EXPLOITING INVARIANT STRUCTURES FOR COMPRESSION IN NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Modern neural networks often require deep compositions of high-dimensional nonlinear functions (wide architecture) to achieve high test accuracy, and thus can have overwhelming number of parameters. Repeated high cost in prediction at test-time makes neural networks ill-suited for devices with constrained memory or computational power. We introduce an efficient mechanism, *reshaped tensor decomposition*, to compress neural networks by exploiting three types of *invariant structures*: periodicity, modulation and low rank. Our reshaped tensor decomposition method exploits such invariance structures using a technique called *tensorization* (reshaping the layers into higher-order tensors) combined with higher order tensor decompositions on top of the tensorized layers. Our compression method improves low rank approximation methods and can be incorporated to (is complementary to) most of the existing compression methods for neural networks to achieve better compression. Experiments on LeNet-5 (MNIST), ResNet-32 (CIFAR10) and ResNet-50 (ImageNet) demonstrate that our reshaped tensor decomposition outperforms (5% test accuracy improvement universally on CIFAR10) the state-of-the-art low-rank approximation techniques under same compression rate, besides achieving orders of magnitude faster convergence rates.

1 INTRODUCTION

Modern neural networks achieve unprecedented accuracy over many difficult learning problems at the cost of deeper and wider architectures with overwhelming number of model parameters. The large number of model parameters causes repeated high cost in test-time as predictions require loading the network into the memory and repeatedly passing the unseen examples through the large network. Therefore, the model size becomes a practical bottleneck when neural networks are deployed on constrained devices, such as smartphones and IoT cameras.

Compressing a successful large network (i.e., reducing the number of parameters), while maintaining its performance, is non-trivial. Many approaches have been employed, including pruning, quantization, encoding and knowledge distillation (see appendix A for a detailed survey). A complementary compression technique, on top of which the aforementioned approaches can be used, is *low rank approximation*. For instance, singular value decomposition (SVD) can be performed on fully connected layers (weights matrices) and tensor decomposition on convolutional layers (convolutional kernels). Low rank approximation methods can work well and reduce the number of parameters by a factor polynomial in the dimension only when the weight matrices or convolutional kernels have low rank structures, which might not always hold in practice.

We propose to exploit additional *invariant structures* in the neural network for compression. A set of experiments on several benchmark datasets justified our conjecture (Section 4): large neural networks have some invariant structures, namely periodicity, modulation and low rank, which make part of the parameters redundant. Consider this toy example of a vector with periodic structure [1,2,3,1,2,3,1,2,3] or modulated structure [1,1,1,2,2,2,3,3,3] in Figure 1. The number of parameters needed to represent this vector, naively, is 9. However if we map or *reshape* the vector into a higher order object, for instance, a matrix [1,1,1;2,2,2;3,3,3] where the columns of the matrix are repeated, then apparently this reshaped matrix can be decomposed into rank one without losing information. Therefore only 6 parameters are needed to represent the original length-9 vector.

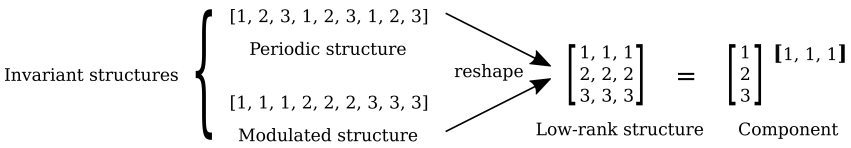


Figure 1: A toy example of invariant structures. The periodic and modulated structures are picked out by exploiting the low rank structure in the reshaped matrix.

Although the invariant structures in large neural networks allow compression of redundant parameters, designing a sophisticated way of storing a minimal representation of the parameters (while maintaining the expressive power of the network) is nontrivial. To solve this problem, we proposed a new framework called *reshaped tensor decomposition* (RTD) which has three phases:

1. **Tensorization**. We reshape the neural network layers into higher-order tensors.
   - For instance, consider a special square tensor convolutional kernel $\mathcal{T} \in \mathbb{R}^{D \times D \times D \times D}$, we reshape $\mathcal{T}$ into a higher m-order tensor $\mathcal{T}' \in \mathbb{R}^{\overbrace{D' \times \cdots \times D'}^{m}}$, so called *tensorization*. Tensorization guarantees that the reshaped tensor has a smaller dimension, i.e., $D' < D$ (as ${D'}^m = D^4$ and $m > 4$). Similar ideas apply to general convolutional kernels.
2. **Higher-order tensor decomposition**. We deploy tensor decomposition (a low rank approximation technique detailed in section 3) on the **tensorized layers** to exploit the periodic, modulated as well as low rank structures in the **original layers**.
   - A rank-R tensor decomposition of the above 4-order tensor $\mathcal{T}$ will result in R number of components (each contains $4D$ parameters), and thus $4DR$ number of parameters in total — smaller than the original D^4 number of parameters if R is small.
   - A rank-R tensor decomposition of the above reshaped m-order kernel tensor $\mathcal{T}'$ maps the layer into $m + 1$ narrower layers. The decomposition will result in R number of components with $mD^{\frac{4}{m}}$ parameters and thus $mD^{\frac{4}{m}}R$ in total — better than the $4DR$ number of parameters required by doing tensor decomposition on the original tensor $\mathcal{T}$ (D is usually large).

   Now the weights of the tensorized neural networks are the components of the tensor, i.e., result of the tensor decomposition. However, decomposing higher order tensors is challenging and known methods are not guaranteed to converge to the minimum error decomposition (Hillar & Lim, 2013). Therefore fine tuning is needed to achieve high performance.
3. **Data reconstruction-based sequential tuning**. We fine-tune the parameters using a *data reconstruction-based sequential tuning* (Seq) method which minimizes the difference between training output of the uncompressed and compressed, layer by layer. Our Seq tuning is a novel approach inspired by a sequential training method proved to converge faster and achieve guaranteed accuracy using a boosting framework (Huang et al., 2017). Unlike traditional *end-to-end* (E2E) backpropagation through the entire network, Seq tunes individual compressed "blocks" one at a time, reducing the memory and complexity required during compression.

Summary of Contributions

- **Novel compression schemes.** We propose new *reshaped tensor decomposition* methods to exploit invariant structures for compressing the parameters in neural networks. By first tensorizing the kernel/weights into a higher-order tensor, our reshaped tensor decomposition discovers extra *invariant structures* and therefore outperform existing "low rank approximation methods".
- **Efficient computational framework.** We introduce a system of *tensor algebra* that enables efficient training and inference for our compressed models. We show that a tensor decomposition on the parameters is equivalent to transforming one layer into multiple narrower sublayers in the compressed model. Therefore, other compression techniques (e.g. pruning) can be applied on top of our method by further compressing the sublayers returned by our method.
- **Sequential knowledge distillation.** We introduce *Seq* tuning to transfer knowledge to a compressed network from its uncompressed counterpart by minimizing the data reconstruction error block by block. With our strategy, only one block of the network is loaded into the GPU "at each

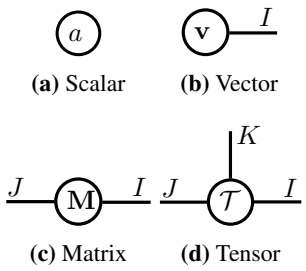
time", therefore allowing compression of large networks on moderate devices. Furthermore, we show empirically that our strategy converges much faster than normal end to end tuning.

- **Comprehensive experiments.** We perform extensive experiments to demonstrate that our *reshaped tensor decomposition* outperforms state-of-the-art low-rank approximation techniques (obtains 5% higher accuracy on CIFAR10 under same compression rates). Our experiments also show that our method scales to deep residual neural networks on large benchmark dataset, ImageNet.

Organization of the paper Section 2 introduces tensor operations, tensor decompositions and their representations in tensor diagrams. In Section 3, we introduce convolutional layer diagram, review existing low-rank approximation techniques, and propose three new schemes to exploit additional *invariant structures*. In Section 4 and Appendix B, we demonstrate by extensive experiments that our compression obtains higher accuracy than existing low-rank approximation techniques. Appendix A surveys compression techniques and discuss how our method is related or complementary to existing techniques. For simplicity, we will use tensor diagrams throughout the text. However we provide a detailed appendix where the tensor operations are mathematically defined.

2 TENSOR PRELIMINARIES

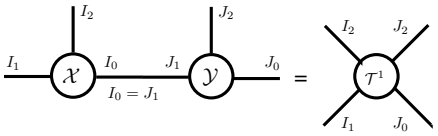
Notations An m-dimensional array $\mathcal{T}$ is defined as an *m-order* tensor $\mathcal{T} \in \mathbb{R}^{I_0 \times \cdots \times I_{m-1}}$. Its $(i_0, \cdots, i_{n-1}, i_{n+1}, \cdots, i_{m-1})^{\text{th}}$ *mode-n fiber*, a vector along the n^{th} axis, is denoted as $\mathcal{T}_{i_0,\cdots,i_{n-1},:,i_{n+1},\cdots,i_{m-1}}$.



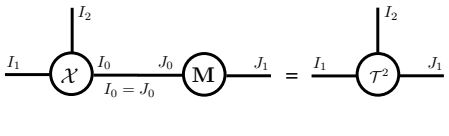
(a) Scalar **(b)** Vector **(c)** Matrix **(d)** Tensor

Figure 2: Diagram of $a \in \mathbb{R}$, $\mathbf{v} \in \mathbb{R}^I$, $\mathbf{M} \in \mathbb{R}^{I \times J}$ and $\mathcal{T} \in \mathbb{R}^{I \times J \times K}$.

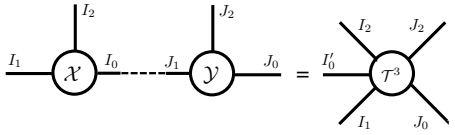
Tensor Diagrams. Following the convention in quantum physics Cichocki et al. (2016), Figure 2 introduces graphical representations for multi-dimensional objects. In *tensor diagrams*, an array (scalar/vector/matrix/tensor) is represented as a *node* in the graph, and its *order* is denoted by the number of *edges* extending from the node, where each edge corresponds to one *mode* (whose dimension is denoted by the number associated to the edge) of the multi-dimensional array.



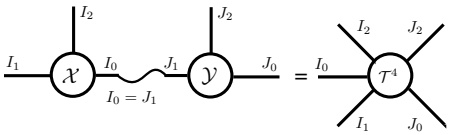
(a) Mode-(0,1) tensor contraction $\mathcal{X} \times_1^0 \mathcal{Y} \to \mathcal{T}^1$
$\mathcal{T}^1_{i_1,i_2,j_0,j_2} = \sum_r \mathcal{X}_{r,i_1,i_2} \mathcal{Y}_{j_0,r,j_2}$



(b) Mode-0 tensor product/multiplication
$\mathcal{X} \times_0 \mathbf{M} \to \mathcal{T}^2$: $\mathcal{T}^2_{j_0,i_1,i_2} = \sum_r \mathcal{X}_{r,i_1,i_2} \mathbf{M}_{j_0,r}$



(c) Mode-(0,1) tensor convolution $\mathcal{X} *_1^0 \mathcal{Y} \to \mathcal{T}^3$
$\mathcal{T}^3_{:,i_1,i_2,j_0,j_2} = \mathcal{X}_{:,i_1,i_2} * \mathcal{Y}_{j_0,:,j_2}$



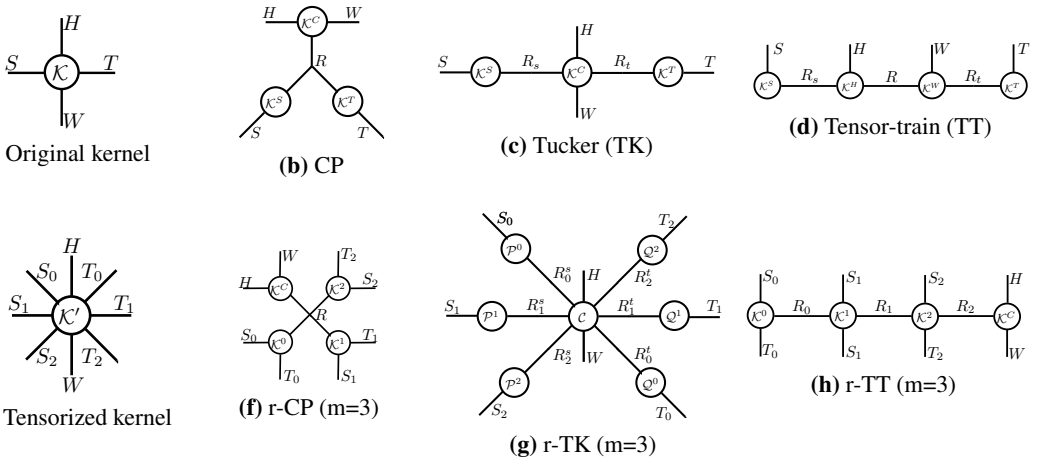
(d) Mode-(0,1) tensor partial outer product
$\mathcal{X} \otimes_1^0 \mathcal{Y} \to \mathcal{T}^4$: $\mathcal{T}^4_{r,i_1,i_2,j_0,j_2} = \mathcal{X}_{r,i_1,i_2} \mathcal{Y}_{j_0,r,j_2}$

Figure 3: Tensor operation illustration. Examples of tensor operations in which $\mathbf{M} \in \mathbb{R}^{J_0 \times J_1}$, $\mathcal{X} \in \mathbb{R}^{I_0 \times I_1 \times I_2}$ and $\mathcal{Y} \in \mathbb{R}^{J_0 \times J_1 \times J_2}$ are input matrix/tensors, and $\mathcal{T}^1 \in \mathbb{R}^{I_1 \times I_2 \times J_0 \times J_2}$, $\mathcal{T}^2 \in \mathbb{R}^{J_0 \times I_1 \times I_2}$, $\mathcal{T}^3 \in \mathbb{R}^{I'_0 \times I_1 \times I_2 \times J_0 \times J_2}$ and $\mathcal{T}^4 \in \mathbb{R}^{I_0 \times I_1 \times I_2 \times J_0 \times J_2}$ are output tensors of corresponding operations. Similar definitions apply to general mode-(i, j) tensor operations.

Tensor Operations. In Figure 3, we use some simple examples to introduce four types of *tensor operations*, which are higher-order generalization of their matrix/vector counterparts, on input tensors $\mathcal{X}$ and $\mathcal{Y}$ and input matrix $\mathbf{M}$. In tensor diagram, an operation is represented by linking edges from the input tensors, where the type of operation is denoted by the shape of line that connects the nodes: solid line stands for *tensor contraction* / *tensor multiplication*, dashed line represents

tensor convolution, and curved line is for *tensor partial outer product*. The rigorous definitions of high-order general tensor operations are defined in Appendix D.

Tensor Decompositions. We introduce generalized *tensor decomposition* as the reverse mapping of the general tensor operations (detailed in Appendix F): given a set of operations and a tensor, the generalized tensor decomposition recovers the factors/components such that the operations on these factors result in a tensor approximately equal to the original one. Several classical types of tensor decompositions (such as CANDECOMP/PARAFAC (CP), Tucker (TK) and Tensor-train (TT) decompositions) are introduced in Appendix F, and their applications on the convolutional kernel in Figure 4a (defined in Section 3) are illustrated as tensor diagrams in Figures 4b, 4c and 4d.



(a) Original kernel **(b)** CP **(c)** Tucker (TK) **(d)** Tensor-train (TT)

(e) Tensorized kernel **(f)** r-CP (m=3) **(g)** r-TK (m=3) **(h)** r-TT (m=3)

Figure 4: Diagrams of kernel decompositions. Figure **(b) (c)** and **(d)** are three types of *plain tensor decomposition* for a traditional convolutional kernel $\mathcal{K}$ in **(a)**. Figure **(f) (g)** and **(h)** are three types of *reshaped tensor decomposition* for our tensorized kernel $\mathcal{K}'$in **(e)** where the reshaping order $m \in \mathbb{Z}$ is chosen to be 3 for illustrative simplicity.

3 Compressing convolutional layer by tensor decompositions

A standard convolutional layer in neural networks is parameterized by a 4-order kernel $\mathcal{K} \in \mathbb{R}^{H\times W\times S\times T}$ where H, W are height/width of the filters, and S, T are the numbers of input/output channels. The layer maps a 3-order input tensor $\mathcal{U} \in \mathbb{R}^{X\times Y\times S}$ (with S number of feature maps of height X and width Y) to another 3-order output tensor $\mathcal{V} \in \mathbb{R}^{X'\times Y'\times T}$ (with T number of feature maps of height X' and width Y') according to the following equation:

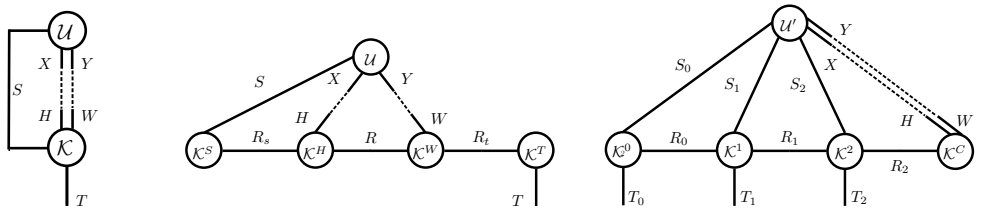
$$\mathcal{V}_{x,y,t} = \sum_{s=0}^{S-1} \sum_{i,j} \mathcal{K}_{i,j,s,t}\, \mathcal{U}_{i+dx,j+dy,s} \tag{3.1}$$

where d is the stride of the convolution. With $HWST$ parameters, it takes $O(HWSTXY)$ operations (FLOPs) to compute the output $\mathcal{V}$. The diagram of the convolutional layer is in Figure 5a.

Plain Tensor Decomposition (PD) Traditional techniques compress a convolutional layer by directly factorizing the kernel $\mathcal{K}$ using tensor decompositions Jaderberg et al. (2014); Lebedev et al. (2014); Kim et al. (2015), such as CANDECOMP/PARAFAC (CP), Tucker (TK) and Tensor-train (TT) decompositions. For example, consider a Tensor-train decomposition on $\mathcal{K}$, the kernel can be factorized and stored as $\mathcal{K}^S \in \mathbb{R}^{S\times R_s}$, $\mathcal{K}^H \in \mathbb{R}^{R_s\times H\times R}$, $\mathcal{K}^W \in \mathbb{R}^{R\times W\times R_t}$ and $\mathcal{K}^T \in \mathbb{R}^{R_t\times T}$, which only requires $(SR_s + HR_sR + WR_tR + TR_t)$ parameters as illustrated in Figure 4d. The decomposition is rigorously defined element-wisely as

$$\mathcal{K}_{i,j,s,t} = \sum_{r_s=0}^{R_s-1} \sum_{r=0}^{R-1} \sum_{r_t=0}^{R_t-1} \mathcal{K}^S_{s,r_s}\, \mathcal{K}^H_{r_s,i,r}\, \mathcal{K}^W_{r,j,r_t}\, \mathcal{K}^T_{r_t,t} \tag{3.2}$$

We defer the details of using CP and TK to Appendix G, although their tensor diagrams are illustrated in Figures 4b, 4c and 4d and their complexities are summarized in Tables 1 and 10.



(a) Uncompressed **(b)** Compression via TT **(c)** Compression via r-TT ($m = 3$)

Figure 5: Convolutional layer diagram. Input $\mathcal{U}$ is passed through the layer kernel $\mathcal{K}$. The forward propogation operation of an uncompressed layer, a plain tensor decomposition compressed layer and our reshaped tensor decomposition compressed layer are illustrated in **(a)**, **(b)** and **(c)** respectively.

Motivation of Tensorization – Invariant Structures Consider a matrix $\mathbf{M} \in \mathbb{R}^{L^2 \times L^2}$ with $\mathbf{M} = \mathbf{a} \otimes \mathbf{b}$, $\mathbf{a}, \mathbf{b} \in \mathbb{R}^{L^2}$, where $\mathbf{a} = [\mathbf{c}; \cdots; \mathbf{c}]$ is *periodic* repetition of the same vector $\mathbf{c} \in \mathbb{R}^L$ and $\mathbf{b}^\top = [d_0 \mathbf{1}^\top; d_1 \mathbf{1}^\top; \cdots; d_{L-1} \mathbf{1}^\top]$ is a *modulated* version of another vector $\mathbf{d} \in \mathbb{R}^L$. Obviously, $\mathbf{M}$ as a rank-1 matrix can be represented by two length-L^2 vectors $\mathbf{a}$ and $\mathbf{b}$, resulting in a total of $2L^2$ parameters. However, if we reshape the matrix $\mathbf{M}$ into a 4-order tensor $\mathcal{T} \in \mathbb{R}^{L \times L \times L \times L}$, it can be factorized by CP decomposition as $\mathcal{T} = \mathbf{1} \otimes \mathbf{c} \otimes \mathbf{d} \otimes \mathbf{1}$ ($\mathbf{1} \in \mathbb{R}^L$), and represented by four length-L vectors, requiring only $4L$ parameters. We refer the process of reshaping an array into a higher-order tensor as *tensorization*, and the use of *tensor decomposition* following tensorization as *reshaped tensor decomposition* (RTD). Therefore, the example above demonstrates that RTD discovers additional *invariant structures* that baseline *plain tensor decomposition* (PD) fails to identify.

Reshaped Tensor Decomposition (RTD) Inspired by this intuition, we *tensorize* the convolutional kernel $\mathcal{K}$ into a higher-order tensor $\mathcal{K}' \in^{H \times W \times S_0 \times \cdots \times S_{m-1} \times T_0 \times \cdots \times T_{m-1}}$. Correspondingly, we define an equivalent *tensorized convolutional layer* to Equation 3.1, by further reshaping input $\mathcal{U}$ and output $\mathcal{V}$ into higher-order tensors $\mathcal{U}' \in \mathbb{R}^{X \times Y \times S_0 \times \cdots \times S_{m-1}}$ and $\mathcal{V}' \in \mathbb{R}^{X' \times Y' \times T_0 \times \cdots \times T_{m-1}}$.

$$\mathcal{V}'_{x,y,t_0,\cdots,t_{m-1}} = \sum_{s_0=0}^{S_0-1} \cdots \sum_{s_{m-1}=0}^{S_{m-1}-1} \sum_{i,j} \mathcal{K}'_{i,j,s_0\cdots,s_{m-1},t_0,\cdots,t_{m-1}} \mathcal{U}'_{i+dx,j+dy,s_0,\cdots,s_{m-1}} \tag{3.3}$$

Now we can compress the convolutional layer by factorizing the tensorized kernel $\mathcal{K}'$ by tensor decompositions, and name the schemes using CP, Tucker and Tensor-train as *reshaped CP* (r-CP), *reshaped Tucker* (r-TK) and *reshaped Tensor-train* (r-TT) respectively. For example, consider a r-TT decomposition on $\mathcal{K}'$, the tensorized kernel can now be stored in $m+1$ factors $\{\mathcal{K}^0, \cdots, \mathcal{K}^{m-1}, \mathcal{K}^C\}$ (a special example of $m = 3$ is illustrated in Figure 4h). The decomposition scheme is rigorously defined element-wisely as

$$\mathcal{K}'_{i,j,s_0,\cdots,s_{m-1},t_0,\cdots,t_{m-1}} = \sum_{r_0=1}^{R_0-1} \cdots \sum_{r_{m-1}=0}^{R_{m-1}-1} \mathcal{K}^0_{s_0,r_0,t_0} \mathcal{K}^1_{r_0,s_1,t_1,r_1} \cdots \mathcal{K}^C_{i,j,r_{m-1}} \tag{3.4}$$

where $\mathcal{K}^l \in \mathbb{R}^{R_{l-1} \times S_l \times T_l \times R_l}, \forall l \in [m]$ and $\mathcal{K}^C \in \mathbb{R}^{R_{m-1} \times H \times W}$. Assuming $S_l = S^{\frac{1}{m}}, T_l = T^{\frac{1}{m}}, R_l = R, \forall l \in [m]$, we require $O(m(ST)^{\frac{1}{m}} R^2 + HWR)$ parameters. We defer the detailed descriptions of r-CP and r-TK to Appendix I, but we illustrate their tensor diagrams in Figures 4f and 4g and summarize their complexities in Tables 1 and 12.

Sequential Tuning Tensor decompositions provide weight estimates in the tensorized convolutional layers. However, decomposing higher order tensors is challenging and known methods are not guaranteed to converge to the minimum error decompositions (Hillar & Lim, 2013). Therefore fine tuning is needed to restore high performance. Analogous to Huang et al. (2017), our strategy of *data reconstruction-based sequential tuning* (Seq) sequentially fine-tunes the parameters layer by layer, using backpropagation to minimize the difference between the outputs from uncompressed layer $\mathcal{V}'$ and the tensorized compressed layer $\widehat{\mathcal{V}}'$.

Computational Complexity As shown in Figure 5c, *reshaped tensor decomposition* maps a layer into multiple (and thus deeper) narrower layers, each of which has width R_l that is usually smaller

than the original width T. We design efficient algorithms for forward/backward propagations for prediction/fine-tuning on these modified layers using *tensor algebra*. (1) *Forward propagation:* computing the output $\mathcal{V}'$, given the input $\mathcal{U}'$ and kernel factors $\{\mathcal{K}^l\}_{l=0}^{m-1}$. (2) *Backward propagation:* computing the derivative of loss function $\mathcal{L}$ with respect to (w.r.t.) the input $\partial\mathcal{L}/\partial\mathcal{U}'$ and kernel factors $\{\partial\mathcal{L}/\partial\mathcal{K}^l\}_{l=0}^{m-1}$, given the derivative w.r.t. the output $\partial\mathcal{L}/\partial\mathcal{V}'$.

A naive forward and backward propagation mechanism is to explicitly reconstruct the original kernel $\mathcal{K}'$ using the factors $\{\mathcal{K}^l\}_{l=0}^{m-1}$, which however makes propagations highly inefficient as shown in Appendix F, G and I. **Alternatively, we propose a framework where both propagations are evaluated efficiently without explicitly forming or computing the original kernel.** The key idea is to interact the input $\mathcal{U}'$ with each of the factors $\mathcal{K}^l$ individually. Taking r-TT as an example, we plug the decomposition 3.4 into 3.3, then the computation of $\mathcal{V}'$ is reduced into $m+1$ steps:

$$\mathcal{U}^l_{x,y,\cdots,r_l} = \sum_{r_{l-1}=0}^{R_{l-1}-1} \sum_{s_l=0}^{S_l-1} \mathcal{K}^{l-1}_{r_{l-1},s_l,t_l,r_l}\, \mathcal{U}^{l-1}_{x,y,s_l,\cdots,s_{m-1},t_0,\cdots,t_{l-1},r_{l-1}},\ \forall l = 1,\cdots,m \tag{3.5}$$

$$\mathcal{V}_{x,y,t_0,\cdots,t_{m-1}} = \sum_{r_{m-1}=0}^{R_{m-1}-1} \sum_{i,j} \mathcal{K}^m_{r_{m-1},i,j}\, \mathcal{U}^m_{i+dx,j+dy,t_0,\cdots,t_{m-1},r_{m-1}} \tag{3.6}$$

where $\mathcal{U}^l$ is the intermediate result after interacting with $\mathcal{K}^{l-1}$, and $\mathcal{U}^0 = \mathcal{U}$. Each step in 3.5 takes $O(\max(S,T)^{1+\frac{1}{m}}RXY)$ operations, while the last step in 3.6 requires $O(HWTRXY)$ [1]. Therefore, the time complexity for the forward pass is $O((m\max(S,T)^{1+\frac{1}{m}}R + HWT)RXY)$, more efficient than uncompressed $O(HWSTXY)$ as $R \ll S,T$. Backpropagation is derived and analyzed in Appendix I, and the analyses of other decomposition are in Appendix G and I, but we summarize their computational complexities in Table 1, 10 and 12.

