LETTER

Symmetric Decomposition of Convolution Kernels*

SUMMARY It is a hot issue that speeding up the network layers and decreasing the network parameters in convolutional neural networks (CNNs). In this paper, we propose a novel method, namely, symmetric decomposition of convolution kernels (SDKs). It symmetrically separates \( k \times k \) convolution kernels into \((k \times 1 \text{ and } 1 \times k)\) or \((1 \times k \text{ and } k \times 1)\) kernels. We conduct the comparison experiments of the network models designed by SDKs on MNIST and CIFAR-10 datasets. Compared with the corresponding CNNs, we obtain good recognition performance, with 1.1×-1.5× speedup and more than 30% reduction of network parameters. The experimental results indicate our method is useful and effective for CNNs in practice, in terms of speedup performance and reduction of parameters.

key words: symmetric decomposition, convolution kernels, speedup, reduction of network parameters

1. Introduction

Convolutional neural networks (CNNs) obtain perfect recognition performance in many research areas [1]–[6]. But, there are considerable redundant parameters caused by two-dimensional convolution kernels, which result in a large amount of computational cost and consume much more hardware resources. The factors have become obstacles to speed up performance of CNNs. Hence, for speeding up the application of CNNs, the effective reduction of parameter redundancy is still an open problem.

In this paper, we propose a new method, symmetric decomposition of convolution kernels (SDKs), for reduction of parameter redundancy and speedup performance. The proposed method symmetrically separates \( k \times k \) convolution kernels into \((k \times 1 \text{ and } 1 \times k)\) or \((1 \times k \text{ and } k \times 1)\) kernels in local or whole convolutional (conv) layers, which is effectively used for simplifying CNNs. The experiments show our method can be effectively used to modify the architecture of CNNs and retain good recognition effectiveness with a decrease of network parameters and speedup performance. Overall, we propose a general method to create the net architectures, which achieve the reduction of parameters and speedup of performance without a loss of recognition effectiveness.

The remainder of this paper is organized as follows. In Sect. 2, we review related work. We describe the ideas of SDKs in Sect. 3. In Sect. 4, we conduct the comparison experiments of the models designed by SDKs on MNIST and CIFAR-10 datasets. Finally, we draw conclusions in Sect. 5.

2. Related Work

Convolution operations consume the bulk of processing time for CNNs. Many researchers have exploited a series of techniques in order to reduce the redundant parameters [7]–[11]. Though the approaches achieve a certain effect, there are several problems yet. Firstly, a few methods obtain speedup performance and reduction of parameters with a loss of accuracy. Secondly, some methods are not the universal methods but useful for the certain network architectures. Thirdly, other methods achieve a drop of parameters with good recognition effectiveness, but there is still lots of redundancy in the networks.

Especially in [12], [13], the researchers speed up CNNs with low rank expansions. It separates \( k \times k \) kernels into \( k \times 1 \) or \( 1 \times k \). Moreover, the methods have been applied to the network architectures, e.g., GoogLeNet [14], Inception V3 [15] and V4 [16]. In Inception V3 and V4, the researchers utilize \( n \times 1, 1 \times n \) and \( 1 \times 1 \) convolutions for very good results. But the method reduces not the whole network parameters but the local network parameters. Despite researchers have achieved some promising results in the above works, the reduction of parameter redundancy is still worthy of considerable attention.

3. Symmetric Decomposition of Convolution Kernels

In this section, we propose the symmetric decomposition of convolution kernels for a single conv layer, then generalize this approach to multiple layers. Note that “*” denotes the convolution product operator in this section.

We exploit the SDKs approach which is based on low-rank decomposition [17], [18]. More specifically, we first use a single conv layer as an example, considering a direct implementation of convolution (as shown in Fig. 1(a)). Given a \( S \)-channel input \( X \in \mathbb{R}^{U \times V} \), an output \( Y \in \mathbb{R}^{U' \times V'} \) is computed as

\[
Y = W_n \ast X
\]

(1)

where \( k \) is the spatial size of the conv kernels, \( N \) is the total number of the kernels, and \( W_n \in \mathbb{R}^{k \times k \times S} \) for \( n \in [1 \ldots N] \). Here, \( N \) conv kernels, i.e., \( W_{n} \in \mathbb{R}^{k \times k} \) with \( n \in [1 \ldots N] \)

Jun OU†, Student Member and Yujian LI*, Nonmember

*The authors are with Faculty of Information Technology, Beijing University of Technology, Beijing, 100124, China.
†This work was supported by the National Natural Science Foundation of China under Grant 61876010.

