CONVOLUTIONAL NEURAL NETWORKS WITH LOW-RANK REGULARIZATION

Cheng Tai[1], **Tong Xiao**[2], **Yi Zhang**[3], **Xiaogang Wang**[2], **Weinan E**[1]
[1]The Program in Applied and Computational Mathematics, Princeton University
[2]Department of Electronic Engineering, The Chinese University of Hong Kong
[3]Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor
{chengt,weinan}@math.princeton.edu; yeezhang@umich.edu
{xiaotong,xgwang}@ee.cuhk.edu.hk

ABSTRACT

Large CNNs have delivered impressive performance in various computer vision applications. But the storage and computation requirements make it problematic for deploying these models on mobile devices. Recently, tensor decompositions have been used for speeding up CNNs. In this paper, we further develop the tensor decomposition technique. We propose a new algorithm for computing the low-rank tensor decomposition for removing the redundancy in the convolution kernels. The algorithm finds the exact global optimizer of the decomposition and is more effective than iterative methods. Based on the decomposition, we further propose a new method for training low-rank constrained CNNs from scratch. Interestingly, while achieving a significant speedup, sometimes the low-rank constrained CNNs delivers significantly better performance than their non-constrained counterparts. On the CIFAR-10 dataset, the proposed low-rank NIN model achieves 91.31% accuracy (without data augmentation), which also improves upon state-of-the-art result. We evaluated the proposed method on CIFAR-10 and ILSVRC12 datasets for a variety of modern CNNs, including AlexNet, NIN, VGG and GoogleNet with success. For example, the forward time of VGG-16 is reduced by half while the performance is still comparable. Empirical success suggests that low-rank tensor decompositions can be a very useful tool for speeding up large CNNs.

1 INTRODUCTION

Over the course of three years, CNNs have revolutionized computer vision, setting new performance standards in many important applications, see e.g., Krizhevsky et al. (2012); Farabet et al. (2013); Long et al. (2014). The breakthrough has been made possible by the abundance of training data, the deployment of new computational hardware (most notably, GPUs and CPU clusters) and large models. These models typically require a huge number of parameters ($10^7 \sim 10^9$) to achieve state-of-the-art performance, and may take weeks to train even with high-end GPUs. On the other hand, there is a growing interest in deploying CNNs to low-end mobile devices. On such processors, the computational cost of applying the model becomes problematic, let alone training one, especially when real-time operation is needed. Storage of millions of parameters also complicates the deployment. Modern CNNs would find many more applications if both the computational cost and the storage requirement could be significantly reduced.

There are only a few recent works for speeding up CNNs. Denton et al. (2014) proposed some low-rank approximation and clustering schemes for the convolutional kernels. They achieved 2x speedup for a single convolutional layer with 1% drop in classification accuracy. Jaderberg et al. (2014) suggested using different tensor decomposition schemes, reporting a 4.5x speedup with 1% drop in accuracy in a text recognition application. Lebedev et al. (2014) further explored the use of CP decomposition to approximate the convolutional kernels. Vanhoucke et al. (2011) showed that using 8-bit quantization of the parameters can result in significant speedup with minimal loss of accuracy. This method can be used in conjunction with low-rank approximations to achieve further speedup.

As convolution operations constitute the bulk of all computations in CNNs, simplifying the convolution layer would have a direct impact on the overall speedup. The convolution kernels in a typical CNN is a 4D tensor. The key observation is that there might be a significant amount of redundancy in the tensor. Ideas based on tensor decomposition seem to be a particularly promising way to remove the redundancy as suggested by some previous works.

In this paper, we further develop the tensor decomposition idea. Our method is based on Jaderberg et al. (2014), but has several significant improvements. The contributions are summarized as follows:

- A new algorithm for computing the low-rank tensor decomposition. Low-rank tensor decompositions are non-convex problems and difficult to compute in general, Jaderberg et al. (2014) use iterative schemes to get an approximate local solution. But we find that the particular form of low-rank decomposition in (Jaderberg et al., 2014) has an exact closed form solution which is the global optimum. Hence we obtain the best data-independent approximation. Furthermore, computing the exact solution is much more effective than iterative schemes. As the tensor decomposition is the most important step in approximating CNNs, being able to obtain an exact solution efficiently thus provides great advantages.

- A new method for training low-rank constrained CNNs from scratch. Most previous works only focus on improving testing time computation cost. This is achieved by approximating and fine-tuning a pre-trained network. Based on the low-rank tensor decomposition, we find that the convolutional kernels can be parameterized in a way that naturally enforces the low-rank constraint. Networks parameterized in this low-rank constrained manner have more layers than their non-constrained counterparts. While it is widely observed that deeper networks are harder to train, we are able to train very deep low-rank constrained CNNs with more than 30 layers with the help of a recent training technique called *batch normalization* Ioffe & Szegedy (2015).