Parallel Computational Complexity Tensor algebra allows us to implement the propagations 3.5 and 3.6 in parallel given enough computational resources, further speeding up prediction. The parallel tine complexities of prediction [2] with our RTD implementation is displayed in Table 2. The prediction time complexity of RTD outperforms the baseline PD, whereas the PD outperforms the original convolutional layers as $R \ll N$ and $m \geq 3$.

Decomp.	O(# of parameters)	O(# of forward ops.)	O(# of backward ops.)
original	k^2N^2	$k^2N^2D^2$	N^2D^4
CP	$(k^2+2N)R$	$(k^2+2N)RD^2$	$(D^2+2N)RD^2$
TK	$(k^2R+2N)R$	$(k^2R+2N)RD^2$	$(D^2R+2N)RD^2$
TT	$2(kR+N)R$	$2(kR+N)RD^2$	$2(DR+N)RD^2$
r-CP	$(k^2+mN^{\frac{2}{m}})R$	$(mN^{1+\frac{1}{m}}+k^2N)RD^2$	$(mN^{1+\frac{1}{m}}+ND^2)RD^2$
r-TK	$(k^2R^{2m-1}+2mN)R$	$(k^2R^{2m-1}+2mN)RD^2$	$(R^{2m-1}D^2+2mN)RD^2$
r-TT	$(mN^{\frac{2}{m}}R+k^2)R$	$(mN^{1+\frac{1}{m}}R+k^2N)RD^2$	$(mN^{1+\frac{1}{m}}R+ND^2)RD^2$

Table 1: Number of parameters and operations required by a compressed convolutional layer by various types of tensor decompositions (for the special case of $X=Y=X'=Y'=D$, $S=T=N$, $H=W=k$ and $D \gg k$). General settings are summarized in Tables 10 and 12.

Decomp.	O(parallel complexity)	Decomp.	O(parallel complexity)
CP	$N+k^2+R$	r-CP	$mN^{\frac{1}{m}}+k^2R$
TK	$N+k^2R+R$	r-TK	$2mN^{\frac{1}{m}}+k^2R^m$
TT	$N+2kR+R$	r-TT	$mN^{\frac{1}{m}}R+k^2R$

Table 2: Parallel time complexity of forward pass using various types of tensor decompositions on convolutional layers. The uncompressed parallel complexity of forward pass is $O(k^2N)$.

[1] The optimal complexity of tensor algebra is NP complete in general Lam et al. (1997), therefore the complexity presented in this paper is the complexity of our implementation.

[2] Assuming adding n terms takes n time in parallel for memory efficiency, although it could be $O(\log n)$.

4 EXPERIMENTS

We evaluate our reshaped tensor decomposition method on the state-of-art networks for a set of benchmark datasets: we evaluate convolutional layer compression on ResNet-32 He et al. (2016) for CIFAR-10; we evaluate fully-connected layer compression on MNIST; and we evaluate the scalability of our compression method on ResNet-50 He et al. (2016) for ImageNet (2012) dataset. The baseline we compare against is the state-of-the-art low-rank approximation methods called *plain tensor decomposition* (PD), as other compression methods are complementary and can be used on top of our *reshaped tensor decomposition* (RTD) method. All types of tensor decomposition (CP, TK, and TT) in baseline PD will be evaluated and compared with corresponding types of tensor decomposition (r-CP, r-TK and r-TT) in our RTD method.

Our primary contribution is to introduce a new framework, reshaped tensor decomposition, that picks out additional invariance structure such as periodicity and modulation, which the low rank approximation baseline, plain tensor decomposition, fails to find. Now we demonstrate that our RTD maintains high accuracy even when the networks are highly compressed on CIFAR-10. We refer to traditional backpropogation-based tuning of the network as end-to-end (E2E) tuning, and to our proposed approach that trains each block individually as *data reconstruction-based sequential* (Seq) tunning.

Decomp.	Compression rate				Decomp.	Compression rate			
	5%	10%	20%	40%		2%	5%	10%	20%
SVD	83.09	87.27	89.58	90.85	r-TR†	-	80.80†	-	90.60
CP	84.02	86.93	88.75	88.75	r-CP	85.7	89.86	**91.28**	-
TK	83.57	86.00	88.03	89.35	r-TK	61.06	71.34	81.59	87.11
TT	77.44	82.92	84.13	86.64	r-TT	78.95	84.26	87.89	-

†The reshaped tensor ring (r-TR) results are cited from Wang et al. (2018), and the accuracy of 80.8% is achieved by 6.67% compression rate.

Table 3: Percentage test accuracy of baseline **PD** with E2E tuning vs. our **RTD** with Seq tuning on CIFAR10. The uncompressed ResNet-32 achieves 93.2% accuracy with 0.46M parameters.

Our algorithm achieves 5% higher accuracy than baseline on ResNet-34 CIFAR10. As in Table 3, using baseline CP decomposition with end-to-end tuning, ResNet-34 is compressed to 10% of its original size, reducing the accuracy from 93.2% to 86.93%. Our reshaped tensor decomposition using r-CP, paired with Seq tuning, increases the accuracy to 91.28% with the same 10% compression rate — **a performance loss of 2% with only 10% of the number of parameters**. It achieves further aggressive compression — **a performance loss of 6% with only 2% of the number of parameters**. We observe similar trends (higher compression and higher accuracy) for Tensor-train decomposition. The structure of the Tucker decomposition (see section I) makes it less effective with very high compression, since the "internal structure" of the network reduces to very low rank, which may lose necessary information. Increasing the network size to 20% of the original provides reasonable performance on CIFAR-10 for Tucker as well.

Table 3 shows that *RTD with Seq tuning* outperforms *PD with end-to-end tuning*. Now we address the following question: is one factor (Seq tuning or RTD) primarily responsible for increased performance, or is the benefit due to synergy between the two?

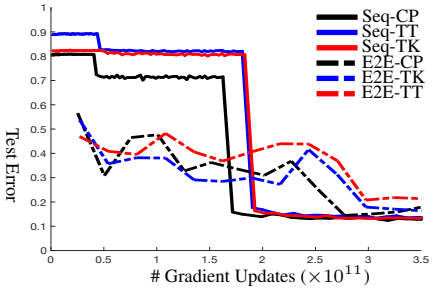


Figure 6: Convergence rate for Seq vs. E2E tuning on CIFAR10.

Seq tuning, reshaped tensor decomposition, or both? (1) We present the effect of different tuning methods on accuracy in Table 4. Other than at very high compression rate (5% column in Table 4), Seq tuning (Seq) consistently outperforms end-to-end (E2E) tuning. In addition, Seq tuning is also *much* faster and leads to more stable convergence compared to end-to-end tuning. Figure 6 plots the compression error over the number of gradient updates for various tuning methods. (2) We present the effect of different compression methods on accuracy in Table 5. Interestingly, if our RTD is used, the test accuracy is restored for

even very high compression ratios [3]. **These results confirm the existence of extra invariant structure in the parameter space of deep neural networks**. Such invariant structure is picked up by our proposed aproach, *tensorization combined with low rank approximation* (i.e., our RTD), but not by low rank approximation itself (i.e., baseline PD). Therefore, our results show that RTD and Seq tuning are symbiotic, and *both* are necessary to simultaneously obtain a high accuracy and a high compression rate.

Decomp.	Compression rate							
	5%		10%		20%		40%	
	Seq	E2E	Seq	E2E	Seq	E2E	Seq	E2E
SVD	74.04	**83.09**	85.28	**87.27**	**89.74**	89.58	**91.83**	90.85
CP	83.19	**84.02**	**88.50**	86.93	**90.72**	88.75	**89.75**	88.75
TK	80.11	**83.57**	**86.75**	86.00	**89.55**	88.03	**91.3**	89.35
TT	**80.77**	77.44	**87.08**	82.92	**89.14**	84.13	**91.21**	86.64

Table 4: Percentage accuracy of **our Seq** vs. **baseline E2E** tuning using PD on CIFAR10.

Decomp.	Compression rate		Decomp.	Compression rate	
	5%	10%		5%	10%
CP	83.19	88.50	r-CP	**89.86**	**91.28**
TK	80.11	86.73	r-TK	71.34	81.59
TT	80.77	87.08	r-TT	**84.26**	**87.89**

Table 5: Percentage accuracy of **our RTD** vs. **baseline PD** using Seq tuning on CIFAR10.

Scalability Finally, we show that our methods scale to state-of-the-art large networks, by evaluating performance on the ImageNet 2012 dataset on a 50-layer ResNet (uncompressed with 76.05% accuracy). Table 6 shows the accuracy of RTD (TT decomposition) with Seq tuning compared to plain tensor decomposition with E2E tuning and the uncompressed network, on ResNet-50 with 10% compression rate. Table 6 shows that Seq tuning of RTD is faster than the alternative. This is an important result because it empirically validates our hypotheses that (1) our RTD compression captures the invariance structure of the ResNet (with few redundancies) better and faster than the baseline PD compression, (2) data reconstruction Seq tuning is effective even on the largest networks and datasets, and (3) our proposed efficient RTD compression methods scale to the state-of-the-art neural networks.

# epochs	# samples	Uncompressed # params. = 25M	TT (E2E) # params. = 2.5M	r-TT (Seq) # params. = 2.5M
0.2	0.24M	4.22	2.78	44.35
0.3	0.36M	6.23	3.99	46.98
0.5	0.60M	9.01	7.48	49.92
1.0	1.20M	17.3	12.80	52.59
2.0	2.40M	30.8	18.17	**54.00**

Table 6: Convergence of percentage accuracies of **uncompressed** vs. **PD** (TT decomposition) vs. **RTD** (r-TT decomposition) achieving 10% compression rate for ResNet-50 ImageNet.

5 Conclusion and Perspectives

We describe an efficient mechanism for compressing neural networks by tensorizing network layers. We implement tensorized decompositions to find approximations of the tensorized kernel, potentially preserving invariance structures missed by implementing decompositions on the original kernels. We extend vector/matrix operations to their higher order tensor counterparts, providing systematic notations and libraries for tensorization of neural networks and higher order tensor decompositions. As a future step, we will explore optimizing the parallel implementations of the tensor algebra.

[3]Note that Tucker remains an exception for aggressive compression due to the low rank internal structure that we previously discussed.

REFERENCES

Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *Advances in neural information processing systems*, pp. 2654–2662, 2014.

Yu Cheng, Felix X Yu, Rogerio S Feris, Sanjiv Kumar, Alok Choudhary, and Shi-Fu Chang. An exploration of parameter redundancy in deep networks with circulant projections. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2857–2865, 2015.

Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282*, 2017.

François Chollet. Xception: Deep learning with depthwise separable convolutions. *arXiv preprint arXiv:1610.02357*, 2016.

Andrzej Cichocki, Namgil Lee, Ivan V Oseledets, Anh Huy Phan, Qibin Zhao, and D Mandic. Low-rank tensor networks for dimensionality reduction and large-scale optimization problems: Perspectives and challenges part 1. *arXiv preprint arXiv:1609.00893*, 2016.

Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in neural information processing systems*, pp. 1269–1277, 2014.

Timur Garipov, Dmitry Podoprikhin, Alexander Novikov, and Dmitry Vetrov. Ultimate tensorization: compressing convolutional and fc layers alike. *arXiv preprint arXiv:1611.03214*, 2016.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pp. 630–645. Springer, 2016.

Christopher J Hillar and Lek-Heng Lim. Most tensor problems are np-hard. *Journal of the ACM (JACM)*, 60(6):45, 2013.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Furong Huang, Jordan Ash, John Langford, and Robert Schapire. Learning deep resnet blocks sequentially using boosting theory. *arXiv preprint arXiv:1706.04964*, 2017.

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

Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. *arXiv preprint arXiv:1511.06530*, 2015.

Tamara G Kolda and Brett W Bader. Tensor decompositions and applications. *SIAM review*, 51(3): 455–500, 2009.

Jean Kossaifi, Aran Khanna, Zachary Lipton, Tommaso Furlanello, and Anima Anandkumar. Tensor contraction layers for parsimonious deep nets. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2017 IEEE Conference on*, pp. 1940–1946. IEEE, 2017a.

Jean Kossaifi, Zachary C Lipton, Aran Khanna, Tommaso Furlanello, and Anima Anandkumar. Tensor regression networks. *arXiv preprint arXiv:1707.08308*, 2017b.

Chi-Chung Lam, P Sadayappan, and Rephael Wenger. On optimizing a class of multi-dimensional loops with reductions for parallel execution. *Parallel Processing Letters*, 7(2):157–168, 1997.

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

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Michael Mathieu, Mikael Henaff, and Yann LeCun. Fast training of convolutional networks through ffts. *arXiv preprint arXiv:1312.5851*, 2013.

Alexander Novikov, Dmitrii Podoprikhin, Anton Osokin, and Dmitry P Vetrov. Tensorizing neural networks. In *Advances in Neural Information Processing Systems*, pp. 442–450, 2015.

Ivan V Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011.

Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2014.

Vikas Sindhwani, Tara Sainath, and Sanjiv Kumar. Structured transforms for small-footprint deep learning. In *Advances in Neural Information Processing Systems*, pp. 3088–3096, 2015.

Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, volume 4, pp. 12, 2017.

Wenqi Wang, Yifan Sun, Brian Eriksson, Wenlin Wang, and Vaneet Aggarwal. Wide compression: Tensor ring nets. *learning*, 14(15):13–31, 2018.

Bichen Wu, Forrest N Iandola, Peter H Jin, and Kurt Keutzer. Squeezedet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving. In *CVPR Workshops*, pp. 446–454, 2017.

Yinchong Yang, Denis Krompass, and Volker Tresp. Tensor-train recurrent neural networks for video classification. *arXiv preprint arXiv:1707.01786*, 2017.

Zichao Yang, Marcin Moczulski, Misha Denil, Nando de Freitas, Alex Smola, Le Song, and Ziyu Wang. Deep fried convnets. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1476–1483, 2015.

Xiangyu Zhang, Jianhua Zou, Xiang Ming, Kaiming He, and Jian Sun. Efficient and accurate approximations of nonlinear convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1984–1992, 2015.

Appendix: Exploiting Invariant Structures for Compression in Neural Networks

A Related Works

A recent survey Cheng et al. (2017) reviews state-of-the-art techniques for compressing neural networks, in which they group the methods into four categories: (1) low-rank factorization; (2) design of compact filters; (3) knowledge distillation; and 4) parameters pruning, quantization and encoding. Generally, our decomposition schemes fall into the category of *low-rank factorization*, but these schemes also naturally lead to novel *designs of compact filters* in the sublayers of the compressed network. On the other hand, our strategy of *sequential tuning* is an advanced scheme of *knowledge distillation* that transfers information from pre-trained teacher network to compressed student network block by block. Furthermore, our method is complementary to the techniques of *parameters pruning, quantization and encoding*, which can be applied on top of our method by further compressing the parameters in the sublayers returned by tensor decomposition.

- **Low-rank Factorization.** Low-rank approximation techniques have been used for a long time to reduce the number of parameters in both fully connected and convolutional layers. Pioneering papers propose to flatten/unfold the parameters in convolutional layer into matrices (a.k.a *matricization*), followed by (sparse) dictionary learning or matrix decomposition (Jaderberg et al., 2014; Denton et al., 2014; Zhang et al., 2015). Subsequently in Lebedev et al. (2014); Kim et al. (2015), the authors show that it is possible to compress the tensor of parameters directly by standard tensor decomposition (in particular CP or Tucker decomposition). The groundbreaking work (Novikov et al., 2015) demonstrates that the parameters in fully connected layer can be efficiently compressed by tensor decomposition by first reshaping the matrix of parameters into higher-order tensor, and the idea is later extended to compress LSTM and GRU layers in recurrent neural networks (Yang et al., 2017).

  Concurrent to Wang et al. (2018), our paper extends this basic idea to convolutional layer by exploiting the invariant structures among the filters. Different from Wang et al. (2018) that only focuses Tensor-ring decomposition, we investigates, analyzes and implements a boarder range of decomposition schemes, besides other benefits discussed below.

- **Design of Compact Filters.** These techniques reduce the number of parameters by imposing additional constraints on linear layers (fully connected or convolutional). For example, the matrix of parameters in fully connected layer is restricted to circular (Cheng et al., 2015), Toeplitz/Vandermonde/Cauchy (Sindhwani et al., 2015), or multiplication of special matrices (Yang et al., 2015). Historically, convolutional layer is proposed as a compact design of fully connected layer, where spatial connections are local (thus sparse) with repeated weights. Recent research further suggests to use more compact convolutional layers, such as 1×1 *convolutional layer* (Szegedy et al., 2017; Wu et al., 2017)(where each filter is simply a scalar), and *depthwise convolutional layer* (Chollet, 2016) (where connections between the feature maps are also sparse). In our paper, we show that the sublayers returned by our decomposition schemes are in fact 1×1 *depthwise convolutional layers*, combing advantages from both designs above.

- **Knowledge Distillation.** The algorithms of knowledge distillation aim to transfer information from a pre-trained teacher network to a smaller student network. In Ba & Caruana (2014); Hinton et al. (2015), the authors propose to train the student network supervised by the logits (the vector before softmax layer) of the teacher network. Romero et al. (2014) extends the idea to matching the outputs from both networks *at each layer*, up to an affine transformation. Our *Seq tuning* strategy is therefore similar to Romero et al. (2014), but we use identical mapping instead of affine transformation, and train the compressed network block by block.

- **Pruning, Quantization and Encoding.** Han et al. (2015) proposes a three-step pipeline to compress a pre-trained network, by (1) pruning uninformative connections, (2) quantizing the remaining weights and (3) encoding the discretized parameters. Since our decomposition schemes effectively transform one layer in the original network into the multiple sublayers, this pipeline can be applied by further compressing all sublayers. Therefore, our method is complementary (and can be used independently) to the techniques in this pipeline.

B SUPPLEMENTARY EXPERIMENTS

Convergence Rate Compared to end-to-end, an ancillary benefit of Seq tuning is *much* faster and leads to more stable convergence. Figure 6 plots compression error over number of gradient updates for various methods. (This experiment is for PD with 10% compression rate.) There are three salient points: first, Seq tuning has very high error in the beginning while the "early" blocks of the network are being tuned (and the rest of the network is left unchanged to tensor decomposition values). However, as the final block is tuned (around 2×10^{11} gradient updates) in the figure, the errors drop to nearly minimum immediately. In comparison, end-to-end tuning requires 50–100% more gradient updates to achieve stable performance. Finally, the result also shows that for each block, Seq tuning achieves convergence very quickly (and nearly monotonically), which results in the stair-step pattern since extra tuning of a block does not improve (or appreciably reduce) performance.

Performance on Fully-Connected Layers An extra advantage of reshaped tensor decomposition compression is that it can apply flexibly to fully-connected as well as convolutional layers of a neural network. Table 7 shows the results of applying reshaped tensor decomposition compression to various tensor decompositions on a variant of LeNet-5 network LeCun et al. (1998). The convolutional layers of the LeNet-5 network were *not* compressed, trained or updated in these experiments. The uncompressed network achieves 99.31% accuracy. Table 7 shows **the fully-connected layers can be compressed to 0.2% losing only about 2% accuracy**. In fact, compressing the dense layers to 1% of their original size reduce accuracy by less then 1%, demonstrating the extreme efficacy of reshaped tensor decomposition compression when applied to fully-connected neural network layers.

	Compression rate		
Method	0.2%	0.5%	1%
r-CP	97.21	97.92	98.65
r-TK	97.71	98.56	98.52
r-TT	97.69	98.43	98.63

Table 7: Reshaped tensor decomposition combined with sequential for fully-connected layers on MNIST. The uncompressed network achieves 99.31% accuracy.

C NOTATIONS

Symbols: Lower case letters (e.g. $\mathbf{v}$) are used to denote column vectors, while upper case letters (e.g. $\mathbf{M}$) are used for matrices, and curled letters (e.g. $\mathcal{T}$) for multi-dimensional arrays (tensors). For a tensor $\mathcal{T} \in \mathbb{R}^{I_0 \times \cdots \times I_{m-1}}$, we will refer to the number of indices as *order*, each individual index as *mode* and the length at one mode as *dimension*. Therefore, we will say that $\mathcal{T} \in \mathbb{R}^{I_0 \times \cdots \times I_{m-1}}$ is an m-order tensor which has dimension I_k at mode-k. *Tensor operations* are extensively used in this paper: The *tensor (partial) outer product* is denoted as $\otimes$, *tensor convolution* as $*$, and finally $\times$ denotes either *tensor contraction* or *tensor multiplication*. Each of these operators will be equipped with subscript and superscript when used in practice, for example $\times_n^m$ denotes mode-(m, n) tensor contraction (defined in Appendix D). Furthermore, the symbol $\circ$ is used to construct *compound operations*. For example, $(* \circ \otimes)$ is a compound operator simultaneously performing tensor convolution and tensor partial outer product between two tensors.

Indexing: In this paragraph, we explain the usages of subscripts/superscripts for both multi-dimensional arrays and operators, and further introduce several functions that are used to alter the layout of multi-dimensional arrays.

- Nature indices start from 0, but reversed indices are used occasionally, which start from -1. Therefore the first entry of a vector $\mathbf{v}$ is v_0, while the last one is v_{-1}.
- For multi-dimensional arrays, the subscript is used to denote an entry or a subarray within an object, while superscript is to index among a sequence of arrays. For example, $\mathbf{M}_{i,j}$ denotes the entry at i^{th} row and j^{th} column of a matrix $\mathbf{M}$, and $\mathbf{M}^{(k)}$ is the k^{th} matrix in a set of N matrices $\{\mathbf{M}^{(0)}, \mathbf{M}^{(1)}, \cdots \mathbf{M}^{(N-1)}\}$. For operators, as we have seen, both subscript and superscript are used to denote the modes involved in the operation.

- The symbol colon ':' is used to slice a multi-dimensional array. For example, $\mathcal{M}_{:,k}$ denotes the k^{th} column of $\mathbf{M}$, and $\mathcal{T}_{:,:,k}$ denotes the k^{th} *frontal slice* of a 3-order tensor $\mathcal{T}$.

- Big-endian notation is adopted in conversion between multi-dimensional array and vectors. Specifically, the function $\mathsf{vec}(\cdot)$ flattens (a.k.a. *vectorize*) a tensor $\mathcal{T} \in \mathbb{R}^{I_0 \times \cdots \times I_{m-1}}$ into a vector $\mathbf{v} \in \mathbb{R}^{\prod_{l=0}^{m-1} I_l}$ such that $\mathcal{T}_{i_0,\cdots,i_{m-1}} = v_{i_{m-1}+i_{m-2}I_{m-1}+\cdots+i_0 I_1 \cdots I_{m-1}}$.

- The function $\mathsf{swapaxes}(\cdot)$ is used to permute ordering of the modes of a tensor as needed. For example, given two tensors $\mathcal{U} \in \mathbb{R}^{I \times J \times K}$ and $\mathcal{V} \in \mathbb{R}^{K \times J \times I}$, the operation $\mathcal{V} = \mathsf{swapaxes}(\mathcal{U})$ convert the tensor $\mathcal{U}$ into $\mathcal{V}$ such that $\mathcal{V}_{k,j,i} = \mathcal{U}_{i,j,k}$.

- The function $\mathsf{flipaxis}(\cdot,\cdot)$ flips a tensor along a given mode. For example, given a tensor $\mathcal{U} \in \mathbb{R}^{I \times J \times K}$ and $\mathcal{V} = \mathsf{flipaxis}(\mathcal{U}, 0)$, the entries in $\mathcal{V}$ is defined as $\mathcal{V}_{i,j,k} = \mathcal{U}_{I-1-i(mod\ I),j,k}$.

D TENSOR OPERATIONS

Operator	Notation	Definition
mode-(k,l) Tensor Contraction	$\mathcal{T}^{(0)} = \mathcal{X} \times_l^k \mathcal{Y}$	$\mathcal{T}^{(0)}_{i_0,\cdots,i_{k-1},i_{k+1},\cdots,i_{m-1},j_0,\cdots,j_{l-1},j_{l+1},\cdots,j_{n-1}}$ $= \left\langle \mathcal{X}_{i_0,\cdots,i_{k-1},:,i_{k+1},\cdots,i_{m-1}}, \mathcal{Y}_{j_0,\cdots,j_{l-1},:,j_{l+1},\cdots,j_{n-1}} \right\rangle$ inner product of mode-k fiber of $\mathcal{X}$ and mode-l fiber of $\mathcal{Y}$
mode-k Tensor Multiplication	$\mathcal{T}^{(1)} = \mathcal{X} \times_k \mathbf{M}$	$\mathcal{T}^{(1)}_{i_0,\cdots,i_{k-1},r,i_{k+1},\cdots,i_{m-1}}$ $= \left\langle \mathcal{X}_{i_0,\cdots,i_{k-1},:,i_{k+1},\cdots,i_{m-1}}, \mathbf{M}_{:,r} \right\rangle$ inner product of mode-k fiber of $\mathcal{X}$ and r^{th} column of $\mathbf{M}$
mode-(k,l) Tensor Convolution	$\mathcal{T}^{(2)} = \mathcal{X} *_l^k \mathcal{Y}$	$\mathcal{T}^{(2)}_{i_0,\cdots,i_{k-1},:,i_{k+1},\cdots,i_{m-1},j_0,\cdots,j_{l-1},j_{l+1},\cdots,j_{n-1}}$ $= \mathcal{X}_{i_0,\cdots,i_{k-1},:,i_{k+1},\cdots,i_{m-1}} * \mathcal{Y}_{j_0,\cdots,j_{l-1},:,j_{l+1},\cdots,j_{n-1}}$ convolution of mode-k fiber of $\mathcal{X}$ and mode-l fiber of $\mathcal{Y}$
mode-(k,l) Partial-Outer Product	$\mathcal{T}^{(3)} = \mathcal{X} \otimes_l^k \mathcal{Y}$	$\mathcal{T}^{(3)}_{i_0,\cdots,i_{k-1},r,i_{k+1},\cdots,i_{m-1},j_0,\cdots,j_{n-1}}$ $= \mathcal{X}_{i_0,\cdots,i_{k-1},r,i_{k+1},\cdots,i_{m-1}} \mathcal{Y}_{j_0,\cdots,j_{l-1},r,j_{l+1},\cdots,j_{n-1}}$ Hadamard product of mode-k fiber of $\mathcal{X}$ and mode-l fiber of $\mathcal{Y}$

Table 8: Summary of tensor operations. In this table, $\mathcal{X} \in \mathbb{R}^{I_0 \times \cdots \times I_{m-1}}$, $\mathcal{Y} \in \mathbb{R}^{J_0 \times \cdots \times J_{n-1}}$ and matrix $\mathbf{M} \in \mathbb{R}^{I_k \times J}$. Mode-$(k,l)$ tensor contraction and mode-(k,l) tensor partial-outer product are legal only if $I_k = J_l$. $\mathcal{T}^{(0)}$ is an $(m+n-2)$-order tensor, $\mathcal{T}^{(1)}$ is an m-order tensor, $\mathcal{T}^{(2)}$ is an $(m+n-1)$-order tensor and $\mathcal{T}^{(3)}$ is an $(m+n-1)$-order tensor.

In this section, we introduce a number of tensor operations that serve as building blocks of *tensorial neural networks*. To begin with, we describe several basic tensor operations that are natural generalization to their vector/matrix counterparts. Despite their simplicity, these basic operations can be combined among themselves to construct complicated compound operators that are actually used in all designs. We will analyze their theoretical *sequential* time complexities in details, and point out the implementational concerns along the way. Although all these operations can in principle be implemented by *parallel* programs, the degree of parallelism depends on their particular software and hardware realizations. Therefore, we will use the sequential time complexity as a rough estimate of the computational expense in this paper.

