Layer-Wise Training to Create Efficient Convolutional Neural Networks

Linghua Zeng and Xinmei Tian(✉)

CAS Key Laboratory of Technology in Geo-spatial Information Processing and Application System, University of Science and Technology of China, Hefei 230027, Anhui, China
zenglh@mail.ustc.edu.cn, xinmei@ustc.edu.cn

Abstract. Recent large CNNs have delivered impressive performance but their storage requirement and computational cost limit a wide range of their applications in mobile devices and large-scale Internet industry. Works focusing on storage compression have led a great success. Recently how to reduce computational cost draws more attention. In this paper, we propose an algorithm to reduce computational cost, which is often solved by sparsification and matrix decomposition methods. Since the computation is dominated by the convolutional operations, we focus on the compression of convolutional layers. Unlike sparsification and matrix decomposition methods which usually derive from mathematics, we receive inspiration from transfer learning and biological neural networks. We transfer the knowledge in state-of-the-art large networks to compressed small ones, via layer-wise training. We replace the complex convolutional layers in large networks with more efficient modules and keep their outputs in each-layer consistent. Modules in the compressed small networks are more efficient, and their design draws on biological neural networks. For AlexNet model, we achieve 3.62× speedup, with 0.11% top-5 error rate increase. For VGG model, we achieve 5.67× speedup, with 0.43% top-5 error rate increase.

Keywords: Deep learning · Network compression · Layer-wise training

1 Introduction

Large CNNs have recently demonstrated state-of-the-art performance in image classification task, which is treated as an important benchmark for computer vision [10]. A well-known competition, Large Scale Visual Recognition Challenge (ILSVRC) [10], and its database have given birth to lots of famous CNNs. These networks (e.g. AlexNet [1] and VGG [2]) are powerful and possess great representation capability. Usually very similar models are used in the training stage and the deployment stage despite their enormously different requirements. In the training stage, most state-of-the-art CNNs focus on decreasing classification error rate. Thus, CNNs are usually designed to have parameters as many as possible if they could achieve lower error rate. Consequently, a huge number of redundant parameters will be generated in this stage [4]. In deployment stage, apart from the error rate, there are strict requirements on storage and computational cost [6]. Improving the efficiency of CNNs is of critical importance.

D. Liu et al. (Eds.): ICONIP 2017, Part II, LNCS 10635, pp. 631–641, 2017.
https://doi.org/10.1007/978-3-319-70096-0_65

To address these issues, there is growing concern about network compression in recent years. It can be roughly divided into storage compression and computation compression. The study of storage compression has been considerably thorough [6]. As a representative work, a three stage pipeline was introduced by Han et al. [6]. Pruning, trained quantization and Huffman coding worked together to reduce the storage requirement by 35× to 49× without increasing the error rate. However, storage compression methods are not remarkably efficient for computation compression. Sparsification and matrix decomposition are fundamental approaches in computation compression. Denton et al. [3] exploited the linear structure of the parameters and found appropriate low-rank approximation of the parameters in different layers. Zhang et al. [5] enabled an asymmetric reconstruction that reduced the rapidly accumulated error when multiple layers were compressed, achieving 5× FLOPs reduction with 1.0% top-5 error rate increase (5× /+1.0) on VGG model. Figurnov et al. [7] sped up the bottleneck convolutional layers by skipping their evaluation in some of the spatial positions, achieving 2× /+2.0 on AlexNet and 1.9× /+2.5 on VGG. Kim et al. [8] used Tucker Decomposition on each layer with the rank determined by a global analytic solution of VBMF, achieving 2.67× /+1.70 on AlexNet and 4.93× /+0.50 on VGG.