- Evaluation on large networks. Previous experiments in Jaderberg et al. (2014) and Denton et al. (2014) give some promises of the effectiveness of low-rank approximations. But these methods have not been tested extensively for large models and generic datasets. Moreover, as iterative methods are used to find the approximation, bad local minima may hurt performance. In this paper, we test the proposed method for various state-of-the-art CNN models, including NIN (Lin et al., 2013), AlexNet (Krizhevsky et al., 2012), VGG (Simonyan & Zisserman, 2014) and GoogleNet (Szegedy et al., 2014). The datasets used include CIFAR-10 and ILSVRC12. We achieved significant speedups for these models with comparable or even better performance. Success on a variety of CNN models give strong evidence that low-rank tensor decomposition can be a very useful tool for simplifying and improving deep CNNs.

Our numerical experiments show that significant speedup can be achieved with minimal loss of performance, which is consistent with previously reported results. Surprisingly, while all previous efforts report a slight decrease or no change in performance, we found a significant increase of classification accuracy in some cases. In particular, on the CIFAR-10 dataset, we achieve 91.31% classification accuracy (without data augmentation) with the low-rank NIN model, which improves upon not only the original NIN but also upon state-of-the-art results on this dataset. We are not aware of significant improvements with low-rank approximations being reported in the previous literature.

The rest of the paper is organized as follows. We discuss some related work in section 2. We then introduce our decomposition scheme in section 3. Results with typical networks including AlexNet, NIN, VGG and GoogleNet on CIFAR10 and ILSVRC12 datasets are reported in section 4. We conclude with the summary and discussion in Section 5.

2 Related Work

Using low-rank filters to accelerate convolution has a long history. Classic examples include high dimensional DCT and wavelet systems constructed from 1D wavelets using tensor products. In the context of dictionary learning, learning separable 1D filters was suggested by Rigamonti et al. (2013).

More specific to CNNs, there are two works that are most related to ours: Jaderberg et al. (2014); Lebedev et al. (2014). For Jaderberg et al. (2014), in addition to the improvements summarized in the previous section, there is another difference in the approximation stage. In Jaderberg et al. (2014), the network is approximated layer by layer. After one layer is approximated by the low-rank filters, the parameters of that layer are fixed, and the layers above are fine-tuned based on a reconstruction error criterion. Our scheme fine-tunes the entire network simultaneously using a discriminative criterion. While Jaderberg et al. (2014) reported that discriminative fine-tuning was inefficient for their scheme, we found that it works very well in our case.

In Lebedev et al. (2014), CP decomposition of the kernel tensors is proposed. Lebedev et al. (2014) used non-linear least squares to compute the CP decomposition. It is also based on the tensor decomposition idea, but our decomposition is based on a different scheme and has some numerical advantages. For the CP decomposition, finding the best low-rank approximation is an ill-posed problem, and the best rank-K approximation may not exist in the general case, regardless the choice of norm (de Silva & Lim, 2008). But for the proposed scheme, the decomposition always exists, and we have an exact closed form solution for the decomposition. In principle, both the CP decomposition scheme and the proposed scheme can be used to train CNNs from scratch. In the CP decomposition, one convolutional layer is replaced with four convolutional layers. Although the effective depth of the network remains the same, it makes optimization much harder as the gradients of the inserted layers are prone to explosion. Because of this, application of this scheme to larger and deeper models is still problematic due to numerical issues.

Lastly, different from both, we consider more and much larger models, which is more challenging. Thus our results provide strong evidence that low-rank approximations can be applicable to a variety of state-of-the-art models.

3 METHOD

In line with the method in Jaderberg et al. (2014), the proposed tensor decomposition scheme is based on a conceptually simple idea: replace the 4D convolutional kernel with two consecutive kernels with a lower rank. In the following, we introduce the details of the decomposition and the algorithms of using the decomposition to approximate a pre-trained network and to train a new one.

3.1 APPROXIMATION OF A PRE-TRAINED CNN

Formally, a convolutional kernel in a CNN is a 4D tensor $\mathcal{W} \in \mathbb{R}^{N \times d \times d \times C}$, where N, C are the numbers of the output and input feature maps respectively and d is the spatial kernel size. We also view $\mathcal{W}$ as an 3D filter array and use notation $\mathcal{W}_n \in \mathbb{R}^{d \times d \times C}$ to represent the n-th filter. Let $\mathcal{Z} \in \mathbb{R}^{X \times Y \times C}$ be the input feature map. The output feature map $\mathcal{F} = (\mathcal{F}_1, \cdots, \mathcal{F}_N)$ is defined as

$$\mathcal{F}_n(x, y) = \sum_{i=1}^{C} \sum_{x'=1}^{X} \sum_{y'=1}^{Y} \mathcal{Z}^c(x', y') \mathcal{W}_n^c(x - x', y - y'),$$

where the superscript is the index of the channels.

