Distribution-Aware Tensor Decomposition for Compression of Convolutional Neural Networks

Alper Kalle[*†]
alper.kalle@cea.fr

Theo Rudkiewicz[*‡]
theo.rudkiewicz@inria.fr

Mohamed-Oumar Ouerfelli[†]
mohamed-oumar.ouerfelli@cea.fr

Mohamed Tamaazousti[†]
mohamed.tamaazousti@cea.fr

Abstract

Neural networks are widely used for image–related tasks but typically demand considerable computing power. Once a network has been trained, however, its memory- and compute-footprint can be reduced by compression. In this work, we focus on compression through tensorization and low-rank representations. Whereas classical approaches search for a low-rank approximation by minimizing an isotropic norm such as the Frobenius norm in weight-space, we use data-informed norms that measure the error in function space. Concretely, we minimize the change in the layer's output distribution, which can be expressed as $\|(W - \widetilde{W})\Sigma^{1/2}\|_F$ where $\Sigma^{1/2}$ is the square root of the covariance matrix of the layer's input and W, $\widetilde{W}$ are the original and compressed weights. We propose new alternating least square algorithms for the two most common tensor decompositions (Tucker-2 and CPD) that directly optimize the new norm. Unlike conventional compression pipelines, which almost always require post-compression fine-tuning, our data-informed approach often achieves competitive accuracy without any fine-tuning. We further show that the same covariance-based norm can be transferred from one dataset to another with only a minor accuracy drop, enabling compression even when the original training dataset is unavailable. Experiments on several CNN architectures (ResNet-18/50, and GoogLeNet) and datasets (ImageNet, FGVC-Aircraft, Cifar10, and Cifar100) confirm the advantages of the proposed method.

1 Introduction

For the past decade, neural networks have consistently outperformed other algorithms across a wide array of tasks particularly in computer vision, including character recognition, image classification, object detection, image segmentation, and depth estimation. Notably, even with the rise of other architectures, Convolutional Neural Networks (CNNs) continue to push the boundaries of performance in computer vision, with recent advancements demonstrating their sustained competitiveness for the state-of-the-art results [Liu et al., 2022, Woo et al., 2023]. A defining characteristic of these networks, in contrast to traditional algorithms (e.g., nearest neighbors, decision trees), is their substantial number of parameters. For instance, prominent architectures such as AlexNet [Krizhevsky et al., 2017], VGG-16 [Simonyan and Zisserman, 2015], ResNet [He et al., 2016] and GoogleNet [Szegedy et al., 2015] possess tens of millions of trainable parameters.

[*] These authors contributed equally

[†] Université Paris-Saclay, CEA, List, F-91120, Palaiseau, France

[‡] TAU team, LISN, Université Paris-Saclay, CNRS, Inria, 91405, Orsay, France

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

Despite the performance benefits often associated with a large parameter count during training, opportunities exist for creating more compact representations of trained models. This is particularly critical for applications requiring efficient execution on resource-constrained embedded devices, such as on-device image classification for smartphones and systems for autonomous vehicles. Consequently, a significant body of research has emerged, focusing on techniques to reduce the parameter count in neural networks [Cheng et al., 2018], including methods like quantization, pruning, knowledge distillation, and low-rank factorization.

Classical approaches to low-rank tensor approximation for network compression typically involve finding a compressed representation that minimizes an isotropic norm, such as the Frobenius norm, of the difference between the original and compressed weights. This minimization is performed directly in the weight space of the network. While mathematically convenient, such an approach does not explicitly consider the impact of weight changes on the network's functional behavior or its output given the data it processes. Consequently, these methods often lead to a notable drop in accuracy, necessitating a subsequent, computationally intensive fine-tuning stage to recover performance. This fine-tuning step typically requires access to the original training dataset, which may not always be available.

