electronics

MDPI

Article

Reducing Parameters of Neural Networks via Recursive Tensor Approximation

Kyuahn Kwon and Jaeyong Chung *

Department of Electronic Engineering, Incheon National University, Incheon 406-772, Korea; gouan3@inu.ac.kr
* Correspondence: jychung@inu.ac.kr

Abstract: Large-scale neural networks have attracted much attention for surprising results in various cognitive tasks such as object detection and image classification. However, the large number of weight parameters in the complex networks can be problematic when the models are deployed to embedded systems. In addition, the problems are exacerbated in emerging neuromorphic computers, where each weight parameter is stored within a synapse, the primary computational resource of the bio-inspired computers. We describe an effective way of reducing the parameters by a recursive tensor factorization method. Applying the singular value decomposition in a recursive manner decomposes a tensor that represents the weight parameters. Then, the tensor is approximated by algorithms minimizing the approximation error and the number of parameters. This process factorizes a given network, yielding a deeper, less dense, and weight-shared network with good initial weights, which can be fine-tuned by gradient descent.

Keywords: tensor approximation; deep learning; machine learning; VLSI

Citation: Kwon, K.; Chung, J. Reducing Parameters of Neural Networks via Recursive Tensor Approximation. *Electronics* **2022**, *11*, 214. https://doi.org/10.3390/electronics11020214

Academic Editors: Andrea Prati, Carlos A. Iglesias, Luis Javier García Villalba and Vincent A. Cicirello

Received: 21 December 2021
Accepted: 9 January 2022
Published: 11 January 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

1. Introduction

Large neural networks such as convolutional neural networks have demonstrated state-of-the-art performance in a number of benchmarks in computer vision, automatic speech recognition, natural language processing, audio recognition, etc. [1–4]. However, these networks have millions and billions of parameters [5], and the evaluation of such models is computationally demanding. While the enormous computing power available today, mainly driven by GPUs, makes us consider the evaluation easy, it comes with large energy consumption. In embedded platforms, the evaluation is still challenging because resources such as computing power, memory, storage, and energy are highly limited. The large energy consumption can also be a critical issue in data centers and can restrict the computing capacity.

Fortunately, weight parameters in neural networks are heavily redundant [6], and exploiting the redundancy, computational cost, and space requirements can be minimized while maintaining the performance. To this end, several methods have been proposed very recently [7–12], and all of these methods assume that neural networks are executed in stored-program computers, including GPU-based machines. The traditional computers have several processing bottlenecks, such as limited memory-bandwidth and a limited number of processing elements, and the performance benefit (e.g., speed-up) by the parameter reduction is not as high as the reduction rate.

Emerging neuromorphic computers emulate biological neural networks, and silicon synapses and neurons are processing elements [13,14]. Connections and units in neural networks are mapped to synapses and neurons, respectively, and these computational resources are not time-multiplexed as in the traditional computers. Thus, the runtime is not a concern because it does not depend on the size of the models. The size of the models affects the required synapses and neurons. To store weight parameters within synapses or nearby, neuromorphic computers use low-density on-chip memories and equip

with at least as many synapses as the parameters [13,15–17]. Without weight-sharing, the number of synapses equals the number of weight parameters. Thus, the number of parameters itself is a very important metric in neuromorphic computers because each weight parameter requires a synpase, a computational resource with its own dedicated memory in neuromorphic systems [18]. In [19], the authors reduced the number of parameters for neuromorphic systems by combining matrix factorization and pruning. Thus, it can be applied to fully connected layers only and cannot take advantage of redundancy across multiple dimensions in high-order tensors.

In this paper, we describe a general parameter reduction method using new tensor approximation methods based on divide-and-conquer [20]. This technique can be applied to store-program computers as well as neuromorphic computers. However, unlike previous works, our main objective was to reduce the size of the models for neuromorphic computers, so we consider the number of parameters as the figure of merit.

This paper makes the following contributions: (1) For neuromorphic computers, we evaluated the methods for reducing the size of the models in terms of the parameter reduction. (2) We approximated a tensor using a more expressive format than the canonical polyadic (CP) format [9], which has been commonly used recently and has shown that higher expressiveness can lead to a better quality of results. (3) For a convolutional network, we compressed all the layers, including fully-connected layers, using a single method and demonstrated that the network with all layers compressed could perform similarly to the original network. (4) We show that the weights of the first fully-connected layers can be tensor-approximated effectively.

2. Related Work

Denil et al. [6] showed that weight parameters are highly redundant by representing a weight matrix as a product of two low-rank matrices. This implies that neural networks are heavily over-parameterized. Their work ignited recent interest in exploiting the massive redundancy to minimize cost at test time.

