Accelerating Training using Tucker Decomposition

Mostafa Elhoushi, Ye Henry Tian, Zihao Chen, Farhan Shafiq, Joey Yiwei Li
Huawei Technologies
Toronto Hetrogeneous Compilers Lab
19 Allstate Parkway
Markham, Ontario, L3R 5B4
Canada

Abstract

Tensor decomposition is one of the well-known approaches to reduce the latency time and number of parameters of a pre-trained convolutional neural network (CNN) model. However, in this paper, we propose an approach to use tensor decomposition to reduce training time of training a model from scratch. This paper explores one type of tensor decomposition: Tucker decomposition. In our approach, we train the model from scratch (i.e., randomly initialized weights) with its original architecture for a small number of epochs, then the model is decomposed, and then continue training the decomposed model till the end. There is an optional step in our approach to convert the decomposed architecture back to the original architecture. We present results of using this approach on both CIFAR10 and Imagenet datasets, and show that there can be upto 1.65$\times$ speed up in training time with accuracy drop of upto 0.82% only. The work in this paper is considered one step towards enabling training on IoT devices. This training acceleration approach is independent of hardware and is expected to have similar speed ups on both CPU and GPU platforms. The code is available online at https://github.com/mostafaelhoushi/tensor-decompositions.

Introduction

While deep learning has obtained high accuracy in computer vision, natural language understanding, and many other fields, one of its main challenges is the extensive computation cost required to train its models. A typical neural network architecture may take upto 2 weeks to train on the Imagenet dataset. Using distributed training, a ResNet50 model can train in 6 minutes, but at the financial and energy cost of 1024 GPUs (Jia et al. 2018). The energy required to train the average deep learning model is equivalent to using fossil fuel releasing around 78,000 pounds of carbon, which is more than half of a car's output during its lifetime (Strubell, Ganesh, and McCallum 2019). The financial and time costs of training deep learning models is making it increasingly difficult for small- and medium- sized companies and research labs to explore various architectures and hyperparameters.

While there has been extensive development to meet the increasing computation cost of training by improving hardware design of GPUs - as well as other forms of specialized hardware - and advancements in distributed training using large number of servers and GPUs, there has been less advancement in reducing the computation costs of training. (Schwartz et al. 2019) estimated that from 2012 to 2018 the computations required for deep learning research have estimated 300,000 $\times$.

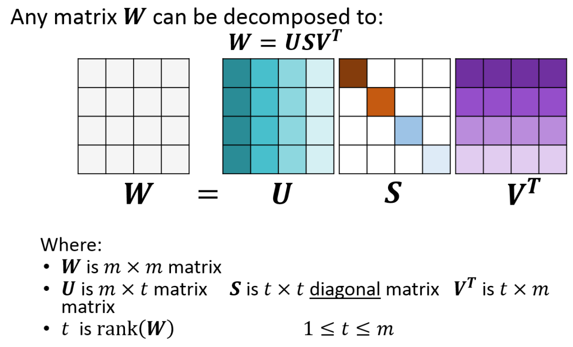
In this paper we propose a hardware independent method to reduce the computation cost of training using tensor decomposition. A lot of research has been made on compressing pre-trained models using tensor decomposition. However, to the best of our knowledge, this paper is the first to propose to use tensor decomposition during training to reduce the computation cost and training time.

In this paper, we will first present related work in reducing the computation cost and latency time of models during inference, followed by related work in reducing training time. Then, we will explain tensor decomposition, and one of its specific methods - Tucker decomposition - that we use in our solution. We then present our proposed solution to decompose the model during training, followed by the results and performances on CIFAR10 and Imagenet datasets.