Tensor contraction Given a m-order tensor $\mathcal{T}^{(0)} \in \mathbb{R}^{I_0 \times \cdots \times I_{m-1}}$ and another n-order tensor $\mathcal{T}^{(1)} \in \mathbb{R}^{J_0 \times \cdots \times J_{n-1}}$, which share the same dimension at mode-k of $\mathcal{T}^{(0)}$ and mode-l of $\mathcal{T}^{(1)}$(i.e. $I_k = J_l$), the mode-(k,l) contraction of $\mathcal{T}^{(0)}$ and $\mathcal{T}^{(1)}$, denoted as $\mathcal{T} \triangleq \mathcal{T}^{(0)} \times_l^k \mathcal{T}^{(1)}$, returns a $(m+n-2)$-order tensor $\mathcal{T} \in R^{I_0 \times \cdots \times I_{k-1} \times I_{k+1} \times \cdots \times I_{m-1} \times J_0 \times \cdots \times J_{l-1} \times J_{l+1} \times \cdots \times J_{n-1}}$, whose

entries are computed as

$$\begin{aligned}
&\mathcal{T}_{i_0,\cdots,i_{k-1},i_{k+1},\cdots,i_{m-1},j_0,\cdots,j_{l-1},j_{l+1},\cdots,j_{n-1}} \\
&= \sum_{r=0}^{I_k-1} \mathcal{T}^{(0)}_{i_0,\cdots,i_{k-1},r,i_{k+1},\cdots,i_{m-1}} \mathcal{T}^{(1)}_{j_0,\cdots,j_{l-1},r,j_{l+1},\cdots,j_{n-1}} && \text{(D.1a)} \\
&= \langle \mathcal{T}^{(0)}_{i_0,\cdots,i_{k-1},:,i_{k+1},\cdots,i_{m-1}}, \mathcal{T}^{(1)}_{j_0,\cdots,j_{l-1},:,j_{l+1},\cdots,j_{n-1}} \rangle && \text{(D.1b)}
\end{aligned}$$

Notice that tensor contraction is a direct generalization of matrix multiplication to higher-order tensor, and it reduces to matrix multiplication if both tensors are 2-order (and therefore matrices). As each entry in $\mathcal{T}$ can be computed as inner product of two vectors, which requires $I_k = J_l$ multiplications, the total number of operations to evaluate a tensor contraction is therefore $O((\prod_{u=0}^{m-1} I_u)(\prod_{v=0,v\neq l}^{n-1} J_v))$, taking additions into account.

Notice that the analysis of time complexity only serves as a rough estimate for actual execution time, because we do not consider the factors of *parallel computing* and *computer systems*. In practice, (1) the modes that are not contracted over can be computed in parallel, and summations can be computed in *logarithmic* instead of linear time; (2) The *spatial locality* of the memory layout plays a key role in speeding up the computation of tensor operations. These arguments equally apply to all tensor operations in this paper, but we will not repeat them in the analyses for simplicity.

Tensor multiplication (Tensor product) Tensor multiplication (a.k.a. tensor product) is a special case of tensor contraction where the second operant is a matrix. Given a m-order tensor $\mathcal{U} \in \mathbb{R}^{I_0 \times \cdots \times I_{m-1}}$ and a matrix $\mathbf{M} \in \mathbb{R}^{I_k \times J}$, where the dimension of $\mathcal{U}$ at mode-k agrees with the number of the rows in $\mathbf{M}$, the mode-k tensor multiplication of $\mathcal{U}$ and $\mathbf{M}$, denoted as $\mathcal{V} \triangleq \mathcal{U} \times_k \mathbf{M}$, yields another m-order tensor $\mathcal{V} \in \mathbb{R}^{I_0 \times \cdots \times I_{k-1} \times J \times I_{k+1} \times \cdots I_{m-1}}$, whose entries are computed as

$$\begin{aligned}
&\mathcal{V}_{i_0,\cdots,i_{k-1},j,i_{k+1},\cdots,i_{m-1}} \\
&= \sum_{r=0}^{I_k-1} \mathcal{U}_{i_0,\cdots,i_{k-1},r,i_{k+1},\cdots,i_{m-1}} \mathbf{M}_{r,j} && \text{(D.2a)} \\
&= \langle \mathcal{U}_{i_0,\cdots,i_{k-1},:,i_{k+1},\cdots,i_{m-1}}, \mathbf{M}_{:,j} \rangle && \text{(D.2b)}
\end{aligned}$$

Following the convention of multi-linear algebra, the mode for J now substitutes the location originally for I_k (which is different from the definition of tensor contraction). Regardlessly, the number of operations for tensor multiplication follows tensor contraction exactly, that is $O((\prod_{u=0}^{m-1} I_u)J)$.

Tensor convolution Given a m-order tensor $\mathcal{T}^{(0)} \in \mathbb{R}^{I_0 \times I_1 \times \cdots \times I_{m-1}}$ and another n-order tensor $\mathcal{T}^{(1)} \in \mathbb{R}^{J_0 \times J_1 \times \cdots \times J_{n-1}}$. The mode-$(k,l)$ convolution of $\mathcal{T}^{(0)}$ and $\mathcal{T}^{(1)}$, denoted as $\mathcal{T} \triangleq \mathcal{T}^{(0)} *_l^k \mathcal{T}^{(1)}$, returns a $(m+n-1)$-order tensor $\mathcal{T} \in \mathbb{R}^{I_0 \times \cdots \times I'_k \times \cdots \times I_{m-1} \times J_0 \times \cdots \times J_{l-1} \times J_{l+1} \times \cdots \times J_{n-1}}$. The entries of $\mathcal{T}$ can be computed using any convolution operation $*$ that is defined for two vectors.

$$\begin{aligned}
&\mathcal{T}_{i_0,\cdots,i_{k-1},:,i_{k+1},\cdots,i_{m-1},j_0,\cdots,j_{l-1},j_{l+1},\cdots,j_{n-1}} \\
&= \mathcal{T}^{(0)}_{i_0,\cdots,i_{k-1},:,i_{k+1},\cdots,i_{m-1}} * \mathcal{T}^{(1)}_{j_0,\cdots,j_{l-1},:,j_{l+1},\cdots,j_{n-1}} && \text{(D.3a)} \\
&= \mathcal{T}^{(1)}_{j_0,\cdots,j_{l-1},:,j_{l+1},\cdots,j_{n-1}} \overline{*} \mathcal{T}^{(0)}_{i_0,\cdots,i_{k-1},:,i_{k+1},\cdots,i_{m-1}} && \text{(D.3b)}
\end{aligned}$$

Here we deliberately omit the exact definition of vector convolution $*$, as it can be defined in multiple forms depending on the user case (Interestingly, the "convolution" in convolutional layer indeed computes *correlation* instead of convolution). Correspondingly, the resulted dimension I'_k at mode-k is determined by the chosen type of convolution. For example, the "convolution" in convolutional layer typically yields $I'_k = I_k$ (with same padding) or $I'_k = I_k - J_l + 1$ (with valid padding). Notice that vector convolution itself is generally asymmetric , i.e. $\mathbf{u} * \mathbf{v} \neq \mathbf{v} * \mathbf{u}$ (except for the case of *circular convolution*). For convenience, we can define its *conjugate* as $\overline{*}$ such that $\mathbf{u} * \mathbf{v} = \mathbf{v} \overline{*} \mathbf{u}$. With this notations, Equation D.3a can also be written as D.3b.

Generally speaking, *Fast Fourier Transform (FFT)* plays a critical role to lower the computational complexities for all types of convolution. In the case of tensor convolution, the number of required operations without FFT is $O((\prod_{u=0}^{m-1} I_u)(\prod_{v=0}^{n-1} J_v))$, while FFT is able to reduce it to

$O((\prod_{u=0,u\neq k}^{m-1} I_u)(\prod_{v=0,v\neq l}^{n-1} J_v)\max(I_k, J_l)\log(\max(I_k, J_l)))$. That being said, FFT is not always necessary: if $\min(I_k, J_l) < \log(\max(I_k, J_l))$ (which is typical in convolutional layers, where I_k is the height/width of the feature maps and J_l is the side length of the square filters), computing the convolution without FFT is actually faster. Furthermore, FFT can be difficult to implement (thus not supported by popular software libraries) if convolution is fancily defined in neural networks (e.g. dilated, atrous). Therefore, we will assume that tensor convolutions are computed without FFT in subsequent sections unless otherwise noted.

Tensor outer product Given a m-order tensor $\mathcal{T}^{(0)} \in \mathbb{R}^{I_0 \times I_1 \times \cdots \times I_{m-1}}$ and another n-order tensor $\mathcal{T}^{(1)} \in \mathbb{R}^{J_0 \times J_1 \times \cdots \times J_{n-1}}$, the outer product of $\mathcal{T}^{(0)}$ and $\mathcal{T}^{(1)}$, denoted $\mathcal{T} \triangleq \mathcal{T}^{(0)} \otimes \mathcal{T}^{(1)}$, concatenates all the indices of $\mathcal{T}^{(0)}$ and $\mathcal{T}^{(1)}$, and returns a $(m+n)$-order tensor $\mathcal{T} \in \mathbb{R}^{I_0 \times \cdots \times I_{m-1} \times J_0 \times \cdots \times J_{n-1}}$ whose entries are computed as

$$\mathcal{T}_{i_0,\cdots,i_{m-1},j_0,\cdots,j_{n-1}} = \mathcal{T}^{(0)}_{i_0,\cdots,i_{m-1}} \mathcal{T}^{(1)}_{j_0,\cdots,j_{n-1}} \tag{D.4}$$

It is not difficult to see that tensor outer product is a direct generalization for outer product for two vectors $\mathbf{M} = \mathbf{u} \otimes \mathbf{v} = \mathbf{u}\,\mathbf{v}^\top$. Obviously, the number of operations to compute a tensor outer product explicitly is $O((\prod_{u=0}^{m-1} I_u)(\prod_{v=0}^{n-1} J_v))$. Tensor outer product is rarely calculated alone in practice because it requires significant amounts of computational and memory resources.

Tensor partial outer product Tensor partial outer product is a variant of tensor outer product defined above, which is widely used in conjunction with other operations. Given a m-order tensor $\mathcal{T}^{(0)} \in \mathbb{R}^{I_0 \times I_1 \times \cdots \times I_{m-1}}$ and another n-order tensor $\mathcal{T}^{(1)} \in \mathbb{R}^{J_0 \times J_1 \times \cdots \times J_{n-1}}$, which share the same dimension at mode-k of $\mathcal{T}^{(0)}$ and mode-l of $\mathcal{T}^{(1)}$ (i.e. $I_k = J_l$), the mode-(k,l) partial outer product of $\mathcal{T}^{(0)}$ and $\mathcal{T}^{(1)}$, denoted as $\mathcal{T} \triangleq \mathcal{T}^{(0)} \otimes_l^k \mathcal{T}^{(1)}$, returns a $(m+n-1)$-order tensor $\mathcal{T} \in \mathbb{R}^{I_0 \times \cdots \times I_{m-1} \times J_0 \times \cdots \times J_{l-1} \times J_{l+1} \times \cdots \times J_{n-1}}$, whose entries are computed as

$$\begin{aligned}
&\mathcal{T}_{i_0,\cdots,i_{k-1},r,i_{k+1},\cdots,i_{m-1},j_0,\cdots,j_{l-1},j_{l+1},\cdots,j_{n-1}} \\
&= \mathcal{T}^{(0)}_{i_0,\cdots,i_{k-1},r,i_{k+1},\cdots,i_{m-1}} \mathcal{T}^{(1)}_{j_0,\cdots,j_{l-1},r,j_{l+1},\cdots,j_{n-1}} &\text{(D.5a)}\\
&= \mathcal{T}^{(0)}_{\cdots,r,\cdots} \otimes \mathcal{T}^{(1)}_{\cdots,r,\cdots} &\text{(D.5b)}
\end{aligned}$$

The operation bears the name "partial outer product" because it reduces to outer product once we fix the indices at mode-k of $\mathcal{T}^{(0)}$ and mode-l of $\mathcal{T}^{(1)}$. Referring to the computational complexity of tensor outer product, the number of operations for each fixed index is $O((\prod_{u=0,u\neq k}^{m-1} I_u)(\prod_{v=0,v\neq l}^{n-1} J_v))$, therefore the total time complexity for the tensor partial outer product is $O((\prod_{u=0}^{m-1} I_u)(\prod_{v=0,v\neq l}^{n-1} J_v))$, the same as tensor contraction.

Precautions of usage

- Similar to matrix multiplication, the operants in tensor operations are not commutative in general. For example, neither $\mathcal{T}^{(0)} \times_l^k \mathcal{T}^{(1)} = \mathcal{T}^{(1)} \times_l^k \mathcal{T}^{(0)}$ nor $\mathcal{T}^{(0)} \times_l^k \mathcal{T}^{(1)} = \mathcal{T}^{(1)} \times_k^l \mathcal{T}^{(0)}$ holds even if the dimensions at the specified modes happen to match.
- Different from matrix multiplication, the law of associative also fails in general. For example, $(\mathcal{T}^{(0)} \times_l^k \mathcal{T}^{(1)}) \times_q^p \mathcal{T}^{(2)} \neq \mathcal{T}^{(0)} \times_l^k (\mathcal{T}^{(1)} \times_q^p \mathcal{T}^{(2)})$, mainly because tensor operations can change the locations of modes in a tensor.
- However, both problems are not fundamental, and can be fixed by adjusting the superscripts and subscripts of the operators carefully (and further permute ordering of the modes in the result accordingly). For example, $\mathcal{T}^{(0)} \times_l^k \mathcal{T}^{(1)} = \mathsf{swapaxes}(\mathcal{T}^{(1)} \times_k^l \mathcal{T}^{(0)})$ holds if $\mathsf{swapaxes}(\cdot)$ is properly performed. Due to space limits, we can not develop general rules in this paper, and will derive such identities as needed. In general, the take away message is a simple statement: *Given an expression that contains multiple tensor operations, these operations need to be evaluated from left to right unless a bracket is explicitly supplied*.

Compound operations: As building blocks, the basic tensor operations defined above can further combined to construct compound operations that perform multiple operations on multiple tensors simultaneously. For simplicity, we illustrate their usage using two representative examples in this

section. More examples will arise naturally when we discuss the derivatives and backpropagation rules for compound operations in Appendix E.

- **Simultaneous multi-operations between two tensors.** For example, given two 3-order tensors $\mathcal{T}^{(0)} \in \mathbb{R}^{R\times X\times S}$ and $\mathcal{T}^{(1)} \in \mathbb{R}^{R\times H\times S}$, we can define a compound operation $\left(\otimes_0^0 \circ *_1^1 \circ \times_2^2\right)$ between $\mathcal{T}^{(0)}$ and $\mathcal{T}^{(1)}$, where mode-$(0,0)$ partial outer product, mode-$(1,1)$ convolution and mode-$(2,2)$ contraction are performed simultaneously, which results in a 2-order tensor $\mathcal{T}$ of $\mathbb{R}^{R\times X'}$ (it is indeed a matrix, though denoted as a tensor). The entries of $\mathcal{T} \triangleq \mathcal{T}^{(0)} \left(\otimes_0^0 \circ *_1^1 \circ \times_2^2\right) \mathcal{T}^{(1)}$ are computed as

$$\mathcal{T}_{r,:} = \sum_{s=0}^{S-1} \mathcal{T}^{(0)}_{r,:,s} * \mathcal{T}^{(1)}_{r,:,s} \tag{D.6}$$

  For commonly used vector convolution, it is not difficult to show that number of operations required to compute the result $\mathcal{T}$ is $O\left(R\max(X,H)\log(\max(X,H))S\right)$ with FFT and $O(RXHS)$ without FFT, as each of the R vectors in $\mathcal{T}$ is computed with a sum of S vector convolutions.

- **Simultaneous operations between a tensor and a set of multiple tensors**. For example, given a 3-order tensor $\mathcal{U} \in \mathbb{R}^{R\times X\times S}$ and a set of three tensors $\mathcal{T}^{(0)} \in \mathbb{R}^{R\times P}$, $\mathcal{T}^{(1)} \in \mathbb{R}^{K\times Q}$ and $\mathcal{T}^{(2)} \in \mathbb{R}^{S\times T}$, we can define a compound operation on $\mathcal{U}$ as $\mathcal{V} \triangleq \mathcal{U}(\otimes_0^0 \mathcal{T}^{(0)} *_0^1 \mathcal{T}^{(1)} \times_0^2 \mathcal{T}^{(2)})$, which performs mode-$(0,0)$ partial outer product with $\mathcal{T}^{(0)}$, mode-$(1,0)$ convolution with $\mathcal{T}^{(1)}$ and mode-$(2,0)$ contraction with $\mathcal{T}^{(2)}$ simultaneously. In this case, a 5-order tensor $\mathcal{V} \in \mathbb{R}^{R\times X'\times P\times Q\times T}$ is returned, with entries calculated as

$$\mathcal{V}_{r,:,p,q,t} = \sum_{s=0}^{S-1} \mathcal{T}^{(0)}_{r,p} \left(\mathcal{U}_{r,:,s} * \mathcal{T}^{(1)}_{:,q}\right) \mathcal{T}^{(2)}_{s,t} \tag{D.7}$$

  The analysis of time complexity of a compound operation with multiple tensors turns out to be a non-trivial problem. To see this, let us first follow the naive way to evaluate the output according to the expression above: (1) each vector in the result $\mathcal{V}_{r,:,p,q,t}$ can be computed with a sum of S vector convolutions, which requires $O(XHS)$ operations; (2) and with $RPQT$ such vectors in the result $\mathcal{V}$, the time complexity for the whole compound operation is therefore $O(RXHPRST)$. However, it is obviously not the best strategy to perform these operations. In fact, the equations can be equivalently rewritten as

$$\mathcal{V}_{r,:,p,q,t} = \left(\left(\sum_{s=0}^{S-1} \mathcal{U}_{r,:,s} \mathcal{T}^{(2)}_{s,t}\right) * \mathcal{T}^{(1)}_{:,q}\right) \mathcal{T}^{(0)}_{r,p} \tag{D.8}$$

  If we follows the supplied brackets and break the evaluation into three steps, it is not difficult to verify that these steps take $O(RXST)$, $O(RXHPT)$ and $O(RX'HPT)$ operations respectively, and result in a total time complexity of $O(RXST + RXHPT + RX'PQT)$ for the compound operation, which is far lower than the one with the naive way.

  Unfortunately, it is an NP-hard problem to determine the best order (with minimal number of operations) to evaluate a compound operation over multiple tensors, therefore in practice the order is either determined by exhaustive search (if there are only a few tensors) or follows a heuristic strategy (if the number of tensors is large).

The examples provided above are by no mean comprehensive, and in fact more complicated compound operations simultaneously perform multiple operations on multiple tensors can be defined, and we will see examples of them in the next section when we derive the backpropagation equations for the compound operations above. Generally, compound operations over multiple tensors are difficult to flatten into mathematical expressions without introducing tedious notations. Therefore, these operations are usually described by graphical representations, which are usually called *tensor network* in the physics literature (not to confuse with *tensorial network* in this paper). Interested readers are referred to the monograph Cichocki et al. (2016), which serves a comprehensive introduction to the application of tensor network in the field of machine learning.

E DERIVATIVES AND BACKPROPAGATION OF TENSOR OPERATIONS

All operations introduced in the last section, both basic and compound, are linear in their operants. Therefore, the derivatives of the result with respect to its inputs are in principle easy to calculate. In this section, we will explicitly derive the derivatives for all operations we have seen in Appendix D.

These derivatives can be further combined with classic chain rule to obtain the corresponding *back-propagation equations* (i.e. how gradient of the loss function propagates backward through tensor operations), which are the cornerstones of modern feed-forward neural networks. In the section, we show that these backpropagation equations can also be characterized by (compound) tensor operations, therefore their computational complexities can be analyzed similarly as in Appendix D.

Interestingly, the backpropagation equations associated with a tensor operation, though typically appear to be more involved, share the same asymptotic complexities as in the forward pass (with tensor convolution as an exception). This observation is extremely useful in the analyses of tensorial neural networks in Appendix G, H and I, which allows us to reuse the same number in the forward pass in the analysis of backward propagation.

In this section, we will assume for simplicity the loss function $\mathcal{L}$ is differentiable. However, all derivatives and backpropagation equations equally apply when $\mathcal{L}$ is only sub-differentiable (piecewise smooth). Also, we will focus on one step of backpropagation, therefore we assume the gradient of the loss function is known to us in prior.

Tensor contraction Recall the definition of tensor contraction in Equation D.1a, the partial derivatives of the result $\mathcal{T}$ with respect to its operants $\mathcal{T}^{(0)}$, $\mathcal{T}^{(1)}$ can be computed at the entries level:

$$\frac{\partial \mathcal{T}_{i_0,\cdots,i_{k-1},i_{k+1},\cdots,i_{m-1},j_0,\cdots,j_{l-1},j_{l+1},\cdots,j_{n-1}}}{\partial \mathcal{T}^{(0)}_{i_0,\cdots,i_{k-1},r,i_{k+1},\cdots,i_{m-1}}} = \mathcal{T}^{(1)}_{j_0,\cdots,j_{l-1},r,j_{l+1},\cdots,j_{n-1}} \tag{E.1a}$$

$$\frac{\partial \mathcal{T}_{i_0,\cdots,i_{k-1},i_{k+1},\cdots,i_{m-1},j_0,\cdots,j_{l-1},j_{l+1},\cdots,j_{n-1}}}{\partial \mathcal{T}^{(1)}_{j_0,\cdots,j_{l-1},r,j_{l+1},\cdots,j_{n-1}}} = \mathcal{T}^{(0)}_{i_0,\cdots,i_{k-1},r,i_{k+1},\cdots,i_{m-1}} \tag{E.1b}$$

With classic chain rule, the derivatives of $\mathcal{L}$ with respect to $\mathcal{T}^{(0)}$ and $\mathcal{T}^{(1)}$ can be obtained through the derivative of $\mathcal{L}$with respect to $\mathcal{T}$.

$$\frac{\partial \mathcal{L}}{\partial \mathcal{T}^{(0)}_{i_0,\cdots,i_{k-1},r,i_{k+1},\cdots,i_{m-1}}} = \sum_{j_0=0}^{J_0-1} \cdots \sum_{j_{l-1}=0}^{J_{l-1}-1} \sum_{j_{l+1}=0}^{J_{l+1}-1} \cdots \sum_{j_{n-1}=0}^{J_{n-1}-1} \frac{\partial \mathcal{L}}{\partial \mathcal{T}_{i_0,\cdots,i_{k-1},i_{k+1},\cdots,i_{m-1},j_0,\cdots,j_{l-1},j_{l+1},\cdots,j_{n-1}}} \mathcal{T}^{(1)}_{j_0,\cdots,j_{l-1},r,j_{l+1},\cdots,j_{n-1}} \tag{E.2a}$$

$$\frac{\partial \mathcal{L}}{\partial \mathcal{T}^{(1)}_{j_0,\cdots,j_{l-1},r,j_{l+1},\cdots,j_{n-1}}} = \sum_{i_0=0}^{I_0-1} \cdots \sum_{i_{k-1}=0}^{I_{k-1}-1} \sum_{i_{k+1}=0}^{I_{k+1}-1} \cdots \sum_{i_{m-1}=0}^{I_{m-1}-1} \frac{\partial \mathcal{L}}{\partial \mathcal{T}_{i_0,\cdots,i_{k-1},i_{k+1},\cdots,i_{m-1},j_0,\cdots,j_{l-1},j_{l+1},\cdots,j_{n-1}}} \mathcal{T}^{(0)}_{i_0,\cdots,i_{k-1},r,i_{k+1},\cdots,i_{m-1}} \tag{E.2b}$$

Though tedious at entries level, it can be simplified with tensor notations in this paper.

$$\frac{\partial \mathcal{L}}{\partial \mathcal{T}^{(0)}} = \mathsf{swapaxes}\left(\frac{\partial \mathcal{L}}{\partial \mathcal{T}}\left(\times_0^{m-1} \circ \cdots \circ \times_{l-1}^{m+l-2} \circ \times_{l+1}^{m+l-1} \circ \cdots \circ \times_{n-1}^{m+n-3}\right)\mathcal{T}^{(1)}\right) \tag{E.3a}$$

$$\frac{\partial \mathcal{L}}{\partial \mathcal{T}^{(1)}} = \mathsf{swapaxes}\left(\frac{\partial \mathcal{L}}{\partial \mathcal{T}}\left(\times_0^{0} \circ \cdots \circ \times_{k-1}^{k-1} \circ \times_{k+1}^{k} \circ \cdots \circ \times_{m-1}^{m-2}\right)\mathcal{T}^{(0)}\right) \tag{E.3b}$$

where $\mathsf{swapaxes}(\cdot)$ is used to align the modes of outputs. Notice that the backpropagation equations are compound operations, even if the original operation is a basic one. It is not difficult to show that the number of operations required for both backpropagation equations are $O((\prod_{u=0}^{m-1} I_u)(\prod_{v=0,v\neq l}^{n-1} J_v))$, which are exactly the same as in the forward pass in Equation D.1a. The result should not surprise us however, since the tensor contraction is a direct generalization to matrix multiplication (where backward propagation has exactly the same time complexity as the matrix multiplication itself).

Tensor multiplication (Tensor product) As a special case of tensor contraction, the derivatives and backpropagation equations for tensor multiplication can be obtained in the same manner. To begin with, the derivatives of $\mathcal{V}$ with respect to $\mathcal{U}$ and $\mathbf{M}$ can be computed from the definition in Equation D.2a.

$$\frac{\partial \mathcal{V}_{i_0,\cdots,i_{k-1},j,i_{k+1},\cdots,i_{m-1}}}{\partial \mathcal{U}_{i_0,\cdots,i_{k-1},r,i_{k+1},\cdots,i_{m-1}}} = \mathbf{M}_{r,j} \tag{E.4a}$$

$$\frac{\partial \mathcal{V}_{i_0,\cdots,i_{k-1},j,i_{k+1},\cdots,i_{m-1}}}{\partial \mathbf{M}_{r,j}} = \mathcal{U}_{i_0,\cdots,i_{k-1},r,i_{k+1},\cdots,i_{m-1}} \tag{E.4b}$$

Subsequently, the derivatives of $\mathcal{L}$ with respect to $\mathcal{U}$ and $\mathbf{M}$ can be computed as with chain rule,

$$\frac{\partial \mathcal{L}}{\partial \mathcal{V}_{i_0,\cdots,i_{k-1},r,i_{k+1},\cdots,i_{m-1}}} = \sum_{j=0}^{J-1} \frac{\partial \mathcal{L}}{\partial \mathcal{U}_{i_0,\cdots,i_{k-1},j,i_{k+1},\cdots,i_{m-1}}} \mathbf{M}_{r,j} \tag{E.5a}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{M}_{r,j}} = \sum_{i_0=0}^{I_0-1} \cdots \sum_{i_{k-1}=0}^{I_{k-1}-1} \sum_{i_{k+1}=0}^{I_{k+1}-1} \cdots \sum_{i_{m-1}=0}^{I_{m-1}-1} \frac{\partial \mathcal{L}}{\partial \mathcal{V}_{i_0,\cdots,i_{k-1},j,i_{k+1},\cdots,i_{m-1}}} \mathcal{U}_{i_0,\cdots,i_{k-1},r,i_{k+1},\cdots,i_{m-1}} \tag{E.5b}$$

Again, the backpropagation equations above can be succinctly written in tensor notations.

$$\frac{\partial \mathcal{L}}{\partial \mathcal{V}} = \frac{\partial \mathcal{L}}{\partial \mathcal{U}} \times_k \mathbf{M}^\top \tag{E.6a}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{M}} = \mathcal{U} \left(\times_0^0 \circ \cdots \circ \times_{k-1}^{k-1} \circ \times_{k+1}^{k+1} \circ \cdots \circ \times_{m-1}^{m-1} \right) \frac{\partial \mathcal{L}}{\partial \mathcal{V}} \tag{E.6b}$$

where the time complexities for both equations are $O((\prod_{u=0}^{m-1} I_u)J)$, which is identical to the forward pass in Equation D.2a (obviously since tensor multiplication is a special of tensor contraction).