A group of works [7–9] used the redundancy to speed up the evaluation of convolutional neural networks. They reduced the number of parameters from a trained network by finding low-rank approximations to weight tensors. Jaderberg et al. [7] approximated the 4-tensor representing a convolutional kernel as a composition of two 3-tensors and obtained parameters that minimized the difference between the outputs of the original and approximated layers using training data. Denton et al. [8] combined filter clustering and a 3-tensor approximation method to approximate a 4-tensor. For the first convolutional layer and the higher convolutional layers, different clustering methods were employed, and for the fully connected layers, low-rank matrix approximation was used. They also suggested fine-tuning the layers above the approximated one when the accuracy is degraded after approximation. Between the two 3-tensor approximation methods they used, the one that produced better results approximated a 3-tensor to a sum of separable 3-tensors, which is known as canonical polyadic (CP) decomposition [9]. They demonstrated 2$\times$ speed-ups of convolutional layers within a 1% accuracy drop. Lebedev et al. [9] used CP decomposition for 4-tensors and fine-tuned the whole network. They demonstrated better speed-up (e.g., 4$\times$ speed-up of a convolutional layer) than the previous methods. In the meantime, they also reported a failure in a fine-tuning that they suspected was caused by the instability of CP decomposition. In the CP format, the tensor approximation problem with a given rank is known to be unstable, and robust numerical procedures such as singular value decomposition (SVD) [21] are in effect not available [22].

To achieve the same goal of fast feedforward execution, Jin et al. [10] trained a network with a small number of parameters in the first place. They applied structural constraints to obtain two separable 3-tensors for each output channel in convolutional layers and achieved 2$\times$ speed-up for the forward pass of the whole network. While reducing the number of parameters decreases the storage and memory requirement, parameters in convolutional layers take up a small portion of the total parameters. In order to reduce the memory

footprint specifically, Yang et al. [11] replaced all fully connected layers with a kernel machine. The space overhead could be minimized by finding a compact representation for weights as well as by reducing the number of parameters. Gong et al. [12] performed scalar quantization using k-means clustering for weights in fully connected layers. This type of method can be combined with parameter reduction methods.

Our work differs from previous works in several ways. First, we aimed to minimize the cost in executing neural networks on neuromorphic computers. To this end, we chose the number of parameters as our cost metric. Second, we propose a new tensor approximation algorithm based on divide-and-conquer [20]and achieved up to 154× reduction in weight parameters within a 1% accuracy drop. To the best of our knowledge, such a high reduction rate has not been reported yet. Last, our method does not have certain limitations that some of the recent methods have. Our method is numerically stable and can be applied to both convolutional layers and fully connected layers. Moreover, the resulting reduced network is end-to-end trainable.

3. Tensor Factorization

In a convolutional neural network for computer vision tasks, the weights in a convolutional layer or the first fully connected layer can be represented in a 4th-order tensor, and we propose a tensor factorization (approximation) method for a weight tensor.

3.1. Existing Methods

For high-order tensor decomposition, several methods are known, such as CP decomposition and Tucker decomposition (also known as high-order singular value decomposition) [23]. In the CP format, the tensor approximation problem with a given rank is known to be unstable, and robust numerical procedures such as the SVD are in effect not available [22]. Tucker decomposition can be computed via several SVDs, but the number of parameters after the decomposition still depends on the dimension exponentially. To tackle this issue, new methods such as tensor-train decomposition [24] and hierarchical Tucker decomposition [25] have been proposed. These methods are different from the proposed method, and they involve complex operations that are difficult to implement in neuromorphic systems, with exception of CP decomposition. We compare our method with CP decomposition later in this paper.

3.2. Proposed Factorization Method

Definition 1. *The d-mode matricization of a tensor* $\chi \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$ *is a matrix defined by* $\mathcal{M} : \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d} \to \mathbb{R}^{n_d \times n_1 n_2 \cdots n_{d-1}}$, $(\mathcal{M}(\chi))_{i_d,(i_1,\ldots,i_{d-1})} = \chi_{i_1,i_2,\cdots,i_d}$ *for any indices* $(i_1, \cdots, i_d)$ *in the multi-index set* $\{1, \cdots, n_1\} \times \cdots \times \{1, \cdots, n_d\}$.

Definition 2. *The vectorization of a tensor* $\chi \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$ *is a vector defined by* $\mathcal{V} : \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d} \to \mathbb{R}^{n_1 n_2 \cdots n_d}$, $(\mathcal{V}(\chi))_{(i_1,\ldots,i_d)} = \chi_{i_1,i_2,\cdots,i_d}$ *for any indices* $(i_1, \cdots, i_d)$ *in the multi-index set* $\{1, \cdots, n_1\} \times \cdots \times \{1, \cdots, n_d\}$.

We assume the reverse lexicographical order of the multi-indices throughout this paper. We now propose a tensor factorization method based on divide-and-conquer [20].