These computation compression methods mainly focus on decomposition algorithms [5, 8] or position choosing algorithm [7] deriving from mathematics. Different from them, we focus on training algorithm inspired by transfer learning. We study how to transfer knowledge in state-of-the-art (big model) networks to compressed networks (small model). Since the computation is dominated by the convolutional operations [3], we focus on the compression of convolutional layers. Firstly we train a big model with redundant parameters to reach the highest possible performance. After that, we propose "layer-wise training", to compress the redundant parameters in the big model and transfer its useful information into the small model. An earlier work proposed by Hinton et al. [4] utilized the last layer output of the big model to train a small model. However, not all of the big model knowledge is included in the last-layer output. Feature maps, generated by convolutional layers, including knowledge about how the network identifies objects, are helpful to teach the small model. For example, when we teach a baby to identify a car, we will not only tell him that this is a car but also tell him that a car has four wheels, the anterior window and others. In our layer-wise training algorithm, we teach the small model not only what the image is but also how to identify the image. Efficient modules in the small model are designed to replace conventional convolutional layers in the big model. We keep the each-layer outputs of the small model and the big model to be consistent. The design of modules draws on biological neural networks, and they also can be explained from the perspective of matrix decomposition.

This paper has the following major contributions:

(1) We propose the layer-wise network training to compress a big model into a small model to reduce computational cost.
(2) We design a set of novel and efficient modules inspired by biological neural networks for layer-wise training and it is the first trying on network compression. Besides they can also be explained from the perspective of matrix decomposition.

(3) We evaluate our methods on large datasets, achieving 3.62× FLOPs reduction with 0.11% top-5 error rate increase on AlexNet model, and 5.67× FLOPs reduction with 0.43% top-5 error rate increase on VGG model.

2 Layer-Wise Training to Create Efficient CNNs

2.1 Overall Framework

In this section, we introduce the scheme of our layer-wise training method. Our scheme consists of three steps: the big model training, layer-wise training for compression, and the small model fine-tuning. Here we use AlexNet [1] as an example, as illustrated in Fig. 1.

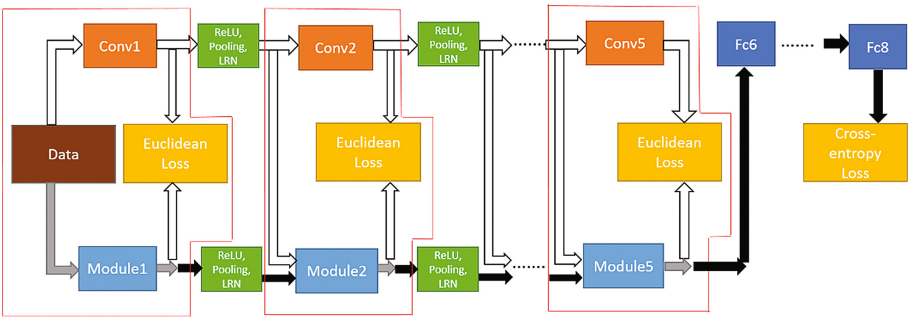

Fig. 1. The scheme of training algorithm. The upper flow in this chart, from conv1 to conv5, represents convolutional layers in the big model. The lower flow, from module1 to module5, represents modules in the small model. The rest part, fc6, fc7, and fc8, represent fully connected layers shared by both big model and small model. Data means input image or we can say training data. Euclidean Loss means the Euclidean Loss layer which computes loss and generate gradients. The red frames are blocks. Since we focus on convolutional layers, ReLU, Pooling and LRN [1] are omitted and represented in a box. (Color figure online)

In the first step, we train a big redundant model (e.g. AlexNet [1], VGG [2]) to achieve the highest possible performance. In the second step, we train modules in the small model to replace convolutional layers in the big model. We keep the each-layer outputs of them to be consistent. We choose Euclidean-Loss function,

$$E = \frac{1}{2}\|x - m\|_2^2. \tag{1}$$

where x and m are the outputs of convolutional layers in the big model and modules in the small model respectively. In this step, as shown in Fig. 1, only white and grey lines work, and the black lines are cut off. The training unit is a block, marked with a red frame. Each block consists of a convolutional layer and a module. In fact, we train modules in each block independently, which means no back propagation between blocks.

