ACCELERATING CONVOLUTIONAL NEURAL NETWORKS USING ITERATIVE TWO-PASS DECOMPOSITION/ CONFERENCE SUBMISSIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

We present the iterative two-pass decomposition flow to accelerate existing convolutional neural networks (CNNs). The proposed rank selection algorithm can effectively determine the proper ranks of the target convolutional layers for the low rank approximation. Our two-pass CP-decomposition helps prevent from the instability problem. The iterative flow makes the decomposition of the deeper networks systematic. The experiment results shows that VGG16 can be accelerated with a 6.2× measured speedup while the accuracy drop remains only 1.2%.

1 INTRODUCTION

Deep learning has become of vital importance in a variety of artificial intelligence applications. Recently, convolutional neural networks (CNNs) have been widely applied to have the breakthrough in improving the recognition accuracy for challenging computer vision tasks such as image classification, localization, object detection, and so on (Russakovsky et al. (2015); Krizhevsky et al. (2012); Simonyan & Zisserman (2014); He et al. (2015)). However, those significant achievements using CNNs come with the cost of larger network size and higher computational complexity, which leads to an increasing difficulty for deploying to resource constrained edge devices, or even for the fast computation on the cloud servers. This paper addresses the acceleration of the existing CNN models to cope with such burden.

1.1 RELATIVE WORKS

There have been research works on accelerating CNNs in many aspects. Several approaches have been proposed to simplify the convolution operations. Vasilache et al. (2014) uses Fast Fourier Transform to accelerate the convolution which is fast for large filters. In Lavin & Gray (2015), Winograd's minimal filtering algorithms have been adopted to speed up small-size filters. Pruning aims at removing unessential weights in the filters to minimize the computation. Weights can be removed after the training (Han et al. (2015); Li et al. (2016); Yu et al. (2017)) or during the training with the sparsity constraint (Wen et al. (2016); Park et al. (2017)). Quantization reduces the precision from the 32-bit floating-point computations to the 8-bit fixed-point ones, or even the binary operations (Hubara et al. (2016); Rastegari et al. (2016)). These approaches are efficient for implementing the hardware accelerators of the faster inference.

In addition to the attempts of accelerating pre-trained models, some works focus on designing new models with more efficient computation (Iandola et al. (2016); Wang et al. (2016); Howard et al. (2017); Zhang et al. (2017)). In general, new models tend to gain more speedup than accelerating pre-trained models. However, constructing a new CNN from scratch might be difficult without using a large amount of computing resources.

In our approach, we aim at accelerating existing CNNs using the low rank approximation that a specific tensor can be represented with several simpler tensors. Jaderberg et al. (2014) has shown the redundancies in convolutional layers can be substituted by lower rank filters with the 4.5× speedup in a text recognition application using a simple four layer CNN. Denton et al. (2014) has presented a

different low-rank decomposition and clustering approach. They reported the $2\times$ speedup on the first layer of a 15-layer CNN for image classification. They also found that the low rank approximation has the potential to improve the generalization ability of the CNN.

Our work was inspired by Lebedev et al. (2014) which adopted the CP-ALS, one of the popular CP decomposition algorithms (Kolda & Bader (2009)). However, only one layer in the network was decomposed. They also observed the instability problem which causes fine-tuning the decomposed networks a difficult problem. Later, Kim et al. (2015) successfully decomposed the whole network with the Tucker Decomposition (Kolda & Bader (2009)). Zhang et al. (2016) took the nonlinear unit into account to obtain the decomposed filters and reduced the accumulated error. Their nonlinear asymmetric 3d reconstruction, combined with the technique in Jaderberg et al. (2014) to further decomposing spatial dimension, achieved the $5\times$ speedup with the accuracy loss of 2% for VGG16. An additional fine-tuning was applied to improve the loss to 1%.

In Astrid & Lee (2017), the iterative fine-tune has been proposed to overcome the CP instability. Instead of the decomposition of the entire network, each iteration decomposes one layer with the fine-tuning of the whole network. Their attempt gradually transforms the dense network into decomposed form layer by layer, which achieves the less accuracy drop. We also adopt and improve this concept to further improve the performance, which will be discussed later in 2.1.3.

