).
[20] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, L.-C. Chen, Mobilenetv2: Inverted residuals and linear bottlenecks, in: Proc. the IEEE Conf. CVPR, 2018, pp. 4510–4520.
[21] N. Ma, X. Zhang, H.-T. Zheng, J. Sun, Shufflenet v2: practical guidelines for efficient CNN architecture design, in: Proc. the European Conference on Computer Vision (ECCV), 2018, pp. 116–131.
[22] S. Ioffe, C. Szegedy, Batch normalization: accelerating deep network training by reducing internal covariate shift, in: Int. Conf. on ICML, 2015, pp. 448–456.
[23] H. Qin, R. Gong, X. Liu, X. Bai, J. Song, N. Sebe, Binary neural networks: a survey, Pattern Recognit. (2020) 107281.
[24] M. Tzelepi, A. Tefas, Improving the performance of lightweight CNNs for binary classification using quadratic mutual information regularization, Pattern Recognit. (2020) 107407.
[25] G. Huang, Y. Sun, Z. Liu, D. Sedra, K.Q. Weinberger, Deep networks with stochastic depth, in: Proc. ECCV, Springer, 2016, pp. 646–661.
[26] K.O. Stanley, R. Miikkulainen, Evolving neural networks through augmenting topologies, Evol. Comput. 10 (2) (2002) 99–127.
[27] C. Fernando, D. Banarse, M. Reynolds, F. Besse, D. Pfau, M. Jaderberg, M. Lanctot, D. Wierstra, Convolution by evolution: differentiable pattern producing networks, in: Proc. the Genetic and Evolutionary Computation Conf. 2016, ACM, 2016, pp. 109–116.
[28] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, K. Keutzer, Fbnet: hardware-aware efficient convnet design via differentiable neural architecture search, in: Proc. CVPR, 2019, pp. 10734–10742.
[29] S. Xie, H. Zheng, C. Liu, L. Lin, SNAS: stochastic neural architecture search, in: Proc. 7th Int. Conf. ICLR, New Orleans, LA, USA, May 6–9, 2019.
[30] Y. Sun, B. Xue, M. Zhang, Evolving deep convolutional neural networks for image classification, arXiv:1710.10741(2017).
[31] D.E. Goldberg, K. Deb, A comparative analysis of selection schemes used in genetic algorithms, in: Foundations of Genetic Algorithms, 1, Elsevier, 1991, pp. 69–93.
[32] G. Larsson, M. Maire, G. Shakhnarovich, Fractalnet: Ultra-deep neural networks without residuals, in: Proc. 5th Int. Conf. ICLR, Toulon, France, April 24–26, 2017, 2017.
[33] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, A.Y. Ng, Reading digits in natural images with unsupervised feature learning (2011).
[34] S. Zagoruyko, N. Komodakis, Wide residual networks, in: Proc. the British Machine Vision Conf., 2016, pp. 87.1–87.12.
[35] E. Real, A. Aggarwal, Y. Huang, Q.V. Le, Regularized evolution for image classifier architecture search, in: Proc. the AAAI Conf. on Artificial Intelligence, 33, 2019, pp. 4780–4789.
[36] T. Elsken, J.H. Metzen, F. Hutter, Efficient multi-objective neural architecture search via lamarckian evolution, in: Proc. 7th Int. Conf. ICLR, New Orleans, LA, USA, May 6–9, 2019.
[37] D.P. Kingma, J. Ba, Adam: a method for stochastic optimization, arXiv:1412.6980(2014).
[38] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, K. Kavukcuoglu, Hierarchical representations for efficient architecture search, in: Int. Conf. on Learning Representations, 2018.
[39] B. Zoph, V. Vasudevan, J. Shlens, Q.V. Le, Learning transferable architectures for scalable image recognition, in: Proc. the IEEE conf. on Computer Vision and Pattern Recognition, 2018, pp. 8697–8710.
[40] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, K. Murphy, Progressive neural architecture search, in: Proc. of the Europ. Conf. on Computer Vision, 2018, pp. 19–34.
[41] P.R. Lorenzo, J. Nalepa, Memetic evolution of deep neural networks, in: Proc. of the Genetic and Evolutionary Computation Conf., 2018, pp. 505–512.
[42] T. DeVries, G.W. Taylor, Improved regularization of convolutional neural networks with cutout, arXiv:1708.04552(2017).
[43] I. Sutskever, J. Martens, G. Dahl, G. Hinton, On the importance of initialization and momentum in deep learning, in: Int. Conf. on Machine Learning, 2013, pp. 1139–1147.
[44] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, A. Lerer, Automatic Differentiation in Pytorch, 2017.

