Accelerating Convolutional Neural Networks for Mobile Applications

Peisong Wang
National Laboratory of Pattern Recognition,
CASIA, Beijing, China
University of Chinese Academy of Sciences,
Beijing, China
peisong.wang@nlpr.ia.ac.cn

Jian Cheng
National Laboratory of Pattern Recognition,
CASIA, Beijing, China
University of Chinese Academy of Sciences,
Beijing, China
CAS Center for Excellence in Brain Science and
Intelligence Technology
jcheng@nlpr.ia.ac.cn

ABSTRACT

Convolutional neural networks (CNNs) have achieved remarkable performance in a wide range of computer vision tasks, typically at the cost of massive computational complexity. The low speed of these networks may hinder real-time applications especially when computational resources are limited. In this paper, an efficient and effective approach is proposed to accelerate the test-phase computation of CNNs based on low-rank and group sparse tensor decomposition. Specifically, for each convolutional layer, the kernel tensor is decomposed into the sum of a small number of low multilinear rank tensors. Then we replace the original kernel tensors in all layers with the approximate tensors and fine-tune the whole net with respect to the final classification task using standard backpropagation.
Comprehensive experiments on ILSVRC-12 demonstrate significant reduction in computational complexity, at the cost of negligible loss in accuracy. For the widely used VGG-16 model, our approach obtains a 6.6× speed-up on PC and 5.91× speed-up on mobile device of the whole network with less than 1% increase on top-5 error.

Keywords

Convolutional Neural Networks; Acceleration; Image Classification; Tensor Decomposition

1. INTRODUCTION

In recent years, convolutional neural networks (CNNs) is continuously setting new state-of-the-art performance in computer vision tasks such as image classification and object detection. However, to a certain extent these breakthroughs have become possible through adding layers and increasing the size of feature maps. One drawback of the resulting networks is that the computation time increases significantly at both training and testing phase. Thanks to specialized hardwares like NVIDIA GPUs and CPU clusters, these CNN models can be trained offline within a relatively affordable time. But the real-world systems may still suffer from long testing time, especially for real-time but computation-limited systems, such as smart phones and automated vehicles. Therefore, it is of central importance to speed up test-phase computation of CNN models.

There have been a banch of studies for CNN model acceleration. One of the most widely used techniques is low-rank approximation. Many low-rank based approaches [3, 6] have been investigated since Denil et al.'s work [2] on reducing redundancy of CNN models. Among these studies, Zhang et al.'s work [18] achieved 3.8× speed-up on VGG-16 model. Low-rank tensor decomposition based methods have also been developed. Lebedev et al. [11] utilized CP-decomposition to approximate the 4D convolutional kernel tensor. But they only handled a single layer for AlexNet. Kim et al. [8] proposed to use Tucker decomposition to reduce the computational complexity of CNN models. Network pruning [4, 12] is another widely used technique for CNN compression and acceleration. The main idea of these methods is to remove low-saliency parameters or small-weight connections [4]. These methods achieve significant theoretical speed-up ratio, but the resulting unstructured sparse connections do not fit well in parallel computation. Beyond the methods mentioned above, many other approaches are proposed. Wu et al. [17] proposed to use product quantization for CNN compression and acceleration at the same time. FFT-based method was also investigated in [13].

In this paper, we propose an acceleration method for CNN models based on tensor low-rank and group sparse decomposition. The resulting architecture gains efficiency from both low-rank and sparsity. We evaluate our proposed method on one of the largest image classification benchmark of ILSVRC-12 [14]. Our approach achieves 6.6× speed-up with less than 1% drop in top-5 classification accuracy. Our contributions can be summarized as follows:

- We propose a tensor Block Term Decomposition (BTD) based acceleration method for convolutional neural networks, which benefits from both low-rank and group sparsity.
- A new alternative least square method, i.e., Principle Component Iteration (PCI), is proposed for efficient tensor block term decomposition.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

MM '16, October 15-19, 2016, Amsterdam, Netherlands
© 2016 ACM. ISBN 978-1-4503-3603-1/16/10...$15.00
DOI: http://dx.doi.org/10.1145/2964284.2967280

- Our method achieves significant speed-up for CNN models. For the widely used VGG-16 [15] model, we obtain 6.6× actual speed-up on PC and 5.91× speed-up on smart phone with less than 1% drop on top-5 classification accuracy.