1.2 Contributions

Our contributions include the following:

- The proposed two-pass decomposition can prevent from possible instability of CP-decomposition and improve the accuracy effectively. Training and fine-tuning can be done with vanilla parameter setting (e.g., learning rate).
- Our iterative flow helps the decomposition of a deeper network in a systematic manner. The rank selection algorithm can be utilized to determine the target ranks for CP-decomposition. The proposed approach can be applied to different kinds of deep convolutional networks, resulting in a general technique to improve the existing models. Our experiment shows that we can achieve the $6.2\times$ speedup for VGG16 (Simonyan & Zisserman (2014)) while with the classification accuracy loss of only 1.2% on ImageNet 2012 validation set. The size of the convolutional layers can be reduced by 85% as well. In addition, our approach can also speed up ResNet50 (He et al. (2015)) by $1.35\times$ faster with the accuracy loss of 1.51%, and the model size reduction of 48%.

1.3 Low Rank Approximation

In this section, we introduce the low rank approximation to accelerate convolutional layers with the CANDECOMP/PARAFAC (CP) decomposition technique (Kiers (2000); Kolda & Bader (2009)).

1.3.1 CP Decomposition for Convolutional Layers

The CP decomposition factorizes the tensors into a sum of series rank-one tensors. Assume W is a third-order tensor and a_r, b_r, c_r for $r = 1, \ldots, R$ are vectors. A rank R decomposition can be expressed as

$$W \approx \sum_{r=1}^{R} a_r \circ b_r \circ c_r, \tag{1}$$

where $W \in \mathbb{R}^{H \times I \times J}, a_r \in \mathbb{R}^H, b_r \in \mathbb{R}^I, c_r \in \mathbb{R}^J$ for $r = 1, \ldots, R$, as shown in Figure 1.

In the case of CNNs, the filters are fourth-order tensors in general. Assume W is a fourth-order tensor and a_r, b_r, c_r, d_r for $r = 1, \ldots, R$ are vectors. A rank R decomposition can be expressed as follows:

$$W \approx \sum_{r=1}^{R} a_r \circ b_r \circ c_r \circ d_r, \tag{2}$$

where $W \in \mathbb{R}^{C_{in} \times W_f \times H_f \times C_{out}}$, $a_r \in \mathbb{R}^{C_{in}}$,$b_r \in \mathbb{R}^{W_f}$, $c_r \in \mathbb{R}^{H_f}$ and $d_r \in \mathbb{R}^{C_{out}}$ for $r = 1, \ldots, R$. Figure 2 shows the original convolution computation.

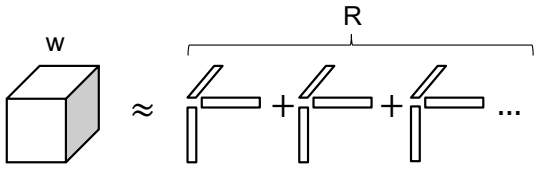

Figure 1: The CP decomposition.

Let X be the input features, Y be the output features, W be the filter weights and $x^* = \frac{W_f - 1}{2}, y^* = \frac{H_f - 1}{2}$. Assume W_f and H_f are both odd numbers. A convolutional layer with C_{in} input channels, C_{out} output channels and filter shape of $W_f \times H_f$, can be expressed as

$$Y(x, y, k) = \sum_{h=1}^{C_{in}} \sum_{i=1}^{W_f} \sum_{j=1}^{H_{in}} W(h, i, j, k) X(i + x - x^*, j + y - y^*, h). \tag{3}$$

Then the weights in Eq. 3 can be substituted with the decomposed tensors in Eq. 2, i.e.,

$$Y(x, y, k) = \sum_{h=1}^{C_{in}} \sum_{i=1}^{W_f} \sum_{j=1}^{H_f} \sum_{r=1}^{R} a_r(h) b_r(i) c_r(j) d_r(k) X(i + x - x^*, j + y - y^*, h) \tag{4}$$

$$= \sum_{h=1}^{C_{in}} a_r(h) \sum_{i=1}^{W_f} b_r(i) \sum_{j=1}^{H_f} c_r(j) \sum_{r=1}^{R} d_r(k) X(i + x - x^*, j + y - y^*, h). \tag{5}$$

Equation 5 can be rewritten as follows:

$$Y_{(1)}(i + x - x^*, j + y - y^*, r) = \sum_{h=1}^{C_{in}} a_r(h) X(i + x - x^*, j + y - y^*, h); \tag{6}$$

$$Y_{(2)}(x, j + y - y^*, r) = \sum_{i=1}^{W_f} b_r(i) Y_{(1)}(i + x - x^*, j + y - y^*, r); \tag{7}$$

$$Y_{(3)}(x, y, r) = \sum_{j=1}^{H_f} c_r(j) Y_{(2)}(x, j + y - y^*, r); \tag{8}$$

$$Y_{(4)}(x, y, k) = \sum_{r=1}^{R} d_r(k) Y_{(3)}(x, y, r). \tag{9}$$

$Y_{(1)}$ and $Y_{(4)}$ perform the 1×1 convolutions on the input and output channels, respectively. In addition, $Y_{(2)}$ and $Y_{(3)}$ perform the depth-wise convolutions along the filters' width and height direction, respectively, as shown in Figure 3. Therefore, the four layers can be applied to substitute the original convolution layer.