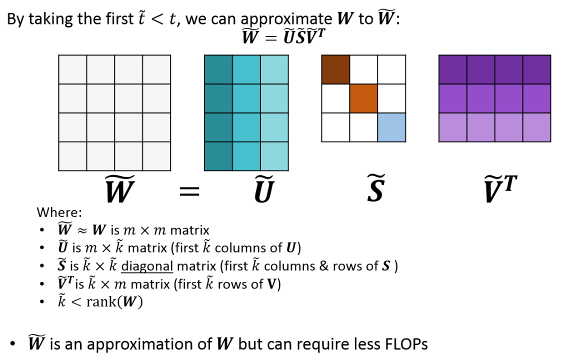
Related Work

Inference Acceleration

Method to reduce CNN model size and speeding up training and/or inference which fall in several categories. First, is searching or designing architectures that have lower number of parameters and hence reduce computation latency time while maintaining reasonable prediction accuracy, e.g., SqueezeNet, MobileNets, and MobileNetV2. Second, is replacing a portion or all of convolution operations in a model with operations that require less computation time or less parameters, e.g., Binarized Neural Networks (Courbariaux and Bengio 2016). Third, is to quantize the parameters of regular convolution operations from 32-bit floating point presentation to smaller number of bits, such as 8-bit integers. Fourth is pruning: removing a portion of convolution filters in each layer - usually - based on some heurestic, e.g., (Liu et al. 2019). Nevertheless, although training a pruned architecture is usually faster than training the original model, obtaining the pruned architecture in the first place requires training the full model in the first place till the end, while our proposed


Any matrix $\boldsymbol{W}$ can be decomposed to:
$\boldsymbol{W} = \boldsymbol{USV^T}$
$\boldsymbol{W}$
$=$
$\boldsymbol{U}$
$\boldsymbol{S}$
$\boldsymbol{V^T}$
Where:
• $\boldsymbol{W}$ is $m \times m$ matrix
• $\boldsymbol{U}$ is $m \times t$ matrix $\boldsymbol{S}$ is $t \times t$ diagonal matrix $\boldsymbol{V^T}$ is $t \times m$ matrix
• t is rank($\boldsymbol{W}$) $1 \leq t \leq m$

(a) Exact Decomposition


By taking the first $\tilde{t} < t$, we can approximate $\boldsymbol{W}$ to $\widetilde{\boldsymbol{W}}$:
$\widetilde{\boldsymbol{W}} = \widetilde{\boldsymbol{U}}\widetilde{\boldsymbol{S}}\widetilde{\boldsymbol{V}}^{\boldsymbol{T}}$
$\widetilde{\boldsymbol{W}}$
$=$
$\widetilde{\boldsymbol{U}}$
$\widetilde{\boldsymbol{S}}$
$\widetilde{\boldsymbol{V}}^{\boldsymbol{T}}$
Where:
• $\widetilde{\boldsymbol{W}} \approx \boldsymbol{W}$ is $m \times m$ matrix
• $\widetilde{\boldsymbol{U}}$ is $m \times \tilde{k}$ matrix (first $\tilde{k}$ columns of $\boldsymbol{U}$)
• $\widetilde{\boldsymbol{S}}$ is $\tilde{k} \times \tilde{k}$ diagonal matrix (first $\tilde{k}$ columns & rows of $\boldsymbol{S}$)
• $\widetilde{\boldsymbol{V}}^{\boldsymbol{T}}$ is $\tilde{k} \times m$ matrix (first $\tilde{k}$ rows of $\mathbf{V}$)
• $\tilde{k} <$ rank($\boldsymbol{W}$)
• $\widetilde{\boldsymbol{W}}$ is an approximation of $\boldsymbol{W}$ but can require less FLOPs

(b) Approximate Decomposition

Figure 1: Singular Value Decomposition.

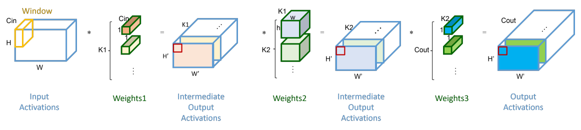


Figure 2: Tucker Decomposition

solution only requires training the first 10 or 30 epochs before compressing.

The fifth categroy is tensor decomposition: separating a regular convolution operation into multiple smaller convolutions whose combined number of parameters or combined latency is less than that of the original operation. This will be explained in the next subsection.

Tensor Decomposition

Tensor decomposition is based on a concept of linear algebra known as Singular Value Decomposition (SVD) that states that any matrix, W, whose dimensions are - without loss of generality - $m \times m$ can be expressed as:

$$W = USV^T \tag{1}$$

As shown in Figure 1, the dimensions of U, S, and V are $m \times k$, $k \times k$, $m \times k$. S is referred to as the unitary matrix and it is a diagonal matrix. Each value along the diagonal represent the "importance" of the corresponding column of U and row of V. The SVD algorithm specifies how to calculate the values of U, S, V matrices in order to hold this equality. k is known as the rank of the matrix W. Most of the time, the rank of a $m \times m$ matrix is $K = m$. If one or more rows and columns of the matrix are linearly dependent on other rows and matrices in the matrix, then the rank is $k < m$. However, in the context of neural networks where the values of the weight matrices are updated during training, this condition is unlikely to happen.