In the third step, we fine-tune the small model using cross-entropy loss. Since we use softmax function after the last layer output, we combine it with loss function,

$$E = -log\frac{e^{f_l}}{\sum_{j=1}^{c} e^{f_j}} = log\sum_{j=1}^{c} e^{f_j} - f_l, \tag{2}$$

where f is the output of last fully connected layer in the small model, c is the channel (dimension) of this output, l is the label index of the training image, and f_l is the scalar of f on label index. Image is filled with noise obstructing the classification, and it is learned by parameters. The experiment on AlexNet shows the existence of noise along parameters after layer-wise training. So we fine-tune the whole network after layer-wise training. In this step, only grey and black lines in Fig. 1 work, and the white lines are cut off. The fully connected layers in the small model are copied from the big model, and they share the same parameters.

2.2 Matrix Decomposition

In this section, we introduce matrix decomposition theory in our work. Formally, each convolutional layer takes a stack of feature maps as input, a 3-D tensor denoted as $Z \in \mathbb{R}^{c\times h\times w}$, where h and w are the height and width of feature maps respectively, and c is the number of feature maps also called channel. The parameters of convolutional layers, also called kernels, are denoted as $W \in \mathbb{R}^{n\times c\times d\times d}$, where n is the number of output feature maps, c is the number of input feature maps also called input channels, and $d \times d$ is the spatial kernel size. The output of a convolutional layer, $A \in \mathbb{R}^{n\times h'\times w'}$ is,

$$\begin{gathered} A_{u,i,j} = \sum_{v=1}^{c}\sum_{m=1}^{d}\sum_{n=1}^{d} Z_{v,(i-1)s+m-p,(j-1)s+n-p} W_{u,v,m,n} \\ i.e.\ A = Z \otimes W, \end{gathered} \tag{3}$$

where p and s are padding size and convolutional stride respectively, and $\otimes$ represents spatial convolution. If we decompose W into two matrices, denoted as $P \in \mathbb{R}^{c\times k}$ and $Q \in \mathbb{R}^{\mathrm{n\times k\times d\times d}}$,

$$W_{u,v,m,n} = \sum_{l=1}^{k} P_{v,l} Q_{u,l,m,n}. \tag{4}$$

$$\begin{aligned} A_{u,i,j} &= \sum_{v=1}^{c}\sum_{m=1}^{d}\sum_{n=1}^{d} Z_{v,(i-1)s+m-p,(j-1)s+n-p} \sum_{l=1}^{k} P_{v,l} Q_{u,l,m,n} \\ &= \sum_{l=1}^{k}\sum_{m=1}^{d}\sum_{n=1}^{d} \left(\sum_{v=1}^{c} Z_{v,(i-1)s+m-p,(j-1)s+n-p} P_{v,l}\right) Q_{u,l,m,n} \\ &i.e.\ A = Z \otimes P \otimes Q. \end{aligned} \tag{5}$$

This formula means that the origin convolutional layer can be decomposed into two sequentially connected convolutional layers: one layer with 1×1 kernels and the other with $d \times d$ kernels. Besides, if we exchange the orders of P and Q, we have

$$W_{u,v,m,n} = \sum_{l=1}^{k} Q_{v,l,m,n} P_{l,u}. \tag{6}$$

In this way, we can obtain a variant of sequentially-connected convolutional layers, which are one layer with $d \times d$ kernels and the other with 1×1 kernels.

2.3 Modules Inspired by Biological Neural Networks

In this section we introduce the principles behind our module design method. The Ventral Stream, going through V1, V2, V4 and the inferior temporal lobe area, is involved in object identification and recognition [9]. We design three kinds of modules, called MODULE-A, MODULE-B, and MODULE-C, shown in Fig. 2. Following the inspiration from Ventral Stream, MODULE-A corresponds to V1 and V2. MODULE-B and MODULE-C correspond to V4 and the inferior temporal lobe area respectively.