Tensor convolution Recall in the definition of tensor convolution in Equation D.3a, we deliberately omit the exact definition of vector convolution for generality. For simplicity, we temporarily limit ourselves to the special case of *circular convolution*. In this case, tensor convolution can be concretely defined by either equation below:

$$\begin{aligned} &\mathcal{T}_{i_0,\cdots,i_{k-1},:,i_{k+1},\cdots,i_{m-1},j_0,\cdots,j_{l-1},j_{l+1},\cdots,j_{n-1}} \\ &= \mathsf{Cir}\left(\mathcal{T}^{(0)}_{i_0,\cdots,i_{k-1},:,i_{k+1},\cdots,i_{m-1}}\right) \mathcal{T}^{(1)}_{j_0,\cdots,j_{l-1},:,j_{l+1},\cdots,j_{n-1}} \end{aligned} \tag{E.7a}$$

$$\begin{aligned} &\mathcal{T}_{i_0,\cdots,i_{k-1},:,i_{k+1},\cdots,i_{m-1},j_0,\cdots,j_{l-1},j_{l+1},\cdots,j_{n-1}} \\ &= \mathsf{Cir}\left(\mathcal{T}^{(1)}_{j_0,\cdots,j_{l-1},:,j_{l+1},\cdots,j_{n-1}}\right) \mathcal{T}^{(0)}_{i_0,\cdots,i_{k-1},:,i_{k+1},\cdots,i_{m-1}} \end{aligned} \tag{E.7b}$$

where $\mathsf{Cir}(\cdot)$ returns a circular matrix of the input vector. Concretely, given a vector $\mathbf{v} \in \mathbb{R}^I$, the circular matrix $\mathsf{Cir}(\mathbf{v})$ is defined as $\mathsf{Cir}(\mathbf{v})_{i,j} = v_{i-j(modI)}$. Now, the derivatives of the result tensor $\mathcal{T}$ with respect to $\mathcal{T}^{(0)}$ and $\mathcal{T}^{(1)}$ can be obtained by matrix calculus.

$$\frac{\partial \mathcal{T}_{i_0,\cdots,i_{k-1},:,i_{k+1},\cdots,i_{m-1},j_0,\cdots,j_{l-1},j_{l+1},\cdots,j_{n-1}}}{\partial \mathcal{T}^{(0)}_{i_0,\cdots,i_{k-1},:,i_{k+1},\cdots,i_{m-1}}} = \mathsf{Cir}\left(\mathcal{T}^{(1)}_{j_0,\cdots,j_{l-1},:,j_{l+1},\cdots,j_{n-1}}\right)^\top \tag{E.8a}$$

$$\frac{\partial \mathcal{T}_{i_0,\cdots,i_{k-1},:,i_{k+1},\cdots,i_{m-1},j_0,\cdots,j_{l-1},j_{l+1},\cdots,j_{n-1}}}{\partial \mathcal{T}^{(1)}_{j_0,\cdots,j_{l-1},:,j_{l+1},\cdots,j_{n-1}}} = \mathsf{Cir}\left(\mathcal{T}^{(0)}_{i_0,\cdots,i_{k-1},:,i_{k+1},\cdots,i_{m-1}}\right)^\top \tag{E.8b}$$

Applying chain rule to the equations above, we arrive at two lengthy equations:

$$\frac{\partial \mathcal{L}}{\partial \mathcal{T}^{(0)}_{i_0,\cdots,i_{k-1},:,i_{k+1},\cdots,i_{m-1}}} = \sum_{j_0=0}^{J_0-1} \cdots \sum_{j_{l-1}=0}^{J_{l-1}-1} \sum_{j_{l+1}=0}^{J_{l+1}-1} \cdots \sum_{j_{n-1}=0}^{J_{n-1}-1} \mathsf{Cir}\left(\mathcal{T}^{(1)}_{j_0,\cdots,j_{l-1},:,j_{l+1},\cdots,j_{n-1}}\right)^\top \frac{\partial \mathcal{L}}{\partial \mathcal{T}_{i_0,\cdots,i_{k-1},:,i_{k+1},\cdots,i_{m-1},j_0,\cdots,j_{l-1},j_{l+1},\cdots,j_{n-1}}} \tag{E.9a}$$

$$\frac{\partial \mathcal{L}}{\partial \mathcal{T}^{(1)}_{j_0,\cdots,j_{l-1},r,j_{l+1},\cdots,j_{n-1}}} = \sum_{i_0=0}^{I_0-1} \cdots \sum_{i_{k-1}=0}^{I_{k-1}-1} \sum_{i_{k+1}=0}^{I_{k+1}-1} \cdots \sum_{i_{m-1}=0}^{I_{m-1}-1} \mathsf{Cir}\left(\mathcal{T}^{(0)}_{i_0,\cdots,i_{k-1},:,i_{k+1},\cdots,i_{m-1}}\right)^\top \frac{\partial \mathcal{L}}{\partial \mathcal{T}_{i_0,\cdots,i_{k-1},:,i_{k+1},\cdots,i_{m-1},j_0,\cdots,j_{l-1},j_{l+1},\cdots,j_{n-1}}} \tag{E.9b}$$

With notations of tensor operations, they can be greatly simplified as

$$\frac{\partial \mathcal{L}}{\partial \mathcal{T}^{(0)}} = \mathsf{swapaxes}\left(\frac{\partial \mathcal{L}}{\partial \mathcal{T}}\left(\times_0^m \cdots \times_{l-1}^{m+l-1} \circ *_l^k \circ \times_{l+1}^{m+l-2} \circ \cdots \circ \times_{n-1}^{m+n-2}\right)\mathsf{flipaxis}(\mathcal{T}^{(1)}, l)\right) \tag{E.10a}$$

$$\frac{\partial \mathcal{L}}{\partial \mathcal{T}^{(1)}} = \mathsf{swapaxes}\left(\frac{\partial \mathcal{L}}{\partial \mathcal{T}}\left(\times_0^0 \circ \cdots \circ \times_{k-1}^{k-1} \circ *_k^k \circ \times_{k+1}^{k+1} \circ \cdots \circ \times_{m-1}^{m-1}\right)\mathsf{flipaxis}(\mathcal{T}^{(0)}, k)\right) \tag{E.10b}$$

Although these backpropagation equations are derived for the special case of circular convolution, they hold for general convolution if we replace $*_l^k$ by its corresponding adjoint operator $(*_l^k)^\top$ (and use the unflipped versions of $\mathcal{T}^{(0)}$ and $\mathcal{T}^{(1)}$). With adjoint of convolution, Equations E.10a and E.10b can be rewritten as

$$\frac{\partial \mathcal{L}}{\partial \mathcal{T}^{(0)}} = \mathsf{swapaxes}\left(\frac{\partial \mathcal{L}}{\partial \mathcal{T}}\left(\times_0^m \cdots \times_{l-1}^{m+l-1} \circ (*_l^k)^\top \circ \times_{l+1}^{m+l-2} \cdots \times_{n-1}^{m+n-2}\right)\mathcal{T}^{(1)}\right) \tag{E.11a}$$

$$\frac{\partial \mathcal{L}}{\partial \mathcal{T}^{(1)}} = \mathsf{swapaxes}\left(\frac{\partial \mathcal{L}}{\partial \mathcal{T}}\left(\times_0^0 \circ \cdots \circ \times_{k-1}^{k-1} \circ (*_k^k)^\top \circ \times_{k+1}^{k+1} \circ \cdots \circ \times_{m-1}^{m-1}\right)\mathcal{T}^{(0)}\right) \tag{E.11b}$$

where the exact form of the adjoint operator $(*_l^k)^\top$ depends on the original definition of vector convolution. Generally, the trick to start with circular convolution and generalize to general cases is very useful to derive backpropagation equations for operations that convolution plays a part.

Despite varieties in the definitions of tensor convolution, the analyses of their time complexities of backpropagation equations are identical, since the numbers of operations only differ by a constant for different definitions (therefore asymptotically the same). With FFT, the number of operations for these two backpropagation equations are $O((\prod_{u=0,u\neq k}^{m-1} I_u)(\prod_{v=0,v\neq l}^{n-1} J_v)\max(I'_k, J_l)\log(\max(I'_k, J_l)))$ and $O((\prod_{u=0,u\neq k}^{m-1} I_u)(\prod_{v=0,v\neq l}^{n-1} J_v)\max(I'_k, I_k)\log(\max(I'_k, I_k)))$, and without FFT $O((\prod_{u=0,u\neq k}^{m-1} I_u)(\prod_{v=0,v\neq l}^{n-1} J_v)I'_k J_l)$ and $O((\prod_{u=0,u\neq k}^{m-1} I_u)(\prod_{v=0,v\neq l}^{n-1} J_v)I'_k I_k)$ respectively. Different from other operations, the time complexities for forward and backward passes are different (with circular convolution as an exception). This asymmetry can be utilized in neural networks (where$I'_k \approx I_k \gg J_l$) to accelerate backpropagation with FFT.

Tensor outer product The derivatives of $\mathcal{T}$ with respect to $\mathcal{T}^{(0)}$ and $\mathcal{T}^{(1)}$ can be directly observed from the definition of tensor outer product in Equation D.4.

$$\frac{\partial \mathcal{T}_{i_0,\cdots,i_{m-1},j_0,\cdots,j_{n-1}}}{\partial \mathcal{T}^{(0)}_{i_0,\cdots,i_{m-1}}} = \mathcal{T}^{(1)}_{j_0,\cdots,j_{n-1}} \tag{E.12a}$$

$$\frac{\partial \mathcal{T}_{i_0,\cdots,i_{m-1},j_0,\cdots,j_{n-1}}}{\partial \mathcal{T}^{(1)}_{j_0,\cdots,j_{n-1}}} = \mathcal{T}^{(0)}_{i_0,\cdots,i_{m-1}} \tag{E.12b}$$

Similar to all previously discussed operations, we first derive the backpropagation equations for $\mathcal{T}^{(0)}$ and $\mathcal{T}^{(1)}$ at the entries level using standard chain rule.

$$\frac{\partial \mathcal{L}}{\partial \mathcal{T}^{(0)}_{i_0,\cdots,i_{m-1}}} = \sum_{j_0=0}^{J_0-1} \cdots \sum_{j_{n-1}=0}^{J_{n-1}-1} \frac{\partial \mathcal{L}}{\partial \mathcal{T}_{i_0,\cdots,i_{m-1},j_0,\cdots,j_{n-1}}} \mathcal{T}^{(1)}_{j_0,\cdots,j_{n-1}} \tag{E.13a}$$

$$\frac{\partial \mathcal{L}}{\partial \mathcal{T}^{(1)}_{j_0,\cdots,j_{n-1}}} = \sum_{i_0=0}^{I_0-1} \cdots \sum_{i_{m-1}=0}^{I_{m-1}-1} \frac{\partial \mathcal{L}}{\partial \mathcal{T}_{i_0,\cdots,i_{m-1},j_0,\cdots,j_{n-1}}} \mathcal{T}^{(0)}_{i_0,\cdots,i_{m-1}} \tag{E.13b}$$

which can then be converted to tensor notations in this paper:

$$\frac{\partial \mathcal{L}}{\partial \mathcal{T}^{(0)}} = \frac{\partial \mathcal{L}}{\partial \mathcal{T}} \left(\times_0^m \circ \cdots \circ \times_{n-1}^{m+n-1}\right) \mathcal{T}^{(1)} \tag{E.14a}$$

$$\frac{\partial \mathcal{L}}{\partial \mathcal{T}^{(1)}} = \frac{\partial \mathcal{L}}{\partial \mathcal{T}} \left(\times_0^0 \circ \cdots \circ \times_{m-1}^{m-1}\right) \mathcal{T}^{(0)} \tag{E.14b}$$

The number of operations required for both equations are $O((\prod_{u=0,u\neq k}^{m-1} I_u)(\prod_{v=0}^{n-1} J_v))$, which are again identical to one in the forward pass in Equation D.4.

Tensor partial outer product Finally, the derivatives of $\mathcal{T}$ with respect to $\mathcal{T}^{(0)}$ and $\mathcal{T}^{(1)}$ can be obtained from the definition of tensor partial outer product in Equation D.5a.

$$\frac{\partial \mathcal{T}_{i_0,\cdots,i_{k-1},r,i_{k+1},\cdots,i_{m-1},j_0,\cdots,j_{l-1},j_{l+1},\cdots,j_{n-1}}}{\partial \mathcal{T}^{(0)}_{i_0,\cdots,i_{k-1},r,i_{k+1},\cdots,i_{m-1}}} = \mathcal{T}^{(1)}_{j_0,\cdots,j_{l-1},r,j_{l+1},\cdots,j_{n-1}} \tag{E.15a}$$

$$\frac{\partial \mathcal{T}_{i_0,\cdots,i_{k-1},r,i_{k+1},\cdots,i_{m-1},j_0,\cdots,j_{l-1},j_{l+1},\cdots,j_{n-1}}}{\partial \mathcal{T}^{(1)}_{j_0,\cdots,j_{l-1},r,j_{l+1},\cdots,j_{n-1}}} = \mathcal{T}^{(0)}_{i_0,\cdots,i_{k-1},r,i_{k+1},\cdots,i_{m-1}} \tag{E.15b}$$

Again with chain rule, the backpropagation equations for $\mathcal{T}^{(0)}$ and $\mathcal{T}^{(1)}$ at the entries level are

$$\frac{\partial \mathcal{L}}{\partial \mathcal{T}^{(0)}_{i_0,\cdots,i_{k-1},r,i_{k+1},\cdots,i_{m-1}}} = \sum_{j_0=0}^{J_0-1} \cdots \sum_{j_{l-1}=0}^{J_{l-1}-1} \sum_{j_{l+1}=0}^{J_{l+1}-1} \cdots \sum_{j_{n-1}=0}^{J_{n-1}-1} \frac{\partial \mathcal{L}}{\partial \mathcal{T}_{i_0,\cdots,i_{k-1},r,i_{k+1},\cdots,i_{m-1},j_0,\cdots,j_{l-1},j_{l+1},\cdots,j_{n-1}}} \mathcal{T}^{(1)}_{j_0,\cdots,j_{l-1},r,j_{l+1},\cdots,j_{n-1}} \tag{E.16a}$$

$$\frac{\partial \mathcal{L}}{\partial \mathcal{T}^{(1)}_{j_0,\cdots,j_{l-1},r,j_{l+1},\cdots,j_{n-1}}} = \sum_{i_0=0}^{I_0-1} \cdots \sum_{i_{k-1}=0}^{I_{k-1}-1} \sum_{i_{k+1}=0}^{I_{k+1}-1} \cdots \sum_{i_{m-1}=0}^{I_{m-1}-1} \frac{\partial \mathcal{L}}{\partial \mathcal{T}_{i_0,\cdots,i_{k-1},r,i_{k+1},\cdots,i_{m-1},j_0,\cdots,j_{l-1},j_{l+1},\cdots,j_{n-1}}} \mathcal{T}^{(0)}_{i_0,\cdots,i_{k-1},r,i_{k+1},\cdots,i_{m-1}} \tag{E.16b}$$

Though the backpropagation equations above appear very similar to the ones for tensor contraction in Equations E.1a and E.1b, written in tensor notations, they are almost the same as the ones for tensor convolution in Equations E.10a and E.10b, except that $(*_l^k)^\top$'s are now replaced by $\otimes_l^k$.

$$\frac{\partial \mathcal{L}}{\partial \mathcal{T}^{(0)}} = \mathsf{swapaxes}\left(\frac{\partial \mathcal{L}}{\partial \mathcal{T}} \left(\times_0^m \circ \cdots \circ \times_{l-1}^{m+l-1} \circ \otimes_l^k \circ \times_{l+1}^{m+l-2} \circ \cdots \circ \times_{n-1}^{m+n-2}\right) \mathcal{T}^{(1)}\right) \tag{E.17a}$$

$$\frac{\partial \mathcal{L}}{\partial \mathcal{T}^{(1)}} = \mathsf{swapaxes}\left(\frac{\partial \mathcal{L}}{\partial \mathcal{T}} \left(\times_0^0 \circ \cdots \circ \times_{k-1}^{k-1} \circ \otimes_k^k \circ \times_{k+1}^{k+1} \circ \cdots \circ \times_{m-1}^{m-1}\right) \mathcal{T}^{(0)}\right) \tag{E.17b}$$

It is not difficult to recognize the time complexity for the two equations above are $O((\prod_{u=0}^{m-1} I_u)(\prod_{v=0,v\neq l}^{n-1} J_v))$, which are identical to the ones in forward pass in Equation D.5a.

Compound operations Up to this point, we have developed the derivatives and backpropagation equations for all basic operations. In this part, we will continue to show similar techniques above equally apply to compound operations, though slightly more involved, and derive the backpropagation equations for the examples we used in Appendix D. Though these equations are not immediately useful in later sections, the techniques to derive them are useful for all other compound operations. Furthermore, these induced equations, which are more complicated than their original definitions, serve as complementary examples of compound operations to the ones in the last section.

- **Simultaneous multi-operations between two tensors.** In Appendix D, we introduced a compound operation $(\otimes_0^0 \circ *_1^1 \circ \times_2^2)$ on two tensors $\mathcal{T}^{(0)} \in \mathbb{R}^{R\times X\times S}$ and $\mathcal{T}^{(1)} \in \mathbb{R}^{R\times H\times S}$, which returns a tensor $\mathcal{T} \in \mathbb{R}^{R\times X'}$. Here, we recap its definitions as follows:

$$\mathcal{T} \triangleq \mathcal{T}^{(0)} \left(\otimes_0^0 \circ *_1^1 \circ \times_2^2\right) \mathcal{T}^{(1)} \tag{E.18a}$$

$$\mathcal{T}_{r,:} \triangleq \sum_{s=0}^{S-1} \mathcal{T}^{(0)}_{r,:,s} * \mathcal{T}^{(1)}_{r,:,s} \tag{E.18b}$$

To ease the derivation, we use the trick to start with circular convolution: directly apply the chain rule, the backpropagation equations at entries level are obtained as follows:

$$\frac{\partial \mathcal{L}}{\partial \mathcal{T}^{(0)}_{r,:,s}} = \mathsf{Cir}\left(\mathcal{T}^{(1)}_{r,:,s}\right)^\top \frac{\partial \mathcal{L}}{\mathcal{T}_{r,:}} \tag{E.19a}$$

$$\frac{\partial \mathcal{L}}{\partial \mathcal{T}^{(1)}_{r,:,s}} = \mathsf{Cir}\left(\mathcal{T}^{(0)}_{r,:,s}\right)^\top \frac{\partial \mathcal{L}}{\mathcal{T}_{r,:}} \tag{E.19b}$$

Now we convert the equations above to tensor notations, and replace the circular convolutions with their adjoints to obtain general backpropagation rules:

$$\frac{\partial \mathcal{L}}{\partial \mathcal{T}^{(0)}} = \frac{\partial \mathcal{L}}{\partial \mathcal{T}} \left(\otimes_0^0 \circ *_1^1\right) \mathsf{flipaxis}(\mathcal{T}^{(1)}) = \frac{\partial \mathcal{L}}{\partial \mathcal{T}} \left(\otimes_0^0 \circ (*_1^1)^\top\right) \mathcal{T}^{(1)} \tag{E.20a}$$

$$\frac{\partial \mathcal{L}}{\partial \mathcal{T}^{(1)}} = \frac{\partial \mathcal{L}}{\partial \mathcal{T}} \left(\otimes_0^0 \circ *_1^1\right) \mathsf{flipaxis}(\mathcal{T}^{(0)}) = \frac{\partial \mathcal{L}}{\partial \mathcal{T}} \left(\otimes_0^0 \circ (*_1^1)^\top\right) \mathcal{T}^{(0)} \tag{E.20b}$$

For simplicity, we assume FFT is not used to accelerate the backpropagation equations. In this case, the derivatives with respect to $\mathcal{T}^{(0)}$ and $\mathcal{T}^{(1)}$ can be computed in $O(RHX'ST)$ and $O(RXX'ST)$ operations respectively. Again, the time complexities of forward and backward passes are not the same when a (compound) tensor operation contains convolution.

- **Simultaneous operations between a tensor and a set of multiple tensors.** Another compound operation presented in Appendix D is defined between a tensor $\mathcal{U} \in \mathbb{R}^{R\times X\times S}$ and a set of tensors $\mathcal{T}^{(0)} \in \mathbb{R}^{R\times P}$, $\mathcal{T}^{(1)} \in \mathbb{R}^{H\times Q}$ and $\mathcal{T}^{(2)} \in \mathbb{R}^{S\times T}$, which returns a tensor $\mathcal{V} \in \mathbb{R}^{R\times X'\times P\times Q\times T}$. Again, we recap its definitions in the following:

$$\mathcal{V} \triangleq \mathcal{U}\left(\otimes_0^0 \mathcal{T}^{(0)} \ *_0^1 \mathcal{T}^{(1)} \ \times_0^2 \mathcal{T}^{(2)}\right) \tag{E.21a}$$

$$\mathcal{V}_{r,:,p,q,t} \triangleq \sum_{s=0}^{S-1} \mathcal{T}^{(0)}_{r,p} \left(\mathcal{U}_{r,:,s} * \mathcal{T}^{(1)}_{:,q}\right) \mathcal{T}^{(2)}_{s,t} \tag{E.21b}$$

In order to derive the backpropagation rule for the *core tensor* $\mathcal{U}$, we follow the standard procedure to (1) first obtain its entries level representation, and (2) explicitly convert it to tensor notations subsequently. Concretely, the backpropagation equation in both formats are displayed as follows:

$$\frac{\partial \mathcal{L}}{\partial \mathcal{U}_{r,:,s}} = \sum_{p=0}^{P-1}\sum_{q=0}^{Q-1}\sum_{t=0}^{T-1} \mathcal{T}^{(0)}_{r,p} \left(\mathsf{Cir}\left(\mathcal{T}^{(1)}_{:,q}\right)^\top \frac{\partial \mathcal{L}}{\partial \mathcal{V}_{r,:,p,q,t}}\right) \mathcal{T}^{(2)}_{s,t} \tag{E.22a}$$

$$\frac{\partial \mathcal{L}}{\partial \mathcal{U}} = \frac{\partial \mathcal{L}}{\partial \mathcal{V}} \left(\left(\otimes_0^0 \circ \times_2^1\right) \mathcal{T}^{(0)} \left((*_1^0)^\top \circ \times_3^1\right) \mathcal{T}^{(1)} \left(\times_2^0 \circ \times_4^1\right) \mathcal{T}^{(2)}\right) \tag{E.22b}$$

Notice that the equation above is indeed **simultaneous multi-operations between a tensor and a set of multiple tensors**, which combines two types of "basic" compound operations introduced in Appendix D. In principle, we can obtain backpropagation equations for $\{\mathcal{T}^{(0)}, \mathcal{T}^{(1)}, \mathcal{T}^{(2)}\}$ in the same manner. However, there is a simpler way to derive them by rewriting the definition as:

$$\mathcal{V} = \mathsf{swapaxes}\left(\mathcal{U}\left(*_0^1 \mathcal{T}^{(1)} \times_0^2 \mathcal{T}^{(2)}\right) \otimes_0^0 \mathcal{T}^{(0)}\right) = \mathsf{swapaxes}\left(\mathcal{U}^{(0)} \otimes_0^0 \mathcal{T}^{(0)}\right) \tag{E.23a}$$

$$= \mathsf{swapaxes}\left(\mathcal{U}\left(\otimes_0^0 \mathcal{T}^{(0)} \times_0^2 \mathcal{T}^{(2)}\right) *_0^1 \mathcal{T}^{(1)}\right) = \mathsf{swapaxes}\left(\mathcal{U}^{(1)} *_1^1 \mathcal{T}^{(1)}\right) \tag{E.23b}$$

$$\mathcal{V} = \mathsf{swapaxes}\left(\mathcal{U}\left(\otimes_0^0 \mathcal{T}^{(0)} *_0^1 \mathcal{T}^{(1)}\right) \times_0^2 \mathcal{T}^{(2)}\right) = \mathsf{swapaxes}\left(\mathcal{U}^{(2)} \times_2^2 \mathcal{T}^{(2)}\right) \tag{E.23c}$$

where $\mathcal{U}^{(0)}$, $\mathcal{U}^{(1)}$ and $\mathcal{U}^{(2)}$ are short-hand notations for $\mathcal{U}(*_0^1\mathcal{T}^{(1)} \times_0^2 \mathcal{T}^{(2)})$, $\mathcal{U}(\otimes_0^0\mathcal{T}^{(0)} \times_0^2 \mathcal{T}^{(2)})$ and $\mathcal{U}(\otimes_0^0\mathcal{T}^{(0)} *_0^1 \mathcal{T}^{(1)})$. With these notations, we are able to reduce these complex expressions to basic ones, by which we can reuse the backpropagation rules derived in this section:

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \mathcal{T}^{(0)}} &= \frac{\partial \mathcal{L}}{\partial \mathcal{V}} \left(\otimes_0^0 \circ \times_1^1 \circ \times_2^3 \circ \times_3^4\right) \mathcal{U}^{(0)} \\
&= \frac{\partial \mathcal{L}}{\partial \mathcal{V}} \left(\otimes_0^0 \circ \times_1^1 \circ \times_2^3 \circ \times_3^4\right) \left(\mathcal{U}\left(*_0^1\mathcal{T}^{(1)} \times_0^2 \mathcal{T}^{(2)}\right)\right) && \text{(E.24a)} \\
\frac{\partial \mathcal{L}}{\partial \mathcal{T}^{(1)}} &= \frac{\partial \mathcal{L}}{\partial \mathcal{V}} \left(\times_0^0 \circ (*_1^1)^\top \circ \times_2^2 \circ \times_3^4\right) \mathcal{U}^{(1)} \\
&= \frac{\partial \mathcal{L}}{\partial \mathcal{V}} \left(\times_0^0 \circ (*_1^1)^\top \circ \times_2^2 \circ \times_3^4\right) \left(\mathcal{U}\left(\otimes_0^0\mathcal{T}^{(0)} \times_0^2 \mathcal{T}^{(2)}\right)\right) && \text{(E.24b)} \\
\left(\frac{\partial \mathcal{L}}{\partial \mathcal{T}^{(2)}}\right)^\top &= \frac{\partial \mathcal{L}}{\partial \mathcal{V}} \left(\times_0^0 \circ \times_1^1 \circ \times_3^2 \circ \times_4^3\right) \mathcal{U}^{(2)} \\
&= \frac{\partial \mathcal{L}}{\partial \mathcal{V}} \left(\times_0^0 \circ \times_1^1 \circ \times_3^2 \circ \times_4^3\right) \left(\mathcal{U}\left(\otimes_0^0\mathcal{T}^{(0)} *_0^1 \mathcal{T}^{(1)}\right)\right) && \text{(E.24c)}
\end{aligned}$$

The complicity of tensor operation culminates at this point: the equations above are examples of **simultaneous multi-operations on multiple tensors**, which we omitted in the discussion in Appendix D due to their complexity. Although the expressions themselves suggest particular orderings to evaluate the compound operations, they are merely the traces of the techniques used in deriving them. It is completely reasonable to reorganize the equations such that they can be computed with more efficient strategies: for instance, one can verify that the following set of equations is actually equivalent to the one above:

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \mathcal{T}^{(0)}} &= \frac{\partial \mathcal{L}}{\partial \mathcal{V}} \left(\left((*_0^1)^\top \circ \times_1^3\right) \mathcal{T}^{(1)} \times_1^4 \mathcal{T}^{(2)}\right) \left(\otimes_0^0 \circ \times_1^1 \circ \times_2^3\right) \mathcal{U} && \text{(E.25a)} \\
\frac{\partial \mathcal{L}}{\partial \mathcal{T}^{(1)}} &= \frac{\partial \mathcal{L}}{\partial \mathcal{V}} \left(\left(\otimes_0^0 \circ \times_1^2\right) \mathcal{T}^{(0)} \times_1^4 \mathcal{T}^{(2)}\right) \left(\times_0^0 \circ (\overline{*}_1^1)^\top \circ \times_2^3\right) \mathcal{U} && \text{(E.25b)} \\
\left(\frac{\partial \mathcal{L}}{\partial \mathcal{T}^{(2)}}\right)^\top &= \frac{\partial \mathcal{L}}{\partial \mathcal{V}} \left(\left(\otimes_0^0 \circ \times_1^2\right) \mathcal{T}^{(0)} \left((*_0^1)^\top \circ \times_1^3\right) \mathcal{T}^{(1)}\right) \left(\times_0^0 \circ \times_1^1\right) \mathcal{U} && \text{(E.25c)}
\end{aligned}$$

As discussed in Appendix D, the problem to find the optimal order to evaluate a compound operation over multiple tensors is NP-hard in general and usually we need to resort to heuristics to obtain a reasonably efficient algorithm. Indeed, one can verify that the second set of equations is more efficient than the first one. For this example, interested readers are encouraged to find the most efficient way by combinatoric search.