In order to decompose W into terms that have fewer number of parameters, we need to set the rank of the transformation $\tilde{k}$ to be less than the rank, k of the matrix W:

$$W \approx \tilde{U}\tilde{S}\tilde{V}^T \tag{2}$$

This results in an approximation. In the extreme case of choosing the rank of decomposition $\tilde{k} = 1$, the matrix can be represented as $m \times 1 + 1 \times m = 2m$ compared to $m \times m = m^2$ parameters of the original W matrix. For large values of m, $2m << m^2$ and hence the storage of the decomposed representation and FLOPs of matrix operation on the decomposed representation is much lower.

The rank has to be selected in an optimal manner in order to balance between the approximation error introduced and the compression obtained. Different methods of rank selection are explained in the following sub-sub-section.

In the context of neural networks, tensor decomposition extends SVD to the 4-dimensional matrices of weights of convolution operators that have dimensions: $C_{out} \times C_{in} \times h \times w$, where C_{out} is the number of channels of the output of the operator, C_{in} is the number of channels of the input image, h is the height of each filter, and w is the width of each filter. There are various types of tensor decompositions: spatial decomposition (Lin et al. 2018), channel decomposition (Zhang et al. 2016), depthwise decomposition (Guo et al. 2018), Tucker decomposition (Tucker 1966). The mathematical expression of each decomposition type and the their derivations from SVD is out of the scope of this paper but they can be found in the reference of each decomposition method. This paper uses Tucker decomposition, and will be explained in more detail in Section .

Rank Selection Some researchers have used time-consuming trial-and-error to select the optimal rank of decomposition of each convolution layer in a network, by analyzing the final accuracy of the model. (Denton et al. 2014) used alternate least squares. (MacKay 1991) proposed a data-driven one-shot decision using *empirical Bayes*. In this paper, we used variational Bayesian matrix factorization (VBMF) (Nakajima et al. 2012).

Training Acceleration

To speed up training neural networks, the main research efforts in industry gear towards designing and enhancing hardware platforms, e.g., vector units in CPUs, NVIDIA's Graphical Processing Units (GPUs), and Googles Tensor Processor Unit (TPU). Another main method to accelerate training is distributed training: training over multiple GPUs on the same workstation, or training over multiple workstations, with each workstation having one or more GPUs.

Our proposed method in accelerating training is hardware independent, i.e., does not require specific hardware design. It can build upon the speed ups by faster hardware designs and distributed approaches.

Other hardware independent approaches in literature include (Sun et al. 2017)that presented a method to speed up training by only updating a portion of the weights during each backpropagation pass. However, the results are only shown for the basic MNIST dataset. (de Gusmo et al. 2016) presented an approach to accelerate training by starting with downsampled kernels and input images to a certain number of epochs, before upscaling to the original input image size and kernel size.

Proposed Method

We use the end-to-end tensor decomposition scheme similar to that proposed by (Kim et al. 2016) that in turn uses the Tucker decomposition algorithm proposed by (Tucker 1966) and the the rank determined by a global analytic solution of variational Bayesian matrix factorization (VBMF) (Nakajima et al. 2012):

1. **Initialize Model**: We start with a model architecture from scratch (i.e., initialized with random weights).
2. **Initial Training**: We then train the weights of the model for a certain number of epochs, e.g., 10 epochs.
3. **Decompose**: Then, we decompose the model and its weights using Tucker decomposition and VBMF rank selection. The decomposed model has a smaller number of weights then the original model, and hence lower training (and inference time), and a sudden drop in accuracy is expected at that point.
4. **Continue Training**: Then, we continue updating those decomposed weights till the end of training.
5. **Reconstruction** [Optional]: Before the end of training by a certain number of epochs (e.g., 10) we reconstruct the original architecture by combining the weight matrices of each set of decomposed convolution operations. The accuracy at this point does not change, as this reconstruction step is lossless.
6. **Fine Tuning** [Optional]: Train the reconstructed model for a few more epochs.