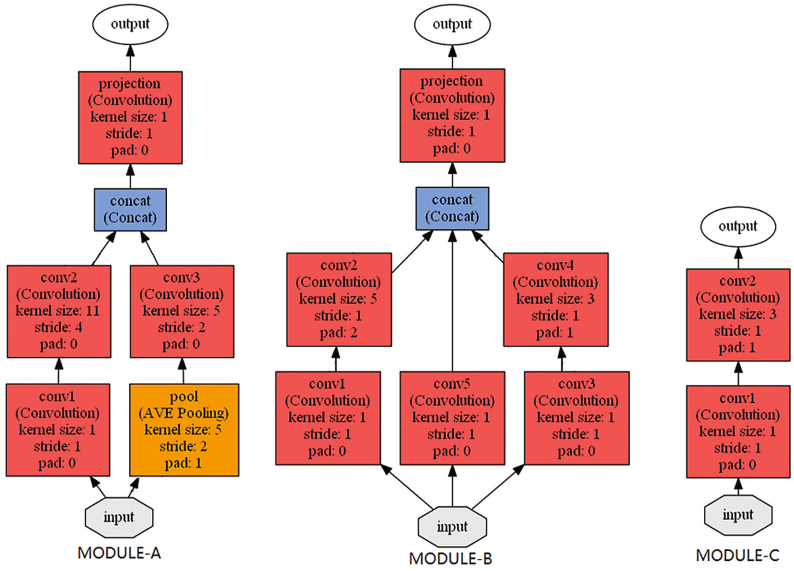

Fig. 2. Three kinds of modules used in our model. A concatenate layer (Concat) concatenates two or more outputs of convolutional layers along the channel axis.

MODULE-A usually takes the role of the first layer in the small model. Similarly V1 and V2 are in the front of visual cortex. Color and shape are treated in different areas in V1 and V2 [9]. As shown in Fig. 2, MODULE-A has two paths which learn shape and color respectively: one path consists of conv1 and conv2, and the other consists of average pooling and conv3. The number of output feature map of conv1 is 1, which means conv1 translates the input color image into a colorless image. Therefore conv2 can only learn the shape. Average pooling is used to decrease the size of input, because we find the color path requires lower resolution. Conv3 receives all color channels, reflecting the color path. The output of MODULE-A is a linear combination of conv2 and conv3 via projection (convolution with 1×1 kernels). The structure of MODULE-A can also be explained from the perspective of matrix decomposition

introduced in Sect. 2.2. Firstly we decompose the original convolutional layer into conv' and projection. Then conv' can be divided into two parts along channel axis, noted as conv2' and conv3'. Then we decompose conv2' into conv1 and conv2. And we decrease kernel size of conv3' to obtain conv3. To keep the receptive fields of two paths consistent, we add an average pooling layer before conv3.

Our MODULE-B, similar with V4, replaces the middle convolutional layers in the big model. V4 has a significant function in visual attention [13]. Inspired by this, MODULE-B focuses on the important part of feature maps and filters the jamming information. MODULE-B has three paths representing different scales of attention. The kernel sizes in conv3, conv4 and conv5 are 5×5, 3×3, and 1×1 respectively. Finally we combine different scales of attentions together via projection. There is also another explanation for MODULE-B from matrix decomposition scheme in Sect. 2.2. Firstly we decompose the original convolutional layer into conv' and projection. Then conv' can be divided into three parts along channel axis, noted as conv1', conv3', and conv5. We decompose conv1' into conv1 and conv2. Similarly we decompose conv3' into conv3 and conv4. Finally we decrease the kernel sizes of conv4 and conv5.

In deeper layers, we adopt MODULE-C. The inferior temporal lobe is capable of remembering particular objects [12]. MODULE-C is very simple, consisting of conv1 and conv2, with 1×1 and 5×5 kernels respectively. Conv2 works as a memory unit, storing the information of object. Conv1 fuses feature maps to match the memory. Conv1 and conv2 in MODULE-C also can be considered as decomposition of a convolutional layer. If we exchange the position of conv1 and conv2, we obtain a variant of MODULE-C. Here conv2 is still a memory unit, but conv1 copes with memory information to form more complicated object.

