Learning Internal Dense But External Sparse Structures of Deep Convolutional Neural Network

Yiqun Duan(✉) and Chen Feng

School of Engineering, The University of British Columbia (Okanagan Campus), Kelowna, BC V1V 1V7, Canada
{yiqun.duan,chen.feng}@ubc.ca

Abstract. Recent years have witnessed two seemingly opposite developments of deep convolutional neural networks (CNNs). On the one hand, increasing the density of CNNs (e.g., by adding cross-layer connections) achieves better performance on basic computer vision tasks. On the other hand, creating sparsity structures (e.g., through pruning methods) achieves a more slim network structure. Inspired by modularity structures in the human brain, we bridge these two trends by proposing a new network structure with internally dense yet externally sparse connections. Experimental results demonstrate that our new structure could obtain competitive performance on benchmark tasks (*CIFAR10, CIFAR100, and ImageNet*) while keeping the network structure slim.

Keywords: Hierarchical CNN · Evolutionary algorithms · Neural network structure

1 Introduction

Deep Convolutional Neural Networks (e.g. [12,18,20,28]) have recently made remarkable success in visual tasks. The very intuitive idea of improving a neural network at the earlier period is to enlarge the scale of the network. However, follow-up papers have shown that when the feedforward network structure has reached a certain depth, neither the best test accuracy nor training accuracy will increase as the network depth increases [15]. An important observation is that by increasing network density and adding long distance connections, the network could be improved such as ResNet [12] and DenseNet [15] have shown. These long-distance connections could create shortcuts between layers in different depth. By propagating loss directly, the network could become deeper and more accurate.

As the network becomes denser, deeper, and more accurate, its scale increases up to millions of parameters. Thus, reducing the network scale and complexity becomes a crucial research task for real-world applications. One promising solution is to increase the network sparsity by pruning redundant connections. For

I. V. Tetko et al. (Eds.): ICANN 2019, LNCS 11728, pp. 247–262, 2019.
https://doi.org/10.1007/978-3-030-30484-3_21

instance, by pruning connections with tiny weights and fine-tuning the pruned network, [11] could reduce the network complexity while only sacrificing about 1% of model accuracy. Similar solutions to create sparsity include channel pruning [14] and structure sparsity [36].

In neuroscience, studies concentrating on the brain structure reveal that neuron connections in human brain perform a locally dense but externally sparse property that the closer two regions are, the denser the connections between them will be [4,9,34]. Also, studies show that while sensory information arrives at the cortex, it is fed up through hierarchical regions from primary area V1 up to higher areas such as V2, V4 and IT [2]. Inside of each cortex layer, tightly packed pyramidal cells consist of basic locally dense structures in the brain.

Based on neuroscientific considerations, the brain does not form connections between every neuron. Instead, the brain forms local modules which consist of internally dense connected neurons inside and forms connections between these dense modules to save space and become more efficient. These internally dense yet externally sparse properties could also be introduced into neural network structures. Thus, we could bridge the trends of being dense and being sparse together in neural network structures. Intuitively, we could acquire good performance while keeping the network structure slim by introducing this brain inspired structure.

We introduce an internally dense yet externally sparse neural network structure by prefixing dense modules and evolving sparse connections between them. The basic building blocks of this network structures are several internally dense modules, where each module consists of several densely connected [15] bottleneck layers to simulate the tightly packed cells. After that, we could form sparse connections between these dense modules for the whole network structure. To give more convincing guidance for forming sparse connections rather than manually design connections based on experience, we design an evolutionary training algorithm (Sect. 3.3) to search optimized connections Moreover, besides merely creating parallel connections between modules, our algorithm could create long-distance connections between the input module and the output module by a transit layer.

The main contribution of this paper is to introduce biologically-inspired internally dense yet externally sparse properties into convolutional neural network by prefixing dense modules and forming sparse connections between dense modules. Instead of empirically constructing module connections, we design an evolutionary training algorithm to search optimized connections. This structure could reach compared slim network structures while keeping competitive model performance on image classification tasks (Exp. Sect. 4.3). We give a detailed analysis of how different sparse connections and different module properties will contribute to the final performance. Moreover, we reveal contribution proportion on the final performance of each connection by several contrast experiments. Thus, we could give intuitive guidance for designing hierarchical network structures.

2 Related Works

Network Architectures Are Becoming Denser

The exploration of network architectures has been an important foundation for all Deep Learning tasks. At the early period of deep learning, increasing the depth of a network might promise a good result as the network structure moved from LeNet to VGG (e.g. [18,20,28]). Since people realize that the increasing depth of the network amplifies problems such as over-fitting and gradient-vanishing [3], parallel structures [32,39] and densely connected layers [15] have been introduced to increase network capacity. In this paper, we refer to the dense block in [15] while constructing internal densely connected modules.