In this paper, we formalize a paradigm shift from classical weight-space error minimization for tensorized layers. Traditional tensorization approaches replace a computational block, typically a single layer f_θ parameterized by θ, with a sequence of more compact layers—for instance, a composition $f_{A,B,C} := g_A \circ h_B \circ i_C$, where A, B, C are the new, smaller parameter tensors. Classical methods then seek to find A, B, C such that their effective combined parameters are close to the original θ i.e. $\|r(A, B, C) - \theta\|_F$ is small (where r is the operator that combines the low rank tensors). In contrast, the data-informed approach aims to ensure that the function implemented by the sequence of compressed layers, $f_{A,B,C}$, closely approximates the original layer's function, f_θ, for the actual data the network encounters. This means we measure the error directly in the function space $\|f_{A,B,C} - f_\theta\|_{\mathrm{L}^2}$, prioritizing the preservation of the layer's input-output behavior rather than just its parameter values. This make it a functional norm and distribution-aware norm. This leads to properly justify the "data norm" that was used by Denton et al. [2014] in the case of SVD decomposition.

To operationalize this concept, we develop novel Alternating Least Squares (ALS) algorithms specifically designed to optimize this new data-informed norm for two of the most widely used tensor decomposition formats: Tucker-2 and Canonical Polyadic Decomposition (CPD). A significant advantage of our approach is its ability to achieve competitive accuracy levels often without any post-compression fine-tuning, thereby streamlining the compression pipeline and reducing its dependency on extensive retraining.

Furthermore, we demonstrate a crucial property of our covariance-based norm: its transferability. We show that the input covariance statistics learned from one dataset can be effectively applied to compress models for different, related datasets with only a minor degradation in accuracy. This finding is particularly valuable as it opens up possibilities for compressing models even when the original training data is inaccessible due to privacy, proprietary restrictions, or other constraints.

Our contributions are:

1. The use of distribution-informed norm for the neural network tensor decomposition.

2. Development of new ALS algorithms for Tucker-2 and CPD tensor decompositions that directly optimize this distribution-informed norm.

3. In depth comparison of tensor decomposition with Tucker and CP decompositions with and without the knowledge of data distribution, and also comparing two different approach: tensor deflation and CP-ALS algorithms with respect to distribution-informed norm.

4. Empirical evidence showing that our approach often achieves competitive accuracy without fine-tuning, and that the learned covariance-based norm can be transferred across datasets.

We validate our proposed methods through comprehensive experiments on several prominent CNN architectures, including ResNet-18, ResNet-50, and GoogLeNet, across diverse benchmark datasets such as ImageNet, FGVC-Aircraft, Cifar10, and Cifar100. The results consistently underscore the advantages of our distribution-informed compression strategy.

2 Related work

Our work intersects three active threads in neural-network compression: tensor-based methods, functional norms, and data-free compression techniques.

Tensorization Tensor decomposition reduces both parameter count and computational cost by approximating weight tensors with low-rank formats. Early efforts include Denton et al. [2014]; subsequent milestones span CP decomposition for CNNs [Lebedev et al., 2015], Tensor-Train for fully connected layers [Novikov et al., 2015], Tucker decomposition [Kim et al., 2016], and Tensor-Ring networks [Zhao et al., 2019]. Most of these methods follow a two-step pipeline: (i) compute a low-rank approximation, then (ii) fine-tune to restore accuracy. Several studies instead alternate tensorization with fine-tuning—e.g., the tensor-power method for successive layer compression [Astrid and Lee, 2017] and the MUSCO pipeline, which repeatedly compresses and fine-tunes [Gusak et al., 2019]. Rank selection has also been explored: variational Bayesian matrix factorization (VBMF) offers a data-free criterion, applied to Tucker layers [Kim et al., 2016] and later to CP layers [Astrid et al., 2018], building on the analytic VBMF solution of Nakajima et al. [2010].

Functional norms Most of the above techniques minimise error in *weight space*, which is not directly linked to a network's functional behaviour. Functional norms instead measure distances between functions. They underpin natural-gradient optimisation [Ollivier, 2017, Schwencke and Furtlehner, 2024] and can guide network growth [Verbockhaven et al., 2024]. In compression, alternative norms are less common; one example is Lohit and Jones [2022], who employ an optimal-transport loss during distillation. Another important example is Denton et al. [2014] who propose the use of two non-isotropic norms: the Mahalanobis distance and the data distance. The latest was not presented as a functional norm but, as we will show later, their data norm can in fact be seen as the $L2$ functional norm. For this data distance, they proposed replacing the approximation criterion from the Frobenius norm with a data covariance distance metric, which accounts for the empirical covariance of the input layer. They recommended using this criterion alongside either the SVD method or a greedy CP algorithm, which iteratively computes rank-one tensors for approximate CP decomposition and conducted experiments for this distance using the SVD method. Furthermore, to the best of our knowledge, we are the first to conduct multiple experiments on several classical CNNs across different datasets to assess the benefits of this new norm. Finally, we also expand the analysis to evaluate the transferability of this norm between different datasets.