a) E-mail: xhogh@hotmail.com (Corresponding author)
DOI: 10.1587/transinf.2018EDL8136

Copyright © 2019 The Institute of Electronics, Information and Communication Engineers
and $s \in [1 \ldots S]$, operate on each input channel $X'$. However, according to Fig. 1 (b), each conv layer is factored as two regular conv layers with symmetric kernels. The first conv layer has $M$ convolution kernels of size $k \times 1$, i.e., $P_m \in \mathbb{R}^{k \times 1 \times S}; \ m \in [1 \ldots M]$, and yields the output feature maps $X' \in \mathbb{R}^{U' \times V' \times S}$. And the second conv layer has $N$ convolution kernels of size $1 \times k$, i.e., $Q_n \in \mathbb{R}^{1 \times k \times M}; n \in [1 \ldots N]$. Similarly, according to Fig. 1 (c), each conv layer is also decomposed as two conv layers: one conv layer with $M$ convolution kernels of size $1 \times k$, i.e., $P_m \in \mathbb{R}^{1 \times k \times S}; m \in [1 \ldots M]$, and the other one with $N$ convolution kernels of size $k \times 1$, i.e., $Q_n \in \mathbb{R}^{k \times 1 \times M}; n \in [1 \ldots N]$. The corresponding output feature maps are $X' \in \mathbb{R}^{U' \times V' \times M}$. Based on the above two kinds of vector kernels, we attain the approximation of vector-kernel convolution by Eq. (2), which is the sum of vector kernels $Q_n^m \ast P_m$.

$$W_n \ast X = \sum_{s=1}^{S} W_n^s \ast X^s \approx Q_n \ast X' = \sum_{m=1}^{M} Q_n^m \ast X'^m = \sum_{s=1}^{S} \left[ \sum_{m=1}^{M} Q_n^m \ast P_m^s \right] \ast X^s$$

(2)

Furthermore, we summarize the network parameters of the SCL and SCL based on SDKs as follows.

- In the SCL, the number of parameters is $SNk^2$.
- In the SCL based on SDKs, the number of parameters is $SMk$ in the first conv layer, while the corresponding value is $MNk$ in the second conv layer. Therefore, the total number of parameters is $MK(S + N)$. According to the above analysis, we assume that $V \gg k$, i.e., the image width is greatly larger than the kernel size, then $V = V'$. Furthermore, if $M(S + N) \ll NSk$ and let $S, M$ and $N$ be of the same order, the modified model reduces the parameters by about $k$ times in comparison with the SCL. Similarly, the conv layers of the deep networks can be dealt with by our method as the same as the SCL.

4. Experimental Results and Analysis

In this section, for convenience, we offer the abbreviations as follows: “API” for “absolute performance improvement”, “RPI” for “relative performance improvement”, “PM” for “parameters”, “M” for “million”, “Arc” for “Architecture” and “BL” for “baseline”. Here, we apply the SDKs for the Arc 1, Arc 2 and VGG-16 [19] (as shown in Tables 1 and 3). Moreover, we evaluate our method on MNIST and CIFAR-10 datasets and perform the comparative experiments of the above three models. Also, the typical architectures modified by SDKs, i.e., Arcs 1 and 2, are listed in Table 2. Especially for VGG-16 [19], we construct VGG-7 (see Table 3), i.e., one of the models designed by our method. The corresponding experimental results are listed in Tables 4–7.

Using TensorFlow, we perform the comparative experiments on a 64-bit windows operating system with an Intel Xeon E5-2603 v4 1.7GHz Processor, 128GB RAM, and two NVIDIA Tesla K40c graphics cards.

In our experiments, we use the following two datasets.

- The MNIST dataset comprises 60,000 training images and 10,000 testing ones, corresponding to 10 classes. Each image size is $28 \times 28$.
- The CIFAR-10 dataset consists of $32 \times 32$ color images.

![Figure 1](image-url)  
**Fig. 1** Illustration of SDKs. (a) A single conv layer (SCL) with a $S$-channel input. (b) An input image is convolved with $k \times 1$ and $1 \times k$ kernels. (c) An input volume is convolved with $1 \times k$ and $k \times 1$ kernels.
containing 50,000 training images and 10,000 testing ones. It falls into 10 categories: airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck.

In Table 1, there are 3 conv layers, i.e., Conv1, Conv2 and Conv3, 3 pool layers, i.e., Pool1, Pool2 and Pool3, and 1 fully connected layer, i.e., FC1, in Arcs 1 and 2, respectively. More concretely, in Table 1 Conv1 means a conv layer with \(N\) kernels of size \(6 \times 6\) for Arc 1 (\(N = 24\)). Also, it means a conv layer with \(N\) kernels of size \(5 \times 5\) for Arc 2 (\(N = 64\)). Meanwhile, Pool1 stands for \(2 \times 2\) pooling.