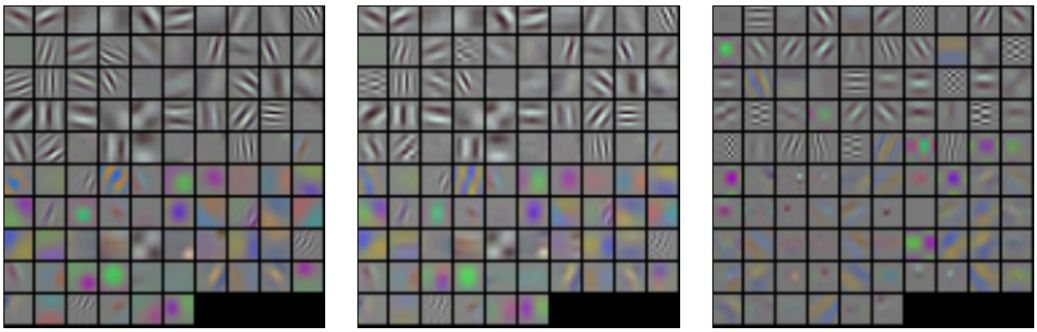
The goal is to find an approximation $\tilde{\mathcal{W}}$ of $\mathcal{W}$ that facilitates more efficient computation while maintaining the classification accuracy of the CNN. We propose the following scheme:

$$\tilde{\mathcal{W}}_n^c = \sum_{k=1}^{K} \mathcal{H}_n^k (\mathcal{V}_k^c)^T, \tag{1}$$

where K is a hyper-parameter controlling the rank, $\mathcal{H} \in \mathbb{R}^{N \times 1 \times d \times K}$ is the horizontal filter, $\mathcal{V} \in \mathbb{R}^{K \times d \times 1 \times C}$ is the vertical filter (we have slightly abused the notations to make them concise, $\mathcal{H}_n^k$ and $\mathcal{V}_k^c$ are both vectors in $\mathbb{R}^d$). Both $\mathcal{H}$ and $\mathcal{V}$ are learnable parameters.

With this form, the convolution becomes:

$$\tilde{\mathcal{W}}_n * \mathcal{Z} = \sum_{c=1}^{C} \sum_{k=1}^{K} \mathcal{H}_n^k (\mathcal{V}_k^c)^T * \mathcal{Z}^c = \sum_{k=1}^{K} \mathcal{H}_n^k * \left(\sum_{c=1}^{C} \mathcal{V}_k^c * \mathcal{Z}^c \right) \tag{2}$$



(a) (b) (c)

Figure 1: (a) Filters in the first layer in AlexNet. (b) Low-rank approximation using the proposed schemes with $K = 8$, corresponding to $3.67\times$ speedup for this layer. Note the low-rank approximation captures most of the information, including the directionality of the original filters. (c) Low-rank filters trained from scratch with $K = 8$.

The intuition behind this approximation scheme is to exploit the redundancy that exist both in the spatial dimensions and across channels. Note the convolutions in the above equation are all one dimensional in space.

We can estimate the reduction in computation with this scheme. Direct convolution by definition requires $\mathcal{O}(d^2NCXY)$ operations. In the above scheme, the computational cost associated with the vertical filters is $\mathcal{O}(dKCXY)$ and with horizontal filters $\mathcal{O}(dNKXY)$, giving a total computational cost of $\mathcal{O}(dK(N+C)XY)$. Acceleration can be achieved if we choose $K < \frac{dNC}{N+C}$. In principle, if $C \ll N$, which is typical in the first layer of a CNN, the acceleration is about d times.

We learn the approximating parameters H and V by a two-step strategy. In the first step, we approximate the convolution kernel $\mathcal{W}$ in each layer by minimizing $\|\tilde{\mathcal{W}} - \mathcal{W}\|_F$ (index of the layers are omitted for notation simplicity). Note that this step can be done in parallel as there is no inter-layer dependence. Then we fine-tune the whole CNN based on the discriminative criterion of restoring classification accuracy.

3.2 Algorithm

Based on the approximation criterion introduced in the previous section, the objective function to be minimized is:

$$(P1) \qquad E_1(\mathcal{H}, \mathcal{V}) := \sum_{n,c} \|\mathcal{W}_n^c - \sum_{k=1}^{K} \mathcal{H}_n^k (\mathcal{V}_k^c)^T\|_F^2. \tag{3}$$

This minimization problem has a closed form solution. This is summarized in the following theorem and the proof can be found in the appendix. The theorem gives us an efficient algorithm for computing the exact decomposition.

Theorem 1. *Define the following bijection that maps a tensor to a matrix $\mathcal{T} : \mathbb{R}^{C\times d\times d\times N} \mapsto \mathbb{R}^{Cd\times dN}$, tensor element (i_1, i_2, i_3, i_4) maps to (j_1, j_2), where*

$$j_1 = (i_1 - 1)d + i_2, \quad j_2 = (i_4 - 1)d + i_3.$$

Define $W := \mathcal{T}[\mathcal{W}]$. Let $W = UDQ^T$ be the Singular Value Decomposition (SVD) of W. Let

$$\begin{aligned} \hat{\mathcal{V}}_k^c(j) &= U_{(c-1)d+j,k}\sqrt{D_{k,k}} \\ \hat{\mathcal{H}}_n^k(j) &= Q_{(n-1)d+j,k}\sqrt{D_{k,k}}, \end{aligned} \tag{4}$$

then $(\hat{\mathcal{H}}, \hat{\mathcal{V}})$ is a solution to $(P1)$.