Tucker decomposition is illustrated in Figure 2. To explain Tucker decomposition, we first express a regular convolution operation of weight $\boldsymbol{W}$ with dimensions $C_{out} \times C_{in} \times h \times w$, acting on an input tensor $\boldsymbol{I}$ with dimensions $H \times W \times C_{in}$ to produce an output tensor $\boldsymbol{O}$ with dimensions $H' \times W' \times C_{out}$:

$$\boldsymbol{O}_{x',y',c_{out}} = \sum_{i=1}^{h}\sum_{j=1}^{w}\sum_{c=1}^{C_{in}} \boldsymbol{W}_{i,j,c_{in},c_{out}} I_{x_i,y_j,c_{in}} \tag{3}$$

where:

$$x_i = (x'-1)S + i - P \tag{4}$$
$$y_j = (j'-1)S + j - P \tag{5}$$
$$\tag{6}$$

where S is the stride and P is the padding.

Tucker decomposition converts this convolution operation into 3 consecutive convolutions:

$$\boldsymbol{O^{(1)}}_{x,y,k_1} = \sum_{c=1}^{C_{in}} \boldsymbol{W^{(1)}}_{c,k_1} \boldsymbol{I}_{x,y,c} \tag{7}$$

$$\boldsymbol{O^{(2)}}_{x',y',k_2} = \sum_{i=1}^{h}\sum_{j=1}^{w}\sum_{k_1=1}^{K_1} \boldsymbol{W^{(2)}}_{i,j,k_1,k_2} \boldsymbol{O^{(1)}}_{x_i,y_j,k_1} \tag{8}$$

$$\boldsymbol{O}_{x',y',c_{out}} = \sum_{k_2=1}^{K_2} \boldsymbol{W^{(3)}}_{c_{out},k_2} \boldsymbol{O^{(2)}}_{x',y',k_2} \tag{9}$$

The first and third convolutions are pointwise convolutions, while the second convolution is a regular spatial convolution with input channels and output channels reduced to K_1 and K_2 respectively. The Tucker decomposition method described in (Tucker 1966) derives the equations to deduce the weights $W^{(1)}$, $W^{(2)}$, $W^{(3)}$ from W. The compression ratio of the decomposition is expressed as:

$$M = \frac{hwC_{in}C_{out}}{C_{in}K_1 + hwK_1K_2 + C_{out}K_2} \tag{10}$$

and

$$E = \frac{hwC_{in}C_{out}H'W'}{C_{in}K_1HW + hwK_1K_2H'W' + C_{out}K_2H'W'} \tag{11}$$

Due to incorporating the height and width of the input and output tensors into the numerator and denominator of the speedup equation, we will notice that speedup in training time is lower than compression ratio.

The values of K_1 and K_2 are determined by the rank selection method, which is in our case is VBMF. It is out of scope to explain the method here.

It is noteworthy that unlike other compression methods such as pruning and distillation, tensor decomposition can be reversed to retain the original architecture without a change in accuracy in a straightforward manner: by simply performing matrix multiplication of the decomposed matrices. The last 2 steps in the process are only to show that the overall training process can retain the original, in case if someone would like to use the original architecture - due to some reason - rather than the decomposed smaller architecture.

Experiments

We have tested our approach by training VGG19, DenseNet40, ResNet56 on CIFAR10 dataset, and ResNet50 in Imagenet dataset. When training on CIFAR10, the batch size used was 128, and the learning rate was initialized at 0.1, reduced to 0.01 at the 100th epoch, and to 0.001 at the 150th epoch. When training on Imagenet, the batch size was 256 and the learning rate was set to 0.1.

In addition to training from scratch the original model, as well as training the decomposed model at an early epoch, and the reconstructed model at a late epoch, we also decomposed the original model after it completed training, and fine-tuned for an additional number of epochs. We did this to compare the accuracy and number of parameters of a model decomposed early during training versus a model decomposed after it completed training.

The results are shown in Tables 1, 2, 3 and 4 and Figures 3, 4, 5, and 6. In those tables, "Dec." is abbreviation for decomposed, and "Rec." is abbreviation for reconstructed.

Results

For VGG19 on CIFAR10, we notice from Table 1 that there was more than $2\times$ speedup in training time, a $20\times$ compression of parameter size, but a drop of almost 2% when decomposed from the 10th epoch. From Figure 3 we notice, a sudden drop in accuracy when decomposition happens, however that drop is compensated for after less than 1000 seconds. When decomposing from later epochs, there was a general trend of decreasing accuracy drop in return for a reduction in model size compression. It may seem that in later epochs, the EVBMF detects more noise in the weight values - as the weights try to fit the training data with higher accuracy and cover more corner cases - and hence selects a higher rank for decomposition. Surprisingly, the scenarios for reconstructing at the 190th epoch, and for decomposing after complete training, did not result in higher best accuracy than decomposing at the 10th epoch without reconstruction.