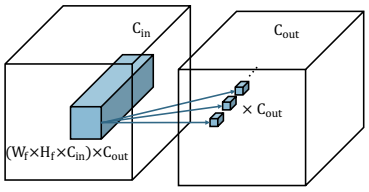

Figure 2: The original convolution.

1.3.2 The Complexity and Speedup

Assume the size of the input features is $W_{in} \times H_{in} \times C_{in}$, where W_{in} and H_{in} are the width and height of the features, and C_{in} is the number of input channels. Let the number of output channels

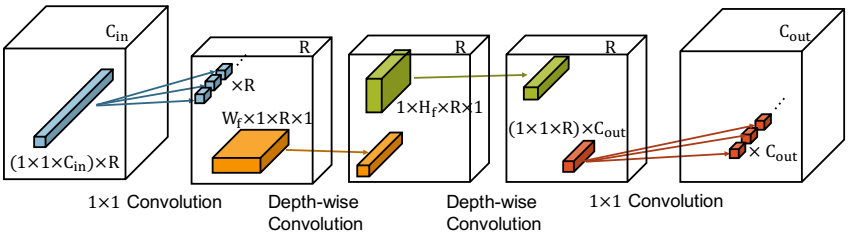

Figure 3: The decomposed convolution.

be C_{out}, the filter shape be $W_f \times H_f$, and the stride be 1. The complexity of the original convolution is $O(C_{in}W_{in}H_{in}C_{out}W_fH_f)$ from Eq. 3. If the convolution layer is decomposed with Eq. 6–9, the complexity becomes $O(W_{in}H_{in}R(C_{in} + W_f + H_f + C_{out}))$. The speedup of decomposing the single convolutional layer with rank R is

$$O(\frac{C_{in}W_{in}H_{in}C_{out}W_fH_f}{W_{in}H_{in}R(C_{in} + W_f + H_f + C_{out})}) = O(\frac{C_{in}C_{out}W_fH_f}{R(C_{in} + W_f + H_f + C_{out})}). \quad (10)$$

While substituting a layer, the smaller the rank R for the decomposed layers, the higher the speedup.

1.3.3 The CP Instability

As Lebedev et al. (2014) mentioned, training the network with CP decomposition may suffer from the instability problem that leads to the gradient explosion. The small learning rate is used to cope with the problem. In addition, part of the decomposed layer is fixed for training. However, with a small learning rate, the accuracy may improve slowly for decomposing deeper networks. Fixing decomposed layers also makes the accuracy hard to improve because only a few parameters could be fine-tuned.

2 Proposed Flow of Accelerating CNNs

In this section, we present the *Two-Pass Decomposition* to avoid the CP Instability. In addition, the *Iterative Two-Pass Decomposition* flow can be applied to improve the accuracy loss when accelerating the deeper networks.

2.1 Two-Pass Decomposition

Our two-pass decomposition consists of four steps to compute the decomposed layer.

Step 1: *The 1^{st} Decomposition*: Apply the CP decomposition on the original filter tensor to get the decomposed tensor. In this work, we adopt the CP-ALS algorithm in Kolda & Bader (2009).

Step 2: *Restoration*: Restore the decomposed tensor back to the dense form.

Step 3: *Optimization*: Replace the original filter tensor with the restored filter tensor in the target convolutional layer. Optimize the updated networks with fine-tuned filter weights.

Step 4: *The 2^{nd} Decomposition*: Decompose the optimized filter tensor again.

The concept to restore the decomposed tensor back to the dense form is to prevent from the CP instability. We observed that if the network is trained in the restored dense form, the training result can be more stable because of its smoother convex. In addition, the structure of the restored dense form tends to be closer to the second low rank form, which also leads to a less decomposition error.