3 Experiment

To validate our algorithm, we reduce computational complexity of state-of-the-art CNNs without much error rate increase. Following [5, 7, 8], we measure the computational complexity as the number of floating point multiply accumulate operations (FLOPs) in the forward propagation through convolutional layers. In our method, conventional convolutional layers in large CNNs are replaced with our proposed modules that need much fewer FLOPs. All CNNs are implemented using Caffe [11]. The layer-wise training adopts batch gradient descent method.

3.1 MNIST

MNIST is a large database of handwritten digits. We introduce lenet-conv, a model in Caffe [11], consisting of two convolutional layers and two fully connected layers. The validation error rate of lenet-conv is 1.03%. We use a variant of MODULE-C to replace convolutional layers in lenet-conv and the new model is denoted as lenet-conv-dec, as shown in Table 1.

Table 1. The structure of lenet-conv and lenet-conv-dec. Conv1, conv2, fc1, and fc2 are layer names. (K × K, N) denotes a convolutional layer with N kernels of K × K size. Fc1 has 500 neurons and fc2 has 10.

model	conv1		conv2		fc1	fc2
lenet-conv	5×5,20		5×5,50		500	10
lenet-conv-dec	5×5,4	1×1,20	5×5,10	1×1,50	500	10

We compare our layer-wise training algorithm with Distilling [4] and label training, to prove that our algorithm enables faster convergence rate. We didn't compare with other methods [5, 7, 8], because they were designed for large networks and results on this small database were not reported. Label training is the original training algorithm of lenet-conv. Distilling and label training run 15K iterations, and the lowest error rates and corresponding iterations were recorded, shown in Table 2. Our algorithm run 500 iterations on layer-wise training and 5000 iterations on fine-tuning. In the compare of error rates, they are all close to lenet-conv, but ours is a little lower. In the compare of training iterations, our algorithm has an advantage.

Table 2. Error and training iterations of different algorithms. Our algorithm run 500 iterations on layer-wise training and 5000 iterations on fine-tuning.

Algorithm	Error	Training iterations
Label training	1.07	12500
Distilling [4]	1.04	11500
Layer-wise training	**0.99**	**500 + 5000**

3.2 ILSVRC

The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [10] evaluates algorithms for image classification at large scale. We adopt two famous CNN models, AlexNet [1] (CaffeNet [11] as a variant) and VGG-16 [2], as our baselines. The AlexNet and VGG-16 are directly downloaded from Caffe's [11] model zoo. In the following experiment, we adopt the increase of top-5 error rate and reduction of FLOPs as benchmarks to compare different algorithms. In this section, we prove our algorithm is better than previous algorithms [5, 7, 8] in both the error rate and compression rate. All cited results come from their papers. We didn't compare Distilling method [4] because the results on this large database were not reported and we achieve poor results with this method.

AlexNet Model

Structures of different models are detailed in Table 3. The increase of top-5 error rate and reduction of FLOPs are given in Table 4. The alex-base model is the original AleNet, with 19.78% top-5 error rate and 666M FLOPs.

Table 3. The structures of five variants of AlexNet. The labels, from conv1 to conv5, represent layer names of five convolutional layers. (K × K, N, %G, /S) represents a convolutional layer with N kernels of K × K size, S strides and G groups. If a hyper-parameter is omitted, it is 1. (avepool5×5, /2) means an average pooling layer with 5 × 5 kernel size and 2 strides.