For DenseNet40 on CIFAR10, we notice a similar drop in accuracy as in VGG19, but less training speedup, despite more than $3\times$ reduction in the number of parameters. This is expected from the compression and speedup ratios expressed in Equations 10 and 11. The results for DenseNet40 also show that both accuracy and model compression are higher for decomposing during training than decomposing after training.

On the other hand, as shown in Table 3, decomposing ResNet56 resulted in an increase accuracy, but less than 10% reduction in training time. This is due to the existence of a large portion of pointwise convolutions that are not possible to decompose using Tucker decomposition.

For Imagenet dataset, the drop in accuracy was less than 0.2% for ResNet50 but the reduction in training time was negligible. Furthermore, decomposing at the 30th epoch resulted in better accuracy than decomposing after complete training.

Conclusion and Future Work

In this paper we have shown that to compress a model using tensor decomposition, we do not have to wait till training ends. We have shown the decomposing at the 10th or 20th epoch of training, results in accuracy close to - and sometimes higher than - that of the original model trained till the end.

We have also shown that in all of the cases on CIFAR10 dataset, the size of a model decomposed after 10 or 20 epochs of training is smaller than that of the model decomposed after complete training. Moreover, we have shown - for CIFAR10 - that training a decomposed model for

Table 1: Performance and size of VGG19 with different scenarios of training on NVIDIA Tesla K40c GPU on CIFAR10 dataset. The epoch at which decomposition or reconstruction happens is mentioned. The total number of epochs for all scenarios is 200, except for the last case where decomposition happens after the 200th epoch, and fine tuned for another 40 epochs.

Model	Accuracy		Params	Training Time
	Best	Final		
Original	93.55%	93.56%	20×10^6	4.77 hr
Dec. @ 10	91.69%	91.39%	749×10^3	2.32 hr
Dec. @ 20	92.10%	91.89%	1.33×10^6	2.75 hr
Dec. @ 30	91.85%	91.78%	1.69×10^6	2.95 hr
Dec. @ 40	92.57%	92.43%	1.75×10^6	3.04 hr
Dec. @ 50	92.51%	92.49%	1.73×10^6	2.41 hr
Dec. @ 10 Rec. @ 190	91.69%	91.53%	749×10^3	2.45 hr
Original then Dec.	91.52%	91.36%	1.33×10^6	4.77 hr + 0.49 hr

Table 2: Performance and size of DenseNet40 with different scenarios of training and decomposition on NVIDIA Tesla P100 GPU on CIFAR10 dataset. The total number of epochs for all scenarios is 160, except for the last case where decomposition happens after the 160th epoch, and fine tuned for another 40 epochs.

Model	Accuracy		Params	Training Time
	Best	Final		
Original	94.00%	93.78%	1.06×10^6	11.13 hr
Dec. @ 20	92.00%	91.82%	270×10^3	8.62 hr
Dec. @ 20 Rec. @ 150	92.00%	91.96%	1.06×10^6	8.83 hr
Original then Dec.	91.49%	91.46%	441×10^3	11.13 hr + 2.17 hr

Table 3: Performance and size of ResNet56 with different scenarios of training on NVIDIA Tesla K40c GPU on CIFAR10 dataset. The epoch at which decomposition or reconstruction happens is mentioned. The total number of epochs for all scenarios is 200, except for the last case where decomposition happens after the 200th epoch, and fine tuned for another 40 epochs.

Model	Accuracy		Params	Training Time
	Best	Final		
Original	91.83%	91.69%	853×10^3	4.39 hr
Dec. @ 10	92.16%	91.97%	508×10^3	4.06 hr
Dec. @ 20	92.27%	92.07%	520×10^3	4.10 hr
Dec. @ 30	91.66%	91.51%	556×10^3	4.15 hr
Dec. @ 40	91.67%	91.50%	550×10^3	4.14 hr
Dec. @ 50	91.65%	91.15%	550×10^6	4.15 hr
Dec. @ 10 Rec. @ 190	92.16%	91.92%	853×10^3	4.07 hr
Original then Dec.	92.32%	92.22%	547×10^3	4.39 hr + 0.85 hr

Table 4: Performance and size of ResNet50 with different scenarios of training on NVIDIA Tesla K40c GPU on Imagenet dataset. The epoch at which decomposition or reconstruction happens is mentioned. The total number of epochs for all scenarios is 90, except for the last case where decomposition happens after the 90th epoch, and fine tuned for another 20 epochs.