Definition 3. *A decomposition tree* $\mathcal{T}$ *of a tensor* χ *is a tree such that (1) each node s is associated with a tensor* $\chi^{(s)}$, *(2) each node has two types of successors, (3) the tensor corresponding to a node is composed of the tensors of its successors of one type, and (4) the root is associated with* χ. *The successor sets are denoted by* S^a *and* S^b, *respectively.*

Given a tensor $\chi \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_e}$, we recursively construct $\mathcal{T}(\chi)$ from the root node. For a node s, the successors of the first type are constructed by the SVD.Let $\chi^{(s)}$ be a d-th order tensor of shape $n_1 \times n_2 \times \cdots \times n_d$. The SVD for the transpose of $\mathcal{M}(\chi)$ yields

$$\mathcal{M}(\chi^{(s)})^T = \mathbf{U}^{(s)} \mathbf{\Sigma}^{(s)} {\mathbf{V}^{(s)}}^T. \tag{1}$$

Let $S^a(s) = \{s_1, \cdots, s_{r^{(s)}}\}$, where $r^{(s)}$ is the rank of $\mathcal{M}(\chi^{(s)})$. Each left-singular vector (i.e., the column vectors of $\mathbf{U}^{(s)}$) is reshaped into a tensor of shape $n_1 \times n_2 \times \cdots \times n_{d-1}$ such that the left-singular vector is the vectorization of the tensor, which results in $r^{(s)}$ (d-1)th order tensors. These are $\chi^{(s_1)}, ..., \chi^{(s_r)}$. Thus, we can write

$$\mathbf{U}^{(s)} = [\mathcal{V}(\chi^{(s_1)}) \cdots \mathcal{V}(\chi^{(s_r)})]. \tag{2}$$

For the node s, the successors of the second type are constructed by subtensors. Let $S^b(s) = \{s'_1, \cdots, s'_{n_d}\}$. We obtain the subtensors by fixing the highest order index (i.e., the d-mode), and they become $\chi^{(s'_1)}, ..., \chi^{(s'_{n_d})}$. Thus, we produce $r^{(s)} + n_d$ one-level lower-order tensors in total using both the methods. Given a tensor, we apply this procedure recursively until it produces vectors (i.e., 1st-order tensors). Since we break the same tensors in the two ways during the recursion, we have two possible factored forms for each tensor and choose either one of them. This allows us to explore numerous factored forms and make our model very expressive.

Let us denote the number of parameters of the tensor associated with a node s by $\eta^{(s)}$. Since $\mathbf{\Sigma}^{(s)}$ can be collapsed into either $\mathbf{U}^{(s)}$ or $\mathbf{V}^{(s)}$, the number of parameters $\eta^{(s)}$ in $\chi^{(s)}$ after the factorization becomes $\Sigma_i^{r^{(s)}} \eta^{(s_i)} + r^{(s)} n_d$ if the SVD form is taken. If the subtensor form is taken, $\eta^{(s)} = \Sigma_i^{n_d} \eta^{(s'_i)}$.

Let $x^{(s)}_{i_1,\cdots,i_d} = \chi^{(s)}_{i_1,\cdots,i_d}$, $v^{(s)}_{i_d,j} = \mathbf{V}^{(s)}_{i_d,j}$, and $\lambda^{(s)}_j = \mathbf{\Sigma}^{(s)}_{j,j}$. Our tensor decomposition can also be written in an elementwise form as

$$x^{(s)}_{i_1,i_2,\cdots,i_d} = \alpha^{(s)} \sum_j^{r^{(s)}} \lambda^{(s)}_j x^{(s_j)}_{i_1,\cdots,i_{d-1}} v^{(s)}_{i_d,j} + (1 - \alpha^{(s)}) x^{(s'_{i_d})}_{i_1,\cdots,i_{d-1}}, \tag{3}$$

where $\alpha^{(s)}$ is a binary free variable that can be set to either one or zero. The decomposition for a 3rd-order tensor is depicted in Figure 1.

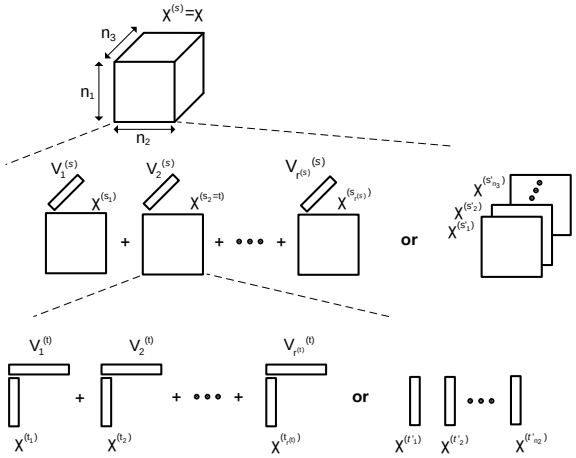


Figure 1. A 3rd-order tensor of shape $n_1 \times n_2 \times n_3$ is decomposed into $r^{(s)} + n_3$ tensors of shape $n_1 \times n_2$. The $r^{(s)}$ tensors are obtained by the SVD, and the n_3 tensors are the 3-mode subtensors. All the 2nd-order tensors are decomposed again in the same manner into 1st-order tensors.