F TENSOR DECOMPOSITIONS

Tensor decompositions are natural extensions of matrix factorizations for multi-dimensional arrays. In this section, we will review three commonly used tensor decompositions, namely *CANDECOMP/PARAFAC (CP) decomposition*, *Tucker (TK) decomposition* and *Tensor-train (TT) decomposition*. CP and Tucker decompositions are classic methods to factorize a tensor and are thoroughly reviewed in an early survey Kolda & Bader (2009), while Tensor-train decomposition is recently proposed in Oseledets (2011), in which the authors show that it has superior numerical stability and scalability than the classic ones. More advanced schemes of tensor decompositions (known as *tensor networks*) are reviewed in Cichocki et al. (2016).

For each of these decompositions, we will present their forms both at the entries level and in tensor notations introduced in Appendix D. When tensor decompositions are used in neural networks, a natural question to ask is how the backpropagation algorithm adapts to the decomposition schemes, i.e. *how the gradient of the original tensor backpropagates to its factors*. In this section, we will follow the standard procedure in Appendix E to derive the corresponding backpropagation equation for each tensor decomposition. Different from previous works (Novikov et al., 2015; Kossaifi et al., 2017b) that use matrix calculus following *matricization*, we present the backpropagation equations directly in tensor notations, which makes our presentation concise and easy to analyze. As we will see in the analyses, backpropagation equations through the original tensor to its factors are

computationally expensive for all decomposition schemes, therefore it is preferable to avoid explicit computation of these equations in practice.

CP decomposition CP decomposition is a direct generalization of singular value decomposition (SVD) which decomposes a tensor into additions of rank-1 tensors (outer product of multiple vectors). Specifically, given an m-order tensor $\mathcal{T} \in \mathbb{R}^{I_0 \times I_1 \times \cdots \times I_{m-1}}$, CP decomposition factorizes it into m factor matrices $\{\mathbf{M}^{(0)}\}_{l=0}^{m-1}$, where $\mathbf{M}^{(l)} \in \mathbb{R}^{R \times I_l}, \forall l \in [m]$, where R is called the *canonical rank* of the CP decomposition, which is allowed to be larger than the I_l's.

$$\mathcal{T}_{i_0,\cdots,i_{m-1}} \triangleq \sum_{r=0}^{R-1} \mathbf{M}^{(0)}_{r,i_0} \cdots \mathbf{M}^{(m-1)}_{r,i_{m-1}} \tag{F.1a}$$

$$\begin{aligned} \mathcal{T} &\triangleq \sum_{r=0}^{R-1} \mathbf{M}^{(0)}_{r,:} \otimes \cdots \otimes \mathbf{M}^{(m-1)}_{r,:} \\ &= \mathbf{1} \times_0^0 \left(\mathbf{M}^{(0)} \otimes_0^0 \cdots \otimes_0^0 \mathbf{M}^{(m-1)} \right) \end{aligned} \tag{F.1b}$$

where $\mathbf{1} \in \mathbb{R}^R$ is an all-ones vector of length R. With CP decomposition, $\mathcal{T}$ can be represented with only $(\sum_{l=0}^{m-1} I_l)R$ entries instead of $(\prod_{l=0}^{m-1} I_l)$ as in the original tensor.

Now we proceed to derive the backpropagation rules for CP decomposition, i.e. the equations relating $\partial \mathcal{L}/\partial \mathcal{T}$ to $\{\partial \mathcal{L}/\partial \mathbf{M}^{(l)}\}_{l=0}^{m-1}$. In order to avoid deriving these equations from the entries level, we first isolate the factor of interest and rewrite the definition of CP decomposition as:

$$\begin{aligned} \mathcal{T} &= \mathsf{swapaxes} \left(\left(\mathbf{M}^{(0)} \cdots \otimes_0^0 \mathbf{M}^{(l-1)} \otimes_0^0 \mathbf{M}^{(l+1)} \cdots \otimes_0^0 \mathbf{M}^{(m-1)} \right) \times_0^0 \mathbf{M}^{(l)} \right) \\ &= \mathsf{swapaxes} \left(\mathcal{A}^{(l)} \times_0^0 \mathbf{M}^{(l)} \right) \end{aligned} \tag{F.2}$$

where we treat the first term as a constant tensor $\mathcal{A}^{(l)}$. Once we reduce the compound operation to a basic one, we can simply refer to the rule derived in Appendix E, which gives us

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \mathbf{M}^{(l)}} &= \frac{\partial \mathcal{L}}{\partial \mathcal{T}} \left(\times_1^0 \cdots \circ \times_l^{l-1} \circ \times_{l+1}^{l+1} \cdots \times_{m-1}^{m-1} \right) \mathcal{A}^{(l)} \\ &= \frac{\partial \mathcal{L}}{\partial \mathcal{T}} \left(\times_1^0 \cdots \circ \times_l^{l-1} \circ \times_{l+1}^{l+1} \cdots \times_{m-1}^{m-1} \right) \\ &\quad \left(\mathbf{M}^{(0)} \otimes_0^0 \cdots \otimes_0^0 \mathbf{M}^{(l-1)} \otimes_0^0 \mathbf{M}^{(l+1)} \otimes_0^0 \cdots \otimes_0^0 \mathbf{M}^{(m-1)} \right) \end{aligned} \tag{F.3}$$

The number of operations to compute one such equation both is $O((\prod_{l=0}^{m-1} I_l)R)$, therefore a total number of $O(m(\prod_{l=0}^{m-1} I_l)R)$ is required for all m equations. Therefore, evaluating these equations are computationally expensive (which takes $O(mR)$ order as many operations as the size $(\prod_{l=0}^{m-1} I_l)$ of the original tensor $\mathcal{T}$), and should be avoided whenever possible.

Tucker decomposition Tucker decomposition provides more general factorization than CP decomposition. Given an m-order tensor $\mathcal{T} \in \mathbb{R}^{I_0 \times I_1 \times \cdots \times I_{m-1}}$, Tucker decomposition factors it into m factor matrices $\{\mathbf{M}^{(l)}\}_{l=0}^{m-1}$, where $\mathbf{M}^{(l)} \in \mathbb{R}^{R_l \times I_l}, \forall l \in [m]$ and an additional m-order core tensor $\mathcal{C} \in \mathbb{R}^{R_0 \times R_1 \times \cdots \times R_{m-1}}$, where the *Tucker ranks* R_l's are required to be smaller or equal than the dimensions at their corresponding modes, i.e. $R_l \leq I_l, \forall l \in [m]$.

$$\mathcal{T}_{i_0,\cdots,i_{m-1}} \triangleq \sum_{r_0=0}^{R_0-1} \cdots \sum_{r_{m-1}=0}^{R_{m-1}-1} \mathcal{C}_{r_0,\ldots,r_{m-1}} \mathbf{M}^{(0)}_{r_0,i_0} \cdots \mathbf{M}^{(m-1)}_{r_{m-1},i_{m-1}} \tag{F.4a}$$

$$\mathcal{T} \triangleq \mathcal{C} \left(\times_0 \mathbf{M}^{(0)} \times_1 \mathbf{M}^{(1)} \cdots \times_{m-1} \mathbf{M}^{(m-1)} \right) \tag{F.4b}$$

Notice that when $R_0 = \cdots = R_{m-1} = R$ and $\mathcal{C}$ is a super-diagonal tensor with all super-diagonal entries to be ones (a.k.a. identity tensor), Tucker decomposition reduces to CP decomposition, and therefore CP decomposition is a special case of Tucker decomposition. With Tucker decomposition, a tensor is approximately by $(\prod_{l=0}^{m-1} R_l + \sum_{l=0}^{m-1} I_l R_l)$ entries.

The backpropagation equations relating $\partial\mathcal{L}/\partial\mathcal{T}$ to $\partial\mathcal{L}/\partial\mathcal{C}$ and $\{\partial\mathcal{L}/\mathbf{M}^{(l)}\}_{l=0}^{m-1}$ can be derived similarly as in CP decomposition. First, we derive the equation for $\mathcal{C}$ at the entries level:

$$\frac{\partial\mathcal{L}}{\partial\mathcal{C}_{r_0,\cdots,r_{m-1}}} = \sum_{i_0=0}^{I_0-1}\cdots\sum_{i_{m-1}=0}^{I_{m-1}-1}\frac{\partial\mathcal{L}}{\partial\mathcal{T}_{i_0,\cdots,i_{m-1}}}\,\mathbf{M}^{(0)}_{r_0,i_0}\cdots\mathbf{M}^{(m-1)}_{r_{m-1},i_{m-1}} \tag{F.5}$$

The equation above, written in tensor notations, reveals an expression in "reversed" Tucker form:

$$\frac{\partial\mathcal{L}}{\partial\mathcal{C}} = \frac{\partial\mathcal{L}}{\partial\mathcal{T}}\left(\times_0(\mathbf{M}^{(0)})^\top\cdots\times_{m-1}(\mathbf{M}^{(m-1)})^\top\right) \tag{F.6}$$

Although the number of operations to evaluate the equation depends on the particular order of tensor multiplications between $\partial\mathcal{L}/\partial\mathcal{T}$ and $\{\mathbf{M}^{(l)})^\top\}_{l=0}^{m-1}$, the time complexity can be bounded by $O((\sum_{l=0}^{m-1}R_l)(\prod_{l=0}^{m-1}I_l))$, where the worst case is achieved when $R_l = I_l, \forall l \in [m]$. Regarding the backpropagation equations for the factors $\{\mathbf{M}^{(l)}\}_{l=0}^{m-1}$, we again isolate the factor of interest and rewrite the definition of Tucker decomposition as:

$$\begin{aligned}\mathcal{T} &= \left(\mathcal{C}\left(\times_0\mathbf{M}^{(0)}\cdots\times_{l-1}\mathbf{M}^{(l-1)}\times_{l+1}\mathbf{M}^{(l+1)}\cdots\times_{m-1}\mathbf{M}^{(m-1)}\right)\right)\times_l\mathbf{M}^{(l)} \\ &= \mathcal{A}^{(l)}\times_l\mathbf{M}^{(l)}\end{aligned} \tag{F.7}$$

where the first term is abbreviated as a tensor $\mathcal{A}^{(l)}$. Subsequently, we apply the standard backpropagation rule of tensor multiplication in Appendix E and obtain the following equation:

$$\begin{aligned}\left(\frac{\partial\mathcal{L}}{\partial\mathbf{M}^{(l)}}\right)^\top =& \frac{\partial\mathcal{L}}{\partial\mathcal{T}}\left(\times_0^0\circ\cdots\circ\times_{l-1}^{l-1}\circ\times_{l+1}^{l+1}\circ\cdots\circ\times_{m-1}^{m-1}\right)\mathcal{A}^{(l)} \\ =& \frac{\partial\mathcal{L}}{\partial\mathcal{T}}\Bigg(\times_0(\mathbf{M}^{(0)})^\top\cdots\times_{l-1}(\mathbf{M}^{(l-1)})^\top\times_{l+1}(\mathbf{M}^{(l+1)})^\top \\ & \cdots\times_{m-1}(\mathbf{M}^{(m-1)})^\top\Bigg)\left(\times_0^0\circ\cdots\circ\times_{l-1}^{l-1}\circ\times_{l+1}^{l+1}\circ\cdots\circ\times_{m-1}^{m-1}\right)\mathcal{C}\end{aligned} \tag{F.8}$$

where the second expression is equivalent to the first one, but requires fewer operations. Though the exact number of operations depends on the order of tensor multiplications, it can be (again) bounded by $O((\sum_{l=0}^{m-1}R_l)(\prod_{l=0}^{m-1}I_l))$. Therefore, the total time complexity for all $(m+1)$ equations (m for $\{\mathbf{M}^{(l)}\}_{l=0}^{m-1}$ and one for $\mathcal{C}$) is bounded by $O((m+1)(\sum_{l=0}^{m-1}R_l)(\prod_{l=0}^{m-1}I_l))$, which is also highly inefficient and should be avoided in practice.

Tensor-train decomposition Tensor-train decomposition factorizes a m-order tensor into m interconnected low-order core tensors $\{\mathcal{T}^{(l)}\}_{l=0}^{m-1}$, where $\mathcal{T}^{(l)} \in \mathbb{R}^{R_l\times I_l\times R_{l+1}}, l = 1,\cdots,m-2$ with $\mathcal{T}^{(0)} \in \mathbb{R}^{I_0\times R_0}$, and $\mathcal{T}^{(m-1)} \in \mathbb{R}^{R_{m-1}\times I_{m-1}}$ such that

$$\mathcal{T}_{i_0,\ldots,i_{m-1}} \triangleq \sum_{r_0=1}^{R_0-1}\cdots\sum_{r_{m-2}=1}^{R_{m-2}-1}\mathcal{T}^{(0)}_{i_0,r_0}\,\mathcal{T}^{(1)}_{r_0,i_1,r_1}\cdots\mathcal{T}^{(m-1)}_{r_{m-2},i_{m-1}} \tag{F.9a}$$

$$\mathcal{T} \triangleq \mathcal{T}^{(0)}\times_0^{-1}\mathcal{T}^{(1)}\times_0^{-1}\cdots\times_0^{-1}\mathcal{T}^{(m-1)} \tag{F.9b}$$

where the R_l's are known as *Tensor-train ranks*, which controls the tradeoff between the number of parameters and accuracy of representation. With Tensor-train decomposition, a tensor is represented by $(R_0I_0 + \sum_{l=1}^{m-2}R_lI_lR_{l+1} + R_{m-1}I_{m-1})$ entries.

The backpropagation equations are derived following the paper Novikov et al. (2015), although we reformat them in tensor notations. To begin with, we introduce two sets of auxiliary tensors $\{\mathcal{P}^{(l)}\}_{l=0}^{m-1}$ and $\{\mathcal{Q}^{(l)}\}_{l=0}^{m-1}$ as follows:

$$\mathcal{P}^{(l)} = \mathcal{T}^{(0)}\times_0^{-1}\cdots\times_0^{-1}\mathcal{T}^{(l)} = \mathcal{P}^{(l-1)}\times_0^{-1}\mathcal{T}^{(l)},\ \forall l = 1,\cdots,m-1 \tag{F.10a}$$

$$\mathcal{Q}^{(l)} = \mathcal{T}^{(l)}\times_0^{-1}\cdots\times_0^{-1}\mathcal{T}^{(m-1)} = \mathcal{T}^{(l)}\times_0^{-1}\mathcal{Q}^{(l+1)},\ \forall l = 0,\cdots,m-2 \tag{F.10b}$$

with corner cases as $\mathcal{P}^{(0)} = \mathcal{T}^{(0)}$ and $\mathcal{Q}^{(m-1)} = \mathcal{T}^{(m-1)}$. Note that both $\{\mathcal{P}^{(l)}\}_{l=0}^{m-1}$ and $\{\mathcal{Q}^{(l)}\}_{l=0}^{m-1}$ can be computed using dynamic programming (DP) using the recursive definitions

above. With these auxiliary tensors, the definition of Tensor-train decomposition in Equation F.9b can be rewritten as:

$$\begin{aligned}\mathcal{T} &= \mathcal{P}^{(l-1)} \times_0^{-1} \mathcal{T}^{(l)} \times_0^{-1} \mathcal{Q}^{(l+1)},\ \forall l \neq 0, m-1 \\ &= \mathcal{T}^{(0)} \times_0^{-1} \mathcal{Q}^{(1)} = \mathcal{P}^{(m-2)} \times_0^{-1} \mathcal{T}^{(m-1)}\end{aligned} \tag{F.11}$$

Applying the backpropagation rule for tensor contraction twice, the backpropagation equations can be obtained in tensor notations as:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \mathcal{T}^{(l)}} &= \mathcal{P}^{(l-1)} \left(\times_0^0 \circ \cdots \circ \times_{l-1}^{l-1}\right) \frac{\partial \mathcal{L}}{\partial \mathcal{T}} \left(\times_1^2 \circ \cdots \circ \times_{m-l-1}^{m-l}\right) \mathcal{Q}^{(l+1)},\ \forall l \neq 0, m-1 \\ \frac{\partial \mathcal{L}}{\partial \mathcal{T}^{(0)}} &= \frac{\partial \mathcal{L}}{\partial \mathcal{T}} \left(\times_1^1 \circ \cdots \circ \times_{m-1}^{m-1}\right) \mathcal{Q}^{(1)},\ \frac{\partial \mathcal{L}}{\partial \mathcal{T}^{(m-1)}} = \mathcal{P}^{(m-2)} \left(\times_0^0 \circ \cdots \circ \times_{m-2}^{m-2}\right) \frac{\partial \mathcal{L}}{\partial \mathcal{T}}\end{aligned} \tag{F.12}$$

Neglecting the expense of precomputing the auxiliary tensors $\{\mathcal{P}^{(l)}\}_{l=0}^{m-1}$ and $\{\mathcal{Q}^{(l)}\}_{l=0}^{m-1}$, the time complexity for the l^{th} equation is $O(R_l R_{l+1}(\prod_{l=0}^{m-1} I_l))$, therefore the total number of operations for all m equations is $O((R_0 + \sum_{l=0}^{m-2} R_l R_{l+1} + R_{m-1})(\prod_{l=0}^{m-1} I_l))$, which is again prohibitively large for practical use.

Variants of standard decompositions In this paper, tensor decompositions are usually used in flexible ways, i.e. we will not stick to the standard formats defined in the previous paragraphs. Indeed, we consider tensor decomposition as a reverse mapping of tensor operations: given a tensor $\mathcal{T}$ and a set of operations, the corresponding tensor decomposition aims to recover the input factors $\{\mathcal{T}^{(l)}\}_{l=0}^{m-1}$ such that the operations on these factors return a tensor approximately equal to the given one. In the following, we demonstrate some possibilities using examples:

- *The ordering of the modes can be arbitrary.* Therefore, CP decomposition of 3-order tensor $\mathcal{T} \in \mathbb{R}^{I_0 \times I_1 \times I_2}$ can be factorized as $\mathcal{T}_{i_0,i_1,i_2} = \sum_{r=0}^{R-1} \mathbf{M}^{(0)}_{i_0,r}\ \mathbf{M}^{(1)}_{i_1,r}\ \mathbf{M}^{(2)}_{i_2,r}$ or $\mathcal{T}_{i_0,i_1,i_2} = \sum_{r=0}^{R-1} \mathbf{M}^{(0)}_{r,i_0}\ \mathbf{M}^{(1)}_{r,i_1}\ \mathbf{M}^{(2)}_{r,i_2}$, or even $\mathcal{T}_{i_0,i_1,i_2} = \sum_{r=0}^{R-1} \mathbf{M}^{(0)}_{i_0,r}\ \mathbf{M}^{(1)}_{r,i_1}\ \mathbf{M}^{(2)}_{i_2,r}$. It is easy to observe these decompositions are equivalent to each other if factor matrices are properly transposed.
- *A tensor may be partially factorized over a subset of modes.* For example, we can define a *partial Tucker decomposition* which factors only the last two modes of a 4-order tensor $\mathcal{T} \in \mathbb{R}^{I_0 \times I_1 \times I_2 \times I_3}$ into a core tensor $\mathcal{C} \in \mathbb{R}^{I_0 \times I_1 \times R_2 \times R_3}$ and two factor matrices $\mathbf{M}^{(2)} \in \mathbb{R}^{R_2 \times I_2}$, $\mathbf{M}^{(3)} \in \mathbb{R}^{R_3 \times I_3}$ such that $\mathcal{T}_{i_0,i_1,i_2,i_3} = \sum_{r_2=0}^{R_2-1} \sum_{r_3=0}^{R_3-1} \mathcal{C}_{i_0,i_1,r_2,r_3}\ \mathbf{M}^{(2)}_{r_2,i_2}\ \mathbf{M}^{(3)}_{r_3,i_3}$ or alternatively $\mathcal{T} = \mathcal{C}\left(\times_2 \mathbf{M}^{(2)} \times_3 \mathbf{M}^{(3)}\right)$ if written in our tensor notations.
- *Multiple modes can be grouped into supermode and decomposed like a single mode.* For example, given a 6-order tensor $\mathcal{T} \in \mathbb{R}^{I_0 \times I_1 \times I_2 \times J_0 \times J_1 \times J_2}$ can be factorized into three factors $\mathcal{T}^{(0)} \in \mathbb{R}^{I_0 \times J_0 \times R_0}$, $\mathcal{T}^{(1)} \in \mathbb{R}^{R_0 \times I_1 \times J_1 \times R_1}$ and $\mathcal{T}^{(2)} \in \mathbb{R}^{R_1 \times I_2 \times J_2}$ such that $\mathcal{T}_{i_0,j_0,i_1,j_1,i_2,j_2} = \sum_{r_2=0}^{R_2-1} \sum_{r_3=0}^{R_3-1} \mathcal{T}^{(0)}_{i_0,j_0,r_0}\ \mathcal{T}^{(1)}_{r_0,i_1,j_1,r_1}\ \mathcal{T}^{(2)}_{r_1,i_2,j_2}$, or more succinctly as $\mathcal{T} = \mathcal{T}^{(0)} \times_0^{-1} \mathcal{T}^{(1)} \times_0^{-1} \mathcal{T}^{(2)}$. where I_0 and J_0 are grouped into a supermode (I_0, J_0) and similarly for (I_1, J_1) and (I_2, J_2).

Decomp.	Notations	O(# of params.)	O(# of backprop ops.)
original	$\mathcal{T}$	I	0
CP	$\mathbf{1} \times_0^0 (\mathbf{M}^{(0)} \otimes_0^0 \cdots \otimes_0^0 \mathbf{M}^{(m-1)})$	$mI^{\frac{1}{m}}R$	mIR
TK	$\mathcal{C}(\times_0 \mathbf{M}^{(0)} \cdots \times_{m-1} \mathbf{M}^{(m-1)})$	$R^m + mI^{\frac{1}{m}}R$	m^2IR
TT	$\mathcal{T}^{(0)} \times_0^{-1} \cdots \times_0^{-1} \mathcal{T}^{(m-1)}$	$mI^{\frac{1}{m}}R^2$	mIR^2

Table 9: Summary of tensor decompositions. In this table, we summarize three types of tensor decompositions in tensor notations, and list their numbers of parameters and time complexities to backpropagate the gradient of a tensor $\mathcal{T} \in \mathbb{R}^{I_0 \times I_1 \times \cdots I_{m-1}}$ to its m factors (and an additional core tensor $\mathcal{C}$ for Tucker decomposition). For simplicity, we assume all dimensions I_l's of $\mathcal{T}$ are equal, and denote the size of $\mathcal{T}$ as the product of all dimensions $I = \prod_{l=0}^{m-1} I_l$. Furthermore, we assume all ranks R_l's (in Tucker and Tensor-train decompositions) share the same number R.

G PLAIN TENSOR DECOMPOSITIONS ON CONVOLUTIONAL LAYER

In this section, we will show how tensor decomposition is able to compress (and accelerate) the standard convolutional layer in neural networks. In order to achieve this, we first represent the operation of a standard convolutional layer in tensor notations. By factorizing the tensor of parameters (a.k.a. *kernel*) into multiple smaller factors, compression is achieved immediately.

As we discussed in Appendix F, learning the factors through the gradient of the original tensor of parameters is highly inefficient. In this section, we provide an alternative strategy that interacts the input with the factors individually, in which explicit reference to the original kernel is avoided. Therefore, our strategy also reduces the computational complexity along with compression.

For simplicity, we assume FFT is not used in computing convolutions, although we show in Appendix E that FFT can possibly speed up the backward pass Mathieu et al. (2013).

Standard convolutional layer In modern convolutional neural network (CNN), a standard convolutional layer is parameterized by a 4-order kernel $\mathcal{K} \in \mathbb{R}^{H \times W \times S \times T}$, where H and W are height and width of the filters (which are typically equal), S and T are the number of input and output channels respectively. A convolutional layer maps a 3-order tensor $\mathcal{U} \in \mathbb{R}^{X \times Y \times S}$ to another 3-order tensor $\mathcal{V} \in \mathbb{R}^{X' \times Y' \times T}$, where X and Y are the height and width for the input feature map, while X' and Y' are the ones for the output feature map, with the following equation:

$$\mathcal{V}_{x,y,t} = \sum_{s=0}^{S-1} \sum_{i,j} \mathcal{K}_{i,j,s,t} \, \mathcal{U}_{i+dx,j+dy,s} \tag{G.1}$$

where d is the stride of the convolution layer and the scopes of summations over i and j are determined by the boundary conditions. Notice that the number of parameters in a standard convolutional layer is $HWST$ and the number of operations needed to evaluate the output $\mathcal{V}$ is $O(HWSTXY)$. With tensor notations in this paper, a standard convolutional layer can be defined abstractly as

$$\mathcal{V} = \mathcal{U} \left(*_0^0 \circ *_1^1 \circ \times_2^2 \right) \mathcal{K} \tag{G.2}$$

which states that the standard convolutional layer in fact performs a compound operation of two tensor convolutions and one tensor contraction simultaneously between the input tensor $\mathcal{U}$ and the kernel of parameters $\mathcal{K}$. Following the standard procedure in Appendix E, we obtain both backpropagation equations in tensor notations as follows:

$$\frac{\partial \mathcal{L}}{\partial \mathcal{U}} = \frac{\partial \mathcal{L}}{\partial \mathcal{V}} \left((*_0^0)^\top \circ (*_1^1)^\top \circ \times_3^2 \right) \mathcal{K}, \; \frac{\partial \mathcal{L}}{\partial \mathcal{K}} = \mathcal{U} \left(\overline{(*_0^0)^\top} \circ \overline{(*_1^1)^\top} \right) \frac{\partial \mathcal{L}}{\partial \mathcal{V}} \tag{G.3}$$

It is not difficult to verify that the numbers of operations to compute these two backpropagation equations are $O(HWSTX'Y')$ and $O(XYSTX'Y')$ respectively.

In the next few paragraphs, we will apply various decompositions in Appendix F as well as *singular value decomposition (SVD)* on the kernel $\mathcal{K}$, and derive the steps to evaluate Equation G.1 that interact the input with the factors individually. Interestingly, these steps are themselves (non-standard) convolutional layers, therefore tensor decomposition on the parameters is equivalent to decoupling a layer in the original model into several sublayers in the compressed network, which can be implemented efficiently using modern deep learning libraries. For simplicity, we assume in the analyses of these decomposition schemes that the output feature maps have approximately the same size as the input ones, i.e. $X \approx X', Y \approx Y'$.