Figure 4 compares the accuracy between our two-pass decomposition and the original CP-ALS decomposition of VGG16. The CNN layers from Conv1_2 to Conv3_1 are evaluated, with Conv1_2 to Conv2_2 decomposed and fixed. The first layer, Conv1_1, is not computationally intensive and not decomposed. The ranks of Conv1_2 Conv2_1, and Conv2_2 are 24, 25, and 28, respectively. In this experiment, different ranks of Conv3_1 are applied to observe the accuracy change. The

comparison shows that the two-pass CP decomposition retains the relatively high accuracy even with smaller ranks, i.e., the ranks of 24 and 32 in the figure. On the contrary, with the original CP decomposition, the accuracy drops significantly when decreasing the rank. Such result leaves our two-pass CP decomposition more room to speed up the network with lower ranks while maintaining the accuracy.

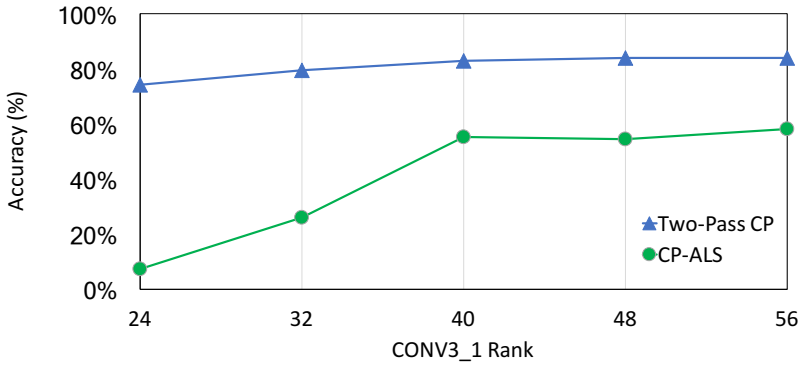

Figure 4: The comparison between the two-pass decomposition and the original CP decomposition, with 500 batches.

2.1.1 Proposed Iterative Two-Pass Decomposition Flow for DNNs

Traditionally, the filter weights of the decomposed layers are fixed during the fine-tuned phase to prevent from the gradient explosion. Our two-pass decomposition provides the better result as compared with the original CP decomposition. However, the accuracy will still degrade for decomposing the deeper networks, since there will be little room for fine-tuning.

For a DNN with many convolution layers, we propose the *Rank Selection* algorithm to effectively determine the rank for each layer. Then our two-pass decomposition technique is applied iteratively to one group of the layers at a time. The overall flow to decompose the whole CNN is shown in Figure 5.

For the decomposition, target rank of each layer is a hyperparameter to decide. Our flow takes a given set of ranks as the input to perform the initial decomposition. After the initial decomposition, the fitness of each decomposed layer can be calculated. Based on the initial ranks and corresponding fitnesses, we propose the Rank Selection algorithm to optimize the target ranks for the decomposition, which will be discussed in Sec. 2.1.2.

The second part of the flow performs the two-pass decomposition iteratively, by grouping the CNN layers. At the end of each iteration, the whole network will be fine-tuned with the decomposed layers fixed. The accuracy of the fine-tuning can be further improved with a few more epochs until the loss is converged. The details will be discussed in Sec. 2.1.3.

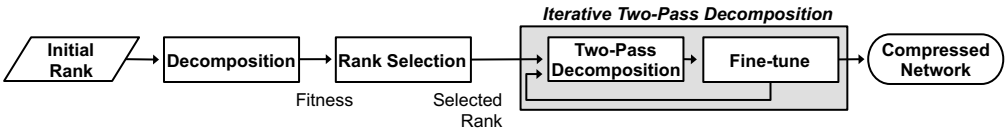

Figure 5: The proposed iterative two-pass decomposition flow.

2.1.2 Our Rank Selection Algorithm

Given any initial ranks for the CNN and a target speedup, the proposed Rank Selection algorithm determines the optimized rank configuration. Let N_l denote the operation complexity of the l^{th} layer in the CNN, and N_l' be its complexity of the decomposed form with the rank of R_l. Therefore, $N_l =$

$C_{in}^{(l)} W_{in}^{(l)} H_{in}^{(l)} C_{out}^{(l)} W_f^{(l)} H_f^{(l)}$ and $N_l' = W_{in}^{(l)} H_{in}^{(l)} R_l (C_{in}^{(l)} + W_f^{(l)} + H_f^{(l)} + C_{out}^{(l)})$ (see Sec. 1.3.2). The overall theoretical speedup S of the CNN can be defined as

$$S = \frac{\sum_{l \in \text{all convolutional layers}} (N_l)}{\sum_{l \in \text{all convolutional layers}} (N_l')} \tag{11}$$

CP-ALS decomposition in Nickel (2016) is adopted for our approach. According to Kolda & Bader (2009); Nickel (2016), the fitness (as the decomposition quality) of a convolutional layer can be defined as

$$F_l = 1 - \frac{\|X_l - \widehat{X_l}\|^2}{\|X_l\|^2} \tag{12}$$

where X_l is the original tensor of the l^{th} layer and $\widehat{X_l}$ is the approximated tensor.