3.3. Low-Rank Tensor Approximation

The parameter reduction by the factorization is usually possible when $r^{(s)}$ is small. Thus, we perform low-rank approximation with the factorization. Since we break a tensor down into smaller, lower-order tensors, the problem of approximating a tensor is now solved by combing small, approximated tensors into a larger tensor. If a tensor is broken down by the SVD, the approximation $\tilde{\chi}^{(s)}$ to the original tensor is obtained as follows: (i) small singular values are replaced by zero, and (ii) $\chi^{(s_i)}$ is approximated by $\tilde{\chi}^{(s_i)}$. If a tensor is broken down into the n_d subtensors, its approximation is simply obtained by combing the approximations to the n_d subtensors. Since a tensor is broken down in both ways, we can choose one of the two approximations to the tensor.

To approximate the given tensor χ using the decomposition, we zero-out small singular values in a recursive manner. For this, we introduce another binary variable $\beta^{(s)}$ and write the approximated tensor $\tilde{\chi}^{(s)}$ in an elementwise form as

$$\tilde{x}^{(s)}_{i_1,i_2,\cdots,i_d} = \alpha^{(s)} \sum_{j}^{r^{(s)}} \lambda^{(s)}_j \beta^{(s)}_j \tilde{x}^{(s_j)}_{i_1,\cdots,i_{d-1}} v^{(s)}_{i_d,j} + (1-\alpha^{(s)}) \tilde{x}^{(s'_{i_d})}_{i_1,\cdots,i_{d-1}}, \tag{4}$$

Unlike the standard low-rank approximation method, we can zero-out a singular value when a smaller singular value than that remains. That is, we do not necessarily zero-out trailing singular values. Thus, we use the set of the binary values $\beta^{(s)}_1, \cdots, \beta^{(s)}_{r^{(s)}}$ instead of the target rank. We define the approximation error by

$$\epsilon^{(s)} \triangleq \|\chi^{(s)} - \tilde{\chi}^{(s)}\|^2_F. \tag{5}$$

where $\|\|_F$ is the Frobenius norm. Then, the approximation error by the proposed method is

$$\epsilon^{(s)} = \alpha^{(s)} \sum_{j=1}^{r^{(s)}} {\lambda^{(s)}_j}^2 \left(\beta^{(s)}_j + (1-\beta^{(s)}_j)\epsilon^{(s_j)}\right) + (1-\alpha^{(s)}) \sum_{j=1}^{n_d} \epsilon^{(s'_j)} \tag{6}$$

since $\mathbf{U}^{(s)}$ and $\mathbf{V}^{(s)}$ are orthogonal matrices.

4. Dual-Objective Optimization

The choices between the two possible approximations and the decisions on the zero-out through the recursion lead to various candidate solutions to the problem of approximating a tensor. Our objective is to minimize both the approximation error and the number of parameters. For this dual-objective optimization, we use two algorithms. The first algorithm finds Pareto optimal solutions in a recursive manner with several pruning methods. The second algorithm is a greedy algorithm that takes a threshold value as input and determines singular values to zero out using the value. This algorithm chooses the one with a smaller cost between the two possible approximations.

4.1. Pareto Front Method

To tackle the optimization problem, we compute the Pareto front in a recursive manner. A tuple (l,m) corresponds to a possible approximation to a tensor, where l and m are the cost and the approximation error, respectively, representing the results of the approximation. We call the tuple the *option*. As an inductive hypothesis, we suppose that for each $\chi^{(s_1)}, \cdots, \chi^{(s_{r^{(s)}})}$ generated by $\chi^{(s)}$, we have a Pareto optimal set of options in Figure 2. Since the approximation to $\chi^{(s)}$ is a composition of the approximations to the one-level lower-order tensors, an option for $\chi^{(s)}$ is associated with a combination of options, each from each Pareto optimal set. In order to obtain the set of options for $\chi^{(s)}$ from all the Pareto sets, the approximation errors in the Pareto set for $\chi^{(s_j)}$ are multiplied by $\lambda^{(s)}_j$ for $j = 1, \cdots, r^{(s)}$. Then, the set of options for $\chi^{(s)}$ is the Minkowiski sum of all the Pareto optimal sets. We also suppose that for each d-mode subtensor of $\chi^{(s)}$, we have a Pareto optimal

set of options. We also perform the Minkowiski operation. Then, we have two option sets for $\chi^{(s)}$. Since we can choose either one of the approximations, we perform the union operation for the two sets and non-Pareto optimal options are discarded, resulting in the Pareto optimal set for $\chi^{(s)}$. In this manner, we obtain the final Pareto optimal set for χ. When $\chi^{(s)}$ is a matrix, we just perform the $k^{(s)}$-rank approximation for $k^{(s)} = 1, ..., \lfloor n_1 n_2/(n_1 + n_2) \rfloor$, which yields a Pareto optimal set. A larger rank than these values results in a larger cost than the non-approximated one.