Because of this Theorem, we call the filters $(\mathcal{H}, \mathcal{V})$ low-rank constrained filters. Note that the solution to $(P1)$ is not unique. Indeed, if $(\mathcal{H}, \mathcal{V})$ is a solution, then $(\alpha\mathcal{H}, 1/\alpha\mathcal{V})$ is also a solution for

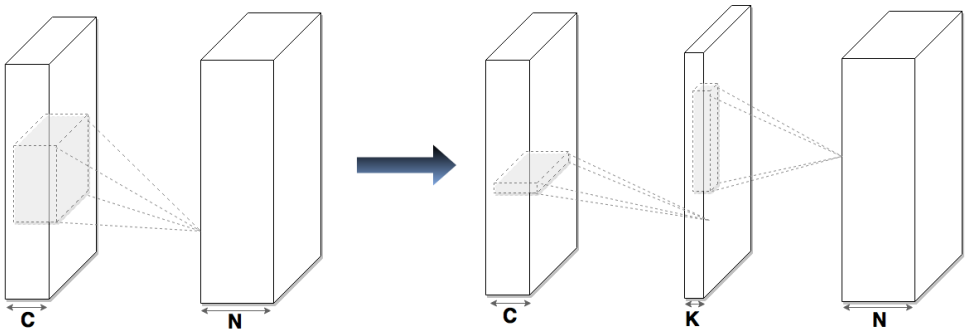


Figure 2: The proposed parametrization for low-rank regularization. Left: The original convolutional layer. Right: low-rank constraint convolutional layer with rank-K.

any $\alpha \neq 0$, but these solutions are equivalent in our application. An illustration of the closed-form approximation is shown in Figure 1.

A different criterion which uses the data distribution is proposed in Denton et al. (2014). But minimization for this criterion is NP-hard. The proof is also included in the appendix.

The algorithm provided by the above theorem is extremely fast. In our experiments, it completes in less than 1 second for most modern CNNs (AlexNet, VGG, GoogLeNet), as they have small convolutional kernels. Iterative algorithms (Denton et al. (2014); Jaderberg et al. (2014) take much longer, especially with the data-dependent criterion. In addition, iterative algorithms often lead to bad local minimum, which leads to inferior performance even after fine-tuning. The proposed algorithm solves this issue, as it directly provides the global minimum, which is the best data-independent approximation. Numerical demonstrations are given in section 4.

3.3 Training Low-rank Constrained CNN From Scratch

Using the above scheme to train a new CNN from scratch is conceptually straightforward. Simply parametrize the convolutional to be of the form in (1), and the rest is not very different from training a non-constrained CNN. Here $\mathcal{H}$ and $\mathcal{V}$ are the trainable parameters. As each convolutional layer is parametrized as the composition of two convolutional layers, the resulting CNN has more layers than the original one. Although the effective depth of the new CNN is not increased, the additional layers make numerical optimization much more challenging due to exploding and vanishing gradients, especially for large networks. To handle this problem, we use a recent technique called Batch Normalization (BN) (Ioffe & Szegedy, 2015). BN transform normalizes the activations of the internal hidden units, hence it can be an effective way to deal with the exploding or vanishing gradients. It is reported in Ioffe & Szegedy (2015) that deeper networks can be trained with BN successfully, and larger learning rates can be used. Empirically, we find BN effective in learning the low-rank constrained networks. An illustration of transformation of a original convolutional layer into a low-rank constraint one is in Figure 2. More details can be found in the numerical experiments section.

4 Experiments

In this section, we evaluate the proposed scheme on the CIFAR-10 and the ILSVRC12 datasets with several CNN models.

4.1 CIFAR-10

CIFAR-10 dataset is small by today's standard, but it is a good testbed for new ideas. We deploy two models as baseline models; one is a customized CNN and the other is the NIN model. We compare their performance with their corresponding low-rank constrained versions. All models on this dataset are learned from scratch.

Table 1: Network structure for CIFAR-10

Layer name	CNN	Low-rank CNN
conv1	$5 \times 5 \times 192$	$K_1 = 12$
conv2	$5 \times 5 \times 128$	$K_2 = 64$
conv3	$5 \times 5 \times 256$	$K_3 = 128$
fc1	2304×512	
fc2	512×10	

Layer name	NIN	Low-rank NIN
conv1	$5 \times 5 \times 192$	$K_1 = 10$
conv2,3	$1 \times 1 \times 160, 1 \times 1 \times 96$	
conv4	$5 \times 5 \times 192$	$K_2 = 51$
conv5,6	$1 \times 1 \times 192, 1 \times 1 \times 192$	
conv7	$3 \times 3 \times 192$	
conv8,9	$1 \times 1 \times 192, 1 \times 1 \times 10$	