**Zhenyu Fang** received the B.Eng. degrees in Electronic and Electrical Engineering in 2016 from the University of Strathclyde Glasgow, Scotland (with first-class honours), and North China Electric Power University, Baoding, China. He received his Ph.D. in Electronic and Electrical Engineering at the University of Strathclyde in July 2020. His main interests are algorithm development for image classification, object detection and face detection.

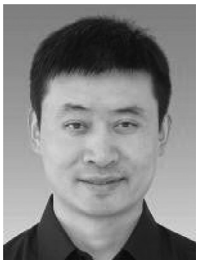
**Jinchang Ren** received the B.E. in computer software, M.Eng. in image processing, and the D.Eng. in computer vision from Northwestern Polytechnical University, Xian, China. He also received a Ph.D. degree in electronic imaging and media communication from the University of Bradford, Bradford, U.K. He has published about 300 articles. His research interests include computer vision and multimedia signal processing, especially on hyperspectral imaging, machine learning and big data analytics. He acts as Associate Editor for five international journals including IEEE Journal of Selected Topics in Applied Earth Observation and Remote Sensing, Journal of the Franklin Institute, IET Image Processing and Big Data Analytics etc.

**Stephen Marshall** received a first class honours degree in Electrical and Electronic Engineering from the University of Nottingham in 1979 and a Ph.D. in Image Processing from the University of Strathclyde in 1989. He is currently a professor in the Department of Electronic and Electrical Engineering, University of Strathclyde. His research activities have been focussed in Nonlinear Image Processing and hyperspectral Imaging. He has published over 200 conference and journal papers. He is a Fellow of the Institution of Engineering and Technology (IET) and a Senior member of the IEEE. Prof. Marshall is also the lead academic for the Vertically Integrated Project Program in Strathclyde.



**Huimin Zhao** reived the B.Sc. and the M.Sc. degrees in signal processing from Northwestern Polytechnical University, Xian, China, in 1992 and 1997, respectively, and the Ph.D. degree in electrical engineering from the Sun Yat-sen University, Guangzhou, China, 2001. He is currently a professor and the Dean with the School of Computer Science, Guangdong Polytechnic Normal University, Guangzhou. His research interests include image, video, and information security technology.

**Song Wang** received the Ph.D. degree in electrical and computer engineering from the University of Illinois at Urbana-Champaign (UIUC), Champaign, IL, USA, in 2002. He was a research assistant with the Image Formation and Processing Group, Beckman Institute, UIUC, from 1998 to 2002. In 2002, he joined the Department of Computer Science and Engineering, University of South Carolina, Columbia, SC, USA, where he is currently a Professor. His current research interests include computer vision, image processing, and machine learning. Prof.Wang is a member of the IEEE Computer Society. He is currently serving as the Publicity/Web Portal Chair for the Technical Committee of Pattern Analysis and Machine Intelligence of the IEEE Computer Society and as an Associate Editor for the IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, Pattern Recognition Letters, and Electronics Letters.



**Xuelong Li** is currently a Full Professor with the Center for Optical Imagery Analysis and Learning (OPTIMAL), School of Computer Science, Northwestern Polytechnical University, Xi'an, China.