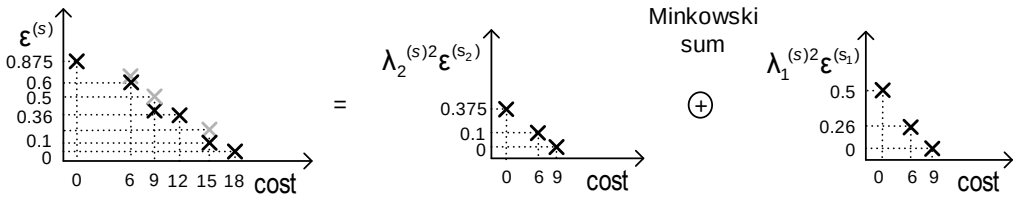


Figure 2. For a tensor of shape $3 \times 3 \times 2$, the merge of Pareto optimal sets is illustrated, assuming $\alpha^{(s)} = 1$. It implies that ${\lambda_1^{(s)}}^2 = 0.5$ and ${\lambda_2^{(s)}}^2 = 0.375$ since $\|\chi^{(s_1)}\|_F = \|\chi^{(s_2)}\|_F = 1$. The option (6, 0.6) is associated with the case that $\beta_1^{(s)} = 0$ and $\beta_2^{(s)} = 1$, showing that zeroing out trailing singular values can result in sub-optimal solutions. This occurs since ${\lambda_1^{(s)}}^2 {\lambda_1^{(s_1)}}^2 < {\lambda_2^{(s)}}^2 {\lambda_1^{(s_2)}}^2$.

4.2. Greedy Method

We also use a greedy algorithm for the optimization problem. This algorithm takes a threshold value τ as input and zero-outs trailing singular values as the standard low-rank approximation method, and the target rank is denoted by $k^{(s)}$. We determine $k^{(s)}$ based on the approximation error to χ per unit cost. Thus, we set $k^{(s)}$ to the largest integer $j \leq r^{(s)}$ such that

$$\frac{\psi^{(s_j)}}{n_1 \times n_2 \times \cdots \times n_{d-1} + n_d} > \tau \tag{7}$$

where $\psi^{(s_j)} = {\lambda_j^{(s)}}^2 \psi^{(s)}$, $\psi^{(s'_j)} = \psi^{(s)}$, and $\psi^{(\phi)} = 1$. To determine $\alpha^{(s)}$, we compare the costs of the two possible approximations and take the one with a smaller cost.

5. Application to Weight Tensors

Let $\chi \in \mathbb{R}^{K_h \times K_w \times C \times F}$ be the weight tensor of a convolutional layer (a fully connected layer), where K_h and K_w are the height and width of the kernel (the input map), respectively, C is the number of input channels, and F is the number of output feature maps. We assume that this tensor is approximated by the greedy method without loss of generality, and we further assume that $\alpha^{(s)} = 1$ for all nodes s in $\mathcal{T}(\chi)$. Let L_i be the set of the nodes at level i in $\mathcal{T}(\chi)$. Then we define

$$R_i = \sum_{s \in L_i} k^{(s)} \tag{8}$$

for $i = 1, \cdots, 3$. In the greedy method, we can enforce $k^{(s)}$ to 1 for every node $s \in L_2 \cup L_3$. Then, our method comes to use the CP-format, and we have $R_1 = R_2 = R_3$. Our decomposition method factorizes the original layer into five sublayers. If we represent the weights of each sublayer in the same format as those of the original layer, the weight tensors of the sublayers are of shape $1 \times 1 \times C \times R_2$, $K_h \times 1 \times R_2 \times R_3$, $1 \times K_w \times R_3 \times R_3$, $1 \times 1 \times R_3 \times R_1$, and $1 \times 1 \times R_1 \times F$, respectively. If the original layer is convolutional, the sublayers are also convolutional. Note that these layers are sparsely connected across channels.

Fine-Tuning

Artificial neural networks are known to be error tolerant. As we reduce the number of parameters, the approximation error increases, but the performance of the networks can remain the same due to the resilience. However, despite the resilience, the performance

of ANNs will drop in the end, as the network diverges from the original one. In the dual-objective optimization, the approximate error is a surrogate metric to evaluate the quality of networks. The true metric is the performance of ANNs, such as classification accuracy. To maximize the performance of ANNs while minimizing the number of parameters, the reduced parameters can be fine-tuned against the training data.