Table 2: CIFAR-10 performance

METHOD	WITHOUT AUG.	WITH AUG.	SPEEDUP
CNN (ours)	15.12%	12.62%	1×
Low-rank CNN (ours)	14.50%	13.10%	2.9×
CNN + Dropout (ours)	13.90%	12.29%	0.96×
Low-rank CNN + Dropout (ours)	13.81%	11.41%	2.8×
NIN (ours)	10.12%	8.19%	1×
Low-rank NIN (ours)	**8.69%**	**6.98%**	1.5×
CNN + Maxout (Goodfellow et al., 2013)	11.68%	9.38%	-
NIN (Lin et al., 2013)	10.41%	8.81%	-
CNN (Srivastava et al., 2014)	12.61%	-	-
NIN + APL units (Agostinelli et al., 2014)	9.59%	7.51%	-

The configurations of the baseline models and their low-rank counterparts are outlined in Table 1. We substitute every single convolutional layer in the baseline models with two convolutional layers with parameter K introduced in the previous section. All other specifications of the network pairs are the same. Rectified Linear Unit (ReLU) is applied to every layer except for the last one. Our implementation of the NIN model is slightly different from the one introduced in Lin et al. (2013). We did not replace the 3×3 convolutional layer because this layer only constitutes a small fraction of the total execution time. Hence the efficiency gain of factorizing this layer is small.

The networks are trained with back propagation to optimize the multinomial logistic regression objective. The batch size is 100. The learning learning rate is initially set to 0.01 and decreases by a factor of 10 every time the validation error stops decreasing. Some models have dropout units with probability 0.25 inserted after every ReLU. For exact specifications of the parameters, the reader may check `https://github.com/chengtaipu/lowrankcnn`. We evaluated the performance of the models both with and without data augmentation. With data augmentation, the images are flipped horizontally with probability 0.5 and translated in both directions by at most 1 pixel. Otherwise, we only subtract the mean of the images and normalize each channel. The results are listed in Table 2.

The performance of the low-rank constrained versions of both networks are better than the baseline networks, with and without data augmentation. Notably, the low-rank NIN model outperforms the baseline NIN model by more than 1%. And as far as we know, this is also better than previously published results.

We then study how the empirical performance and speedup change as we vary the rank K. We choose the CNN+Dropout as baseline model with data augmentation described above. The results are listed in Table 3.

The number of parameters in the network can be reduced by a large factor, especially for the second and third layers. Up to $7\times$ speedup for a specific layer and 2-3$\times$ speedup for the whole network can be achieved. In practice, it is difficult for the speedup to match the theoretical gains based on the number of operations, which is roughly proportional to the reduction of parameters. The actual gain also depends on the software and hardware optimization strategies of convolutions. Our results in Table 3 are based on Nvidia Titan GPUs and Torch 7 with cudnn backend.

Interestingly, even with significant reductions in the number of parameters, the performance does not decrease much. Most of the networks listed in Table 3 even outperform the baseline model.

Table 3: Speedup and performance change. Performance change is relative to the baseline CNN+Dropout model with accuracy 87.71%.

LAYER	K_1	K_2	K_3	ACCURACY CHANGE	SPEEDUP (LAYER)	SPEEDUP (NET)	REDUCTIONS (WEIGHTS)
First	4	64	256	+0.69%	1.20×	2.91×	3.5×
	8	64	256	+0.85%	1.13×	2.87×	1.8×
	12	64	256	+0.94%	1.05×	2.85×	1.2×
Second	12	8	256	-0.02%	7.13×	3.21×	47.5×
	12	16	256	+0.50%	6.76×	3.21×	23.8×
	12	32	256	+0.89%	6.13×	3.13×	12.0×
	12	64	256	+0.94%	3.72×	2.86×	6.0×
	12	128	256	+1.32%	2.38×	2.58×	3.0×
	12	256	256	+1.40%	1.25×	1.92×	1.5×
Third	12	64	8	-2.25%	6.98 ×	3.11×	52.5×
	12	64	16	+0.21%	6.89×	3.11×	26.4×
	12	64	32	+0.19%	5.82×	3.10×	13.3×
	12	64	64	+0.19%	3.74×	2.96×	6.7×
	12	64	128	+0.94%	2.38×	2.86×	3.3×
	12	64	256	+1.75%	1.31×	2.30×	1.7×

Applying the low-rank constraints for all convolutional layers, the total number of parameters in the convolutional layers can be reduced by a large factor without degrading much performance. For example, with $K_1 = 12, K_2 = 16$ and $K_3 = 32$, the parameters in the convolutional kernels are reduced by 91% and the relative performance is +0.25%.

Nevertheless, the parameters in the fully connected layers still occupy a large fraction. This limits the overall compression ability of the low-rank constraint. There are some very recent works focusing on reducing the parameters in the fully connected layers (Novikov et al., 2015), combining these techniques with the proposed scheme will be explored in future research.