Layer	alexnet-base	alexnet-dec-base	alexnet-dec-back	alexnet-dec-front	alexnet-dec-mod		
conv1	11×11,96,/4	11×11,96,/4	11×11,48,/4	11×11,96,/4	1×1,1		avepool5×5,/2
					11×11,24,/4		5×5,40,/2
			1×1,96		1×1,96		
conv2	5×5,256,%2	5×5,256,%2	5×5,128,%2	1×1,48,%2	1×1,16	1×1,12	1×1,16
						3×3,48	5×5,64
			1×1,256,%2	5×5,256,%2	1×1,256		
conv3	3×3,384	3×3,384	3×3,192	1×1,128	1×1,128		
			1×1,384	3×3,384	3×3,384,%2		
conv4	3×3,384,%2	3×3,384,%2	3×3,192,%2	1×1,192,%2	1×1,96		
			1×1,384,%2	3×3,384,%2	3×3,384,%2		
conv5	3×3,256,%2	3×3,256,%2	3×3,128,%2	1×1,192,%2	1×1,128		
			1×1,256,%2	3×3,256,%2	3×3,256,%4		

Table 4. Comparison of top-5 error rate increase and FLOPs reduction based on AlexNet.

Model	alexnet-dec-base	alexnet-dec-back	alexnet-dec-front	alexnet-dec-mod	Kim's [8]	Figurnov's [7]
error↑	+0.01	+0.04	−0.10	**+0.11**	+1.70	+2.0
FLOPs↓	1×1	2.16×	1.65×	**3.62**×	2.67×	2.0×

alexnet-dec-base

This model is used to show that layer-wise training can reproduce convolutional layers in alexnet-base from scratch. The structure of alexnet-dec-base is the same with alexnet-base and convolutional layers in alexnet-dec-base work as modules. We use the convolutional layer in alexnet-dec-base to learn the one in alexnet-base. We did not fine-tune the network after layer-wise training. The error rate of this model is close to alexnet-base. We visualized the kernels of conv1 in two models, and discovered they were almost the same except that kernels in alexnet-dec-base had more noise. Though noise didn't jeopardize this model, it become complicated when models have less redundancy. So in the following experiments we fine-tune the model after layer-wise training.

alexnet-dec-back and alexnet-dec-front

These two models are used to validate our decomposition scheme in Sect. 2.2. We replace each convolutional layer in alexnet-base with a sequence of two convolutional layers, consisting of d × d kernels and 1 × 1 kernels respectively, and the new model is called alexnet-dec-back. The alexnet-dec-front is similar but its first layer is unchanged. Our alexnet-dec-back model achieved 2.16× FLOPs reduction, with 0.04% increase in the error rate. It proved that decomposition were capable of achieving an acceptable compression rate with little increase in the error rate. Our alexnet-dec-front achieved

1.65× FLOPs reduction, lower than alexnet-dec-back but it achieved 0.10% decrease in the error rate. It proved the existence of redundant parameters.

alexnet-dec-mod

Finally we used the modules in Sect. 2.3. The first convolutional layer is replace with MODULE-A, the second layer is replaced with MODULE-B, and the following layers are replaced with MODULE-C. We try to imitate the structure of Ventral Stream. We achieve 3.62× FLOPs reduction, with 0.11% increase in the error rate (3.62× /+0.11). Comparing to the methods of Kim (2.67× /+1.70) [8] and Figurnov (2× /+2.0) [7], our method has enormous advantages for the error rate and compression rate.

VGG-16

VGG-16 has 15.35G FLOPs, with 10.10% top-5 error rate. Structures of vgg-base (VGG-16) and our vgg-dec-mod model are shown in Table 5. The increase of top-5 error rate and reduction of FLOPs are given in Table 6.

Conv1_1 is unchanged. MODULE-B is used to replace conv1_2, conv2_1, conv2_2, conv3_1, and conv3_2. MODULE-C is used to replace the other convolutional layers. Here MODULE-B is a little different from the prototype. The kernel size of original convolutional layers is 3 × 3. So the convolutional layer with 5 × 5 kernel size in MODULE-B and the layer in front of it are not necessary. Finally we achieve 5.67× FLOPs reduction, with 0.43% increase in top-5 error rate (5.67× /+0.43). Comparing to the methods of Kim (4.93× /+0.50%) [8], Figurnov (1.9× /+2.5%) [7] and Zhang (5× /+1.0%) [5], our method has advantages on both the error rate and compression rate.

Table 5. The structure of vgg-16-base (original VGG-16) and vgg-dec-mod. The meaning of each label is the same with Table 3.