The product of the fitness F_l and the operation complexity N_l is used to estimate the profit to improve a specific layer l. If a layer has higher fitness and complexity, we tend to reduce its rank for the speedup. Otherwise, its ranks can be increased for the accuracy. Algorithm 1 shows the proposed Rank Selection to help determine the optimized rank configuration. Note that the CP decomposition with a large rank is time-consuming. So a linear approximation is used to predict the fitness in this stage.

Algorithm 1 Rank Selection

Input: R_l, F_l, N_l, N_l' for $l \in$ all convolutional layers, S_{target} : target speedup, ϵ : speedup margin
Output: Selected Ranks for each layer

Calculate Initial Speedup S
while $|S - S_{\text{target}}| > \epsilon$ **do**
  if $S < S_{\text{target}}$ **then**
    Let T be the layer with the largest $F_T \times N_T$ among all layers for the speedup.
    $F_T = F_T \times \frac{(R_T - 1)}{R_T}$ ▷ Linear approximation of the new fitness.
    $R_T = R_T - 1$
  else
    Let T be the layer with the smallest $F_T \times N_T$ among all layers for the accuracy.
    $F_T = F_T \times \frac{(R_T + 1)}{R_T}$ ▷ Linear approximation of the new fitness.
    $R_T = R_T + 1$
  end if
  Update N_l', S
end while

2.1.3 Iterative Two-Pass Decomposition Flow

To prevent from the accuracy degradation, our two-pass decomposition approach can be performed iteratively, based on the concept from Astrid & Lee (2017). Figure 6 illustrates the three iterations applied to VGG16 model. Each iteration consists of five steps. The first four steps perform the proposed two-pass decomposition of the target group. E.g., the group of Conv1_2, Conv2_1, Conv2_2 is decomposed and restored back to the dense form in the first iteration (see Figure 6). Then the whole network is fine-tuned to optimize the weights. Afterward, the three target layers are decomposed again. In the fifth step, the whole network is fine-tuned again with these decomposed layers fixed. This additional step can improve the degraded accuracy further.

The second iteration deals with the group of Conv3_1, Conv3_2, and Conv3_3. Note that the previously decomposed layers remain fixed in the Optimization and Fine-tuning steps. The process continues until all the target layers are decomposed.

3 Experimental Results and Discussions

3.1 Decomposing VGG16

The VGG16 model modified from machrisaa (2016) is applied to verify our iterative two-pass decomposition flow. ImageNet 2012 training set is used for training and the validation set is used

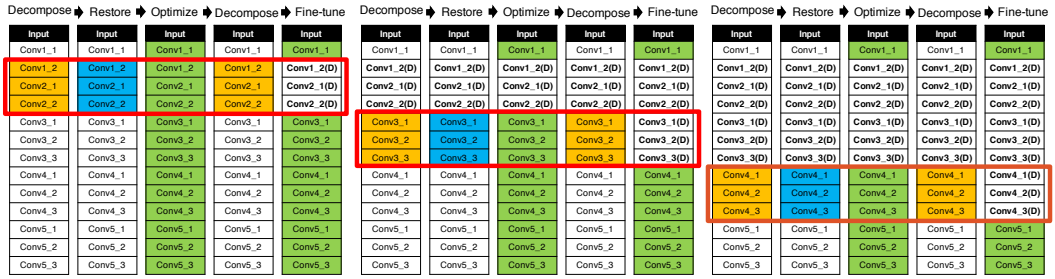

Figure 6: Iterative two-pass decomposition for VGG16.

for the top-5 single view accuracy measurement. The accuracy of the pre-trained VGG16 model is 89.9% by using AdaDelta Optimizer with default setting in Tensorflow. The speedup is measured on a single-thread 2.7GHz Intel Core i5 CPU.