Data-free compression Several mainstream compression techniques now operate without access to the original training data. Data-free knowledge distillation was introduced by Lopes et al. [2017]. Zero-shot quantization approaches such as ZeroQ [Cai et al., 2020] and DFQ [Nagel et al., 2019] remove the need for calibration data. For pruning, Srinivas and Babu [2015] merge redundant neurons, Tang et al. [2021] prune using synthetically generated data, and SynFlow prunes networks even before training [Tanaka et al., 2020]. To the best of our knowledge, no existing work tackles the data-free high order tensor approximation problem. We address this by evaluating the use of a generic distinct dataset to the one used during model training.

Alternative Norms Defining an importance score, or norm, is central to structured network compression. Prevailing methods rely on simple weight magnitude or more complex, loss-based metrics derived from the Hessian Matrix [LeCun et al., 1989] or Fisher Information Matrix [Tu et al., 2016]. Our work explores an alternative based on activation statistics. Specifically, our distribution-aware norm uses the activation covariance matrix to model the complete second-order statistical structure of feature maps. This provides a richer measure of importance than common activation-based heuristics, which are often limited to first-order statistics like mean magnitude [Rhu et al., 2018]. By capturing inter-channel dependencies, our norm offers a more holistic criterion for guiding the compression.

3 Background on compression by tensor decomposition

The general principle of tensor decomposition is to replace a part of the neural network by a lighter one. For example if we have a neural network $f = q \circ l \circ p$ where q and p are two neural networks and l is one layer of the neural network, we can replace l by a new squence of layers $l' := l_1 \circ l_2 \circ \cdots \circ l_n$. The new neural network is then $f' := q \circ l' \circ p$. The goal of the tensor decomposition is to find the best

l' such that f and f' are as close as possible. In most of the case the layer l is either a fully connected layer or a convolutional layer. In the case of a fully connected layer, the tensor decomposition is done through the singular value decomposition (SVD) and lead to replace the layer by a couple of fully connected layers. In the case of a convolutional layer, the tensor decomposition can be done through the CP decomposition or the Tucker decomposition as described in the next section.

Convolution A convolution $\mathbf{Conv}_{\mathcal{K}}$ parameterized by a tensor $\mathcal{K}$ of size (T, S, H, W) is a function from the space of images with S channels like $\mathcal{X} \in \mathbb{R}^{S \times H_y \times W_x}$ where H_y, W_x represent the height and width of the image to the space of images with T channels like $\mathcal{Y} \in \mathbb{R}^{T \times H'_y \times W'_x}$. The convolution layer has $SHWT$ parameters and the cost of the convolution operation given above is $(SHWT)(H'_y W'_x)$ additions-multiplications.

By decomposing the kernel tensor we can replace the convolution by a series of smaller layers in terms parameters and computational cost. In the next sections, we present two possible decompositions that can be seen as generalizations of singular value decomposition (SVD) to the tensor case.

3.1 CP Decomposition for Convolutional Layer Compression

The CANDECOMP/PARAFAC (CP) decomposition [Hitchcock, 1927] represents a tensor as sum of rank 1 tensors, where the rank 1 tensors are them self decomposed as the outer product of vectors. For a rank R, CP decomposition of a kernel tensor $\mathcal{K} \in \mathbb{R}^{T \times S \times H \times W}$ is given by the following formula:

$$\widetilde{\mathcal{K}} = \sum_{r=1}^{R} \boldsymbol{U}_r^{(T)} \otimes \boldsymbol{U}_r^{(S)} \otimes \boldsymbol{U}_r^{(H)} \otimes \boldsymbol{U}_r^{(W)} \tag{1}$$

where $\boldsymbol{U}_r^{(n)} \in \mathbb{R}^n$ for $n \in [T, S, H, W]$ are the equivalent of the singular vectors and $\otimes$ denotes the outer product.