Layer	vgg-16-base	vgg-dec-mod		Layer	vgg-16-base	vgg-dec-mod	
conv1_1	3×3,64	3×3,64		conv1_2	3×3,64	1×1,12	1×1,8
						3×3,24	
						1×1,64	
conv2_1	3×3,128	1×1,24	1×1,8	conv2_2	3×3,128	1×1,36	1×1,16
		3×3,48,%2				3×3,48,%2	
		1×1,128				3×3,512,%2	
conv3_1	3×3,256	1×1,36	1×1,16	conv3_2	3×3,256	1×1,48	1×1,32
		3×3,96,%2				3×3,96,%2	
		1×1,256				1×1,256	
conv3_3	3×3,256	1×1,96		conv4_1	3×3,512	1×1,96	
		3×3,256,%2				3×3,512,%2	
conv4_2	3×3,512	1×1,144		conv4_3	3×3,512	1×1,128	
		3×3,512,%2				3×3,512,%2	
conv5_1	3×3,512	1×1,256		conv5_2	3×3,512	1×1,256	
		3×3,512,%4				3×3,512,%4	
conv5_3	3×3,512	1×1,256					
		3×3,512,%4					

Table 6. Comparison of top-5 error rate increase and FLOPs reduction based on VGG-16.

Model	vgg-16-dec-mod	Kim's [8]	Figurnov's [7]	Zhang's [5]
error↑	**+0.43**	+0.50	+2.5	+1.0
FLOPs↓	**5.67**×	4.93×	1.9×	5×

4 Conclusion

In this work, we propose a layer-wise training algorithm to create efficient convolutional neural networks and achieve better results than previous works, validated by better compression rate, lower error rate and faster convergence rate. Inspiration from visual cortex of brain help us design efficient modules to replace conventional convolutional layers. Besides our method is extremely flexible. It is easy to adopt other module design methods to achieve higher compression rate and lower error rate in the future.

Acknowledgements. This work is supported by the 973 project 2015CB351803, NSFC No. 61572451 and No. 61390514, Youth Innovation Promotion Association CAS CX2100060016, and Fok Ying Tung Education Foundation WF2100060004.

References

1. Krizhevsky, A., Sutskever, I., Hinton, G.: Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems, pp. 1106–1114 (2012)
2. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition (2014). arXiv preprint: arXiv:1409.1556
3. Denton, E., 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)
4. Hinton, G., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network (2015). arXiv preprint: arXiv:1503.02531
5. Zhang, X., Zou, J., He, K., Sun, J.: Accelerating very deep convolutional networks for classification and detection. IEEE Trans. Pattern Anal. Mach. Intell. **38**(10), 1943–1955 (2016)
6. Han, S., Mao, H., Dally, W.J.: Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding (2015). arXiv preprint: arXiv:1510.00149
7. Figurnov, M., Ibraimova, A., Vetrov, D.P., Kohli, P.: PerforatedCNNs: acceleration through elimination of redundant convolutions. In: Advances in Neural Information Processing Systems, pp. 947–955 (2016)
8. 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 (2015). arXiv preprint: arXiv:1511.06530
9. Nicholls, J.G., Martin, A.R., Wallace, B.G., Fuchs, P.A.: From Neuron to Brain. Sinauer Associates, Sunderland (2001)
10. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., et al.: Imagenet large scale visual recognition challenge. Int. J. Comput. Vis. **115**(3), 211–252 (2015)

11. Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., et al.: Caffe: convolutional architecture for fast feature embedding. In: Proceedings of the 22nd ACM International Conference on Multimedia, pp. 675–678 (2014)
12. Chelazzi, L., Miller, E.K., Duncanf, J.: A neural basis for visual search in inferior temporal lobe. Nature **363**(6427), 345–347 (1993)
13. Roe, A.W., Chelazzi, L., Connor, C.E., Conway, B.R., Fujita, I., et al.: Toward a unified theory of visual area V4. Neuron **74**(1), 12–29 (2012)