SVD-convolutional layer Many researchers propose to compress a convolutional layer using singular value decomposition, under the name of *dictionary learning* (Jaderberg et al., 2014; Denton et al., 2014; Zhang et al., 2015). These methods differ in their matricization of the tensor of parameters $\mathcal{K}$, i.e. how to group the four modes into two and flatten the kernel $\mathcal{K}$ into a matrix. By simple combinatorics, it is not difficult to show there are seven different types of matricization in total. Here, we only pick to present the one by Jaderberg et al. (2014), which groups filter height

and input channels as a supermode (H, S) and filter width and output channels (W, T) as another.

$$\mathcal{K}_{i,j,s,t} = \sum_{r=0}^{R-1} \mathcal{K}^{(0)}_{i,s,r}\, \mathcal{K}^{(1)}_{j,r,t} \tag{G.4a}$$

$$\mathcal{K} = \mathsf{swapaxes}\left(\mathcal{K}^{(0)} \times_1^2 \mathcal{K}^{(1)}\right) \tag{G.4b}$$

where $\mathcal{K}^{(0)} \in \mathbb{R}^{H\times S\times R}$ and $\mathcal{K}^{(1)} \in \mathbb{R}^{W\times R\times T}$ are the two factor tensors. It is easy to see an SVD-convolutional layer has $(HS + WT)R$ parameters in total (HSR in $\mathcal{K}^{(0)}$ and WTR in $\mathcal{K}^{(1)}$). Now we plug the Equation G.4a into G.1, and break the evaluation of $\mathcal{V}$ into two steps such that only one factor is involved at each step.

$$\mathcal{U}^{(0)}_{x,j+dy,r} = \sum_{s=0}^{S-1} \sum_{i} \mathcal{K}^{(0)}_{i,s,r}\, \mathcal{U}_{i+dx,j+dy,s} \tag{G.5a}$$

$$\mathcal{V}_{x,y,t} = \sum_{r=0}^{R-1} \sum_{j} \mathcal{K}^{(1)}_{j,r,t}\, \mathcal{U}^{(0)}_{x,j+dy,r} \tag{G.5b}$$

where $\mathcal{U}^{(0)} \in \mathbb{R}^{X'\times Y\times R}$ is the intermediate result after the first step. This two-steps procedure requires only requires $O((HSXY + WTX'Y)R)$ operations in the forward pass ($O(HSRXY)$ at the first step and $O(WTRX'Y)$ at the second). This number is smaller than $O(HWSTXY)$ as in the standard convolutional layer, since $(HS + WT)R \leq HWST$ implies $(HSXY + WTX'Y)R \leq (HS + WT)RXY \leq HWSTXY$. Therefore, SVD-convolutional layer also outperforms the standard one in efficiency. (Notice that if $\mathcal{K}$ is reconstructed explicitly, the forward pass requires at least $O(HWSTXY)$ operations.) In tensor notations, these steps can be written concisely as follows:

$$\mathcal{U}^{(0)} = \mathcal{U}\left(*_0^0 \circ \times_1^2\right) \mathcal{K}^{(0)} \tag{G.6a}$$

$$\mathcal{V} = \mathcal{U}^{(0)}\left(*_1^1 \circ \times_1^2\right) \mathcal{K}^{(1)} \tag{G.6b}$$

After decomposition, each operation is still a compound operation of tensor convolution and tensor contraction, and therefore itself a convolutional layer whose filters have size either $H \times 1$ and $1 \times W$. Effectively, SVD-convolutional layer is in fact a concatenation of two convolutional layers without nonlinearity in between. Now we proceed to derive the corresponding backpropagation equations for these two steps following the procedure in Appendix E, which are presented in the following:

$$\frac{\partial \mathcal{L}}{\partial \mathcal{U}^{(0)}} = \frac{\partial \mathcal{L}}{\partial \mathcal{V}}\left((*_0^1)^\top \circ \times_2^2\right)\ \mathcal{K}^{(1)},\ \frac{\partial \mathcal{L}}{\partial \mathcal{K}^{(1)}} = \mathcal{U}^{(0)}\left(\times_0^0 \circ \overline{(\overline{*}_1^1)^\top}\right)\frac{\partial \mathcal{L}}{\partial \mathcal{V}} \tag{G.7a}$$

$$\frac{\partial \mathcal{L}}{\partial \mathcal{U}} = \frac{\partial \mathcal{L}}{\partial \mathcal{U}^{(0)}}\left((*_0^0)^\top \circ \times_2^2\right)\ \mathcal{K}^{(0)},\ \frac{\partial \mathcal{L}}{\partial \mathcal{K}^{(0)}} = \mathcal{U}\left(\overline{(\overline{*}_0^0)^\top} \circ \times_1^1\right)\frac{\partial \mathcal{L}}{\partial \mathcal{U}^{(0)}} \tag{G.7b}$$

It is not hard to show the number of operations required to obtain the derivatives with respect to $\mathcal{U}$ and $\mathcal{U}^{(0)}$ is $O((HSX'Y + WTX'Y')R)$, and the one for the factors is $O((XSX'Y + YTX'Y')R)$.

CP-convolutional layer Both Lebedev et al. (2014); Denton et al. (2014) propose to decompose the kernel $\mathcal{K}$ using CP decomposition, differing at whether the height H and width W of the filters are grouped into a *supermode*. For simplicity, we follow the scheme in Denton et al. (2014):

$$\mathcal{K}_{i,j,s,t} = \sum_{r=0}^{R-1} \mathcal{K}^{(0)}_{s,r}\, \mathcal{K}^{(1)}_{i,j,r}\, \mathcal{K}^{(2)}_{r,t} \tag{G.8a}$$

$$\mathcal{K} = \mathbf{1} \times_2^0 \left(\mathcal{K}^{(1)} \otimes_1^2 \mathcal{K}^{(0)} \otimes_0^2 \mathcal{K}^{(2)}\right) \tag{G.8b}$$

where $\mathcal{K}^{(0)} \in \mathbb{R}^{S\times R}$, $\mathcal{K}^{(1)} \in \mathbb{R}^{H\times W\times R}$ and $\mathcal{K}^{(2)} \in \mathbb{R}^{R\times T}$ are three factor tensors, which contain $(HW + S + T)R$ parameters in total. Again, plugging Equation G.8a into G.1 yields a three-steps

procedure to evaluate $\mathcal{V}$:

$$\mathcal{U}^{(0)}_{i+dx,j+dy,r} = \sum_{s=0}^{S-1} \mathcal{K}^{(0)}_{s,r}\, \mathcal{U}_{i+dx,j+dy,s} \tag{G.9a}$$

$$\mathcal{U}^{(1)}_{x,y,r} = \sum_{i,j} \mathcal{K}^{(1)}_{i,j,r}\, \mathcal{U}^{(0)}_{i+dx,j+dy,r} \tag{G.9b}$$

$$\mathcal{V}_{x,y,t} = \sum_{r=0}^{R-1} \mathcal{K}^{(2)}_{r,t}\, \mathcal{U}^{(1)}_{x,y,r} \tag{G.9c}$$

where $\mathcal{U}^{(0)} \in \mathbb{R}^{X\times Y\times R}$ and $\mathcal{U}^{(1)} \in \mathbb{R}^{X'\times Y'\times R}$ are two intermediate tensors. Written in tensor notations, these equations are represented as:

$$\mathcal{U}^{(0)} = \mathcal{U} \times_2 \mathcal{K}^{(0)} \tag{G.10a}$$

$$\mathcal{U}^{(1)} = \mathcal{U}^{(0)} \left(*^0_0 \circ *^1_1 \circ \otimes^2_2\right) \mathcal{K}^{(1)} \tag{G.10b}$$

$$\mathcal{V} = \mathcal{U}^{(1)} \times_2 \mathcal{K}^{(2)} \tag{G.10c}$$

After CP decomposition, the first and third steps are basic tensor multiplications on the input/intermediate tensor, which are usually named (weirdly) as 1×1 *convolutional layers* despite that no convolution is involved at all, while the second step is a compound operation of two tensor convolutions and one partial outer product, which is known as *depth-wise convolutional layer* (Chollet, 2016). The number of operations for these three steps are $O(SRXY)$, $O(HWRXY)$ and $O(TRX'Y')$ respectively, resulting in a time complexity of $O((SXY + HWXY + TX'Y')R)$ for the forward pass, which is faster than the standard convolutional layer, since $(HW+S+T)R \leq HWST$ implies $(SXY + HWXY + TX'Y')R \leq HWSTXY$. Now we proceed to obtain their backpropagation equations following the procedure in Appendix E:

$$\frac{\partial \mathcal{L}}{\partial \mathcal{U}^{(1)}} = \frac{\partial \mathcal{L}}{\partial \mathcal{V}} \times_2 (\mathcal{K}^{(2)})^\top, \ \frac{\partial \mathcal{L}}{\partial \mathcal{K}^{(2)}} = \mathcal{U}^{(1)} \left(\times^0_0 \circ \times^1_1\right) \frac{\partial \mathcal{L}}{\partial \mathcal{V}} \tag{G.11a}$$

$$\frac{\partial \mathcal{L}}{\partial \mathcal{U}^{(0)}} = \frac{\partial \mathcal{L}}{\partial \mathcal{U}^{(1)}} \left((*^0_0)^\top \circ (*^1_1)^\top \circ \otimes^2_2\right) \mathcal{K}^{(1)}, \frac{\partial \mathcal{L}}{\partial \mathcal{K}^{(1)}} = \frac{\partial \mathcal{L}}{\partial \mathcal{U}^{(1)}} \left(\overline{(*^0_0)^\top} \circ \overline{(*^1_1)^\top} \circ \otimes^2_2\right) \mathcal{U}^{(1)}$$

$$\frac{\partial \mathcal{L}}{\partial \mathcal{U}} = \frac{\partial \mathcal{L}}{\partial \mathcal{U}^{(0)}} \times_2 (\mathcal{K}^{(0)})^\top, \ \frac{\partial \mathcal{L}}{\partial \mathcal{K}^{(0)}} = \mathcal{U} \left(\times^0_0 \circ \times^1_1\right) \frac{\partial \mathcal{L}}{\partial \mathcal{U}^{(0)}} \tag{G.11b}$$

The number of operations in all three steps to calculate the derivatives with respect to input/intermediate tensors can be counted as $O((SXY + HWX'Y' + TX'Y')R)$, while the one for the factors as $O((SXY + XYX'Y' + TX'Y')R)$.

Tucker-convolutional layer The use of Tucker decomposition to compress and accelerate convolutional layers is proposed in Kim et al. (2015). Despite the name of Tucker decomposition, they in fact suggest a *partial Tucker decomposition*, which only factorizes the modes over the numbers of input/output filters and keeps the other two modes for filter height/width untouched.

$$\mathcal{K}_{i,j,s,t} = \sum_{r_s=0}^{R_s-1} \sum_{r_t=0}^{R_t-1} \mathcal{K}^{(0)}_{s,r_s}\, \mathcal{K}^{(1)}_{i,j,r_s,r_t}\, \mathcal{K}^{(2)}_{r_t,t} \tag{G.12a}$$

$$\mathcal{K} = \mathcal{K}^{(1)} \left(\times_2 (\mathcal{K}^{(0)})^\top \times_3 \mathcal{K}^{(2)}\right) \tag{G.12b}$$

where $\mathcal{K}^{(0)} \in \mathbb{R}^{S\times R_s}$, $\mathcal{K}^{(1)} \in \mathbb{R}^{H\times W\times R_s\times R_t}$ and $\mathcal{K}^{(2)} \in \mathbb{R}^{R_t\times T}$ are three factor tensors, with a total of $(SR_s + HWR_sR_t + R_tT)$ parameters. All that follow are identical to the ones for SVD and CP layers. A three-steps forward pass procedure is obtained by plugging Equation G.12a in G.1.

$$\mathcal{U}^{(0)}_{i+dx,j+dy,r_s} = \sum_{s=0}^{S-1} \mathcal{K}^{(0)}_{s,r_s}\, \mathcal{U}_{i+dx,j+dy,s} \tag{G.13a}$$

$$\mathcal{U}^{(1)}_{x,y,r_t} = \sum_{r_s=0}^{R_s-1} \sum_{i,j} \mathcal{K}^{(1)}_{i,j,r_s,r_t}\, \mathcal{U}^{(0)}_{i+dx,j+dy,r_s} \tag{G.13b}$$

$$\mathcal{V}_{x,y,t} = \sum_{r_t=0}^{R_t-1} \mathcal{K}^{(2)}_{r_t,t}\, \mathcal{U}^{(1)}_{x,y,r_t} \tag{G.13c}$$

where $\mathcal{U}^{(0)} \in \mathbb{R}^{X \times Y \times R_s}$ and $\mathcal{U}^{(1)} \in \mathbb{R}^{X' \times Y' \times R_t}$ are two intermediate tensors. These three steps, with number of operations of $O(R_s SXY)$, $O(HWR_s R_t XY)$ and $O(R_t TX'Y')$ respectively, leads to a total time complexity of $O(SR_s XY + HWR_s R_t XY + R_t TX'Y')$ for the forward pass. Like CP and SVD convolutional layers, Tucker-convolutional layer is faster than the standard convolutional layer, since $SR_s + HWR_s R_t + R_t T \leq HWST$ implies $SR_s XY + HWR_s R_t XY + R_t TX'Y' \leq HWSTXY$. These equations, again, can be concisely written in tensor notations:

$$\mathcal{U}^{(0)} = \mathcal{U} \times_2 \mathcal{K}^{(0)} \tag{G.14a}$$

$$\mathcal{U}^{(1)} = \mathcal{U}^{(0)} \left(*_0^0 \circ *_1^1 \circ \times_2^2\right) \mathcal{K}^{(1)} \tag{G.14b}$$

$$\mathcal{V} = \mathcal{U}^{(1)} \times_2 \mathcal{K}^{(2)} \tag{G.14c}$$

where the first and the third steps are two 1×1 convolutional layers, and the second step is itself a standard convolutional layer, which only differs from CP-convolutional layer at the second step. For completeness, we summarize all backpropagation equations in the following:

$$\frac{\partial \mathcal{L}}{\partial \mathcal{U}^{(1)}} = \frac{\partial \mathcal{L}}{\partial \mathcal{V}} \times_2 (\mathcal{K}^{(2)})^\top, \frac{\partial \mathcal{L}}{\partial \mathcal{K}^{(2)}} = \mathcal{U}^{(1)} \left(\times_0^0 \circ \times_1^1\right) \frac{\partial \mathcal{L}}{\partial \mathcal{V}} \tag{G.15a}$$

$$\frac{\partial \mathcal{L}}{\partial \mathcal{U}^{(0)}} = \frac{\partial \mathcal{L}}{\partial \mathcal{U}^{(1)}} \left((*_0^0)^\top \circ (*_1^1)^\top \circ \times_3^2\right) \mathcal{K}^{(1)}, \frac{\partial \mathcal{L}}{\partial \mathcal{K}^{(1)}} = \frac{\partial \mathcal{L}}{\partial \mathcal{U}^{(1)}} \left(\overline{(*_0^0)^\top} \circ \overline{(*_1^1)^\top}\right) \mathcal{U}^{(1)} \tag{G.15b}$$

$$\frac{\partial \mathcal{L}}{\partial \mathcal{U}} = \frac{\partial \mathcal{L}}{\partial \mathcal{U}^{(0)}} \times_2 (\mathcal{K}^{(0)})^\top, \frac{\partial \mathcal{L}}{\partial \mathcal{K}^{(0)}} = \mathcal{U} \left(\times_0^0 \circ \times_1^1\right) \frac{\partial \mathcal{L}}{\partial \mathcal{U}^{(0)}} \tag{G.15c}$$

Referring to the CP-convolutional layer, the time complexity for the backward pass is obtained with slight modification: the number of operations for the input/intermediate tensors is $O(SR_s XY + HWR_s R_t X'Y' + R_t TX'Y')$, and the one for factors is $O(SR_s XY + XYHWX'Y' + R_t TX'Y')$.

Tensor-train-convolutional layer Lastly, we propose to apply Tensor-train decomposition to compress a convolutional layer. However, naive Tensor-train decomposition on the kernel $\mathcal{K}$ may give inferior results (Garipov et al., 2016), and careful reordering of the modes is necessary. In this paper, we propose to reorder the modes as (input channels S, filter height H, filter width W, output channels T), and decompose the kernel as

$$\mathcal{K}_{i,j,s,t} = \sum_{r_s=0}^{R_s-1} \sum_{r=0}^{R-1} \sum_{r_t=0}^{R_t-1} \mathcal{K}^{(0)}_{s,r_s} \mathcal{K}^{(1)}_{r_s,i,r} \mathcal{K}^{(2)}_{r,j,r_t} \mathcal{K}^{(3)}_{r_t,t} \tag{G.16a}$$

$$\mathcal{K} = \mathsf{swapaxes}\left(\mathcal{K}^{(0)} \times_0^{-1} \mathcal{K}^{(1)} \times_0^{-1} \mathcal{K}^{(2)} \times_0^{-1} \mathcal{K}^{(3)}\right) \tag{G.16b}$$

where $\mathcal{K}^{(0)} \in \mathbb{R}^{S \times R_s}$, $\mathcal{K}^{(1)} \in \mathbb{R}^{R_s \times H \times R}$, $\mathcal{K}^{(2)} \in \mathbb{R}^{R \times W \times R_t}$ and $\mathcal{K}^{(3)} \in \mathbb{R}^{R_t \times T}$ are factors, which require $(SR_s + HR_s R + WRR_t + R_t T)$. Once we plug the decomposition scheme in Equation G.16a into G.1, the evaluation of $\mathcal{V}$ is decoupled into four steps , with number of operations as $O(R_s SXY)$, $O(HR_s RXY)$, $O(WRR_t X'Y)$ and $O(R_t TX'Y')$ respectively, resulting in a total time complexity as $O(R_s SXY + HR_s RXY + WRR_T X'Y + R_t TX'Y')$ in the forward pass.

$$\mathcal{U}^{(0)}_{i+dx,j+dy,r_s} = \sum_{s=0}^{S-1} \mathcal{K}^{(0)}_{s,r_s} \mathcal{U}_{i+dx,j+dy,s} \tag{G.17a}$$

$$\mathcal{U}^{(1)}_{x,j+dy,r} = \sum_{r_s=0}^{R_s-1} \sum_{i} \mathcal{K}^{(1)}_{r_s,i,r} \mathcal{U}^{(0)}_{i+dx,j+dy,r_s} \tag{G.17b}$$

$$\mathcal{U}^{(2)}_{x,y,r_t} = \sum_{r=0}^{R-1} \sum_{j} \mathcal{K}^{(2)}_{r,j,r_t} \mathcal{U}^{(1)}_{x,j+dy,r} \tag{G.17c}$$

$$\mathcal{V}_{x,y,t} = \sum_{r_t=0}^{R_t-1} \mathcal{K}^{(3)}_{r_t,t} \mathcal{U}^{(2)}_{x,y,r_t} \tag{G.17d}$$

where $\mathcal{U}^{(0)} \in \mathbb{R}^{X \times Y \times R_s}$, $\mathcal{U}^{(1)} \in \mathbb{R}^{X' \times Y \times R}$ and $\mathcal{U}^{(2)} \in \mathbb{R}^{X' \times Y' \times R_t}$ are three intermediate tensors. In tensor notations, these equations can be rewritten as as follows:

$$\mathcal{U}^{(0)} = \mathcal{U} \times_2 \mathcal{K}^{(0)} \tag{G.18a}$$
$$\mathcal{U}^{(1)} = \mathcal{U}^{(0)} \left(*_1^0 \circ \times_0^2\right) \mathcal{K}^{(1)} \tag{G.18b}$$
$$\mathcal{U}^{(2)} = \mathcal{U}^{(1)} \left(*_1^1 \circ \times_0^2\right) \mathcal{K}^{(2)} \tag{G.18c}$$
$$\mathcal{V} = \mathcal{U}^{(2)} \times_2 \mathcal{K}^{(3)} \tag{G.18d}$$

Tensor-train-convolutional layer is concatenation of four sub-layers, where the first and the last ones are 1×1 convolutional layers, while the other two in between are convolutional layers with rectangular kernels. In fact, Tensor-train-convolutional layer can either be interpreted as (1) a Tucker-convolutional layer where the second sublayer is further compressed by a SVD, or (2) a SVD-convolutional layer where both factors are further decomposed again by SVD. Referring to the previous results, the corresponding backpropagation equations are easily derived as

$$\frac{\partial \mathcal{L}}{\partial \mathcal{U}^{(2)}} = \frac{\partial \mathcal{L}}{\partial \mathcal{V}} \times_2 (\mathcal{K}^{(3)})^\top, \frac{\partial \mathcal{L}}{\partial \mathcal{K}^{(3)}} = \mathcal{U}^{(2)} \left(\times_0^0 \circ \times_1^1\right) \frac{\partial \mathcal{L}}{\partial \mathcal{V}} \tag{G.19a}$$
$$\frac{\partial \mathcal{L}}{\partial \mathcal{U}^{(1)}} = \frac{\partial \mathcal{L}}{\partial \mathcal{U}^{(2)}} \left((*_0^1)^\top \circ \times_2^2\right) \mathcal{K}^{(2)}, \frac{\partial \mathcal{L}}{\partial \mathcal{K}^{(2)}} = \mathcal{U}^{(1)} \left(\times_0^0 \circ \overline{(*_1^1)^\top}\right) \frac{\partial \mathcal{L}}{\partial \mathcal{U}^{(2)}} \tag{G.19b}$$
$$\frac{\partial \mathcal{L}}{\partial \mathcal{U}^{(0)}} = \frac{\partial \mathcal{L}}{\partial \mathcal{U}^{(1)}} \left((*_0^0)^\top \circ \times_2^2\right) \mathcal{K}^{(1)}, \frac{\partial \mathcal{L}}{\partial \mathcal{K}^{(2)}} = \mathcal{U}^{(0)} \left(\overline{(*_0^0)^\top} \circ \times_1^1\right) \frac{\partial \mathcal{L}}{\partial \mathcal{U}^{(1)}} \tag{G.19c}$$
$$\frac{\partial \mathcal{L}}{\partial \mathcal{U}} = \frac{\partial \mathcal{L}}{\partial \mathcal{U}^{(0)}} \times_2 (\mathcal{K}^{(0)})^\top, \frac{\partial \mathcal{L}}{\partial \mathcal{K}^{(0)}} = \mathcal{U} \left(\times_0^0 \circ \times_1^1\right) \frac{\partial \mathcal{L}}{\partial \mathcal{U}^{(0)}} \tag{G.19d}$$

Similar to all previous layers, the time complexities for input/intermediate tensors and factors can be calculated as $O(SR_sXY + HR_sRX'Y + WR_tTX'Y' + R_tTX'Y')$ and $O(SR_sXY + XR_sRX'Y + YR_tRX'Y' + R_tTX'Y')$ respectively.

Decomp.	O(# of params.) O(# of forward ops.)	O(# of backward ops. for inputs) O(# of backward ops. for params.)
original	$HWST$ $HWSTXY$	$HWSTX'Y'$ $XYSTX'Y'$
SVD	$(HS + WT)R$ $HSXY + WTX'Y)R$	$(HSX'Y + WTX'Y')R$ $(XSX'Y + YTX'Y')R$
CP	$(HW + S + T)R$ $(SXY + HWXY + TX'Y')R$	$(SXY + HWX'Y' + TX'Y')R$ $(SXY + XYX'Y' + TX'Y')R$
TK	$(HWR_sR_t +$ $SR_s + R_tT)$ $(HWR_sR_tXY +$ $SR_sXY + R_tTX'Y')$	$(HWR_sR_tX'Y' +$ $SR_sXY + R_tTX'Y')$ $(XYR_sR_tX'Y' +$ $SR_sXY + R_tTX'Y')$
TT	$(SR_s + HR_sR +$ $WR_tR + R_tT)$ $(SR_sXY + HR_sRXY +$ $WR_tRX'Y + R_tTX'Y')$	$(SR_sXY + HR_sRX'Y +$ $WR_tTX'Y' + R_tTX'Y')$ $(SR_sXY + XR_sRXY +$ $YR_tRX'Y' + R_tTX'Y')$

Table 10: Summary of plain tensor decomposition on convolutional layer. We list the number of parameters and the number of operations required by forward/backward passes for various plain tensor decomposition on convolutional layer. For reference, a standard convolutional layer maps a set of S feature maps with height X and width Y, to another set of T feature maps with height X' and width Y'. All filters in the convolutional layer share the same height H and width W.

H RESHAPED TENSOR DECOMPOSITIONS ON DENSE LAYER

The operation of dense layer (a.k.a. fully connected layer) in neural network can be simply characterized by a matrix-vector multiplication, which maps a vector $\mathbf{u} \in \mathbb{R}^S$ to another vector $\mathbf{v} \in \mathbb{R}^T$,

where S and T are the number of units for the input and output respectively.

$$v_t = \sum_{s=0}^{S-1} \mathbf{K}_{s,t} u_s \tag{H.1a}$$

$$\mathbf{v} = \mathbf{K}\,\mathbf{u} \tag{H.1b}$$

It is easy to see that a dense layer is parameterized by a matrix $\mathbf{K}$ with ST parameters, and evaluating the output $\mathbf{v}$ requires $O(ST)$ operations. With a matrix at hand, the simplest compression is via *singular value decomposition (SVD)*, which decomposes $\mathbf{K}$ into multiplication of two matrices $\mathbf{K} = \mathbf{P}\,\mathbf{Q}$, where $\mathbf{P} \in \mathbb{R}^{S\times R}$, $\mathbf{Q} \in \mathbb{R}^{R\times T}$ with $R \leq \min(S,T)$. With SVD decomposition, the number of parameters is reduced from ST to $((S+T)R)$ and time complexity from $O(ST)$ to $O((S+T)R)$.