All the convolutional layers except the first one, Conv1_1, are decomposed. We adopt the ranks proposed in Zhang et al. (2016) as the Baseline rank configuration (see Table 1). With the Baseline configuration as the initial ranks, The Rank-Selection configuration is obtained with the theoretical speedup of 8.4, which achieves the 6.2× measured speedup of on the single thread CPU.

Table 1: Two rank configurations for VGG16.

Layer Name	Baseline	RankSelection	Layer Name	Baseline	RankSelection
Conv1_2	11	16	Conv4_1	104	339
Conv2_1	25	68	Conv4_2	92	205
Conv2_2	28	53	Conv4_3	100	246
Conv3_1	52	159	Conv5_1	232	450
Conv3_2	46	93	Conv5_2	224	431
Conv3_3	56	115	Conv5_3	214	416

To compare the different decomposition sequences, two grouping schemes are evaluated, as shown in Table 2. The In-Order scheme groups the convolutional layers based on their connection order. On the other hand, The Fitness-Based scheme groups the layers based on the sorting order of the fitness, from the smallest to the largest. In this experiment, each scheme has three groups of layers.

Table 3 shows the accuracy at the optimization step and final fine-tuning step of each iteration for the Baseline configuration and Rank-Selection configuration, respectively. Two different grouping schemes are also applied for each rank configuration. For the quick evaluation, one epoch is applied for each fine-tuning. We can observe that the Rank-Selection configuration improves the accuracy significantly. In addition, the In-Order grouping scheme is much better than the Fitness-Based scheme for the Baseline rank configuration. However, their difference is vague when applying the Rank-Selection configuration. We will discuss it further in Sec. 3.3

Table 2: Two grouping schemes for the iterative two-pass decomposition.

	Iteration 1	Iteration 2	Iteration 3
In-Order	Con1_2, Conv2_1, Conv2_2, Conv3_1	Con3_2, Conv3_3, Conv4_1, Conv4_2	Con4_3, Conv5_1, Conv5_2, Conv5_3
Fitness-Based	Conv2_2, Conv3_2, Conv3_3, Conv4_2, Conv4_3	Conv4_1, Conv5_1, Conv5_2, Conv5_3	Conv1_2, Conv2_1, Conv3_1

To compare with previous works (Jaderberg et al. (2014); Zhang et al. (2016)), our experiment is extended with additional epochs to fine-tune until the accuracy improvement is smaller than 0.1%. Table 4 shows the accuracy drop for each iteration. In this case, the Fitness-Based configuration is better than the In-Order one (see the discussion in Sec. 3.3). As a result, our method achieves the

Table 3: Accuracy comparison among different configurations for VGG16 (%).

		Iteration 1		Iteration 2		Iteration 3	
		Optimize	Fine-tune	Optimize	Fine-tune	Optimize	Fine-tune
Baseline	In-Order	88.76	88.74	85.34	84.54	80.62	80.06
	Fitness-Based	85.59	85.06	82.55	82.34	80.75	70.34
Rank Selection	In-Order	90.05	90.04	89.17	89.08	88.25	88.12
	Fitness-Based	89.11	88.97	88.52	88.44	88.13	87.95

highest measured speedup with the lowest accuracy drop among the works utilizing decomposition techniques, as shown in Table 5. Note that the asymmetric 3d approach in Zhang et al. (2016) results in a 2.0% accuracy drop. An additional fine-tuning is explicitly applied to meet the final 1.0% accuracy drop with 5 more epochs and the learning rate of 10^{-5}. Our method only relies on vanilla fine-tuning to reach the low accuracy drop. The learning rate we used is 10^{-3}.

The CP decomposition can also compress the filter size effectively. With the Rank-Selection configuration, our approach reduces 85% of the convolutional layers (from 57MB to 8.3MB). However, due to the large size of fully connected layers (about 470MB) in VGG16, the overall reduction of the entire networks is 9% (from 528MB to 480MB) after the two-pass decomposition flow.

Table 4: Comparison of the accuracy drop among different configurations with the convergence constraint for VGG16 (%).

		Iteration 1		Iteration 2		Iteration 3	
		Optimize	Fine-tune	Optimize	Fine-tune	Optimize	Fine-tune
In-Order (rs)	Accuracy Drop	-0.15	-0.14	0.47	0.57	1.48	1.59
	Epochs	1	1	4	2	1	1
Fitness-Based (rs)	Accuracy Drop	0.32	0.44	0.97	1.04	1.17	1.20
	Epochs	8	2	1	1	2	2

Table 5: Comparison among different works.