This decomposition leads as suggested by Lebedev et al. [2015] to replace the convolution by a series of four layers: a 1×1 convolution parameterized by $(\boldsymbol{U}_r^{(S)})_r$, a $H \times 1$ convolution parameterized by $(\boldsymbol{U}_r^{(H)})_r$, a $1 \times W$ convolution parameterized by $(\boldsymbol{U}_r^{(W)})_r$ and a 1×1 convolution parameterized by $(\boldsymbol{U}_r^{(T)})_r$. The number of parameters of the CP decomposition is $R \times (T + S + H + W)$ which for a small rank R is smaller than the number of parameters of the original convolutional layer; the reduction factor is the same for the computational cost.

To choose a rank for the decompostion without reliying on trial and errors, we use the VBMF (Variational Bayesian Matrix Factorization) algorithm as suggested by Astrid et al. [2018]. We note R_{VBMF} the maximum of ranks computed through application of VBMF to the each possible unfolding of the kernel tensor and $R_{\max}$ an upper bound of the rank with $R_{\max} := \frac{TSHW}{\max\{T,S,H,W\}}$. We use a rank R_α that is a linear combination of those ranks determined by a parameter α that we call VBMF ratio:

$$R_\alpha = R_{\text{VBMF}} + (1 - \alpha)(R_{\max} - R_{\text{VBMF}}). \tag{2}$$

3.2 Tucker Decomposition for Convolutional Layer Compression

The Tucker decomposition [Tucker, 1966] countrary to the CP decomposition use a different number of eigenvectors for each mode. In the case of convolutional layers, as the two mode corresponding to the kernel size (H and W) are small, we do not decompose them which leads to the Tucker2 decomposition. For a kernel tensor $\mathcal{K} \in \mathbb{R}^{T \times S \times H \times W}$, the Tucker2 decomposition with rank R_T and R_S is $\widetilde{\mathcal{K}} = \mathcal{G} \times_1 \boldsymbol{U}^{(T)} \times_2 \boldsymbol{U}^{(S)}$*.

As suggested by Kim et al. [2016], this leads to replace the convolution by a series of two layers: a 1×1 convolution parameterized by $\boldsymbol{U}^{(T)}$, a full convolution parameterized by $\mathcal{G}$ and a 1×1 convolution parameterized by $\boldsymbol{U}^{(S)}$.

To select the ranks R_T and R_S, we use the same method as for CP decomposition. We compute the ranks R_{T-VBMF} and R_{S-VBMF} by applying the VBMF algorithm to the unfolding matrices of sizes $T \times SHW$ and $S \times THW$. Then we compute the ranks R_T and R_S using the formula given

*$\times_i$ indicates a product along the ith axis of the tensor $\mathcal{G}$

in equation (2) on the VBMF ranks R_{T-VBMF} and R_{S-VBMF}, respectively (with the $R_{\max}$ being either T or S).

4 The Distribution-Aware Tensor Decomposition

4.1 Functional Metric for Network Compression

In general when replacing the layer l_θ from $f = q \circ l_\theta \circ p$ to get $f' = q \circ l'_{\theta'} \circ p$ we don't want to retrain the network so we aim to have $f \approx f'$. Mathematically for a data distribution $\mathcal{D}$, for a classification task we want to have $\mathbb{E}_{x\sim\mathcal{D}}(d(f(x), f'(x))) \approx 0$ where d is a way to measure the distance between two distributions like the Wasserstein distance or the KL divergence.