Inspired by the intuition that *invariant structures* can be exploited by tensor decompositions in Section 3, we *tensorize* the matrix $\mathbf{K}$ into a tensor $\mathcal{K} \in \mathbb{R}^{S_0\times\cdots\times S_{m-1}\times T_0\times\cdots\times T_{m-1}}$ such that $S = \prod_{l=0}^{m-1} S_i$, $T = \prod_{l=0}^{m-1} T_l$ and $\mathsf{vec}(\mathcal{K}) = \mathsf{vec}(\mathbf{K})$. Correspondingly, we reshape the input/output $\mathbf{u}$, $\mathbf{v}$ into $\mathcal{U} \in \mathbb{R}^{S_0\times\cdots\times S_{m-1}}$, $\mathcal{V} \in \mathbb{R}^{T_0\times\cdots\times T_{m-1}}$ such that $\mathsf{vec}(\mathcal{U}) = \mathbf{u}$, $\mathsf{vec}(\mathcal{V}) = \mathbf{v}$ and present an (uncompressed) tensorized dense layer as follows:

$$\mathcal{V}_{t_0,\cdots,t_{m-1}} = \sum_{s_0=0}^{S_0-1} \cdots \sum_{s_{m-1}=0}^{S_{m-1}-1} \mathcal{K}_{s_0,\cdots,s_{m-1},t_0,\cdots,t_{m-1}}\, \mathcal{U}_{s_0,\cdots,s_{m-1}} \tag{H.2a}$$

$$\mathcal{V} = \mathcal{U}\left(\times_0^0 \circ \cdots \circ \times_{m-1}^{m-1}\right) \mathcal{K} \tag{H.2b}$$

Therefore, a tensorized dense layer, parameterized by a $2m$-order tensor $\mathcal{K}$, maps an m-order tensor $\mathcal{U}$ to another m-order tensor $\mathcal{V}$. It is straightforward to observe that the tensorized dense layer is mathematically equivalent is to the dense layer in Equation H.1a. Correspondingly, its backpropagation equations can be obtained by simply reshaping the ones for standard dense layer:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{u}} = \mathbf{K}^\top \frac{\partial \mathcal{L}}{\partial \mathbf{v}},\ \frac{\partial \mathcal{L}}{\partial \mathbf{K}} = \mathbf{u}\left(\frac{\partial \mathcal{L}}{\partial \mathbf{v}}\right)^\top = \mathbf{u} \otimes \frac{\partial \mathcal{L}}{\partial \mathbf{v}} \tag{H.3}$$

$$\frac{\partial \mathcal{L}}{\partial \mathcal{U}} = \mathcal{K}\left(\times_0^m \cdots \times_{m-1}^{2m-1}\right) \frac{\partial \mathcal{L}}{\partial \mathcal{V}},\ \frac{\partial \mathcal{L}}{\partial \mathcal{K}} = \mathcal{U} \otimes \frac{\partial \mathcal{L}}{\partial \mathcal{V}} \tag{H.4}$$

In the section, we will compress the tensorized dense layer by decomposing the kernel $\mathcal{K}$ into multiple smaller factors. As we will see, the schemes of tensor decompositions used in this section are not as straightforward as in Appendix F and G, and our principles in their designs are analyzed at the end of this section. Again, learning the the factors through the gradient of the original tensor is extremely costly, therefore a multi-steps procedure to compute the output by interacting the input with the factors individually is desirable. For simplicity of analyses, we will assume for the rest of the paper that S and T are factored evenly, that is $S_l \approx S^{\frac{1}{m}}, T_l \approx T^{\frac{1}{m}}, \forall l \in [m]$ and all ranks are equal to a single number R.

r-CP-dense layer Obviously, the simplest way to factorize $\mathcal{K}$ is to perform naive CP decomposition over all $2m$ modes without grouping any supermode. However, such naive decomposition leads to significant loss of information, as we discuss at the end of this section. In this paper, we instead propose to factor the kernel $\mathcal{K}$ by grouping (S_l, T_l)'s as supermodes. Concretely, the tensor of parameters $\mathcal{K}$ is decomposed as:

$$\mathcal{K}_{s_0,\cdots,s_{m-1},t_0,\cdots,t_{m-1}} = \sum_{r=0}^{R-1} \mathcal{K}^{(0)}_{r,s_0,t_0} \cdots \mathcal{K}^{(m-1)}_{r,s_{m-1},t_{m-1}} \tag{H.5a}$$

$$\mathcal{K} = \mathsf{swapaxes}\left(\mathbf{1} \times_0^0 (\mathcal{K}^{(0)} \otimes_0^0 \cdots \otimes_0^0 \mathcal{K}^{(m-1)})\right) \tag{H.5b}$$

where $\mathcal{K}^{(l)} \in \mathbb{R}^{R\times S_l\times T_l}, \forall l \in [m]$ are m factors and R is the canonical rank that controls the tradeoff between the number of parameters and the fidelity of representation. Therefore, the total number of parameters of a r-CP-dense layer is approximately $\sum_{l=0}^{m-1} S_l T_l R \approx m(ST)^{\frac{1}{m}} R$, which is significantly smaller than ST given that R is reasonably small. The next step to derive the sequential procedure mirrors the ones in all schemes in Appendix G, by plugging the Equation H.5a into H.2a,

we arrive at a multi-steps procedure for the forward pass.

$$\mathcal{U}^{(0)}_{r,s_0,\cdots,s_{m-1}} = \mathcal{U}_{s_0,\cdots,s_{m-1}} \tag{H.6a}$$

$$\mathcal{U}^{(l+1)}_{r,s_{l+1},\cdots,s_{m-1},t_0,\cdots,t_l} = \sum_{s_l=0}^{S_l-1} \mathcal{K}^{(l)}_{r,s_l,t_l}\, \mathcal{U}^{(l)}_{r,s_l,\cdots,s_{m-1},t_0,\cdots,t_{l-1}} \tag{H.6b}$$

$$\mathcal{V}_{t_0,t_1,\cdots,t_{m-1}} = \sum_{r=0}^{R-1} \mathcal{U}^{(m)}_{r,t_0,\cdots,t_{m-1}} \tag{H.6c}$$

where the m intermediate results $\mathcal{U}^{(l)} \in \mathbb{R}^{R\times S_l\times\cdots S_{m-1}\times T_0\times\cdots\times T_{l-1}}, \forall l \in [m]$ are $(m+1)$-order tensors, and the $(l+1)^{\text{th}}$ step above (i.e. the equation interacting $\mathcal{U}^{(l)}$ with $\mathcal{K}^{(l)}$) requires $O((\prod_{k=0}^{l} S_k)(\prod_{k=l}^{m-1} T_k)R)$ operations. The number can be bounded by $O(\max(S,T)^{1+\frac{1}{m}}R)$, and equality is achieved when $S_l = T_l = S^{\frac{1}{m}} = T^{\frac{1}{m}}$. Since the first and last steps require $O(SR)$ and $O(TR)$ that are negligible compared to the other m steps, therefore the total number of operations is bounded by $O(m\max(S,T)^{1+\frac{1}{m}}R)$. These steps in tensor notations are presented as follows:

$$\mathcal{U}^{(0)} = \mathbf{1} \otimes \mathcal{U} \tag{H.7a}$$

$$\mathcal{U}^{(l+1)} = \mathcal{U}^{(l)} \left(\otimes_0^0 \circ \times_1^1\right) \mathcal{K}^{(l)} \tag{H.7b}$$

$$\mathcal{V} = \mathbf{1} \times_0^0 \mathcal{U}^{(m)} \tag{H.7c}$$

Following the procedure in Appendix E, their backpropagation equations are obtained as:

$$\frac{\partial \mathcal{L}}{\partial \mathcal{U}^{(l)}} = \mathcal{K}^{(l)} \left(\otimes_0^0 \circ \times_{-1}^2\right) \frac{\partial \mathcal{L}}{\partial \mathcal{U}^{(l+1)}} \tag{H.8a}$$

$$\frac{\partial \mathcal{L}}{\partial \mathcal{K}^{(l)}} = \mathcal{U}^{(l)} \left(\otimes_0^0 \circ \times_1^2 \circ \cdots \circ \times_{m-1}^m\right) \frac{\partial \mathcal{L}}{\partial \mathcal{U}^{(l+1)}} \tag{H.8b}$$

$$\frac{\partial \mathcal{L}}{\partial \mathcal{U}^{(m)}} = \mathbf{1} \otimes \frac{\partial \mathcal{L}}{\partial \mathcal{V}},\ \frac{\partial \mathcal{L}}{\partial \mathcal{U}} = \mathbf{1} \times_0^0 \frac{\partial \mathcal{L}}{\partial \mathcal{U}^{(0)}}$$

As we discussed in Appendix E, if a compound operation does not contain convolution, the time complexity of backpropagation is identical to the forward pass. Therefore, we claim the number of operations required for backward pass is also bounded by $O(m\max(S,T)^{1+\frac{1}{m}}R)$.

r-Tucker-dense layer The application of TK decomposition is rather straightforward, which factors the tensor of parameters $\mathcal{K}$ exactly the same as in Appendix F.

$$\mathcal{K}_{s_0,\cdots,s_{m-1},t_0,\cdots,t_{m-1}} = \sum_{r_0^s=0}^{R_0^s-1} \cdots \sum_{r_{m-1}^s=0}^{R_{m-1}^s-1} \sum_{r_0^t=0}^{R_0^t-1} \cdots \sum_{r_{m-1}^t=0}^{R_{m-1}^t-1}$$

$$\mathbf{P}^{(0)}_{s_0,r_0^s} \cdots \mathbf{P}^{(m-1)}_{s_{m-1},r_{m-1}^s} \mathcal{C}_{r_0^s,\cdots,r_{m-1}^s,r_0^t,\cdots,r_{m-1}^t} \mathbf{Q}^{(0)}_{r_0^t,t_0} \cdots \mathbf{Q}^{(m-1)}_{r_{m-1}^t,t_{m-1}} \tag{H.9a}$$

$$\mathcal{K} = \mathcal{C}\left(\times_0 (\mathbf{P}^{(0)})^\top \cdots \times_{m-1} (\mathbf{P}^{(m-1)})^\top \times_m \mathbf{Q}^{(0)} \cdots \times_{2m-1} \mathbf{Q}^{(m-1)}\right) \tag{H.9b}$$

where $\mathbf{P}^{(l)} \in \mathbb{R}^{S_l\times R_l^s}, \forall l \in [m]$ are named as input factors, $\mathcal{C} \in \mathbb{R}^{R_0^s\times\cdots\times R_{m-1}^s\times R_0^t\times\cdots\times R_{m-1}^t}$ as core factor, and lastly $\mathbf{Q}^{(l)} \in \mathbb{R}^{R_l^t\times T_l}, \forall l \in [m]$ as output factors. Similar to Tucker-convolutional layer, the procedure to evaluate the result $\mathcal{V}$ can be broken into three steps, by plugging Equation H.9a in H.2a.

$$\mathcal{U}^{(0)}_{r_0^s,\cdots,r_{m-1}^s} = \sum_{s_0=0}^{S_0-1} \cdots \sum_{s_{m-1}=0}^{S_{m-1}-1} \mathbf{P}^{(0)}_{s_0,r_0^s} \cdots \mathbf{P}^{(m-1)}_{s_{m-1},r_{m-1}^s} \mathcal{U}_{s_0,\cdots,s_{m-1}} \tag{H.10a}$$

$$\mathcal{U}^{(1)}_{r_0^t,\cdots,r_{m-1}^t} = \sum_{r_0^s=0}^{R_0^s-1} \cdots \sum_{r_{m-1}^s=0}^{R_{m-1}^s-1} \mathcal{C}_{r_0^s,\cdots,r_{m-1}^s,r_0^t,\cdots r_{m-1}^t} \mathcal{U}^{(0)}_{r_0^s,\cdots,r_{m-1}^s} \tag{H.10b}$$

$$\mathcal{V}_{t_0,\cdots,t_{m-1}} = \sum_{r_0^t=0}^{R_0^t-1} \cdots \sum_{r_{m-1}^t=0}^{R_{m-1}^t-1} \mathbf{Q}^{(0)}_{r_0^t,t_0} \cdots \mathbf{Q}^{(m-1)}_{r_{m-1}^t,t_{m-1}} \mathcal{U}^{(1)}_{r_0^t,\cdots,r_{m-1}^t} \tag{H.10c}$$

where the first and last steps are compound operations between a tensor and a set of multiple tensors, while the middle step is a multi-operations between two tensors. Under the assumptions that $S_l \approx S^{\frac{1}{m}}, T_l \approx T^{\frac{1}{m}}, R_l^s = R_l^t = R, \forall l \in [m]$, the order to contract the factors in the first and last steps makes no difference, therefore we assume the order follows the indices without loss of generality. With this strategy, the contraction with the l^{th} factor takes $O((\prod_{k=0}^{l} R_k)(\prod_{k=l}^{m-1} S_k))$ operations, which is roughly $O(R^{l+1} S^{\frac{m-l}{m}})$ and can be further bounded by $O(SR)$ since $R \leq S^{\frac{1}{m}}$ by the definition of Tucker decomposition: therefore, the time complexity to contract all m factors is at most $O(mSR)$. Likewise, the number of operations for the last step can also be bounded by $O(mTR)$. Lastly, it is easy to see the middle step needs $O(R^{2m})$ operations, therefore leads to a total time complexity of $O(m(S+T)R + R^{2m})$ for the three-step procedure. In tensor notations, these equations can be concisely written as

$$\mathcal{U}^{(0)} = \mathcal{U} \left(\times_0 \mathbf{P}^{(0)} \cdots \times_{m-1} \mathbf{P}^{(m-1)} \right) \tag{H.11a}$$

$$\mathcal{U}^{(1)} = \mathcal{U}^{(0)} \left(\times_0^0 \circ \cdots \circ \times_{m-1}^{m-1} \right) \mathcal{C} \tag{H.11b}$$

$$\mathcal{V} = \mathcal{U}^{(1)} \left(\times_0 \mathbf{Q}^{(0)} \cdots \times_{m-1} \mathbf{Q}^{(m-1)} \right) \tag{H.11c}$$

Though compound in nature, the procedure to derive their backpropagation rules are pretty straightforward: notice the equations for the first and last steps have the exactly the same form as standard Tucker decomposition in Appendix F. Therefore, we can simply modify the variable names therein to obtain the backpropagation equations for these two steps.

$$\frac{\partial \mathcal{L}}{\partial \mathcal{U}^{(1)}} = \frac{\partial \mathcal{L}}{\partial \mathcal{V}} \left(\times_0 (\mathbf{Q}^{(0)})^\top \cdots \times_{m-1} (\mathbf{Q}^{(m-1)})^\top \right) \tag{H.12a}$$

$$\left(\frac{\partial \mathcal{L}}{\partial \mathbf{Q}^{(l)}} \right)^\top = \frac{\partial \mathcal{L}}{\partial \mathcal{V}} \left(\times_0^0 \circ \cdots \circ \times_{l-1}^{l-1} \circ \times_{l+1}^{l+1} \circ \cdots \circ \times_{m-1}^{m-1} \right)$$
$$\left(\mathcal{U}^{(1)} \left(\times_0 \mathbf{Q}^{(0)} \cdots \times_{l-1} \mathbf{Q}^{(l-1)} \times_{l+1} \mathbf{Q}^{(l+1)} \cdots \times_{m-1} \mathbf{Q}^{(m-1)} \right) \right) \tag{H.12b}$$

$$\frac{\partial \mathcal{L}}{\partial \mathcal{U}} = \frac{\partial \mathcal{L}}{\partial \mathcal{U}^{(0)}} \left(\times_0 (\mathbf{P}^{(0)})^\top \cdots \times_{m-1} (\mathbf{P}^{(m-1)})^\top \right) \tag{H.12c}$$

$$\left(\frac{\partial \mathcal{L}}{\partial \mathbf{P}^{(l)}} \right)^\top = \frac{\partial \mathcal{L}}{\partial \mathcal{U}^{(0)}} \left(\times_0^0 \circ \cdots \circ \times_{l-1}^{l-1} \circ \times_{l+1}^{l+1} \circ \cdots \circ \times_{m-1}^{m-1} \right)$$
$$\left(\mathcal{U} \left(\times_0 \mathbf{P}^{(0)} \cdots \times_{l-1} \mathbf{P}^{(l-1)} \times_{l+1} \mathbf{P}^{(l+1)} \cdots \times_{m-1} \mathbf{P}^{(m-1)} \right) \right) \tag{H.12d}$$

The step in the middle is itself a tensorized layer defined in Equation H.2b, therefore its backpropagation rules can be obtained by renaming the variable in Equations H.4.

$$\frac{\partial \mathcal{L}}{\partial \mathcal{U}^{(0)}} = \mathcal{C} \left(\times_0^m \circ \cdots \circ \times_{m-1}^{2m-1} \right) \frac{\partial \mathcal{L}}{\partial \mathcal{U}^{(1)}} \tag{H.13a}$$

$$\frac{\partial \mathcal{L}}{\partial \mathcal{C}} = \mathcal{U}^{(0)} \otimes \frac{\partial \mathcal{L}}{\partial \mathcal{U}^{(1)}} \tag{H.13b}$$

Despite their technical complicity, we can resort to conclusion that the complexities for forward and backward passes are the same for operations without convolution, and claim that the number of operations required for the backpropagation equations above is bounded by $O(m(S+T)R + R^{2m})$.

r-Tensor-train-dense layer The layer presented in this part follows closely the pioneering work in compressing network using tensor decompositions (Novikov et al., 2015), except that we replace the backpropagation algorithm in the original paper (as discussed in Appendix F) with a multi-step procedure similar to all other layers in this paper. With the replacement, the efficiency of the backward is greatly improved compared to original design. Similar to r-CP-dense layer, we will group (S_l, T_l)'s as supermodes and decomposed the kernel $\mathcal{K}$ by Tensor-train decomposition following the order of their indices:

$$\mathcal{K}_{s_0,\cdots,s_{m-1},t_0,\cdots,t_{m-1}} = \sum_{r_0=0}^{R_0-1} \cdots \sum_{r_{m-2}=0}^{R_{m-2}-1} \mathcal{K}^{(0)}_{s_0,t_0,r_0} \mathcal{K}^{(1)}_{r_0,s_1,t_1,r_1} \cdots \mathcal{K}^{(m-1)}_{r_{m-2},s_{m-1},t_{m-1}} \tag{H.14a}$$

$$\mathcal{K} = \mathsf{swapaxes} \left(\mathcal{K}^{(0)} \times_0^{-1} \mathcal{K}^{(1)} \times_0^{-1} \cdots \times_0^{-1} \mathcal{K}^{(m-1)} \right) \tag{H.14b}$$

where the factor tensors are $\mathcal{K}^{(l)} \in \mathbb{R}^{R_{l-1} \times S_l \times T_l \times R_l}, \forall l = 1, \cdots, m-2$, with two corner cases $\mathcal{K}^{(0)} \in \mathbb{R}^{S_0 \times T_0 \times R_0}$ and $\mathcal{K}^{(m-1)} \in \mathbb{R}^{R_{m-2} \times S_{m-1} \times T_{m-1}}$. The number of parameters in its l^{th} factor $\mathcal{K}^{(l)}$ is approximately $(ST)^{\frac{1}{m}} R^2$, therefore the layer has $O(m(ST)^{\frac{1}{m}} R^2)$ parameters in total. For aestheticism, we add singleton mode $R_{-1} = 1$ to $\mathcal{U}, \mathcal{K}^{(0)}$ such that $\mathcal{U} \in \mathbb{R}^{S_0 \times \cdots \times S_{m-1} \times R_{-1}}, \mathcal{K}^{(0)} \in \mathbb{R}^{R_{-1} \times S_0 \times T_0 \times R_0}$, and rename $\mathcal{U}$ as $\mathcal{U}^{(0)}$. Now insert the Equation H.14a into H.2a and expand accordingly, we obtain an $(m+1)$-steps procedure to evaluate the output $\mathcal{V}$:

$$\mathcal{U}^{(l+1)}_{s_{l+1}, \cdots, s_{m-1}, t_0, \cdots, t_l, r_l} = \sum_{r_{l-1}=0}^{R_{l-1}-1} \sum_{s_l=0}^{S_l-1} \mathcal{K}^{(l)}_{r_{l-1}, s_l, t_l, r_l} \mathcal{U}^{(l)}_{s_l, \cdots, s_{m-1}, t_0, \cdots, t_{l-1}, r_{l-1}} \tag{H.15}$$

where the last result $\mathcal{U}^{(m)}$ is equal to the final output $\mathcal{V}$, and the other m tensors $\mathcal{U}^{(l)}$'s are intermediate results. Similar to r-CP-dense layer, the number of operations at the l^{th} step is $O((\prod_{k=0}^{l} S_k)(\prod_{k=l}^{m-1} T_l) R_{l-1} R_l)$ and can be bounded by $O(\max(S,T)^{1+\frac{1}{m}} R^2)$, therefore the time complexity for all m-steps is bounded by $O(m \max(S,T)^{1+\frac{1}{m}} R^2)$. These steps very simple in tensor notations.

$$\mathcal{U}^{(l+1)} = \mathcal{U}^{(l)} \left(\times_1^0 \circ \times_0^{-1} \right) \mathcal{K}^{(l)} \tag{H.16}$$

Observe that the operations in the forward pass are entirely tensor contractions, therefore their backpropagation equations are easily derived following the procedure in Appendix E.

$$\frac{\partial \mathcal{L}}{\partial \mathcal{U}^{(l)}} = \mathcal{K}^{(l)} \left(\times_{-2}^1 \circ \times_{-1}^2 \right) \frac{\partial \mathcal{L}}{\partial \mathcal{U}^{(l+1)}} \tag{H.17a}$$

$$\frac{\partial \mathcal{L}}{\partial \mathcal{K}^{(l)}} = \mathsf{swapaxes} \left(\mathcal{U}^{(l)} \left(\times_0^1 \circ \cdots \circ \times_{m-2}^{m-1} \right) \frac{\partial \mathcal{L}}{\partial \mathcal{U}^{(l+1)}} \right) \tag{H.17b}$$

In the analysis of the backward pass, we can again take advantage of the argument that the forward and backward passes share the same number of operations. Therefore, we claim the time complexity for backpropagation is bounded by $O(m \max(S,T)^{1+\frac{1}{m}} R^2)$.

Relation to tensor contraction layer: In Kossaifi et al. (2017a), the authors propose a novel tensor contraction layer, which takes a tensor of arbitrary order as input and return a tensor of the same order. Formally, a tensor contraction layer, parameterized by a set of m matrices $\{\mathbf{M}^{(l)}\})_{l=0}^{m-1}$, with $\mathbf{M}^{(l)} \in \mathbb{R}^{S_l \times T_l}, \forall l \in [m]$, maps a m-order tensor $\mathcal{U} \in \mathbb{R}^{S_0 \times \cdots \times S_{m-1}}$ to another m-order tensor $\mathcal{V} \in \mathbb{R}^{T_0 \times \cdots T_{m-1}}$ such that

$$\mathcal{V}_{t_0, \cdots, t_{m-1}} = \sum_{s_0=0}^{S_0-1} \cdots \sum_{s_{m-1}=0}^{S_{m-1}-1} \mathcal{U}_{s_0, \cdots, s_{m-1}} \mathbf{M}^{(0)}_{s_0, t_0} \cdots \mathbf{M}^{(m-1)}_{s_{m-1}, t_{m-1}} \tag{H.18a}$$

$$\mathcal{V} = \mathcal{U} \left(\times_0 \mathbf{M}^{(0)} \cdots \times_{m-1} \mathbf{M}^{(m-1)} \right) \tag{H.18b}$$

It is not difficult to observe that the tensor contraction layer is in fact special case of r-CP-dense layer where the kernel is restricted to rank-1, that is $\mathcal{K}_{s_0, \cdots, s_{m-1}, t_0, \cdots, t_{m-1}} = \mathbf{M}^{(0)}_{s_0, t_0} \cdots \mathbf{M}^{(m-1)}_{s_{m-1}, t_{m-1}}$, or equivalently $\mathcal{K} = \mathbf{M}^{(0)} \otimes \cdots \otimes \mathbf{M}^{(m-1)}$ in our tensor notations.

Relation to tensor regression layer: Along with the tensor contraction layer, tensor regression layer is also proposed in Kossaifi et al. (2017b), which takes a tensor of arbitrary order as input and maps it to a scalar. Formally, given an m-order tensor $\mathcal{U} \in \mathbb{R}^{S_0 \times \cdots \times S_{m-1}}$, it is reduced to a scalar v by contracting of all the modes with another tensor of the same size $\mathcal{K} \in \mathbb{R}^{S_0 \times \cdots \times S_{m-1}}$.

$$v = \sum_{s_0=0}^{S_0-1} \cdots \sum_{s_{m-1}=0}^{S_{m-1}-1} \mathcal{U}_{s_0, \cdots, s_{m-1}} \mathcal{K}_{s_0, \cdots, s_{m-1}} \tag{H.19a}$$

$$v = \mathcal{U} \left(\times_0^0 \circ \times_1^1 \circ \cdots \circ \times_{m-1}^{m-1} \right) \mathcal{K} \tag{H.19b}$$

where the tensor $\mathcal{K}$ is stored in Tucker-format as in Equation F.4a. Therefore, the tensor regression layer is effectively parameterized by a set of matrices $\{\mathbf{M}^{(l)}\}_{l=0}^{m-1}, \mathbf{M}^{(l)} \in \mathbb{R}^{S_l \times R_l}, \forall l \in [m]$, with

an additional core tensor $\mathcal{C} \in \mathbb{R}^{R_0 \times \cdots \times R_{m-1}}$. Therefore the definition of tensor regression layer in Equation H.19a can also be rephrased as

$$v = \sum_{s_0=0}^{S_0-1} \cdots \sum_{s_{m-1}=0}^{S_{m-1}-1} \mathcal{U}_{s_0,\cdots,s_{m-1}} \mathbf{M}^{(0)}_{s_0,r_0} \cdots \mathbf{M}^{(m-1)}_{s_{m-1},r_{m-1}} \mathcal{C}_{r_0,\cdots,r_{m-1}} \tag{H.20a}$$

$$= \mathcal{U} \left(\times_0 \mathbf{M}^{(0)} \cdots \times_{m-1} \mathbf{M}^{(m-1)} \right) \left(\times_0^0 \circ \cdots \circ \times_{m-1}^{m-1} \right) \mathcal{C} \tag{H.20b}$$

Now we are able to observe that the tensor regression layer is indeed a special case of r-Tucker-dense layer where the input factors $\mathbf{P}^{(l)} = \mathbf{M}^{(l)}, \forall l \in [m]$, while the output factors $\mathbf{Q}^{(l)}$'s are simply scalar 1's with $T_l = 1, \forall l \in [m]$.

Comments on the designs: As we can observe, the design of r-Tucker-dense is different from the other two layers using CP and Tensor-train decompositions, in which (S_l, T_l)'s are first grouped into supermodes before factorization. Indeed, it is a major drawback in the design of r-Tucker-dense layer: notice that the first intermediate result $\mathcal{U}^{(0)}$ obtained in the forward pass is a tensor of size $R_0^s \times R_1^s \cdots \times R_{m-1}^s$, which becomes very tiny if the kernel $\mathcal{K}$ is aggressively compressed. Therefore, the size of the intermediate tensor poses an *"information bottleneck"*, causing significant loss during the forward pass, which is verified by our experimental results in Section app:experiments. Therefore, the use of r-Tucker-dense layer is not recommended when we expect excessive compression rate. On the other hand, by grouping (S_l, T_l)'s as supermodes in r-CP-dense layer and r-Tensor-train-dense layer, all intermediate tensors $\mathcal{U}^{(l)}$'s have similar size as the input, therefore the bottleneck in r-Tucker-dense layer is completely avoided.

But why do we not group (S_l, T_l)'s together in the design of Tucker-dense layer at the beginning? In theory, we are for sure able to factorize the kernel as

$$\mathcal{K} = \sum_{r_0=1}^{R_0-1} \cdots \sum_{r_{m-1}=0}^{R_{m-1}-1} \mathcal{C}_{r_0,\cdots,r_{m-1}} \mathcal{K}^{(0)}_{r_0,s_0,t_0} \cdots \mathcal{K}^{(m-1)}_{r_{m-1},s_{m-1},t_{m-1}} \tag{H.21}$$

However, the contractions among the input and the factors become problematic: (1) interacting the input with the factors $\mathcal{K}^{(l)}$'s yields an intermediate tensor of size $T_0 \times \cdots \times T_{m-1} \times R_0 \times \cdots \times R_{m-1}$, which is too large to fit into memory; (2) while reconstructing the kernel $\mathcal{K}$ from $\mathcal{K}^{(l)}$'s and subsequently invoking the Equation H.2a will make the time complexity for backward pass intractable as we discussed in Appendix F. Therefore, we have to abandon this attempting design, in order to maintain a reasonable time complexity.

As a compensation for the possible loss, the current design of Tucker-dense layer actually has one benefit over the other two layers: the numbers of operations for the backward pass remains at the same order as the number of parameters, while the number of operations required by r-CP-dense and r-Tensor-train-dense layers are orders higher. As a result, r-Tucker-dense layer is much faster than r-CP-dense and r-Tensor-train-dense layers at the same compression rate. Therefore, r-Tucker-dense layer is more desirable if we value speed over accuracy and the compression rate is not too high.

Decomp.	O(# of params.)	O(# of forward/backprop ops.)
original	ST	ST
SVD	$(S+T)R$	$(S+T)R$
r-CP	$m(ST)^{\frac{1}{m}}R$	$m\max(S,T)^{1+\frac{1}{m}}R$
r-TK	$m(S^{\frac{1}{m}}+T^{\frac{1}{m}})R+R^{2m}$	$m(S+T)R+R^{2m}$
r-TT	$m(ST)^{\frac{1}{m}}R^2$	$m\max(S,T)^{1+\frac{1}{m}}R^2$

Table 11: Summary of reshaped tensor decomposition on dense layer. In this table, we list the numbers of parameters and time complexities of forward/backward passes required by various reshaped tensor decompositions on dense layer. For simplicity, we assume that the number of input units S and outputs units T are factorized evenly, i.e. $S_l = S^{\frac{1}{m}}, T_l = T^{\frac{1}{m}}, \forall l \in [m]$ and all ranks (in r-TK and r-TT) share the same number R, i.e. $R_l = R, \forall l \in [m]$.