Method	Actual Speedup	Accuracy Drop (%)
Jaderberg et al. (2014) (reported in Zhang et al. (2016))	4×	9.70%
Zhang et al. (2016)	4×	3.84%
Zhang et al. (2016) 3d	5×	2.00%
Zhang et al. (2016) 3d (fine-tuned)	5×	1.00%
In-Order (rs)	6.2×	1.59%
Fitness-Based (rs)	6.2×	1.20%

3.2 DECOMPOSING RESNET50

The deeper CNN, ResNet50, is also evaluated. The accuracy of the pre-trained model which is modified from ethereon (2016) is 92.02% with the same training environment of the previous experiment, except for the learning rate of 10^{-5}.

In this experiment, the 1×1 convolutional layers are excluded for the decomposition. The initial rank is set to 50 for all target layers. The result of our iterative decomposition is shown in Table 6. Each iteration is trained with one epoch. After the iterative two-pass decomposition, the measured speedup is 1.35× and the model size reduces by 48% (98MB→51MB).

Table 6: Accuracy of ResNet50 using the iterative two-pass decomposition (%)

Accuracy	Iteration 1		Iteration 2		Iteration 3	
	Optimize	Fine-tune	Optimize	Fine-tune	Optimize	Fine-tune
In-Order	91.40	91.42	90.84	90.90	90.32	90.51

3.3 DISCUSSION

Astrid & Lee (2017) suggests that freezing layers make the fine-tuning greedy, which might cause the optimization to be stuck in a local minimal. However, our approach works efficiently by iteratively decomposing a group of layers with previously layers frozen. The attempt also makes the optimization converge faster and prevents from the gradient explosion.

Our first evaluation in Table 3 shows that the In-Order grouping scheme performs better in a typical training and fine-tuning, especially for the Baseline rank configuration. The In-Order scheme may be suitable for the decomposition with smaller rank configuration. Because the Fitness-Based approach decomposes those layers of smaller ranks first. But it also makes the accuracy drop quickly in the former layers of small ranks.

For the Rank-Selection configuration, the Fitness-Based scheme does not lead to a significant accuracy drop due to the relatively higher ranks and fitnesses. The layers of lower fitness are decomposed first. Their accuracy drop can be compensated by fine-tuning the latter layers with higher fitness. The result in Table 4 also shows that a better accuracy drop may be achieved with a proper convergence constraint.

4 CONCLUSION

The iterative two-pass decomposition flow has been presented to accelerate existing deep CNNs. Our two-pass decomposition effectively prevents from the CP instability. The Rank Selection algorithm provides the fine-grained rank configuration to achieve the target speedup while maintaining the accuracy. The experiment results show that VGG16 can be accelerated by 6.2 times with the accuracy drop of only 1.20% and the size reduction of 85%. In addition, ResNet50 can be speeded up by 1.35 times with the accuracy drop of 1.51% and the size reduction of 48%.

The future works include the improvement of the grouping scheme and the decomposing order, and a smarter rank selection with non-linear fitness estimation. In addition, accelerating 1×1 convolutional layers will also be considered for the further improvement on the advanced CNNs.

REFERENCES

M. Astrid and S.-I. Lee. CP-decomposition with Tensor Power Method for Convolutional Neural Networks Compression. *ArXiv e-prints*, January 2017.

E. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting Linear Structure Within Convolutional Networks for Efficient Evaluation. *ArXiv e-prints*, April 2014.

ethereon. Caffe to tensorflow. `https://github.com/ethereon/caffe-tensorflow`, 2016.

S. Han, H. Mao, and W. J. Dally. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *ArXiv e-prints*, October 2015.

K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. *ArXiv e-prints*, December 2015.

A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *ArXiv e-prints*, April 2017.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 29*, pp. 4107–4115. Curran Associates, Inc., 2016. URL `http://papers.nips.cc/paper/6573-binarized-neural-networks.pdf`.

F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and ¡ 0.5MB model size. *ArXiv e-prints*, February 2016.

M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up Convolutional Neural Networks with Low Rank Expansions. *ArXiv e-prints*, May 2014.

Henk A. L. Kiers. Towards a standardized notation and terminology in multiway analysis. *JOURNAL OF CHEMOMETRICS*, 2000.

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 e-prints*, November 2015.