Model	Best Accuracy		Params	Training Time
	Top1	Top5		
Original	75.65%	92.85%	25.6×10^6	185.4 hr
Dec. @ 30	75.34%	92.68%	17.6×10^6	179.2 hr
Original then Dec.	69.26%	89.38%	8.2×10^6	185.4 hr + 20.56 hr

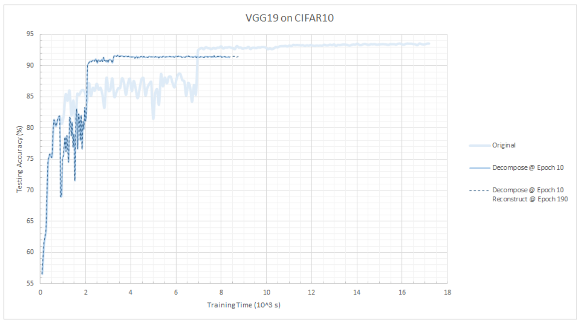


Figure 3: Training progress of VGG19 model on CIFAR10 dataset on NVIDIA Tesla K40c GPU for 200 epochs with batch size 128.

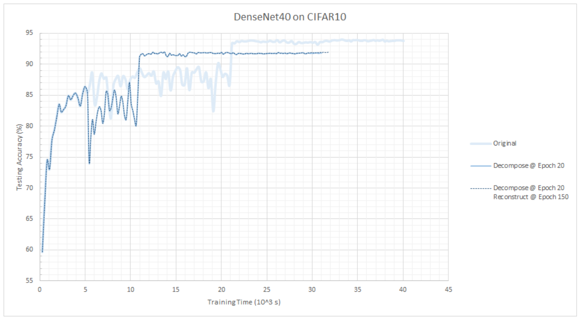


Figure 4: Training progress of DenseNet40 model on CIFAR10 dataset on NVIDIA Tesla P100 GPU for 160 epochs with batch size 128.

VGG and DenseNet architectures results in considerable faster training time: more than $2\times$ for VGG19 and $1.3\times$ for DenseNet40. However, the speedup obtained for ResNet architecture was negligible.

For future work, there is a need to explore ways to reduce the accuracy drop in accuracy for our "decomposition-in-training" approach for some models, and to increase the training speedup for other architectures especially ResNet by looking into solutions to decompose pointwise convolutions.

Acknowledgments

We would like to thank Prof. Andreas Moshovos for providing us the template to design Fig. 2. We would like to thank Jacob Gildenblat (Gildenblat 2018) and Ruihang Du (Du 2018) for providing open-sourced code for Tucker decomposition using PyTorch. We also thank Yerlan Idelbayev for providing open-sourced code and model files to reproduce the accuracies of the original ResNet (Idelbayev 2018), DenseNet, and VGG papers (Mingjie 2018) results on CIFAR10.

References

Courbariaux, M., and Bengio, Y. 2016. Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1. *CoRR* abs/1602.02830.

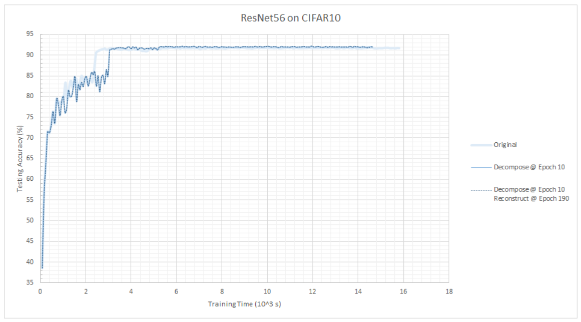


Figure 5: Training progress of ResNet56 model on CIFAR10 dataset on NVIDIA Tesla K40c GPU for 200 epochs with batch size 128.

de Gusmo, P. P. B.; Francini, G.; Lepsy, S.; and Magli, E. 2016. Fast training of convolutional neural networks via kernel rescaling.

Denton, E.; Zaremba, W.; Bruna, J.; LeCun, Y.; and Fergus, R. 2014. Exploiting linear structure within convolutional networks for efficient evaluation. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'14, 1269–1277. Cambridge, MA, USA: MIT Press.

Du, R. 2018. Decompose-cnn. https://github.com/larry0123du/Decompose-CNN. Accessed: 2019-09-05.