6. Experiments

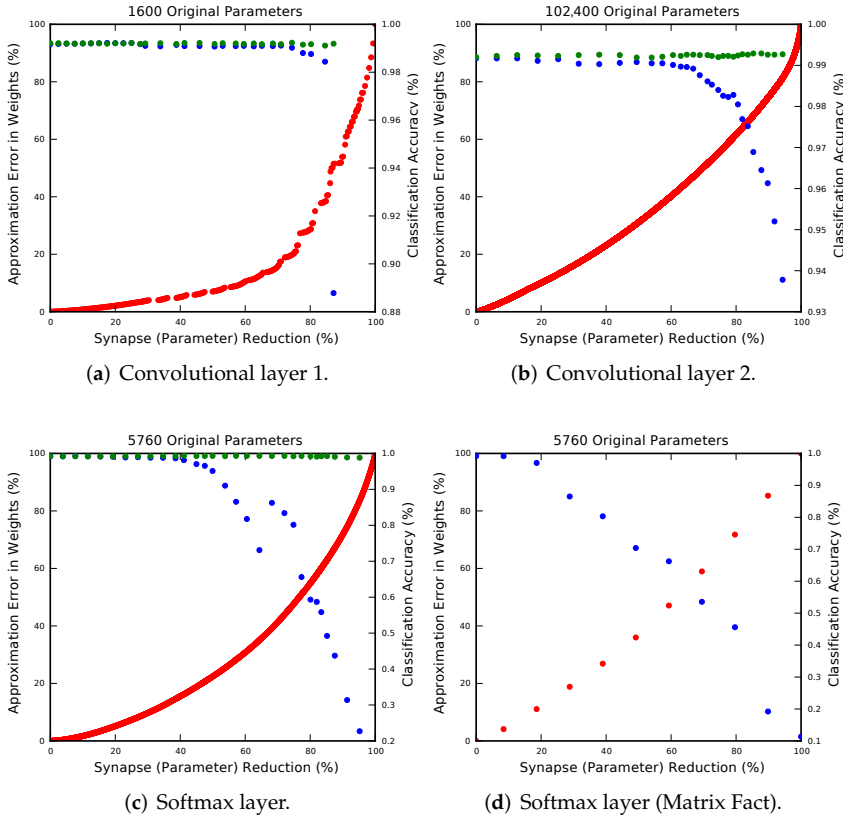
We implemented the proposed algorithms in Python on top of Pylearn2 [26]. The proposed methods were evaluated on MNIST, CIFAR-10, and ILSVRC 2012 image classification datasets. We used a convolutional neural network (CNN) for MNIST. The network had two convolutional layers with 5×5 filters and a final softmax layer. Each convolutional layer included a max-pooling with sizes of 4 and strides of 2 and the rectified linear unit as the activation function. The weight decay was set to 0.00005; the momentum was set to 0.5; the learning rate was set to 0.01. The baseline model had 109,760 parameters and achieved 99.18% classification accuracy after training for 33 epochs. Our method can be applied to any layer, and we first performed layer-wise experiments. The results are shown in Figure 3. The classification accuracy dropped gradually as the approximation error increased, which shows that the surrogate metric to evaluate the quality of networks worked well. The second convolutional layer had 102,400 parameters, which was 93.3% of the total parameters. When we reduced it by 43$\times$ to 2375 parameters, the classification accuracy dropped to 76.45%. However, after fine-tuning the factored network again for 30 epochs, including the other layers, the accuracy was recovered to 99.25%. Thus, we achieved the 43$\times$ reduction without loss of accuracy. If some accuracy loss would have been acceptable, we could have reduced it by 164$\times$ to 627 parameters at the cost of 0.75% accuracy loss.

For the following experiments, we used the greedy algorithm. We used a CNN described in [27] for CIFAR-10. The network had three convolutional maxout layers, a fully connected maxout layer, and a fully connected softmax layer. The baseline model achieved 88.29% accuracy. We performed experiments on the fully connected maxout layer with 12 M parameters, which was 73.9% of the total parameters. We reduced it by 154$\times$ to 77.8 K at the expense of 0.9% accuracy drop.

We used AlexNet [1] implemented in [28] for ILSVRC 2012. The network contained five convolutional layers and three fully connected layers. The baseline model achieved 56.4% top-1 accuracy. We performed experiments on the first fully connected layer with 37.7 M parameters, which was 61.9% of the total parameters. We reduced it by 28$\times$ to 1.3 M parameters at the expense of 0.83% top-1 accuracy drop. We could have reduced it further by 58$\times$ to 654 K parameters by sacrificing 2.25% top-1 accuracy.

Comparison with Existing Parameter Reduction Techniques

Since all the existing methods use different neural network architectures, learning techniques, hyperparameters, and data sets, it is difficult to compare them directly. Nevertheless, we mention some results here that can be roughly compared with our results. Considering that we did not use any adaption and dropout to keep the MNIST experiments simple, our MNIST results may be compared with the jointly trained Fastfood-1024 in [11], which had 38,821 total parameters and achieved 0.83% error. When we applied our techniques to the layer with the largest number of parameters only, the resulting network had 9375 total parameters and achieved 0.75% error. When we applied our technique to all the layers, the resulting network had 3270 total parameters and achieved 0.81% error. The results are summarized in Table 1. The resulting model by our method was very small compared with the other models.



(**a**) Convolutional layer 1. (**b**) Convolutional layer 2.

(**c**) Softmax layer. (**d**) Softmax layer (Matrix Fact).

Figure 3. The Pareto optimal solutions to the dual-objective problem of cost and approximation error are shown in red. For 30 sampled solutions, the classification accuracy is shown in blue, and classification accuracy after re-training is in green. In the re-training, all parameters including the other layers are fine-tuned. Comparing (**c**) with (**d**) shows that it is better to approximate the weights of the layer as a tensor rather than as a matrix.

Table 1. Parameter reduction results on MNIST dataset.