2. PROPOSED METHOD

Throughout this paper vectors are denoted boldface lowercase letters, e.g., $\mathbf{x}$. Boldface capital letters, e.g., $\mathbf{X}$, are utilized to represent matrices. Euler script letters are used to denote higher-order tensors, e.g., $\mathcal{X}$. The symbol $\times_n$ represent n-mode product of a tensor with a matrix [9].

2.1 Low-rank and Group Sparse Tensor Approximation

In CNN models, the convolutional kernel is a 4-way tensor of size W×H×S×T, where W/H is the width/height of the kernel and S/T is the number of input/output channels. For simplicity, we combine width/height together and denote the spatial dimension as P(=WH). Thus the kernel tensor becomes a 3D tensor. The proposed method exploits low-rank and group sparse decomposition on the 3D convolutional kernel tensor.

Our assumption is twofold. First, we assume that the learned kernel tensors are low rank along the input and output channels. The resulting architecture is called *bottleneck* [5]. Bottleneck architecture consists of three layers where the first 1×1 convolution is used to reduce input channel dimension, the last 1×1 convolution layer is responsible for restoring output channel dimension, and the middle layer is a regular W×H convolution but with smaller input/output dimensions.

We also assume that connections of the second layer of the bottleneck architecture is sparse in group. The resulting architecture is a layer with multiple convolutions which is (channel-wise) locally connected. We refer to this architecture as *group* like in the implementation of Caffe [7].

The approach presented here benefits from both low-rank and group sparsity. The method proposed has the form of tensor block term decomposition [1]. Thus we denote our acceleration method as BTD.

2.2 Block Term Decomposition on Convolutional Kernels

Throughout this paper we represent convolutional kernels as a 3D tensor $\mathcal{T} \in \mathbb{R}^{S \times T \times P}$, where S and T are the numbers of input and output channels and P is the spatial dimension, e.g., for a 3×3 kernel, P equals to 9.

The basic idea of tensor block term decomposition is to approximate a tensor as a sum of several low-rank tensors. Each low-rank tensor is in the form of a core tensor multiplied by a factor matrix along each dimension. For the convolutional kernel tensor $\mathcal{T}$, the block term decomposition is

$$\mathcal{T} \approx \sum_{r=1}^{R} \mathcal{G}_r \times_1 \mathbf{A}_r \times_2 \mathbf{B}_r \times_3 \mathbf{C}_r \qquad (1)$$

where $\mathcal{G}_r \in \mathbb{R}^{s \times t \times p}$ is the core tensor of r-th subtensor and $\mathbf{A}_r \in \mathbb{R}^{S \times s}$, $\mathbf{B}_r \in \mathbb{R}^{T \times t}$ and $\mathbf{C}_r \in \mathbb{R}^{P \times p}$ are factor matrices along each dimension. Here the lower-case letter s, t and p represent the rank in each dimension respectively.

In our method, we don't exploit the low rank of spatial dimension, thus we utilize the decomposition of the following form:

$$\mathcal{T} \approx \sum_{r=1}^{R} \mathcal{G}_r \times_1 \mathbf{A}_r \times_2 \mathbf{B}_r \qquad (2)$$

where $\mathcal{G}_r \in \mathbb{R}^{s \times t \times P}$ is the core tensor of r-th subtensor and $\mathbf{A}_r \in \mathbb{R}^{S \times s}$ and $\mathbf{B}_r \in \mathbb{R}^{T \times t}$ are factor matrices along input and output channel dimensions respectively. We can think that the factor matrix along spatial dimension in (2) is an identity matrix. The decomposition is visualized in Figure 1.

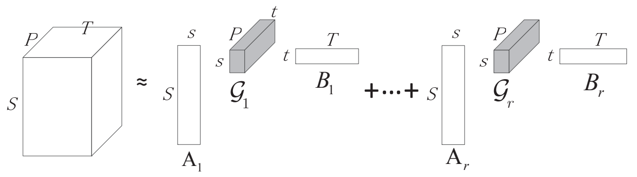

Figure 1: Visualization of tensor block term decomposition.