Deep Neural Network Compression

Besides increasing model capacity, deep neural network compression is another active domain concentrating on acquiring slim models by eliminating network redundancy. These methods could be summarized by the three basic aspects: **1. Numerical approximation of kernels**, which includes binarization [7,25], quantization [40], weight sharing or coding method [10] and mainly uses numerical method to approximate kernel with smaller scale; **2. Sparse regularization on kernels**, which mainly prunes connections based on regularization on kernels, such as weights/channel pruning [14,23] and structure sparsity learning [22,36]; **3. Decomposition of kernels**, which mainly uses smaller groups of low-rank kernel instead of a larger whole kernel, such as [6,8,17] and [37]. Instead of pre-training then pruning the whole network structure, we use an evolutionary programming method to determine sparse connections between dense modules, while keeping the modules slim. Competitive performances are obtained.

Neural Network Structure Search

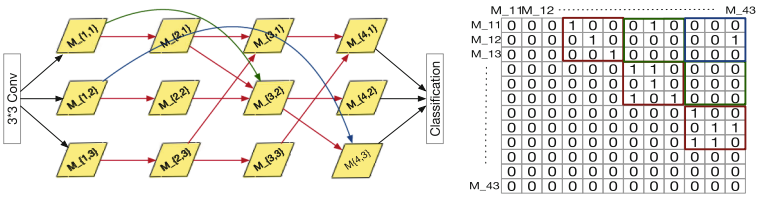
At early period, papers [1,5,24,31] have developed methods that evolve both topologies and weights on simple neural networks. Recently, papers concentrate on evolving structures have risen again along with the rapid development of deep learning. Genetic CNN [38] concentrates on using genetic algorithm to evolve skip-connections on a straight forward convolutional neural network. Then, Google [26] shows great potential about structure search on image classification tasks. Google also proposed a state-of-the-art deep neural network structure NasNet [41] to search both the parameters and the structures. However, the huge scale of these networks still remains a problem. In our paper, the evolving algorithm is a tool to reveal properties of internally dense yet externally sparse structures. Moreover, we analyze how each connections could contribute to the final model performance.

3 Methodology

In order to introduce internal dense yet external sparse properties into deep convolutional neural networks, we proposed a new network structure which prefixes

internal dense modules and evolves sparse connections between dense modules. We define M as the set of dense modules. For a clear identification of these dense modules, we divide these dense modules into D layers, where each layer contains W modules. In particular, we define a dense module in the set M as $M_{d,w}, d \in \{0, 1, ...D\}, w \in \{0, 1, ...W\}$, where index d denotes the depths in layer wise and w denotes the module index among all of the others in the same depth. These modules M are sparsely connected by directed edges/connections. We define P as the adjacency matrix to represent these directed connections between modules. Clearly, the whole neural network structure can be defined as a directed graph $G(M, P)$, where M denotes the set of internal dense modules $M_{d,w}$, P denotes the adjacency matrix, which is used to represent sparse connections between modules.

For example, Fig. 1 shows a set of dense modules in subfigure Fig. 1(a) with depth $D = 4$ and $W = 3$. We use an adjacency matrix P to represent connections between modules as it shows in Fig. 1(b). In this example, firstly a 3*3 convolutional layer processed the input images into some feature maps. Then the feature maps are divided into several groups in channel wise. Each group is sent into a dense module as the input feature maps. After the features flow through sparse dense convolutional neural networks, the outputs are concatenated together for final output layer.



(a) Sparse connected modules (b) Matrix representation

Fig. 1. Example of a network's structure obtained by fixing the dense modules M in advance and using the adjacency matrix P to represent sparse connections between modules. Figure (a) denotes the network structure. Figure (b) denotes the adjacency matrix corresponding to Fig. (a), where red rectangle area denotes connections with distance 1, green rectangle denotes connections with distance 2, blue area denotes connections with distance 3. (Color figure online)

Naturally, three main questions occur: 1. What is the exact structure and definition of a dense module $M_{d,w}$? 2. What does a connection represent in this convolutional neural network? 3. How shall we connect these dense modules and decide the adjacency matrix P?

The exact structure of an internal dense module $M_{d,w}$ is defined in Sect. 3.1. In Sect. 3.2, we answer the second question by specifying a connection in neural networks. In Sect. 3.3, we solve the problem of connecting these modules by evolving sparse connections between them.

3.1 Build Internal Densely Connected Modules

In this section, we will introduce the detailed structure of a dense module $M_{d,w}$. From a high level, a dense module $M_{d,w}$ receives feature maps from other dense modules as its input, then outputs down-scaled feature maps. In other words, we densely connect our *bottleneck layers* and a *transit layer* as the main structure inside a dense module as it is shown in Fig. 2.

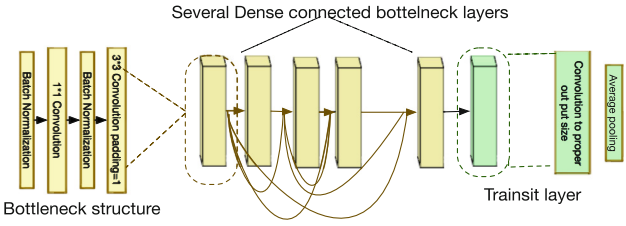


Fig. 2. An example of structure for a prefixed dense module as shown above, where yellow layer represents several densely connect bottleneck layers (it means each intermediate output has a direct connection to the output layer). The detailed structure of a bottleneck layer is shown left. After the final layer, the green layer represents a transition layer to control the feature map size. The depth of dense blocks in our experiment usually varied from 6 to 20. (Color figure online)

This structure uses a *bottleneck layer* as the basic building block of the dense module. Following [15,33], we design our own bottleneck layer as below. A bottleneck layer consists of sequential layers as follows: {BN - 1*1conv -BN - 3*3conv}, where BN denotes a batch normalization layer [16] which could normalize the feature maps by adjusting and scaling after activation functions. The BN layer is followed by a 1*1 convolution layer which keeps the feature map size the same. Then it is followed by a BN layer and a 3*3 convolution layer with zero padding. The bottleneck layer l keeps the output feature map size unchanged and controls the channel number of the feature maps always be constant number k (the meaning of k will be explained later).