I RESHAPED TENSOR DECOMPOSITIONS ON CONVOLUTIONAL LAYER

In Appendix H, we *tensorize* the parameters into higher-order tensor in order to exploit their invariance structures. It is tempting to extend the same idea to convolutional layer such that similar structures can be discovered. In the section, we propose several additional convolutional layers based on the same technique as in Appendix H: the input and output tensors are folded as $\mathcal{U} \in \mathbb{R}^{X \times Y \times S_0 \times \cdots \times S_{m-1}}$ and $\mathcal{V} \in \mathbb{R}^{X' \times Y' \times T_0 \times \cdots \times T_{m-1}}$, while the tensor of parameters are reshaped into $\mathcal{K} \in \mathbb{R}^{H \times W \times S_0 \times \cdots \times S_{m-1} \times T_0 \times \cdots \times T_{m-1}}$. Similar to the tensorized dense layer in Equation H.2a, we define a (uncompressed) tensorized convolutional layer equivalent to Equation G.1:

$$\mathcal{V}_{x,y,t_0,\cdots,t_{m-1}} = \sum_{s_0=0}^{S_0-1} \cdots \sum_{s_{m-1}=0}^{S_{m-1}-1} \sum_{i,j} \mathcal{K}_{i,j,s_0\cdots,s_{m-1},t_0,\cdots,t_{m-1}} \, \mathcal{U}_{i+dx,j+dy,s_0,\cdots,s_{m-1}} \tag{I.1a}$$

$$\mathcal{V} = \mathcal{U} \left(*_0^0 \circ *_1^1 \circ \times_2^2 \circ \cdots \circ \times_{m+1}^{m+1} \right) \mathcal{K} \tag{I.1b}$$

Their corresponding backpropagation equations are then easily obtained by reshaping the ones for standard convolutional layer in Equation G.3.

$$\frac{\partial \mathcal{L}}{\partial \mathcal{U}} = \mathcal{K} \left((*_0^0)^\top \circ (*_1^1)^\top \circ \times_2^{m+2} \cdots \times_{m+1}^{2m+1} \right) \frac{\partial \mathcal{L}}{\partial \mathcal{V}} \tag{I.2a}$$

$$\frac{\partial \mathcal{L}}{\partial \mathcal{K}} = \mathcal{U} \left((*_0^0)^\top \circ (*_1^1)^\top \right) \frac{\partial \mathcal{L}}{\partial \mathcal{V}} \tag{I.2b}$$

What follows are almost replications of Appendix G and H: applying tensor decompositions in Appendix F to the kernel $\mathcal{K}$ and derive the corresponding multi-steps procedures. As we shall expect, all layers in this section mimic their counterparts in Appendix H (in fact they will reduce to counterpart layers when original layer is 1×1 convolutional layer). Therefore in this section, we will borrow results from last section whenever possible and only emphasize their differences.

r-CP-convolutional layer In this part, CP decomposition is used in a similar way as in r-CP-dense layer, which decomposes the kernel $\mathcal{K}$ grouping (S_l, T_l)'s as supermodes, and (H, W) as an additional supermode. Specifically, the tensor $\mathcal{K}$ takes the form as

$$\mathcal{K}_{i,j,s_0,\cdots,s_{m-1},t_0,\cdots,t_{m-1}} = \sum_{r=0}^{R-1} \mathcal{K}^{(0)}_{r,s_0,t_0} \cdots \mathcal{K}^{(m-1)}_{r,s_{m-1},t_{m-1}} \, \mathcal{K}^{(m)}_{r,i,j} \tag{I.3a}$$

$$\mathcal{K} = \mathsf{swapaxes} \left(\mathbf{1} \times_0^0 \left(\mathcal{K}^{(0)} \otimes_0^0 \cdots \otimes_0^0 \mathcal{K}^{(m)} \right) \right) \tag{I.3b}$$

where $\mathcal{K}^{(l)} \in \mathbb{R}^{R \times S_l \times T_l}, \forall l \in [m]$ and $\mathcal{K}^{(m)} \in \mathbb{R}^{R \times H \times W}$ are $(m+1)$ factors. Notice that Equation I.3a only differs from Equation H.5a in r-CP-dense layer by an additional factor $\mathcal{K}^{(m)}$, therefore has HWR more parameters and reaches $O(m(ST)^{\frac{1}{m}}R + HWR)$ in total. Accordingly, the multi-steps procedure to evaluate the output $\mathcal{V}$ now has one extra step at the end, and the $(m+2)$-steps algorithm is presented at the entries level as follows:

$$\mathcal{U}^{(0)}_{r,i+dx,j+dy,s_0,\cdots,s_{m-1}} = \mathcal{U}_{i+dx,j+dy,s_0,\cdots,s_{m-1}} \tag{I.4a}$$

$$\mathcal{U}^{(l+1)}_{r,i+dx,j+dy,s_{l+1},\cdots,s_{m-1},t_0,\cdots,t_l} = \sum_{s_l=0}^{S_l-1} \mathcal{K}^{(l)}_{r,s_l,t_l} \, \mathcal{U}^{(l)}_{r,i+dx,j+dy,s_l,\cdots,s_{m-1},t_0,\cdots,t_{l-1}} \tag{I.4b}$$

$$\mathcal{V}_{x,y,t_0,\cdots,t_{m-1}} = \sum_{r=0}^{R-1} \sum_{i,j} \mathcal{K}^{(m)}_{r,i,j} \, \mathcal{U}^{(m)}_{r,i+dx,j+dy,t_0,\cdots,t_{m-1}} \tag{I.4c}$$

where $\mathcal{U}^{(l)} \in \mathbb{R}^{R \times S_l \times \cdots \times S_{m+1} \times T_0 \times \cdots \times T_{l-1}}, \forall l \in [m]$ are m intermediate tensors. Notice that the order to interact the $(m+1)$ factors is arbitrary, that is the convolutional factor $\mathcal{U}^{(m)}$ can be convoluted over at any step during the forward pass. In this paper, we place the convolutional factor $\mathcal{U}^{(m)}$ to the end simply for implementational convenience: it is not difficult to recognize that the last step is a *3D-convolutional layer* with R input feature volumes and one output feature volume, if we treat the number of feature maps T as depth of the feature volumes. The time complexity in the forward

pass are easily obtained through the results of r-CP-dense layer: compared to r-CP-dense layer, each of the existing m steps will be scaled by a factor of XY, while the additional last step requires $O(HWTRXY)$ operations. Therefore, the total number of operations for r-CP-convolutional layer is $O(m \max(S,T)^{1+\frac{1}{m}} RXY + HWTRXY)$. In tensor notations, these steps are rephrased as:

$$\mathcal{U}^{(0)} = \mathbf{1} \otimes \mathcal{U} \tag{I.5a}$$

$$\mathcal{U}^{(l+1)} = \mathcal{U}^{(l)} \left(\otimes_0^0 \circ \times_1^3\right) \mathcal{K}^{(l)} \tag{I.5b}$$

$$\mathcal{V} = \mathcal{U}^{(m)} \left(\times_0^0 \circ *_1^1 \circ *_2^2\right) \mathcal{K}^{(m)} \tag{I.5c}$$

The backpropagation equations are also very similar to their r-CP-dense layer counterparts. For completeness, we list all of them in the following:

$$\frac{\partial \mathcal{L}}{\partial \mathcal{U}^{(m)}} = \mathcal{K}^{(m)} \left((*_0^1)^\top \circ (*_1^2)^\top\right) \frac{\partial \mathcal{L}}{\partial \mathcal{V}} \tag{I.6a}$$

$$\frac{\partial \mathcal{L}}{\partial \mathcal{K}^{(m)}} = \mathcal{U}^{(m)} \left((*_0^1)^\top \circ (*_1^2)^\top \times_2^3 \circ \cdots \circ \times_{m+1}^{m+2}\right) \frac{\partial \mathcal{L}}{\partial \mathcal{V}} \tag{I.6b}$$

$$\frac{\partial \mathcal{L}}{\partial \mathcal{U}^{(l)}} = \mathsf{swapaxes}\left(\mathcal{K}^{(l)} \left(\otimes_0^0 \circ \times_{-1}^2\right) \frac{\partial \mathcal{L}}{\partial \mathcal{U}^{(l+1)}}\right) \tag{I.6c}$$

$$\frac{\partial \mathcal{L}}{\partial \mathcal{K}^{(l)}} = \mathcal{U}^{(l)} \left(\otimes_0^0 \circ \otimes_1^1 \circ \otimes_2^2 \circ \times_3^4 \circ \cdots \circ \times_{m+2}^{m+3}\right) \frac{\partial \mathcal{L}}{\partial \mathcal{U}^{(l+1)}} \tag{I.6d}$$

$$\frac{\partial \mathcal{L}}{\partial \mathcal{U}} = \mathbf{1} \times_0^0 \frac{\partial \mathcal{L}}{\partial \mathcal{U}^{(0)}} \tag{I.6e}$$

In the analyses of these backpropagation equations, it is again convenient to make connections to their r-CP-dense layer counterparts: the time complexities for the first m steps are scaled by XY, while the last step of 3D-convolutional layer requires $O(HWTRX'Y')$ operations for the derivative with respect to $\mathcal{U}^{(m)}$, and $O(XYTRX'Y')$ for the gradient of $\mathcal{K}^{(m)}$.

r-Tucker-convolutional layer Incorporating the features from both Tucker-convolutional layer in Appendix G and r-Tucker-dense layer in Appendix H, we propose to apply partial Tucker decomposition on the tensorized kernel $\mathcal{K}$ over all modes except the filter height H and width W. Concretely, the tensorized kernel $\mathcal{K}$ is factorized as:

$$\mathcal{K}_{i,j,s_0,\cdots,s_{m-1},t_0,\cdots,t_{m-1}} = \sum_{r_0^s=0}^{R_0^s-1} \cdots \sum_{r_{m-1}^s=0}^{R_{m-1}^s-1} \sum_{r_0^t=0}^{R_0^t-1} \cdots \sum_{r_{m-1}^t=0}^{R_{m-1}^t-1}$$

$$\mathbf{P}^{(0)}_{s_0,r_0^s} \cdots \mathbf{P}^{(m-1)}_{s_{m-1},r_{m-1}^s} \mathcal{C}_{i,j,r_0^s,\cdots,r_{m-1}^s,r_0^t,\cdots,r_{m-1}^t} \mathbf{Q}^{(0)}_{r_0^t,t_0} \cdots \mathbf{Q}^{(m-1)}_{r_{m-1}^t,t_{m-1}} \tag{I.7a}$$

$$\mathcal{K} = \mathcal{C} \left(\times_2 (\mathbf{P}^{(0)})^\top \cdots \times_{m+1} (\mathbf{P}^{(m-1)})^\top \times_{m+2} \mathbf{Q}^{(0)} \cdots \times_{2m+1} \mathbf{Q}^{(m-1)}\right) \tag{I.7b}$$

where $\mathbf{P}^{(l)} \in \mathbb{R}^{S_l \times R_l^s}, \forall l \in [m]$, $\mathcal{C} \in \mathbb{R}^{H \times W \times R_0^s \times \cdots \times R_{m-1}^s \times R_0^t \times \cdots \times R_{m-1}^t}$ and $\mathbf{Q}^{(l)} \in \mathbb{R}^{R_l^t \times T_l}, \forall l \in [m]$ are again named as input factors, core factor and output factors respectively. The reason that (S_l, T_l)'s are not grouped into supermodes follows exactly the same arguments as in Appendix H. Compared to r-Tensor-train-dense layer, the only difference is that the core tensor now has two extra modes for filter height and width, and therefore the number of parameters is magnified by a factor of HW. Similar to Tucker-convolutional and r-Tucker-dense layers, the procedure to evaluate $\mathcal{V}$ can be sequentialized into three steps:

$$\mathcal{U}^{(0)}_{i+dx,j+dy,r_0^s,\cdots,r_{m-1}^s} = \sum_{s_0=0}^{S_0-1} \cdots \sum_{s_{m-1}=0}^{S_{m-1}-1} \mathbf{P}^{(0)}_{s_0,r_0^s} \cdots \mathbf{P}^{(m-1)}_{s_{m-1},r_{m-1}^s} \mathcal{U}_{i+dx,j+dy,s_0,\cdots,s_{m-1}} \tag{I.8a}$$

$$\mathcal{U}^{(1)}_{x,y,r_0^t,\cdots,r_{m-1}^t} = \sum_{r_0^s=0}^{R_0^s-1} \cdots \sum_{r_{m-1}^s=0}^{R_{m-1}^s-1} \sum_{i,j} \mathcal{C}_{i,j,r_0^s,\cdots,r_{m-1}^s,r_0^t,\cdots r_{m-1}^t} \mathcal{U}^{(0)}_{i+dx,j+dy,r_0^s,\cdots,r_{m-1}^s} \tag{I.8b}$$

$$\mathcal{V}_{x,y,t_0,\cdots,t_{m-1}} = \sum_{r_0^t=0}^{R_0^t-1} \cdots \sum_{r_{m-1}^t=0}^{R_{m-1}^t-1} \mathbf{Q}^{(0)}_{r_0^t,t_0} \cdots \mathbf{Q}^{(m-1)}_{r_{m-1}^t,t_{m-1}} \mathcal{U}^{(1)}_{x,y,r_0^t,\cdots,r_{m-1}^t} \tag{I.8c}$$

where $\mathcal{U}^{(0)} \in \mathbb{R}^{X \times Y \times R_0^s \times \cdots \times R_{m-1}^s}$ and $\mathcal{U}^{(1)} \in \mathbb{R}^{X' \times Y' \times R_0^t \times \cdots \times R_{m-1}^t}$ are two intermediate tensors. Referring to r-Tucker-dense layer, the number of operations of the first step is scaled up by XY, the number for the middle step by $HWXY$ and the last step by $X'Y'$. Therefore, the time complexity for the three-steps process is $O(mS^{\frac{1}{m}}XY + HWR^{2m}XY + mT^{\frac{1}{m}}X'Y')$. Subsequently, we can rewrite these steps concisely in tensor notations.

$$\mathcal{U}^{(0)} = \mathcal{U}\left(\times_2 \mathbf{P}^{(0)} \cdots \times_{m+1} \mathbf{P}^{(m-1)}\right) \tag{I.9a}$$

$$\mathcal{U}^{(1)} = \mathcal{U}^{(0)}\left(*_0^0 \circ *_1^1 \times_2^2 \circ \cdots \circ \times_{m-1}^{m-1}\right)\mathcal{C} \tag{I.9b}$$

$$\mathcal{V} = \mathcal{U}^{(1)}\left(\times_2 \mathbf{Q}^{(0)} \cdots \times_{m+1} \mathbf{Q}^{(m-1)}\right) \tag{I.9c}$$

In principle, the backpropagation rules for the sequential steps can be derived almost identically as in the r-Tucker-dense layer. For reference, we list all equations for the first and last steps as follows:

$$\frac{\partial \mathcal{L}}{\partial \mathcal{U}^{(1)}} = \frac{\partial \mathcal{L}}{\partial \mathcal{V}}\left(\times_2 (\mathbf{Q}^{(0)})^\top \cdots \times_{m+1} (\mathbf{Q}^{(m-1)})^\top\right) \tag{I.10a}$$

$$\left(\frac{\partial \mathcal{L}}{\partial \mathbf{Q}^{(l)}}\right)^\top = \frac{\partial \mathcal{L}}{\partial \mathcal{V}}\left(\times_0^0 \circ \cdots \circ \times_{l+1}^{l+1} \circ \times_{l+3}^{l+3} \circ \cdots \circ \times_{m+1}^{m+1}\right)$$
$$\left(\mathcal{U}^{(1)}\left(\times_2 \mathbf{Q}^{(0)} \cdots \times_{l+1} \mathbf{Q}^{(l-1)} \times_{l+3} \mathbf{Q}^{(l+1)} \cdots \times_{m+1} \mathbf{Q}^{(m-1)}\right)\right) \tag{I.10b}$$

$$\frac{\partial \mathcal{L}}{\partial \mathcal{U}} = \frac{\partial \mathcal{L}}{\partial \mathcal{U}^{(0)}}\left(\times_2 (\mathbf{P}^{(0)})^\top \cdots \times_{m+1} (\mathbf{P}^{(m-1)})^\top\right) \tag{I.10c}$$

$$\left(\frac{\partial \mathcal{L}}{\partial \mathbf{P}^{(l)}}\right)^\top = \frac{\partial \mathcal{L}}{\partial \mathcal{U}^{(0)}}\left(\times_0^0 \circ \cdots \circ \times_{l+1}^{l+1} \circ \times_{l+3}^{l+3} \circ \cdots \circ \times_{m+1}^{m+1}\right)$$
$$\left(\mathcal{U}\left(\times_2 \mathbf{P}^{(0)} \cdots \times_{l+1} \mathbf{P}^{(l-1)} \times_{l+3} \mathbf{P}^{(l+1)} \cdots \times_{m+1} \mathbf{P}^{(m-1)}\right)\right) \tag{I.10d}$$

Notice that the middle step itself is a tensorized convolutional layer defined in Equation I.1b, therefore its backpropagation equations are exactly the same as the ones in Equations I.2a and I.2b.

$$\frac{\partial \mathcal{L}}{\partial \mathcal{U}^{(0)}} = \mathcal{C}\left((*_0^0)^\top \circ (*_1^1)^\top \circ \times_2^{m+2} \circ \cdots \circ \times_{m+1}^{2m+1}\right)\frac{\partial \mathcal{L}}{\partial \mathcal{U}^{(1)}} \tag{I.11a}$$

$$\frac{\partial \mathcal{L}}{\partial \mathcal{C}} = \mathcal{U}^{(0)}\left((*_0^0)^\top \circ (*_1^1)^\top\right)\frac{\partial \mathcal{L}}{\partial \mathcal{U}^{(1)}} \tag{I.11b}$$

The analyses for the backpropagation equations mimic the ones in the forward pass, again by comparison against the ones in r-Tucker-dense layer: the time complexity to obtain the derivatives in the first step is magnified by XY, while the ones for the middle step and the last step are scaled by $HWX'Y'$ and $X'Y'$ respectively. Therefore, the total number of operations for the derivatives with respect to input/intermediate results is $O(mS^{\frac{1}{m}}RXY + HWR^{2m}X'Y' + mT^{\frac{1}{m}}RXY)$, and the number for the factors is $O(mS^{\frac{1}{m}}RXY + XYR^{2m}X'Y' + mT^{\frac{1}{m}}RX'Y')$.

r-Tensor-train-convolutional layer Following the r-CP-convolutional layer, we propose to apply Tensor-train decomposition to the tensorized kernel $\mathcal{K}$ by grouping (S_l, T_l)'s and filter height/width (H, W) as supermodes. In Tensor-train decomposition, these supermodes are ordered by their indices, with the extra supermode (H, W) appended to the end. Concretely, the tensorized kernel $\mathcal{K}$ is decomposed as:

$$\mathcal{K}_{i,j,s_0,\cdots,s_{m-1},t_0,\cdots,t_{m-1}} = \sum_{r_0=0}^{R_0-1} \cdots \sum_{r_{m-1}=0}^{R_{m-1}-1} \mathcal{K}^{(0)}_{s_0,t_0,r_0}\mathcal{K}^{(1)}_{r_0,s_1,t_1,r_1}\cdots\mathcal{K}^{(m)}_{r_{m-1},i,j} \tag{I.12a}$$

$$\mathcal{K} = \mathsf{swapaxes}\left(\mathcal{K}^{(0)} \times_0^{-1} \mathcal{K}^{(1)} \times_0^{-1} \cdots \times_0^{-1} \mathcal{K}^{(m)}\right) \tag{I.12b}$$

where $\mathcal{K}^{(0)} \in \mathbb{R}^{S_0 \times T_0 \times R_0}$, $\mathcal{K}^{(l)} \in \mathbb{R}^{R_{l-1} \times S_l \times T_l \times R_l}$ and $\mathcal{K}^{(m)} \in \mathbb{R}^{R_{m-1} \times H \times W}$ are $(m+1)$ factor tensors. Compared to r-Tensor-train-dense layer, the r-Tensor-train-convolutional layer has an additional factor $\mathcal{K}^{(m)}$ that contains RHW parameters, which leads to a total number of $O((m(ST)^{\frac{1}{m}} + HW)R)$. For conciseness, we follow the preprocessing steps to add singleton mode

$R_{-1} = 1$ to $\mathcal{U}$ and $\mathcal{K}^{(0)}$ such that $\mathcal{U} \in \mathbb{R}^{X \times Y \times S_0 \times \cdots \times S_{m-1} \times R_{-1}}$ and $\mathcal{K}^{(0)} \in \mathbb{R}^{R_{-1} \times S_0 \times T_0 \times R_0}$ and rename $\mathcal{U}$ as $\mathcal{U}^{(0)}$. As we shall expect, the multi-steps procedure to evaluate $\mathcal{V}$ now has $(m+1)$ steps, with the last step as a 3D-convolutional layer:

$$\mathcal{U}^{(l)}_{i+dx,j+dy,\cdots,r_l} = \sum_{r_{l-1}=0}^{R_{l-1}-1} \sum_{s_l=0}^{S_l-1} \mathcal{K}^{(l)}_{r_{l-1},s_l,t_l,r_l} \mathcal{U}^{(l)}_{i+dx,j+dy,s_l,\cdots,s_{m-1},t_0,\cdots,t_{l-1},r_{l-1}} \tag{I.13a}$$

$$\mathcal{V}_{x,y,t_0,\cdots,t_{m-1}} = \sum_{r_{m-1}=0}^{R_{m-1}-1} \sum_{i,j} \mathcal{K}^{(m)}_{r_{m-1},i,j} \mathcal{U}^{(m)}_{i+dx,j+dy,t_0,\cdots,t_{m-1},r_{m-1}} \tag{I.13b}$$

where $\mathcal{U}^{(l)} \in \mathbb{R}^{X \times Y \times S_l \times \cdots \times S_{m-1} \times T_0 \times \cdots \times T_{l-1} \times R_{l-1}}, \forall l \in [m]$ are the intermediate results. The number of operations for the first m steps is $O(m \max(S,T)^{1+\frac{1}{m}} RXY)$ by comparing against the corresponding steps in r-Tensor-train-dense layer, and the last step of 3D-convolutional layer requires $O(HWRTXY)$ operations in the forward pass. In tensor notations, these steps are nicely represented as:

$$\mathcal{U}^{(l)} = \mathcal{U}^{(l)} \left(\times_1^2 \circ \times_0^{-1} \right) \mathcal{K}^{(l)} \tag{I.14a}$$

$$\mathcal{V} = \mathcal{U}^{(m)} \left(*_1^0 \circ *_2^1 \circ \times_0^{-1} \right) \mathcal{K}^{(m)} \tag{I.14b}$$

The backpropagation rules are easily obtained by modifying the ones in r-Tensor-train-dense layer. For completeness, we list all backpropagation equations in the following:

$$\frac{\partial \mathcal{L}}{\partial \mathcal{U}^{(m)}} = \frac{\partial \mathcal{L}}{\partial \mathcal{V}} \left((*_1^0)^\top \circ (*_1^2)^\top \right) \mathcal{K}^{(m)} \tag{I.15a}$$

$$\frac{\partial \mathcal{L}}{\partial \mathcal{K}^{(m)}} = \frac{\partial \mathcal{L}}{\partial \mathcal{V}} \left((*_0^0)^\top \circ (*_1^1)^\top \circ \times_2^2 \cdots \times_{m+1}^{m+1} \right) \mathcal{U}^{(m)} \tag{I.15b}$$

$$\frac{\partial \mathcal{L}}{\partial \mathcal{U}^{(l)}} = \mathsf{swapaxes} \left(\mathcal{K}^{(l)} \left(\times_{-2}^2 \circ \times_{-1}^3 \right) \frac{\partial \mathcal{L}}{\partial \mathcal{U}^{(l+1)}} \right) \tag{I.15c}$$

$$\frac{\partial \mathcal{L}}{\partial \mathcal{K}^{(l)}} = \mathsf{swapaxes} \left(\mathcal{U}^{(l)} \left(\times_0^0 \circ \times_1^1 \circ \times_2^3 \cdots \times_m^{m+1} \right) \frac{\partial \mathcal{L}}{\partial \mathcal{U}^{(l+1)}} \right) \tag{I.15d}$$

The analyses of the backward steps again follow by building connections with r-Tensor-train-dense layer: each of the first m steps is scaled by XY, while the last step requires $O(HWTRX'Y')$ for $\mathcal{U}^{(m)}$ and $O(XYTRX'Y')$ for $\mathcal{K}^{(m)}$.

Decomp.	O(# of params) O(# of forward ops.)	O(# of back ops. for inputs) O(# of back ops. for params.)
original	$HWST$ $HWSTXY$	$HWSTX'Y'$ $XYSTX'Y'$
r-CP	$\#\mathbf{Params_{CP}} + HWR$ $\#\mathbf{Ops_{CP}}XY + HWTRXY$	$\#\mathbf{Ops_{CP}}XY + HWTRX'Y'$ $\#\mathbf{Ops_{CP}}XY + XYTRX'Y'$
r-TK	$(HW\#\mathbf{Params_{TK_C}} +$ $\#\mathbf{Params_{TK_I}} + \#\mathbf{Params_{TK_O}}$ $(HW\#\mathbf{Ops_{TK_C}}XY +$ $\#\mathbf{Ops_{TK_I}}XY + \#\mathbf{Ops_{TK_O}}X'Y')$	$(HW\#\mathbf{Ops_{TK_C}}X'Y' +$ $\#\mathbf{Ops_{TK_I}}XY + \#\mathbf{Ops_{TK_O}}X'Y')$ $(XY\#\mathbf{Ops_{TK_C}}X'Y' +$ $\#\mathbf{Ops_{TK_I}}XY + \#\mathbf{Ops_{TK_O}}X'Y')$
r-TT	$\#\mathbf{Params_{TT}} + HWR$ $\#\mathbf{Ops_{TT}}XY + HWTRXY$	$\#\mathbf{Ops_{TT}}XY + HWTRX'Y'$ $\#\mathbf{Ops_{TT}}XY + XYTRX'Y'$

†In order to compare against reshaped tensor decompositions on dense layer in Table 11, we denote the numbers for the dense layers as:

$\#\mathbf{Params_{CP}} = m(ST)^{\frac{1}{m}}R$	$\#\mathbf{Params_{TK}} = \sum_{\mathbf{L}} \#\mathbf{Params_{TK_L}}$	$\#\mathbf{Params_{TT}} = m(ST)^{\frac{1}{m}}R^2$
$\#\mathbf{Ops_{CP}} = m\max(S,T)^{1+\frac{1}{m}}$	$\#\mathbf{Ops_{TK}} = \sum_{\mathbf{L}} \#\mathbf{Ops_{TK_L}}$	$\#\mathbf{Ops_{TT}} = m\max(S,T)^{1+\frac{1}{m}}R^2$
$\#\mathbf{Params_{TK_I}} = mS^{\frac{1}{m}}R$	$\#\mathbf{Params_{TK_C}} = R^{2m}$	$\#\mathbf{Params_{TK_O}} = mT^{\frac{1}{m}}R$
$\#\mathbf{Ops_{TK_I}} = mSR$	$\#\mathbf{Ops_{TK_C}} = R^{2m}$	$\#\mathbf{Ops_{TK_O}} = mTR$

Table 12: Summary of reshaped tensor decomposition on convolutional layer. In this table, we list the number of parameters and time complexities required by various reshaped tensor decompositions on convolutional layer. Recall that a convolutional layer, composed with ST filters of size $H \times W$, maps a set of S feature maps of size $X \times Y$ to another set of T feature maps of size $X' \times Y'$. For simplicity, we assume the numbers of input/output feature maps S, T are factorized evenly, i.e. $S_l = S^{\frac{1}{m}}, T_l = T^{\frac{1}{m}}, \forall l \in [m]$, and all ranks are equal to R, i.e $R_l = R, \forall l \in [m]$.