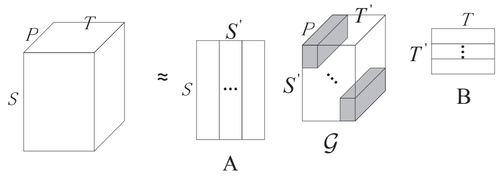

Figure 2: Visualization of tensor block term decomposition with a group sparse core tensor.

The next section will discuss how to solve this decomposition problem efficiently. If the decomposition is known, each component of the subtensors can be concatenated, i.e., concatenating $\mathbf{A}_1, \cdots, \mathbf{A}_r$ into a big matrix $\mathbf{A}$, and $\mathbf{B}_1, \cdots, \mathbf{B}_r$ into a big matrix $\mathbf{B}$, and $\mathcal{G}_1, \cdots, \mathcal{G}_r$ into a big core tensor $\mathcal{G}$. Here the resulting big core tensor $\mathcal{G}$ is group sparse, i.e., non-zero values are along the diagonal blocks, as shown in Figure 2. The matrices $\mathbf{A}$ and $\mathbf{B}$ correspond to the first and last 1×1 convolutional layers of the bottleneck architecture, and the core tensor $\mathcal{G}$ is the second convolution kernel of size $W \times H$ which can be constructed using multiple groups. Figure 3 shows how convolution layers can be replaced by new layers for speed-up.

2.3 Principle Component Iteration

Tensor decomposition can be seen as a generalization of matrix decomposition to multidimensional case and has been actively studied in recent years. Many approaches are proposed for tensor decomposition, however, few of them consider the block term decomposition. An intuitive method for solving the decomposition problem of (1) is in the greedy way, e.g., greedily finding a best Tucker approximation of

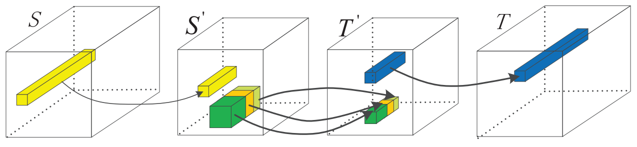

Figure 3: Block term decomposition used for speeding-up convolution.

the residual tensor. But this greedy method may suffer from high approximation error. In our experiment we find that a minor increase of the approximation error may result in a huge drop on the final accuracy.

The authors of [1] gives an alternating least squares (ALS) algorithms. But in our experiments, we find that this method has a large memory consumption and runs quite slow, which is prohibitive for practical use. Nonlinear least squares (NLS) algorithms are also quite slow.

In this section, a much more efficient algorithm called Principle Component Iteration (PCI) is proposed, which is a generalization of higher order orthogonal iteration (HOOI) [9] used in the Tucker decomposition. The basic idea of PCI is to find the principle component (a subtensor) of residual tensor approximated using other components at every iteration. PCI is much more efficient than existing methods. For example, the decomposition time of a single layer for VGG-16 model is more than ten hours using NLS algorithm, while PCI can achieve comparable approximation error within half an hour. The details of our proposed PCI method is presented in Algorithm 1.

Algorithm 1 Principle Component Iteration

```
1: procedure PCI($\mathcal{T}$)
2:     initialize $\mathcal{T}_r \leftarrow 0$ for $r = 1, \cdots, R$
3:     repeat
4:         for $r = 1$ to $R$ do
5:             $\mathcal{T}_{res} \leftarrow \mathcal{T} - \mathtt{sum}(\mathcal{T}_1, \cdots, \mathcal{T}_{r-1}, \mathcal{T}_{r+1}, \cdots, \mathcal{T}_R)$
6:             $\mathcal{T}_r \leftarrow \mathtt{HOOI}(\mathcal{T}_{res})$
7:         end for
8:         $\mathcal{T}_{res} \leftarrow \mathcal{T} - \mathtt{sum}(\mathcal{T}_1, \cdots, \mathcal{T}_R)$
9:     until $\mathcal{T}_{res}$ ceases to decrease or maximum iterations reached
10:    return $\mathcal{T}_1, \cdots, \mathcal{T}_R$
11: end procedure
```