In that case, the bottleneck layers could be densely connected [15] to increase density of dense module $M_{d,w}$. The densely connectivity could be presented as $x_l = H_l(x_0, x_1, x_2, ...x_{l-1})$, where H_l represents nonlinear operation on feature maps in bottleneck layer l and $(x_0, x_1, x_2, ...x_{l-1})$ represents the concatenation (channel-wise) of all previous outputs. It means that the input of layer l depends on outputs of every previous layers. As each bottleneck layer produces feature maps with k channels, the layer l has concatenated input feature maps with $k_0 + k \times (l-1)$ channels, where k_0 is the channel number of input feature maps in this dense module. In that case, as layer number l goes deeper, the channel number of input feature maps will grow rapidly as defined above. Following [15], we define k as the growth rate of the module, which could control the module scale. In experiments of this paper, we keep the growth rate k the same for all dense modules M.

We define a transit layer as the output layer of every dense module $M_{d,w}$. The transit layer consists of sequential layers {BN - Relu -1*1conv - Average-pooling}, where BN is the batch normalization layer introduced above, Relu is the activation function. The 1*1 convolution layer controls the output feature map channels as a constant number k_0 for all dense modules M. The average pooling layer mainly down scale the feature map size for final classification task. It should be noted that, the transit layer could control the output feature maps of dense modules to have a certain shape.

3.2 Explore External Sparse Connections

Since we have defined the whole network structure as a directed graph $G(M, P)$ and explained definition of dense modules M in Sect. 3.1, we will explain what a connection denotes in a neural network in this section. As we have explained before, M could be regarded as a set of nodes in the directed graph $G(M, P)$ and P is the adjacency matrix to represent the directed connections between modules M. The directed connections denote the feature map flow in our neural networks. Once there is a connection between two modules, it means one module will accept the output feature map of the other module as part of its inputs. For example, in Fig. 1, the module $M_{3,1}$ receives two directed connections from $M_{2,1}$ and $M_{2,3}$, it means module $M_{3,1}$ receives feature maps both from $M_{2,1}$ and $M_{2,3}$. Similarly, the module $M_{3,1}$ sends a directed connection to $M_{4,1}$, this means $M_{3,1}$ send output feature maps to $M_{4,1}$ as one of its inputs.

According to the *transit layers* defined in Sect. 3.1, output feature maps of all modules have the same channel number k_0. However, the sizes of feature maps from different depth are different. So, how can we make a module to accept output feature maps from different depth of feature maps? After the example, we will introduce how we exactly make a module to accept changeable feature maps from multiple other dense modules.

Methods for Local Connections with Same Depth

Here we define the distance as the difference $d_1 - d_2$ of two connected modules M_{d_1,w_1} and M_{d_2,w_2} in depth. If the distance of the connection is 1 (e.g., connection between $M_{3,3}$ and $M_{2,3}$ in Fig. 1), we call it a local connection. As we defined in Sect. 3.1, each output feature map of a dense module with same depth shares the same feature map size and the same channel number k_0. If there is only one local connection, we could naturally follow the down sampling flow of deep CNN and directly send the feature map output of the previous depth to the current module as: $O_{d,w} = M_{d,w}(O_{d-1,w_2})$, where O_{d-1,w_2} denotes the output feature map of previous depth M_{d-1,w_2}.

But what if the module $M_{d,w}$ has several local connections? We use an *Addition Method* to connect multiple feature map inputs which could be defined as: $O_{d,w} = M_{d,w}(O_{d-1,w_1} + O_{d-1,w_2}...)$, where O_{d_1,w_1} and $O_{d_1,w_2}...$ denote the multiple input feature maps. These additional methods directly add all the input feature maps from local connections and fit the required input feature map size of $M_{d,w}$. Actually we also try the *Concatenation Methods* which concatenate

all input feature maps channel wise and use a single transit layer introduced in Sect. 3.1 to reduce the channel number to required k_0. The experiment results show that the two connection methods have very similar results on the same structure. As addition methods don't need extra transit layers when changing the input feature map numbers, we choose addition methods as our connection methods.