As shown in Table 2, \(X\) denotes the number of the conv layers. For example, in Arc 1, Conv1_X means two conv layers, i.e., Conv1, with \(M\) kernels of size \(1 \times 6\) and Conv1, with \(N\) kernels of size \(6 \times 1\) (\(M = N = 24\)). Meanwhile, in Arc 2, Conv1_X means two conv layers, i.e., Conv1, with \(M\) kernels of size \(1 \times 5\) and Conv1, with \(N\) kernels of size \(5 \times 1\) (\(M = N = 64\)).

In Table 3, we show the main difference between VGG_7 and VGG-16. Here, \(X\) also indicates the number of the conv layers. To be specific, Conv1_X means 4 conv layers for VGG_7 and 2 conv layers for VGG-16. That is to say, in VGG_7 Conv1_X is composed of Conv1_1 with \(M\) kernels of size \(3 \times 1\), Conv1_2 with \(N\) kernels of size \(1 \times 3\), Conv1_3 with \(M\) kernels of size \(3 \times 1\) and Conv1_4 with \(N\) kernels of size \(1 \times 3\) (\(M = N = 64\)). Meanwhile, in VGG-16 Conv1_X includes Conv1_1 with \(N\) kernels of size \(3 \times 3\) and Conv1_2 with \(N\) kernels of size \(3 \times 3\) (\(N = 64\)).

Table 4 Comparison of Arcs 1 and 2 on MNIST and CIFAR-10 (C10), respectively.

Table 5 Comparison of Arc1, Arc 1_1, Arc 1_2, Arc 2_8, Arc 2_2, Arc 2_1 and Arc 2_2.

Table 6 Comparison of VGG-16 and VGG_X with \(X \in [1, 7]\) on CIFAR-10.

Table 7 Comparison of VGG-1, VGG-2, VGG-7 and VGG-16 on CIFAR-10.
As shown in Table 5, Arc 1.8 outperforms Arc 1, i.e. the standard CNNs, with about $1.3\times$ speedup and approximately 32% decrease of network parameters. Moreover, it attains more than 10% RPI and up to $1.21\times$ speedup in comparison with the other decomposed models, e.g., Arc 1.1 and 1.2. Similarly, compared with Arc 2 on CIFAR-10, Arc 2.8 attains about 3% API, near $1.5\times$ speedup and 36.4% reduction of parameters. Meanwhile, it achieves about $1.2\times$ speedup and more than 11% RPI in comparison to Arc 2.1 or Arc 2.2.

In Tables 6 and 7, we find VGG-16 designed by SDks is not only superior to these models modified by other decomposition methods but also superior to VGG-16. For instance, VGG $X \ (X \in [3, 7])$ outperforms VGG_1 or VGG_2 in terms of time (H) and accuracy. Meanwhile, VGG $X \ (X \in [3, 7])$ attains a speedup of $1.06\times$-$1.11\times$ and a reduction of network parameters by one-third in comparison to VGG-16. Especially for VGG_7, it not only attains 3-5% RPI and more than $1\times$ speedup in comparison to VGG_1 or VGG_2 but also obtains a slight gain in term of accuracy compared with VGG-16.

Based on the above experimental results, we see the models with the local modified conv layers by SDks (MLM) usually have lower recognition performance than these with the whole modified conv layers by SDks (MWM). That is to say, there are one or more conv layers with a pair of symmetric kernels in MLM, e.g., $1 \times k$ and $k \times 1$ kernels or $k \times 1$ and $1 \times k$ kernels. Furtherly, each conv layer has a pair of kernels in MWM, i.e., $1 \times k$ and $k \times 1$ kernels or $k \times 1$ and $1 \times k$ kernels. For example, in Table 4, Arcs 2.4 and 2.5 belong to MLM and Arcs 1.8, 2.7 and 2.8 belong to MWM. Meanwhile, in Table 6 VGG_6 belongs to MLM and VGG_7 belongs to MWM.

Although the MLM have an increasing accuracy with an increment of number of the modified conv layers, they have not always the corresponding decrease of training time. For example, it takes about 1.42 hours for Arc 2.5 while 1.4 hours for Arc 2.4. The former has two designed conv layers but the latter has only one. However, as for MWM, they always have good recognition performance and remarkable reduction of parameters, e.g., Arc 1.8, Arc 2.8 and VGG_7.