Most of the works use as proxy the Frobenius norm $\|\theta - \theta'\|_F$ where θ are the parameters of the layer l_θ and θ' are the combined parameters of the new layer $l'_{\theta'}$ (this combination is different depending on the method (see previous section)). Here we propose instead to minimize the functional norm $\|l_\theta \circ p - l'_{\theta'} \circ p\|_{\mathsf{L}^2}$ where p is the network before the layer l_θ where we take as measure the data distribution $\mathcal{D}$. Hence, we have:

$$\|l_\theta \circ p - l'_{\theta'} \circ p\|_{\mathsf{L}^2} = \sqrt{\mathbb{E}_{x\sim\mathcal{D}}\Big(\|l_\theta \circ p(x) - l'_{\theta'} \circ p(x)\|_F^2\Big)}. \tag{3}$$

To compute the $\|\cdot\|_{\mathsf{L}^2}$ in the case where l_θ is a convolution we use the following result:

Proposition 1. *Consider a distribution $\mathcal{D}$, a partial neural network p, and two convolution $\mathbf{Conv}_\mathcal{K}$ and $\mathbf{Conv}_{\widetilde{\mathcal{K}}}$ parametrized by the kernel tensor $\mathcal{K} \in \mathbb{R}^{T\times S\times H\times W}$ and $\widetilde{\mathcal{K}}$. Under reasonable assumptions* [†]*, we can define $\Sigma := \mathbb{E}_{x\sim\mathcal{D}}\big(u(p(x))u(p(x))^\top\big)$ where u is the unfolding operator* [‡] *that transforms the image $p(x)$ into a matrix that can be used to compute the convolution as a matrix product. We can also define $\Sigma^{1/2}$ the square root of Σ such that $\Sigma^{1/2}(\Sigma^{1/2})^\top = \Sigma$. Then, we have:*

$$\left\|\mathbf{Conv}_\mathcal{K} \circ p - \mathbf{Conv}_{\widetilde{\mathcal{K}}} \circ p\right\|_{\mathsf{L}^2} = \left\|\Big(\mathcal{K} - \widetilde{\mathcal{K}}\Big)_{(1)} \Sigma^{1/2}\right\|_F \tag{4}$$

where $(\cdot)_{(1)}$ is the reshaping of the convolution kernel into $(T, S \times H \times W)$.

See appendix A for the proof.

Frobenius norm Here we can notice that in the case where $\Sigma = I_n$, for example if we consider that $\mathcal{D} = \mathcal{N}(0, I)$, we recover the Frobenius norm over the parameters of the layer. Hence the functional norm is a generalization of the Frobenius norm over the parameters of the layer, suited for the case of a non isotropic data distributions. In the following, we call the distribution-aware norm given in the Eq. 4 as Sigma norm.

4.2 Decomposition Computation

Although taking the input data distribution into account lead to better compression results, we cannot directly apply the highly efficient algorithms used for kernel approximation based on the Frobenius norm. In this paper, we introduce new efficient algorithms, equivalent to the widely used CP-ALS and Tucker-ALS methods for the Frobenius norm, that approximate the kernel using this new metric.

4.2.1 Alternating Least Square Algorithm for CP with Distribution-Aware Norm

The CP decomposition as presented in Eq. 1 involves 4 terms. The main idea of the ALS algorithm is to minimize successively each of the terms and then to iterate this process. The minimization of each term is convex and can be done with a close formula using the pseudo-inverse. We details below how the minimization can be done with Sigma norm.

[†] The reasonable assumptions are that the function p is in L^2 for the distribution $\mathcal{D}$. This is likely the case in our range of applications since p is a neural network that is in most cases continuous and $\mathcal{D}$ can be considered as restricted to a compact set.

[‡] https://docs.pytorch.org/docs/stable/generated/torch.nn.Unfold.html

Firstly, we note the properties of vectorization of matrices which we will use to adopt the Sigma norm for ALS algorithm. Let $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{n \times k}$ be two matrices. Then, we can state the followings, for all $i \in [\![1, m]\!], j \in [\![1, n]\!]$,

$$\mathrm{Vec}(A)[m(j-1)+i] = A[i,j], \text{ and} \tag{5}$$

$$\mathrm{Vec}(AB) = (B^\top \otimes \mathrm{Id}(m))\ \mathrm{Vec}(A) = (\mathrm{Id}(k) \otimes A)\ \mathrm{Vec}(B). \tag{6}$$