Methods	Layer	Baseline		Reduced Model	
		Params	Error	Params	Error
[11]	Conv. layers FC layer	25,500 405,000	0.87%	25,500 13,321	0.83%
[10]	Conv. layer 1 Conv. layer 2 Conv. layer 3 FC layers	7200 307,200 819,200 NA	0.38%	7200 30,912 102,144 NA	0.44%
CP-decomp.	Conv. layer 1 Conv. layer 2 FC layer	1600 102,400 5760	0.82%	600 828 1920	1.14%
This work	Conv. layer 1 Conv. layer 2 FC layer	1600 102,400 5760	0.82%	588 824 1858	0.81%

For a recent method based on CP decomposition [9], we performed a direct comparison by implementing it within our environments. We used alternating least squares (ALS) for CP decomposition. The method usually finds a better approximation to a given tensor with a same cost budget. This seems to be caused by the facts that (1) we find orthogonal vectors

during the SVD, and (2) our method approximates decomposed tensors independently. Better approximation provides a good initial solution to the fine-tuning after approximation, and the initial solution can be close to the original solution. However, a slightly worse starting point is easily compensated for during the fine-tuning. Our factored form was a generalization to the CP format and was more expressive. This expressiveness paid off after the fine-tuning. We set the target rank of CP decomposition such that the number of parameters would become similar to that of our reduced model. In CP decomposition, the sizes of resulting models are coarse-grained, and the reduced model by CP decomposition was slightly larger than our reduced model. We fine-tuned the approximated model using learning rates 0.01, 0.001, and 0.0001, and the best result was selected, which is also shown in Table 1. In the previous experiment for CIFAR-10, the greedy algorithm achieved 87.42% accuracy when the size of the first fully connected layer was reduced to 77,750 parameters. For the same experiment, the CP decomposition achieved 86.31% accuracy when it was reduced to 78,358 parameters.

7. Discussion

As a part of our research toward building a neuromorphic computer that forward-executes neural networks with minimum energy consumption, we presented a parameter reduction technique based on our own tensor approximation method and achieved up to 154× reduction within negligible loss of performance.

The proposed method provided an enormous reduction rate for convolutional layers and first densely connected layers. For subsequent densely connected layers, the proposed method was reduced to the standard low-rank approximation method. A possible research direction for future works is to reshape a low-order tensor (e.g., a matrix) into a higher-order tensor. We believe that this could be possible because the first densely connected layers were compressed significantly even without fast-decaying singular values. Another interesting research avenue is to combine the strengths of our methods and CP decomposition. We may be able to come up with better approximation algorithms to the proposed factored format.

Author Contributions: Conceptualization, J.C.; methodology, J.C.; software, K.K. and J.C.; validation, J.C.; formal analysis, J.C.; investigation, J.C.; resources, J.C.; data curation, J.C.; writing—original draft preparation, K.K. and J.C.; writing—review and editing, J.C.; visualization, J.C.; supervision, J.C.; project administration, J.C.; funding acquisition, J.C. All authors have read and agreed to the published version of the manuscript.

Funding: This work is supported by an Incheon National University Grant in 2018.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: The authors also would like to thank the reviewers and editors for their reviews of this research.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. Imagenet classification with deep convolutional neural networks. *Adv. Neural Inf. Process. Syst.* **2012**, *25*, 1097–1105. [CrossRef]
2. Alsharif, O.; Pineau, J. End-to-end text recognition with hybrid hmm maxout models. *arXiv* **2013**, arXiv:1310.1811.
3. Razavian, A.S.; Azizpour, H.; Sullivan, J.; Carlsson, S. CNN Features off-the-shelf: An Astounding Baseline for Recognition. In Proceedings of the Computer Vision and Pattern Recognition Workshops (CVPRW), Columbus, OH, USA, 23–28 June 2014; pp. 512–519.
4. Taigman, Y.; Yang, M.; Ranzato, M.; Wolf, L. Deepface: Closing the gap to human-level performance in face verification. In Proceedings of the Computer Vision and Pattern Recognition (CVPR), Columbus, OH, USA, 23–28 June 2014; pp. 1701–1708.
5. Sermanet, P.; Eigen, D.; Zhang, X.; Mathieu, M.; Fergus, R.; LeCun, Y. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv* **2013**, arXiv:1312.6229.