Tamara G. Kolda and Brett W. Bader. Tensor Decompositions and Applications. *SIAM, Society for Industrial and Applied Mathematics*, 51(3):455–500, 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (eds.), *Advances in Neural Information Processing Systems 25*, pp. 1097–1105. Curran Associates, Inc., 2012. URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

A. Lavin and S. Gray. Fast Algorithms for Convolutional Neural Networks. *ArXiv e-prints*, September 2015.

V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky. Speeding-up Convolutional Neural Networks Using Fine-tuned CP-Decomposition. *ArXiv e-prints*, December 2014.

H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning Filters for Efficient ConvNets. *ArXiv e-prints*, August 2016.

machrisaa. Tensorflow vgg16 and vgg19. https://github.com/machrisaa/tensorflow-vgg, 2016.

Maximilian Nickel. scikit-tensor. https://github.com/mnick/scikit-tensor, 2016.

Jongsoo Park, Sheng Li, Wei Wen, Ping Tak Peter Tang, Hai Li, Yiran Chen, and Pradeep Dubey. FASTER CNNS WITH DIRECT SPARSE CONVOLUTIONS AND GUIDED PRUNING. *ICLR,International Conference on Learning Representations*, 2017.

M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. *ArXiv e-prints*, March 2016.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *ArXiv e-prints*, September 2014.

N. Vasilache, J. Johnson, M. Mathieu, S. Chintala, S. Piantino, and Y. LeCun. Fast Convolutional Nets With fbfft: A GPU Performance Evaluation. *ArXiv e-prints*, December 2014.

M. Wang, B. Liu, and H. Foroosh. Design of Efficient Convolutional Layers using Single Intra-channel Convolution, Topological Subdivisioning and Spatial "Bottleneck" Structure. *ArXiv e-prints*, August 2016.

W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning Structured Sparsity in Deep Neural Networks. *ArXiv e-prints*, August 2016.

Xiyu Yu, Tongliang Liu, Xinchao Wang, and Dacheng Tao. On Compressing Deep Models by Low Rank and Sparse Decomposition. *CVPR,Conference on Computer Vision and Pattern Recognition*, 2017.

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*, 38:1943–1955, 2016.

X. Zhang, X. Zhou, M. Lin, and J. Sun. ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. *ArXiv e-prints*, July 2017.

APPENDIX A

Figure 7 shows the initial fitnesses of the baseline rank configuration and fitness-based configuration. Note that the layer order is sorted by the fitnesses of the baseline rank configuration. Because there is only a slight difference between the sorting of the two configuration, the fitness-based grouping scheme is based on the sorting of the baseline rank configuration.

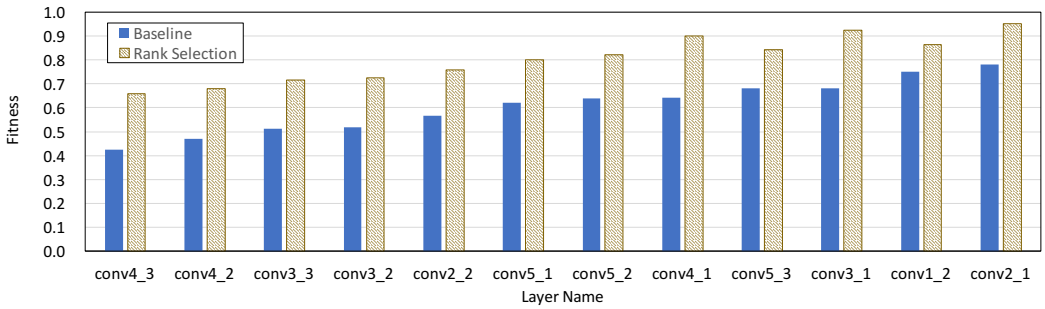

Figure 7: The initial fitnesses of the baseline rank configuration and fitness-based configuration.

Table 7 lists the rank of each layer after the Rank Selection.

Table 7: Rank configuration for ResNet50.

Iteration 1		Iteration 2		Iteration 3	
Layer Name	Rank	Layer Name	Rank	Layer Name	Rank
res2b_branch2b	24	res4b_branch2b	66	res5a_branch2c	250
res2c_branch2b	32	res4c_branch2b	100	res5b_branch2a	199
res3a_branch2b	37	res4d_branch2b	103	res5b_branch2b	153
res3b_branch2b	54	res4e_branch2b	86	res5b_branch2c	250
res3c_branch2b	20	res4f_branch2b	76	res5c_branch2a	250
res3d_branch2b	45	res5a_branch2a	99	res5c_branch2b	128
res4a_branch2b	64	res5a_branch2b	164	res5c_branch2c	250