Methods for Long Distance Connections from Different Depth

Within the same definition of distance above, a typical example of multiple long distance connections is as shown in Fig. 1, where $M_{4,3}$ receives directed connections from both $M_{3,2}$ (with depth 3) and $M_{1,2}$ (with depth 1). It should be noted that the existence of long distance connections means that feature maps could flow through different numbers of dense modules in this network structure. Here we also use the addition methods introduced above. Since the fact that the output feature maps of all the modules have same channel number k_0, the only problem for adding feature maps from different depth is the different feature map size. In that case, before addition, we implement an average pooling layer T_d according to the depth d to change the feature map size into current requirement of a module. After that, we could add all the adjusted input feature maps as the input of $M_{d,w}$. The math process can be define as $O_{d,w} = M_{d,w}(T_{d_1}(O_{d_1,w_1})) + T_{d_2}(O_{d_2,w_2})...)$, where $O_{d,w}$ denotes the output feature map of module $M_{d,w}$ and T_d denotes the pooling layer from depth d. In this case, we can achieve long distance connections.

3.3 Evolution Algorithm to Search External Sparse Connections

Since we've answered the question of what is a dense module in Sect. 3.1 and how the modules are connected in Sect. 3.2, this section will introduce how we decide the connection topologies.

One crucial problem in creating sparse connections between dense modules is that there has not been a convincing theory on what can be called an efficient connection. In that case, we decide to make the neural network evolving optimized sparse connections by itself. In this paper, we use a genetic algorithm [29] to search the proper connections. We take the adjacency matrix P as shown above as the gene for evolving. In each iteration, the genetic algorithm generates several new 'individuals' with genes from the mutation of the best 'individual' in last iteration. The set of generated 'individuals' is called 'population'. Genetic algorithm evolves by selecting best performance individual in every iteration.

Encoding: Here we take the adjacent matrix P to represent connection topology during training. In implementation details, we use a connection list of each module to reduce the storage space.

Initial State: The initial adjacency matrix is shown as Fig. 3-*Initial State* which only has direct local connections. We randomly initialize the weights value of modules at the first iteration in the training process. Since a deep neural network needs a long time to train, restricted to our computation capacity, we set the

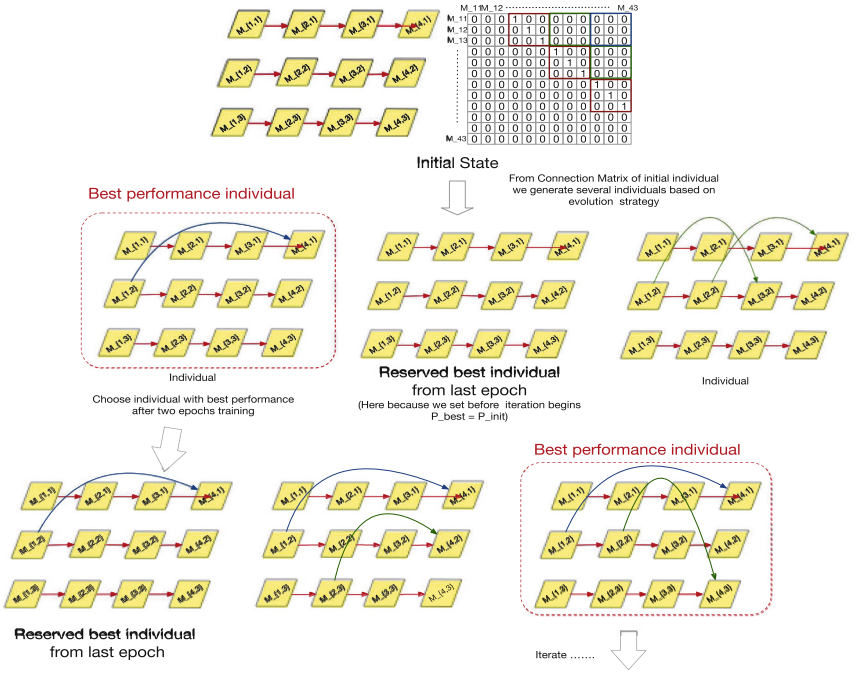


Fig. 3. An example of the Network Structure Evolving. *Initial state* denotes the initial adjacency matrix P. As we set before first iteration $P_{best} = P_{init}$, based on P_{best} we generate 2 individuals below. All together these 3 individuals form the population which to be trained simultaneously in iteration 1. Then, we choose the individual with the best performance, and based on that we form population for iteration 2. Following this principle we keep the network evolving.

population between 2 to 3 individuals. We define the adjacency matrix of the initial individual state as P_{init}, the best performance individual of the previous iteration as P_{best}, and others as P_i at beginning of each iteration. An example of evolving process is shown in Fig. 3.

Evolution Strategy: At each iteration, the mutation function will generate several new individuals based on the best individual from the previous iteration P_{best}. The mutation function could be defined as:

$$P_1, P_2... = Mutation(P_{best}) \tag{1}$$

Where it accepts P_{best} as its input, then generates several mutation individuals $P_1, P_2...$ based on P_{best}. The mutation function randomly picks two possible connections and changed their connectivity based on the input adjacency matrix. It means that, if we randomly pick an unconnected connection, we set it connected. And for already connected connection, we set it disconnected. Then we treat the set of $P_{best}, P_1, P_2...$ as population in this iteration and separately resume training the whole network under these connection conditions for an epoch and choose