2.4 Complexity Analysis

For every spatial position and every convolutional layer of the original networks, the normal convolution requires $S \times T \times W \times H$ multiplication-addition operations. While in our method, the required operations of the corresponding three layers (note that we replace the original convolutional layer with three convolutional layers) are $S \times S'$, $S' \times T' \times W \times H/R$, and $T \times T'$ respectively, where R represents the number of groups. Thus the theoretical speed-up ratio is given by:

$$ratio = \frac{STWH}{SS' + S'T'WH/R + TT'} \qquad (3)$$

3. EXPERIMENTS

In this section, we evaluate our approach on ILSVRC-12 [14] image classification benchmark. Experiments are conducted on two of the mostly used CNN models, i.e., AlexNet [10] and VGG-16 [15]. Single-layer performance for AlexNet is firstly evaluated using our approach. Based on the experimental results, we further show the power of our method on whole-model acceleration of very deep VGG model [15]. Through out this section, we use Berkeley's *Caffe* [7] routines for our experiments. We utilize Intel MKL and Eigen3 library for PC and smart phone respectively.

3.1 Speed-Up Results for AlexNet

In this subsection, we evaluate the acceleration performance on AlexNet. We process the second convolutional layer of AlexNet, as in [3, 6, 11, 17], because the second convolution is the most time-consuming layer during test-phase. Table 1 shows the comparison with existing methods. All results in Table 1 are the best results reported by the authors in their papers except for [6], which is reproduced by [11]. For our method, we set the selected rank $S' = 40, T' = 120$ and the group number R is set to be 20.

Table 1: Comparison with other methods for the second convolutional layer of AlexNet.

Method	Speed-up	Top-5 Err. ↑
Biclustering + SVD [3]	2.7×	0.95%
CP Decomposition [11]	4.5×	1.22%
Q-CNN [17]	6.1×	0.31%
Low rank [6]	6.6×	≈1%
BTD	8.1×	0.98%

From Table 1 we can conclude that with about 1% drop in accuracy, our method achieves more than 8× speed-up, which is much higher than other methods.

3.2 Whole-Model Acceleration for VGG-16

In this subsection, we evaluate our approach on VGG-16 model for ImageNet classification. We mainly focus on the convolutional layers in our experiments. But our approach can be directly combined with other approaches e.g., the product quantization method [17] or SVD used in [3, 18]. We report actual running time and speed-up ratio for each layer as well as for the whole VGG-16 model in Table 2. All results in this subsection are conducted on personal computer with i7 CPU (3.4GHz). We don't process the first convolutional layer because of the small complexity.

As can been seen from Table 2, our method can achieves about 6 $\sim$ 13× CPU speed-up. Our method obtains 6.6× whole-model speed-up (column denoted by "Whole") even without accelerating the fully connected layers at the cost of 1% drop in accuracy. Note that the first fully connected layer can be treated as convolution with large filters. Thus we can further improve speed-up ratio to 7.4× (last column denoted by "Whole*"), which is almost twice as much of the best existing method as shown in Table 3. Note that the theoretical speed-up ratios shown in Table 3 are calculated using the multiplication-addition operations of all convolutional layers before and after accelerating. Non-convolutional layers, such as pooling layers, are also time-consuming. Thus the actual speed-up ratios are much lower than the theoretical ones, especially at higher speed-up ratios.

Comparison with current state-of-the-art methods are illustrated in Table 3. All results are the best results reported by the authors except for the method used in [6], which is the result reproduced by the authors of [18]. The different theoretical speed-up ratio are got by setting different hyper-parameters, for example the rank used by these methods.

From Table 3, we find that our method outperforms all the rest methods significantly. Even without accelerating fully connected layers, our method can still achieves 6.6× actual speed-up, which is much higher compared to the 3.8× acceleration of [18].

Table 2: Performance of accelerating VGG-16 model. Runing time is in milliseconds for a single view on CPU using Intel MKL (single thread, with SSE).