4.2 ILSVRC12

ILSVRC12 (Russakovsky et al., 2015) is a well-known large-scale benchmark dataset for image classification. We adopt three famous CNN models, AlexNet (Krizhevsky et al., 2012) (CaffeNet (Jia et al., 2014) as an variant), VGG-16 (Simonyan & Zisserman, 2014), and GoogLeNet (Szegedy et al., 2014) (BN-Inception (Ioffe & Szegedy, 2015) as an variant) as our baselines. The CaffeNet and VGG-16 are directly downloaded from Caffe's model zoo and then fine-tuned on the training set until convergence, while the BN-Inception model is trained from scratch by ourselves.

The introduced low-rank decomposition is applied to each convolutional layer that has kernel size greater than 1×1. Input images are first warped to 256×256 and then cropped to 227×227 or 224×224 for different models. We use the single center crop during the testing stage, and evaluate the performance by the top-5 accuracy on the validation set. Detailed training parameters are available at `https://github.com/chengtaipu/lowrankcnn`.

As before, the hyper-parameter K controls the trade-off between the speedup factor and the classification performance of the low-rank models. Therefore, we first study its effect for each layer, and then use the information to configure the whole low-rank model for better overall performance. We decompose a specific layer with a different K each time, while keeping the parameters of all the other layers fixed. The performance after fine-tuning with respect to the theoretical layer speedup is demonstrated in Figure 3. In general, we choose for each layer the value of K that most accelerates the forward computation while does not hurt the performance significantly ($< 1\%$). A more automatic way for choosing K is based on Eigengap, such that the first K eigenvectors account for 95% of the variations. This is similar to choosing the number of principal components in PCA. The detailed low-rank model structures are listed in Table 4.

The proposed closed form solution provides the optimal data-independent initialization to the low-rank model. As indicated in Figure 4, there is a performance gap between the low-rank models and

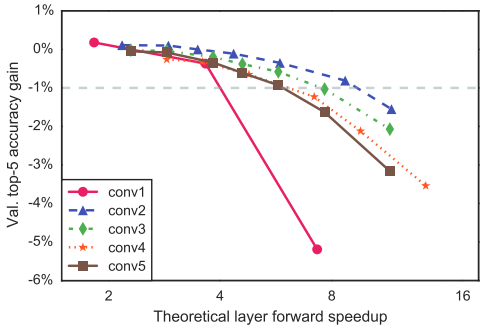


Figure 3: The performance w.r.t. the theoretical layer speedup. Only the conv1-conv5 layers of the AlexNet are shown.

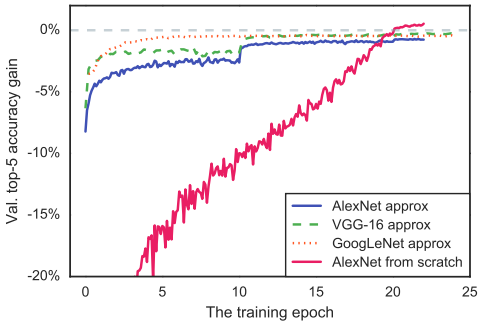


Figure 4: The performance w.r.t. the fine-tuning epoch when using the proposed closed form solution as initialization.

Table 4: Low-rank models for ILSVRC12. For VGG-16, each convolution module contains two or three sub-convolutional layers. For GoogLeNet, each inception module contains one 3×3 and two consecutive 3×3 convolutional layers. Their corresponding Ks are shown in a cell for brevity.

(a) AlexNet

Layer	K
conv1	8
conv2	40
conv3	60
conv4	100
conv5	200

(b) VGG-16

Layer	K
conv1	5, 24
conv2	48, 48
conv3	64, 128, 160
conv4	192, 192, 256
conv5	320, 320, 320

(c) GoogLeNet

Layer	K	Layer	K
conv1	8	inception(4b)	64, 64, 80
conv2	48	inception(4c)	64, 64, 64
inception(3a)	32, 32, 48	inception(4d)	64, 96, 96
inception(3b)	32, 32, 48	inception(4e)	64, 128, 160
inception(3c)	80, 32, 48	inception(5a)	128, 96, 128
inception(4a)	32, 64, 80	inception(5b)	128, 96, 128

their baselines at the beginning, but the performance is restored after fine-tuning. It is claimed in Denton et al. (2014) that data-dependent criterion leads to better performance, we found that this is true upon approximation, but after fine-tuning, the difference between the two criteria is negligible ($< 0.1\%$).

At last, we compare the low-rank models with their baselines from the perspective of classification performance, as well as the time and space consumption. The results are summarized in Table 5. We can see that all the low-rank models achieve comparable performances. Those initialized with closed form weights approximation (cf. approximation rows in Table 5) are slightly inferior to their baselines. While the low-rank AlexNet trained from scratch with BN could achieve even better performance. This observation again reveals that the low-rank CNN structure could have better discriminative power and generalization ability. On the other hand, both the running time and the number of parameters are consistently reduced. Note that the large gaps between the theoretical and the actual speedup are mainly due to the CNN implementations, and the current BN operations significantly slow down the forward computation. This suggests room for accelerating the low-rank models by designing specific numerical algorithms.