Algorithm 1. Evolutionary Connectivity Algorithm

procedure EVOLVE($G(M, P_{init}), Data, n$) ▷ *Data*: Training data, $G(M, P_{init})$: Given network structure with modules M and initial adjacency matrix P_{init}, n: Total iteration
  $P_{best} \leftarrow P_{init}$
  for n iterations **do**
    $P_1, P_2 ... P_{k-1}, P_k \leftarrow Mutation(P_{best})$ ▷ k,Number of individuals in a generation, $P_k = P_{best}$
    $checkpoint \leftarrow G(M, P_{best})$
    for k iterations **do**
      Resume weights of M in the *Checkpoint*: $G(M, P_k) \leftarrow checkpoint$
      train $G(M, P_k)$ on *Data* and get validation accuracy
      if $P_k.accuracy > P_{best}.accuracy$ **then**
        $P_{best} \leftarrow P_k$
        $best - check \leftarrow G(M, P_k)$
      end if
    end for
    Resume weights of M in the $best - check$: $G(M, P_{best}) \leftarrow best - check$
  end for
  Return P_{best}
end procedure

the individual with the best accuracy as P_{best} of current iteration. Adjacency matrix of the best performance individual P_{best} will remain to next iteration. And based on it we mutate new individuals. The whole process is shown in Algorithm 1.

4 Experiments

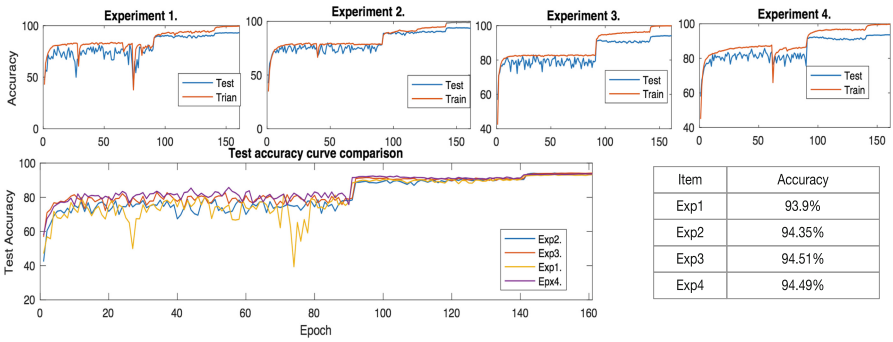
In this section, we apply several experiments to reveal interesting phenomena and properties of the internally dense yet externally sparse deep neural network structures. All of the experiments in this section are based on CIFAR10, CIFAR100 and ImageNet datasets for image classification tasks. Section 4.1 shows the efficiency of the evolutionary algorithm by several repeatability experiments. Section 4.2 discusses how the growth rate of each module will affect the model performance. Section 4.3 gives our performance benchmark compared to state-of-the-art models. At last, we do a detailed discussion about which connections are important for the whole model in Sect. 4.4.

4.1 Evolving Sparse Connections

This experiment is used to prove the efficiency of the Evolutionary Connectivity Algorithm introduced in Sect. 3.3. We evaluate the network structure efficiency on the classification task using benchmark dataset CIFAR10. In implementation details, we prefix the dense modules having 4 different depth, where in each

depth, it has 3 modules. The total of 12 modules M have the growth rate k of 12. The dense modules in depth 1, 2, 3, 4 respectively have 6, 12, 24, 16 bottleneck layers inside each module. The input feature map channels of each dense module k_0 is 32. The preprocessed images firstly flow through a 3*3 convolution layer and generate feature maps with 96 channels. Then the feature maps are divided into 3 feature map groups with 32 channels each. And the three groups are separately fed to input dense modules with depth 1 ($M_{1,1}, M_{1,2}, M_{1,3}$).

Through the training process, the network evolves sparse connections between prefixed dense modules according to the Evolutionary Connectivity Algorithm 1. We set the total iteration number to be 160, with weight decay of 5e−4. The training uses SGD with momentum 0.9 for gradient descent. The learning rate strategy is the same as most of the papers that during epoch 0–90 the learning rate is 0.1, during 90–140 learning rate is 0.01, and during 140–160 learning rate is 0.001. It should be noted that changing the learning rate will lead to accuracy 'step jumps' such as Figs. 4, 5 and 6 show. Restricted to our computation power, we set the number of individuals generated in each iteration to be 2. The training curves of P_{best} are shown in Fig. 4. All the experiments are trained on NVIDIA AWS P3.x2large instance.



Item	Accuracy
Exp1	93.9%
Exp2	94.35%
Exp3	94.51%
Exp4	94.49%

Fig. 4. Several Repeatable Experiments on Sparse Connection Evolving. The upper four figures denote the training curve & testing curve of each experiment. The lower figure denotes the comparison of test accuracy of each experiment. All accuracy step jumps are caused by learning rate change strategy in Sect. 4.1.

According to the repeatable experiment results, although randomness of forming the early generations may lead to variation and fluctuation on the testing performance curve, the training curve will finally converge to the same trend. This shows the *repeatability* of our algorithm. Based on these experiments, we found that the optimized adjacency matrix is not unique to achieve good performances. The evolving results are shown in Fig. 5. However, we could still find some similarity between those evolving results of these experiments. It denotes that the modules with shallow depth are more likely to form a long-distance connection. This means that the distance between the input and the output are

shortened under that situation. This perfectly fits a current trend observed by various papers [6,12,15,27,32,33] that skip/direct connections are important.