Gildenblat, J. 2018. Pytorch tensor decompositions. https://github.com/jacobgil/pytorch-tensor-decompositions. Accessed: 2019-09-05.

Guo, J.; Li, Y.; Lin, W.; Chen, Y.; and Li, J. 2018. Network decoupling: From regular to depthwise separable convolutions. In *British Machine Vision Conference 2018, BMVC 2018, Northumbria University, Newcastle, UK, September 3-6, 2018*, 248.

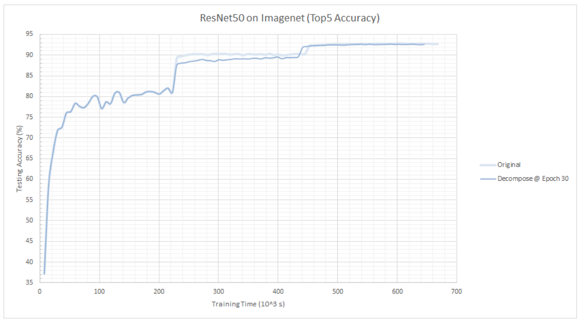
Idelbayev, Y. 2018. Proper resnet implementation for cifar10/cifar100 in pytorch. https://github.com/akamaster/pytorch_resnet_cifar10. Accessed: 2019-05-23.

Jia, X.; Song, S.; He, W.; Wang, Y.; Rong, H.; Zhou, F.; Xie, L.; Guo, Z.; Yang, Y.; Yu, L.; Chen, T.; Hu, G.; Shi, S.; and Chu, X. 2018. Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes. *ArXiv* abs/1807.11205.

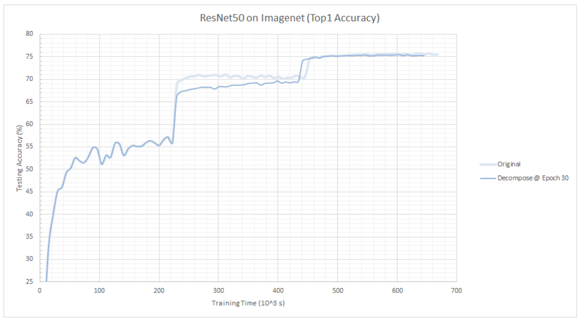
Kim, Y.; Park, E.; Yoo, S.; Choi, T.; Yang, L.; and Shin, D. 2016. Compression of deep convolutional neural networks for fast and low power mobile applications. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.

Lin, S.; Ji, R.; Chen, C.; Tao, D.; and Luo, J. 2018. Holistic cnn compression via low-rank decomposition with knowledge transfer. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 1–1.

Liu, Z.; Sun, M.; Zhou, T.; Huang, G.; and Darrell, T. 2019. Rethinking the value of network pruning. In *ICLR*.



(a) Top5 Accuracy



(b) Top1 Accuracy

Figure 6: Training progress of ResNet50 model on Imagenet dataset on NVIDIA Tesla V100 GPU for 90 epochs with batch size of 256.

MacKay, D. J. 1991. Bayesian interpolation. *NEURAL COMPUTATION* 4:415–447.

Mingjie, E. 2018. Network slimming (pytorch). https://github.com/Eric-mingjie/network-slimming. Accessed: 2019-09-05.

Nakajima, S.; Tomioka, R.; Sugiyama, M.; and Babacan, S. D. 2012. Perfect dimensionality recovery by variational bayesian pca. In Pereira, F.; Burges, C. J. C.; Bottou, L.; and Weinberger, K. Q., eds., *Advances in Neural Information Processing Systems 25*. Curran Associates, Inc. 971–979.

Schwartz, R.; Dodge, J.; Smith, N. A.; and Etzioni, O. 2019. Green AI. *CoRR* abs/1907.10597.

Strubell, E.; Ganesh, A.; and McCallum, A. 2019. Energy and policy considerations for deep learning in nlp. In *ACL*.

Sun, X.; Ren, X.; Ma, S.; and Wang, H. 2017. meProp: Sparsified back propagation for accelerated deep learning with reduced overfitting. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, 3299–3308.

Tucker, L. R. 1966. Some mathematical notes on three-mode factor analysis. *Psychometrika* 31(3):279–311.

Zhang, X.; Zou, J.; He, K.; and Sun, J. 2016. Accelerating very deep convolutional networks for classification and detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 38(10):19431955.