5 Discussion

In this paper, we explored using tensor decomposition techniques to speedup convolutional neural networks. We have introduced a new algorithm for computing the low-rank tensor decomposition and a new method for training low-rank constrained CNNs from scratch. The proposed method is evaluated on a variety of modern CNNs, including AlexNet, NIN, VGG, GoogleNet with success. This gives a strong evidence that low-rank tensor decomposition can be a generic tool for speeding up large CNNs.

Table 5: Comparisons between the low-rank models and their baselines. The theoretical speedup and weights reduction are computed concerning only the convolutional layers to be decomposed. While the actual speedup is based on the forward computation time of the whole net.

METHOD	TOP-5 VAL. ACCURACY	THEORETICAL SPEEDUP	ACTUAL SPEEDUP	WEIGHTS REDUCTION
AlexNet (original)	80.03%	1×	1×	1×
Low-rank (cf. approximation)	79.66%	5.27×	1.82×	5.00×
Low-rank (from scratch with BN)	80.56%	5.24×	1.09×	4.94×
VGG-16 (original)	90.60%	1×	1×	1×
Low-rank (cf. approximation)	90.31%	3.10×	2.05×	2.75×
GoogLeNet (original)	92.21%	1×	1×	1×
Low-rank (cf. approximation)	91.79%	2.89×	1.20×	2.84×

On the the other hand, the interesting fact that the low-rank constrained CNNs sometimes outperform their non-constrained counterparts points to two things. One is the local minima issue. Although the expressive power of low-rank constrained CNNs is strictly smaller than that of the non-constrained one, we have observed in some cases that the former have smaller training error. This seems to suggest the low-rank form helps the CNNs begin with a better initialization and settles at a better local minimum. The other issue is over-fitting. This is shown by the observation that in many cases the constrained model has higher training error but generalizes better. Overall, this suggests room for improvement in both the numerical algorithms and the regularizations of the CNN models.

ACKNOWLEDGMENTS

This work is supported in part by the 973 project 2015CB856000 of the Chinese Ministry of Science and Technology and the DOE grant DE-SC0009248.

REFERENCES

Agostinelli, Forest, Hoffman, Matthew, Sadowski, Peter, and Baldi, Pierre. Learning activation functions to improve deep neural networks. *arXiv preprint arXiv:1412.6830*, 2014.

de Silva, Vin and Lim, Lek-Heng. Tensor Rank and the Ill-Posedness of the Best Low-Rank Approximation Problem. *SIAM Journal on Matrix Analysis and Applications*, 30(3):1084–1127, September 2008.

Denton, Emily L, Zaremba, Wojciech, Bruna, Joan, LeCun, Yann, and Fergus, Rob. Exploiting linear structure within convolutional networks for efficient evaluation. In *NIPS*, 2014.

Farabet, Clement, Couprie, Camille, Najman, Laurent, and LeCun, Yann. Learning hierarchical features for scene labeling. *TPAMI*, 35(8):1915–1929, 2013.

Gillis, Nicolas and Glineur, François. Low-rank matrix approximation with weights or missing data is np-hard. *SIAM Journal on Matrix Analysis and Applications*, 32(4):1149–1165, 2011.

Goodfellow, Ian J, Warde-Farley, David, Mirza, Mehdi, Courville, Aaron, and Bengio, Yoshua. Maxout networks. *arXiv preprint arXiv:1302.4389*, 2013.

Ioffe, Sergey and Szegedy, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Jaderberg, Max, Vedaldi, Andrea, and Zisserman, Andrew. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014.

Jia, Yangqing, Shelhamer, Evan, Donahue, Jeff, Karayev, Sergey, Long, Jonathan, Girshick, Ross, Guadarrama, Sergio, and Darrell, Trevor. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

Lebedev, Vadim, Ganin, Yaroslav, Rakhuba, Maksim, Oseledets, Ivan, and Lempitsky, Victor. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *arXiv preprint arXiv:1412.6553*, 2014.

Lin, M., Chen, Q., and Yan, S. Network In Network. *ArXiv e-prints*, December 2013.

Long, Jonathan, Shelhamer, Evan, and Darrell, Trevor. Fully convolutional networks for semantic segmentation. *arXiv preprint arXiv:1411.4038*, 2014.

Novikov, A., Podoprikhin, D., Osokin, A., and Vetrov, D. Tensorizing Neural Networks. *ArXiv e-prints*, September 2015.

Rigamonti, Roberto, Sironi, Amos, Lepetit, Vincent, and Fua, Pascal. Learning separable filters. In *CVPR*, 2013.

Russakovsky, Olga, Deng, Jia, Su, Hao, Krause, Jonathan, Satheesh, Sanjeev, Ma, Sean, Huang, Zhiheng, Karpathy, Andrej, Khosla, Aditya, Bernstein, Michael, Berg, Alexander C., and Fei-Fei, Li. ImageNet Large Scale Visual Recognition Challenge. *IJCV*, pp. 1–42, April 2015.

Simonyan, Karen and Zisserman, Andrew. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Srivastava, Nitish, Hinton, Geoffrey, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan. Dropout: A simple way to prevent neural networks from overfitting. In *ICML*, 2014.