6. Denil, M.; Shakibi, B.; Dinh, L.; de Freitas, N. Predicting parameters in deep learning. *Adv. Neural Inf. Process. Syst.* **2013**, *2*, 2148–2156.
7. Jaderberg, M.; Vedaldi, A.; Zisserman, A. Speeding up convolutional neural networks with low rank expansions. *arXiv* **2014**, arXiv:1405.3866.
8. Denton, E.L.; Zaremba, W.; Bruna, J.; LeCun, Y.; Fergus, R. Exploiting linear structure within convolutional networks for efficient evaluation. *Adv. Neural Inf. Process. Syst.* **2014**, *1*, 1269–1277.
9. Lebedev, V.; Ganin, Y.; Rakhuba, M.; Oseledets, I.; Lempitsky, V. Speeding-up Convolutional Neural Networks Using Fine-tuned CP-Decomposition. *arXiv* **2014**, arXiv:1412.6553.
10. Jin, J.; Dundar, A.; Culurciello, E. Flattened Convolutional Neural Networks for Feedforward Acceleration. *arXiv* **2014**, arXiv:1412.5474.
11. Yang, Z.; Moczulski, M.; Denil, M.; de Freitas, N.; Alex Smola, L.S.; Wang, Z. Deep Fried Convnets. *arXiv* **2014**, arXiv:1412.7149.
12. Gong, Y.; Liu, L.; Yang, M.; Bourdev, L. Compressing Deep Convolutional Networks using Vector Quantization. *arXiv* **2014**, arXiv:1412.6115.
13. Merolla, P.A.; Arthur, J.V.; Alvarez-Icaza, R.; Cassidy, A.S.; Sawada, J.; Akopyan, F.; Jackson, B.L.; Imam, N.; Guo, C.; Nakamura, Y.; et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science* **2014**, *345*, 668–673. [CrossRef] [PubMed]
14. Snider, G. Molecular-Junction-Nanowire Crossbar-Based Neural Network. U.S. Patent 20040150010, 15 April 2008.
15. Cassidy, A.S.; Merolla, P.; Arthur, J.V.; Esser, S.K.; Jackson, B.; Alvarez-Icaza, R.; Datta, P.; Sawada, J.; Wong, T.M.; Feldman, V.; et al. Cognitive Computing Building Block: A Versatile and Efficient Digital Neuron Model for Neurosynaptic Cores. In Proceedings of the Neural Networks (IJCNN), The 2013 International Joint Conference on. IEEE, Dallas, TX, USA, 4–9 August 2013; pp. 1–10.
16. Amir, A.; Datta, P.; Risk, W.P.; Cassidy, A.S.; Kusnitz, J.A.; Esser, S.K.; Andreopoulos, A.; Wong, T.M.; Flickner, M.; Alvarez-Icaza, R.; et al. Cognitive computing programming paradigm: A corelet language for composing networks of neurosynaptic cores. In Proceedings of the Neural Networks (IJCNN), The 2013 International Joint Conference on. IEEE, Dallas, TX, USA, 4–9 August 2013; pp. 1–10.
17. Esser, S.K.; Andreopoulos, A.; Appuswamy, R.; Datta, P.; Barch, D.; Amir, A.; Arthur, J.; Cassidy, A.; Flickner, M.; Merolla, P.; et al. Cognitive computing systems: Algorithms and applications for networks of neurosynaptic cores. In Proceedings of the Neural Networks (IJCNN), The 2013 International Joint Conference on. IEEE, Dallas, TX, USA, 4–9 August 2013; pp. 1–10.
18. DeBole, M.V.; Taba, B.; Amir, A.; Akopyan, F.; Andreopoulos, A.; Risk, W.P.; Kusnitz, J.; Ortega Otero, C.; Nayak, T.K.; Appuswamy, R.; et al. TrueNorth: Accelerating From Zero to 64 Million Neurons in 10 Years. *Computer* **2019**, *52*, 20–29. [CrossRef]
19. Chung, J.; Shin, T. Simplifying deep neural networks for neuromorphic architectures. In Proceedings of the 2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC), Austin, TX, USA, 5–9 June 2016; pp. 1–6. [CrossRef]
20. Cormen, T.H.; Leiserson, C.E.; Rivest, R.L.; Stein, C. *Introduction to Algorithms*; MIT Press: Cambridge, MA, USA, 2009.
21. Klema, V.; Laub, A. The singular value decomposition: Its computation and some applications. *IEEE Trans. Autom. Control* **1980**, *25*, 164–176. [CrossRef]
22. Oseledets, I.V.; Tyrtyshnikov, E.E. Breaking the curse of dimensionality, or how to use SVD in many dimensions. *SIAM J. Sci. Comput.* **2009**, *31*, 3744–3759. [CrossRef]
23. De Lathauwer, L.; De Moor, B.; Vandewalle, J. A multilinear singular value decomposition. *SIAM J. Matrix Anal. Appl.* **2000**, *21*, 1253–1278. [CrossRef]
24. Oseledets, I.V. Tensor-train decomposition. *SIAM J. Sci. Comput.* **2011**, *33*, 2295–2317. [CrossRef]
25. Grasedyck, L. Hierarchical singular value decomposition of tensors. *SIAM J. Matrix Anal. Appl.* **2010**, *31*, 2029–2054. [CrossRef]
26. Goodfellow, I.J.; Warde-Farley, D.; Lamblin, P.; Dumoulin, V.; Mirza, M.; Pascanu, R.; Bergstra, J.; Bastien, F.; Bengio, Y. Pylearn2: A machine learning research library. *arXiv* **2013**, arXiv:1308.4214.
27. Goodfellow, I.; Warde-Farley, D.; Mirza, M.; Courville, A.; Bengio, Y. Maxout Networks. In Proceedings of the 30th International Conference on Machine Learning, Atlanta, GA, USA, 16–21 June 2013; pp. 1319–1327.
28. Ding, W.; Wang, R.; Mao, F.; Taylor, G. Theano-based Large-Scale Visual Recognition with Multiple GPUs. *arXiv* **2014**, arXiv:1412.2302.