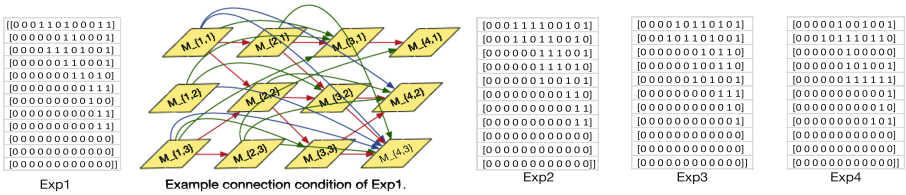


Fig. 5. Connection Matrix with Best Performance on each Experiment. We also give an example connection status of Exp1.

4.2 How Growth Rate of the Dense Module Influences the Final Result

As we mentioned above in Sect. 3.1, growth rate k is an important parameter which controls the model scale. In order to figure out the influence of growth rate k, this subsection introduces the contrast experiment by controlling all other factors the same except the growth rate k in the prefix dense modules. The prefixed modules and training parameters are the same as those used in experiment Sect. 4.1. We train the network with the same strategy and the same device above. The results are shown in Fig. 6.

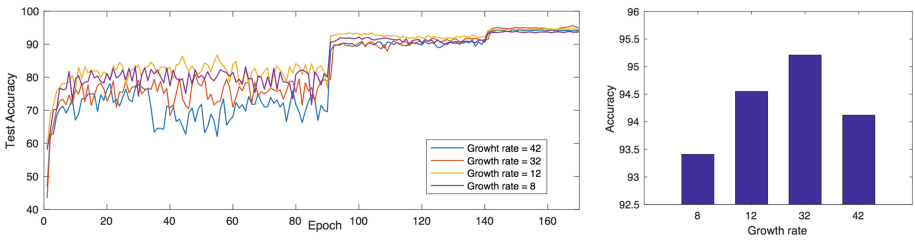


Fig. 6. Test Accuracy Curve Comparison on Different Growth Rate. Each color represents test accuracy curve of experiments on different growth rate.

Clearly, the networks with smaller growth rate converge faster and have flatter curve shapes compared to those with larger growth rates at the earlier period of training. It means that the modules with smaller scale are easier to train while the connections of this network evolving. We can also see that, although modules with smaller growth rates converge really fast, the final test accuracy is not as high as those modules with larger growth rate. However, experiment results also

demonstrate that the network redundancy is not the 'larger the better'. As it shows in Fig. 6, after the growth rate is larger than 32, the test accuracy will not increase anymore. It is also rational because if the capacity of each module is too large, the unstable input features may make the network harder to train. On another hand, the increasing growth rate, which leads to the increasing of model scale, increases the risk of over-fitting.

4.3 Performance Benchmark

Although our paper emphasizes on how sparse connections will change the model performance, we still give performance scores on the benchmark dataset as shown in Tables 1 and 2. Since the aim of this paper is to obtain slim structures while keeping the model's capacity and achieve internally dense yet externally sparse network structures, the test accuracy on both *ImageNet* and *CIFAR* is not that high compared to state-of-the-art models. However, we still get competitive results on both datasets.

Table 1. Test error rate performance on CIFAR dataset. Note results with * are the best result run by ourselves.

Method	Params	Depth	CIFAR-10	CIFAR-100
Network in network [21]	–	–	8.81	35.68
VGG19 [28]	–	–	6.58	27.09
Highway Network [30]	–	–	7.72	32.29
DFN [35]	3.9M	50	6.40	27.61
Fractal Net [19]	38.6M	21	5.22	23.30
Resnet [12]	1.7M	110	5.46 5.58*	27.62
Pre-activated Resnet [13]	1.7M	164	4.72 5.12*	25.6
Wide Resnet [39]	7.4M	32	5.4	23.55
Densenet ($k = 12$) [15]	1M	40	5.24 5.43*	24.42 24.98*
Densenet-BC ($k = 12$)	0.8M	100	4.51	22.27
Densenet121 ($k = 24$)	15.2M	121	4.68*	21.49*
SDMN, growth rate $k = 8$, 6 modules	0.4M	–	6.97*	–
SDMN, growth rate $k = 8$, 8 modules	1.3M	–	6.59*	25.6*
SDMN, growth rate $k = 8$, 12 modules	2.3M	–	5.97*	24.8*
SDMN, growth rate $k = 12$, 12 modules	3.7M	–	5.35*	23.41*
SDMN, growth rate $k = 32$, 12 modules	22M		4.79*	21.9*

4.4 Separable of Sparse Connections

In this subsection, we discuss which connections are important based on one of our evolved adjacency matrices. We separately cut off one sparse connection

Table 2. Test accuracy rate performance on ImageNet dataset, compared with slim network models.

Model	Params	Top1/Top5 Acc.
MobileNetV1	4.2M	70.6%/89.5%
ShuffleNet (2x)	4.4M	70.9%/89.8%
MobileNetV2 (1.4)	6.9M	74.7%/â
NASNet-A (N = 4, F = 44)	5.1M	74.0%/91.3%
Sparse-Dense-Modules (k = 12)	3.7M	71.1%/90.0%

from the whole network structure each time and test the remaining accuracy on *CIFAR10* dataset. Then we come up with a matrix, which suggests the accuracy decreasing while losing each connection. The matrix is shown in Fig. 7.