Szegedy, Christian, Liu, Wei, Jia, Yangqing, Sermanet, Pierre, Reed, Scott, Anguelov, Dragomir, Erhan, Dumitru, Vanhoucke, Vincent, and Rabinovich, Andrew. Going deeper with convolutions. *arXiv preprint arXiv:1409.4842*, 2014.

Vanhoucke, Vincent, Senior, Andrew, and Mao, Mark Z. Improving the speed of neural networks on cpus. In *Deep Learning and Unsupervised Feature Learning, NIPS Workshop*, 2011.

APPENDIX

PROOF OF THEOREM 1

Proof. Consider the following minimization problem:

$$\begin{aligned} (P2) \qquad & E_2(\tilde{W}) := \|\tilde{W} - W\|_F^2 \\ & \text{subject to} \quad \text{Rank}(\tilde{W}) \leq K. \end{aligned} \tag{5}$$

Let $(\mathcal{H}^*, \mathcal{V}^*)$ be a solution to (P1), then we can construct a solution to $(P2)$ as follows:

$$\tilde{W} = \sum_{k=1}^{K} \begin{bmatrix} \mathcal{V}_k^1 \\ \mathcal{V}_k^2 \\ \vdots \\ \mathcal{V}_k^C \end{bmatrix} \left[\mathcal{H}_1^k, \mathcal{H}_2^k, \cdots, \mathcal{H}_N^k\right].$$

Because of the separability of the Frobenius norm,

$$E_1(\mathcal{H}^*, \mathcal{V}^*) = E_2(\tilde{W}).$$

Moreover, as $\text{Rank}(\tilde{W}) \leq K$, hence $\tilde{W}$ is feasible for $(P2)$. We have

$$E_2(W^*) \leq E_1(\mathcal{H}^*, \mathcal{V}^*) = E_2(\tilde{W}), \tag{6}$$

where W^* is any solution to $(P2)$.

On the other hand, let W^* be a solution to $(P2)$, then we construct a solution $(\hat{\mathcal{H}}, \hat{\mathcal{V}})$ to $(P1)$ as (4). Hence

$$E_1(\mathcal{H}^*, \mathcal{V}^*) \leq E_1(\hat{\mathcal{H}}, \hat{\mathcal{V}}).$$

Together with (6),

$$E_1(\hat{\mathcal{H}}, \hat{\mathcal{V}}) = E_2(\mathcal{W}^*) = E_1(\mathcal{H}^*, \mathcal{V}^*). \tag{7}$$

We have proved $(\hat{\mathcal{H}}, \hat{\mathcal{V}})$ is a solution to $(P1)$. □

HARDNESS OF THE DATA-DEPENDENT APPROXIMATION

Using the data-dependent criterion, the minimization problem is:

$$E(\mathcal{H}, \mathcal{V}) := \sum_{i=1}^{M} \sum_{n=1}^{N} \sum_{c=1}^{C} \|\mathcal{W}_n^c * \mathcal{Z}_i^c - \sum_{k=1}^{K} \mathcal{H}_n^k (\mathcal{V}_k^c)^T * \mathcal{Z}_i^c\|_F^2. \tag{8}$$

For fixed stride s, define the linear map $P_m : \mathbb{R}^{X \times Y} \mapsto \mathbb{R}^{d \times d}$, $P_m(z)$ samples the m-th $d \times d$ patch from $z \in \mathbb{R}^{X \times Y}$ followed by flipping the patch horizontally and vertically. Then

$$\sum_c \|\mathcal{W}_n^c * \mathcal{Z}_n^c\|_F^2 = \sum_{m,c} \langle \mathcal{W}_n^c, P_m \mathcal{Z}_i^c \rangle^2.$$

Let

$$Z_{im} = \begin{bmatrix} P_m \mathcal{Z}_i^1 \\ P_m \mathcal{Z}_i^2 \\ \vdots \\ P_m \mathcal{Z}_i^c \end{bmatrix} \otimes \underbrace{(1, 1, \cdots, 1)}_{N}.$$

Similar as in Criterion 1, the approximation problem is equivalent to the following minimization program:

$$\begin{aligned} E(\tilde{W}) &:= \sum_{i,m} \|(W - \tilde{W}) \circ Z_{im}\|_F^2 \\ \text{subject to} \qquad & \text{Rank}(\tilde{W}) \leq K, \end{aligned} \tag{9}$$

where $\circ$ is the Hadamard product.

This is a weighted low-rank approximation problem:

$$\begin{aligned} E(\tilde{W}) &:= \sum_{ij} G_{ij} (W_{ij} - \tilde{W}_{ij})^2 \\ \text{subject to} \qquad & \text{Rank}(\tilde{W}) \leq K, \end{aligned} \tag{10}$$

where $G = \sum_{im} Z_{im} \circ Z_{im}$.

Although it appears very similar to the problem in Criterion 1, which has a closed form solution, this is much more difficult to solve except for a few special cases. (E.g., when the weight matrix is identity or has rank one.) In fact, it can be proved that this problem is NP-hard (Gillis & Glineur, 2011).