5. Conclusion

We have presented a novel method, i.e., SDks, for reducing the network parameters and speeding up the processing time of network layers. It replaces convolution kernels with $(k \times 1$ and $1 \times k$) or $(1 \times k$ and $k \times 1$) kernels. Our method can be used for designing local or whole conv layers. Normally, the nets designed by SDks achieve speedup. Also, the nets with the whole modified conv layers obtain better recognition than those with the local modified conv layers. Overall, the models designed by SDks obtain speedup performance and a decrease of network parameters, with a good recognition accuracy. Therefore, it is a universal and useful method in practice.

References

[1] Q. Jing, J. Zhu, and C. Chen, “Very Fast Semantic Image Segmentation Using Hierarchical Dilation and Feature Refining,” Cognitive Computation, vol.10, no.1, pp.62–72, 2018.

[2] L. Mou, P. Ghamisi, and X.X. Zhu, “Unsupervised spectral-spatial feature learning via deep residual conv-deconv network for hyperspectral image classification,” IEEE Trans. Geosci. Remote Sens., vol.56, no.1, pp.391–406, Jan. 2018.

[3] V. Badrinarayanan, A. Kendall, and R. Cipolla, “Segnet: A deep convolutional encoder-decoder architecture for image segmentation,” IEEE Trans. Pattern Anal. Mach. Intell., vol.39, no.12, pp.2481–2495, 2017.

[4] B.-B. Gao, C. Xing, C.-W. Xie, J. Wu, and X. Geng, “Deep label distribution learning with label ambiguity,” IEEE Trans. Image Process., vol.26, no.6, pp.2825–2838, 2017.

[5] T. Tan, Y. Qian, H. Hu, Y. Zhou, W. Ding, and K. Yu, “Adaptive very deep convolutional residual network for noise robust speech recognition,” IEEE/ACM Transactions on Audio, Speech, and Language Processing, vol.26, no.8, pp.1393–1405, Aug. 2018.

[6] S. Zhang, S. Zhang, T. Huang, and W. Gao, “Speech Emotion Recognition Using Deep Convolutional Neural Network and Discriminant Temporal Pyramid Matching,” IEEE Trans. Multimedia, vol.20, no.6, pp.1576–1590, June 2018.

[7] Y. He, X. Zhang, and J. Sun, “Channel pruning for accelerating very deep neural networks,” Proc. Int. Conf. Comput. Vis., pp.1398–1406, 2017.

[8] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, “Learning efficient convolutional networks through network slimming,” Proc. Int. Conf. Comput. Vis., pp.2755–2763, 2017.

[9] X. Zhang, J. Zou, K. He, and J. Sun, “Accelerating very deep convolutional networks for classification and detection,” IEEE Trans. Pattern Anal. Mach. Intell., vol.38, no.10, pp.1943–1955, 2016.

[10] S. Han, J. Pool, J. Tran, and W. Dally, “Learning both weights and connections for efficient neural network,” Advances in Neural Information Processing Systems, pp.1135–1143, 2015.

[11] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, “Learning efficient convolutional networks through network slimming,” Proc. IEEE Int. Conf. Comput. Vis., pp.2755–2763, 2017.

[12] J. Yue, J. Li, and Y. Gong, “Restructuring of deep neural network acoustic models with singular value decomposition,” Interspeech, pp.2365–2369, Aug. 2013.

[13] E.L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, “Exploiting linear structure within convolutional networks for efficient evaluation,” Proc. Adv. Neural Inf. Process. Syst., pp.1269–1277, 2016.

[14] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” Proc. Comput. vis. pattern recognit., pp.1–9, June 2015.

[15] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Re-thinking the inception architecture for computer vision,” Proc. Computer vision and pattern recognition (CVPR), pp.2818–2826, 2016.

[16] C. Szegedy, S. Ioffe, V. Vanhoucke, and A.A. Alemi, “Inception-v4, inception-resnet and the impact of residual connections on learning,” Proc. AAAI, vol.4, p.12, Feb. 2017.

[17] M. Jaderberg, A. Vedaldi, and A. Zisserman, “Speeding up convolutional neural networks with low rank expansions,” arXiv preprint arXiv:1405.3866, 2014.

[18] A. Sironi, B. Tekin, R. Rigamonti, V. Lepetit, and P. Fua, “Learning separable filters,” IEEE Trans. Pattern Anal. Mach. Intell., vol.37, no.1, pp.94–106, 2015.

[19] Simonyan, Karen, and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” arXiv preprint arXiv:1409.1556, 2014.