The red rectangle area denotes the direct connections; the green and blue rectangle area denote the long-distance connections. According to the accuracy loss distribution, local and direct connections are of vital importance for a neural network. This is rational because the deep learning method needs a compared invariant forward and backward feature flow path for loss propagation. We can also see the accuracy loss is larger along the diagonal to the high left of the matrix. It means that connections within shallow depth play a more important role in conducting features/patterns than deeper connections. It is also rational because the shallower connections simultaneously mean the features that flow through such connections have not been extract to some level of abstraction.

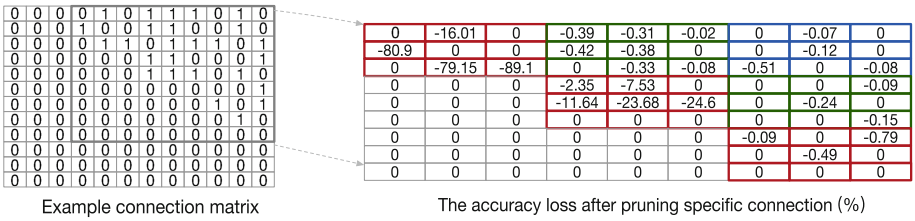


Fig. 7. Example connection matrix shows the selected best connection from a typical experiment. Right part of the figure shows how much accuracy will loss if we cut off the corresponding connection in the connection matrix. (Color figure online)

5 Conclusions and Future Work

In this paper, we firstly introduce locally dense but externally sparse structures of deep convolutional neural network by prefixing some dense modules M and evolving sparse connections between them. Experiment results demonstrate

that evolving sparse connections between dense modules could reach competitive results on benchmark datasets. In order to analyze the properties of these biologically plausible structures, we apply several sets of contrast experiments and show in *Experiment* section. By changing the growth rate of each dense module, we analyze how model scale will influence the model performance. Similar to most of the related works, redundancy of each dense module is not 'the larger the better', where the test accuracy will first increase with the growth rate increasing, but finally drop while the growth has reached some thresholds. We also analyze the contribution of each connection to the whole model by disconnecting each connection and separately testing the accuracy of the model with the disconnected connection. It shows that local connections are important for baseline accuracy, while long-distance connections could improve the accuracy by small steps.

The combination of being dense and being sparse is an interesting area. The internally dense and externally sparse structures also coincide with the modularity in human brain. We demonstrate the feasibility of these structures and give a simple algorithm to search best connections. We also notice that the connection matrix is not unique for reaching good performance. We will concentrate on revealing the relationship between these similar connection matrices and the corresponding features behind it. In this case, we may acquire state-of-the-art performance on other datasets and tasks in our future work. Moreover, as these structures have various direct paths between input and output, separating a network into several small networks without big accuracy loss is also a promising topic.

Acknowledgment. We thank the anonymous reviewers for their comments. This work was supported by Mitacs Accelerate Grant IT08175 with Two Hat Security Research Corporation. Part of this work has been prototyped in the CEASE.ai project at Two Hat Security.

References

1. Angeline, P.J., Pollack, J.: Evolutionary module acquisition. In: Proceedings of the Second Annual Conference on Evolutionary Programming, pp. 154–163. Citeseer (1993). https://doi.org/10.1007/978-3-540-24650-3_17
2. Belliveau, J., et al.: Functional mapping of the human visual cortex by magnetic resonance imaging. Science **254**(5032), 716–719 (1991). https://doi.org/10.1126/science.1948051
3. Bengio, Y., Simard, P., Frasconi, P.: Learning long-term dependencies with gradient descent is difficult. IEEE Trans. Neural Networks **5**(2), 157–166 (1994). https://doi.org/10.1109/72.279181
4. Betzel, R.F., et al.: The modular organization of human anatomical brain networks: accounting for the cost of wiring. Netw. Neurosci. **1**(1), 42–68 (2017). https://doi.org/10.1162/NETN_a_00002
5. Braun, H., Weisbrod, J.: Evolving neural feedforward networks. In: Albrecht, R.F., Reeves, C.R., Steele, N.C. (eds.) Artificial Neural Nets and Genetic Algorithms, pp. 25–32. Springer, Vienna (1993). https://doi.org/10.1007/978-3-7091-7533-0_5. https://doi.org/10.1109/72.80206