Layers	$C1_1$	$C1_2$	$C2_1$	$C2_2$	$C3_1$	$C3_2$	$C3_3$	$C4_1$	$C4_2$	$C4_3$	$C5_1$	$C5_3$	$C5_3$	Whole	Whole*
VGG-16	10.9	200.4	86.4	171.2	79.0	159.7	159.9	76.9	154.1	153.6	42.8	42.8	42.7	1442.0	1442.0
BTD	10.95	33.25	9.46	18.87	7.58	12.41	14.01	6.20	10.40	13.00	6.64	6.63	6.62	218.33	194.88
Speed-up	1×	6.0×	9.1×	9.0×	10.4×	12.9×	11.4×	12.4×	14.8×	11.8×	6.4×	6.5×	6.5×	6.6×	7.4×

Table 3: Comparison with other methods for whole-model acceleration of VGG-16. BTD*/BTD is obtained with/without processing the fully connected layer. Different theoretical speed-up ratios result from different hyper-parameters

Method	Theoretical	Actual	Top-5 Err. ↑
Zhang [18]	4×	3.8×	0.3%
	5×	-	1.0%
Jaderberg [6]	3×	-	2.3%
	4×	3.8×	9.7%
	5×	-	29.7%
Tai [16]	3×	2.1×	0.3%
Q-CNN [17]	4×	-	0.5%
BTD	12.1×	6.6×	1.0%
BTD*	12.2×	7.4×	1.3%

3.3 Results on Mobile Device

In this subsection, we evaluate the performance of our method on smart phone. Results are conducted on Huawei Mate 7 which is equipped with a 1.8GHz Kirin 925 CPU. We adapt the code of Caffe onto Android and use Eigen3 library for matrix operations. We use the same accelerated model as in previous subsection. Thus the accuracy drop for mobile devices is the same as reported in previous subsection. The results are shown in Table 4. Speed-up ratio are based on runing time of VGG-16 on smart phone using single thread.

Table 4: Comparison on time consumption of the original and the accelerated VGG-16 model on smart phone.

Model	#threads	Time (s)	speed-up
VGG-16	1	10.52	1×
	4	4.29	2.45×
BTD	1	1.78	5.91×
	4	0.96	10.95×

From Table 4 we can see that our model runs much faster than the original VGG-16 model. The actual speed-up is 5.91× using single thread.

4. CONCLUSIONS

In this paper, we propose an acceleration method for CNN models based on tensor low-rank and group sparse decomposition. The resulting architecture gains efficiency from both low-rank and sparsity that cannot be matched by either method. We also propose a new alternative least square method, i.e., priciple componant iteration, for tensor block term decomposition, which is much more efficient. Comprehensive experiments conducted on ILSVRC-12 benchmark [14] demonstrate significant reduction in computational complexity, at the cost of negligible loss in accuracy.

Acknowledgements

This work was supported in part by National Natural Science Foundation of China (No.61332016) and Beijing Brain Science Research Project (No.Z161100000216146).

5. REFERENCES

[1] L. De Lathauwer and D. Nion. Decompositions of a higher-order tensor in block terms-part iii: Alternating least squares algorithms. *SIAM journal on Matrix Analysis and Applications*, 30(3):1067–1083, 2008.

[2] M. Denil, B. Shakibi, L. Dinh, N. de Freitas, et al. Predicting parameters in deep learning. In *Advances in Neural Information Processing Systems*, pages 2148–2156, 2013.

[3] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in Neural Information Processing Systems*, pages 1269–1277, 2014.

[4] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, pages 1135–1143, 2015.

[5] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

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

[7] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the ACM International Conference on Multimedia*, pages 675–678. ACM, 2014.

[8] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. *arXiv preprint arXiv:1511.06530*, 2015.

[9] T. G. Kolda and B. W. Bader. Tensor decompositions and applications. *SIAM review*, 51(3):455–500, 2009.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[11] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *arXiv preprint arXiv:1412.6553*, 2014.

[12] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Pensky. Sparse convolutional neural networks. In

IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 806–814, 2015.

[13] M. Mathieu, M. Henaff, and Y. LeCun. Fast training of convolutional networks through ffts. *arXiv preprint arXiv:1312.5851*, 2013.

[14] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[15] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[16] C. Tai, T. Xiao, X. Wang, et al. Convolutional neural networks with low-rank regularization. *arXiv preprint arXiv:1511.06067*, 2015.

[17] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng. Quantized convolutional neural networks for mobile devices. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[18] X. Zhang, J. Zou, K. He, and J. Sun. Accelerating very deep convolutional networks for classification and detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2015.