6. Chollet, F.: Xception: deep learning with depthwise separable convolutions. arXiv preprint, pp. 1610–02357 (2017). https://doi.org/10.1109/CVPR.2017.195
7. Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., Bengio, Y.: Binarized neural networks: training deep neural networks with weights and activations constrained to +1 or −1. arXiv preprint arXiv:1602.02830 (2016)
8. Denton, E.L., Zaremba, W., Bruna, J., LeCun, Y., Fergus, R.: Exploiting linear structure within convolutional networks for efficient evaluation. In: Advances in Neural Information Processing Systems, pp. 1269–1277 (2014)
9. Gazzaniga, M.S.: Brain modularity: towards a philosophy of conscious experience (1988)
10. Han, S., Mao, H., Dally, W.J.: Deep compression: compressing deep neural networks with pruning, trained quantization and Huffman coding. arXiv preprint arXiv:1510.00149 (2015). https://doi.org/10.1145/2351676.2351678
11. Han, S., Pool, J., Tran, J., Dally, W.: Learning both weights and connections for efficient neural network. In: Advances in Neural Information Processing Systems, pp. 1135–1143 (2015)
12. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016). https://doi.org/10.1109/CVPR.2016.90
13. He, K., Zhang, X., Ren, S., Sun, J.: Identity mappings in deep residual networks. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) ECCV 2016. LNCS, vol. 9908, pp. 630–645. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46493-0_38
14. He, Y., Zhang, X., Sun, J.: Channel pruning for accelerating very deep neural networks. In: International Conference on Computer Vision (ICCV), vol. 2, p. 6 (2017)
15. Huang, G., Liu, Z., Weinberger, K.Q., van der Maaten, L.: Densely connected convolutional networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, vol. 1, p. 3 (2017). https://doi.org/10.1109/CVPR.2017.243
16. Ioffe, S., Szegedy, C.: Batch normalization: accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167 (2015)
17. Kim, Y.D., Park, E., Yoo, S., Choi, T., Yang, L., Shin, D.: Compression of deep convolutional neural networks for fast and low power mobile applications. arXiv preprint arXiv:1511.06530 (2015)
18. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems, pp. 1097–1105 (2012). https://doi.org/10.1145/3065386
19. Larsson, G., Maire, M., Shakhnarovich, G.: FractalNet: ultra-deep neural networks without residuals. arXiv preprint arXiv:1605.07648 (2016)
20. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998). https://doi.org/10.1109/5.726791
21. Lin, M., Chen, Q., Yan, S.: Network in network. arXiv preprint arXiv:1312.4400 (2013)
22. Mao, H., et al.: Exploring the regularity of sparse structure in convolutional neural networks. arXiv preprint arXiv:1705.08922 (2017)
23. Park, J., et al.: Faster CNNs with direct sparse convolutions and guided pruning. arXiv preprint arXiv:1608.01409 (2016)
24. Pujol, J.C.F., Poli, R.: Evolving the topology and the weights of neural networks using a dual representation. Appl. Intell. **8**(1), 73–84 (1998). https://doi.org/10.1023/a:1008272615525

25. Rastegari, M., Ordonez, V., Redmon, J., Farhadi, A.: XNOR-Net: ImageNet classification using binary convolutional neural networks. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) ECCV 2016. LNCS, vol. 9908, pp. 525–542. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46493-0_32
26. Real, E., Aggarwal, A., Huang, Y., Le, Q.V.: Regularized evolution for image classifier architecture search. arXiv preprint arXiv:1802.01548 (2018)
27. Redmon, J., Farhadi, A.: YOLOv3: an incremental improvement. arXiv preprint arXiv:1804.02767 (2018)
28. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
29. Srinivas, M., Patnaik, L.M.: Genetic algorithms: a survey. Computer **27**(6), 17–26 (1994). https://doi.org/10.1109/2.294849
30. Srivastava, R.K., Greff, K., Schmidhuber, J.: Highway networks. arXiv preprint arXiv:1505.00387 (2015)
31. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evol. Comput. **10**(2), 99–127 (2002). https://doi.org/10.1162/106365602320169811
32. Szegedy, C., Ioffe, S., Vanhoucke, V., Alemi, A.A.: Inception-v4, inception-ResNet and the impact of residual connections on learning. In: AAAI, vol. 4, p. 12 (2017)
33. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the inception architecture for computer vision. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2818–2826 (2016). https://doi.org/10.1109/CVPR.2016.308
34. Sztarker, J., Tomsic, D.: Brain modularity in arthropods: individual neurons that support "what" but not "where" memories. J. Neurosci. **31**(22), 8175–8180 (2011). https://doi.org/10.1523/jneurosci.6029-10.2011
35. Wang, J., Wei, Z., Zhang, T., Zeng, W.: Deeply-fused nets. arXiv preprint arXiv:1605.07716 (2016)
36. Wen, W., Wu, C., Wang, Y., Chen, Y., Li, H.: Learning structured sparsity in deep neural networks. In: Advances in Neural Information Processing Systems, pp. 2074–2082 (2016)
37. Xie, G., Wang, J., Zhang, T., Lai, J., Hong, R., Qi, G.J.: IGCV 2: interleaved structured sparse convolutional neural networks. arXiv preprint arXiv:1804.06202 (2018). https://doi.org/10.1109/CVPR.2018.00922
38. Xie, L., Yuille, A.: Genetic CNN. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 1379–1388 (2017)
39. Zagoruyko, S., Komodakis, N.: Wide residual networks. arXiv preprint arXiv:1605.07146 (2016). https://doi.org/10.5244/C.30.87
40. Zhou, A., Yao, A., Guo, Y., Xu, L., Chen, Y.: Incremental network quantization: towards lossless CNNs with low-precision weights. arXiv preprint arXiv:1702.03044 (2017)
41. Zoph, B., Vasudevan, V., Shlens, J., Le, Q.V.: Learning transferable architectures for scalable image recognition (2017)