Given that the first unfolding of CP decomposition is $\widetilde{\mathcal{K}}_{(1)} = \boldsymbol{U}^{(T)}\ (\boldsymbol{U}^S \odot \boldsymbol{U}^H \odot \boldsymbol{U}^W)^\top$ where $\odot$ denotes the Khatri–Rao product, we have:

$$\begin{aligned}
\mathrm{Vec}(\widetilde{\mathcal{K}}_{(1)}\Sigma^{1/2}) &= \Big((\Sigma^{1/2})^\top \otimes \mathrm{Id}(T)\Big)\ \mathrm{Vec}(\widetilde{\mathcal{K}}_{(1)}) && (7)\\
&= \Big((\Sigma^{1/2})^\top \otimes \mathrm{Id}(T)\Big)\ \mathrm{Vec}(\boldsymbol{U}^{(T)}\ (\boldsymbol{U}^S \odot \boldsymbol{U}^H \odot \boldsymbol{U}^W)^\top) && (8)\\
&= \Big((\Sigma^{1/2})^\top \otimes \mathrm{Id}(T)\Big)\ \Big((\boldsymbol{U}^{(S)} \odot \boldsymbol{U}^{(H)} \odot \boldsymbol{U}^{(W)}) \otimes \mathrm{Id}(T)\Big)\ \mathrm{Vec}(\boldsymbol{U}^{(T)}). && (9)
\end{aligned}$$

Thus, we can iterate over each factor matrices to minimize the error between the kernel tensor $\mathcal{K}$ and its approximation $\widetilde{\mathcal{K}}$ under Sigma norm. In other words, for the factor matrix $\boldsymbol{U}^{(T)}$ we have the following minimization problem:

$$\min_{U^{(T)}} \left\| \mathrm{Vec}(\mathcal{K}\Sigma^{1/2}) - \boldsymbol{P}^{(T)}\mathrm{Vec}(\boldsymbol{U}^{(T)}) \right\|_F, \tag{10}$$

where $\boldsymbol{P}^{(T)} = \big((\Sigma^{1/2})^\top \otimes \mathrm{Id}(T)\big)\ \big((\boldsymbol{U}^{(S)} \odot \boldsymbol{U}^{(H)} \odot \boldsymbol{U}^{(W)}) \otimes \mathrm{Id}(T)\big)$, and similarly for the other matrices corresponding to the components S, H, W, which are detailed in the appendix B.2.

Since the least square minimization problems have a closed form solution, we can deduce the factor matrix $U^{(T)}$ (and similarly the other factors) with the following formula:

$$\mathrm{Vec}(U^{(T)}) \leftarrow (\boldsymbol{P}^{(T)})^\dagger \mathrm{Vec}(\mathcal{K}\Sigma^{1/2}) \tag{11}$$

where $\boldsymbol{A}^\dagger$ denotes the Moore–Penrose inverse of the matrix $\boldsymbol{A}$. As a result, we present the full algorithm called CP-ALS-Sigma in algorithm 1 and we refer the appendix B.4 for the practical use of the algorithm 1.

Algorithm 1 CP-ALS-Sigma

```
function [$\boldsymbol{U}^{(T)}, \boldsymbol{U}^{(S)}, \boldsymbol{U}^{(W)}, \boldsymbol{U}^{(H)}, m$] = CP-ALS-SIGMA
    Give initializations for matrices $\boldsymbol{U}^{(T)}, \boldsymbol{U}^{(S)}, \boldsymbol{U}^{(H)}, \boldsymbol{U}^{(W)}$
    for $n = 1, \ldots, m$ do
        $\mathrm{Vec}(\boldsymbol{U}^{(T)}) \leftarrow (\boldsymbol{P}^{(T)})^\dagger\ \mathrm{Vec}(\mathcal{K}\Sigma^{1/2})$    ▷ update $\boldsymbol{U}^{(T)}$ with a least square solution
        $\mathrm{Vec}(\boldsymbol{U}^{(S)}) \leftarrow (\boldsymbol{P}^{(S)})^\dagger\ \mathrm{Vec}(\mathcal{K}\Sigma^{1/2})$    ▷ update $\boldsymbol{U}^{(S)}$ with a least square solution
        $\mathrm{Vec}(\boldsymbol{U}^{(H)}) \leftarrow (\boldsymbol{P}^{(H)})^\dagger\ \mathrm{Vec}(\mathcal{K}\Sigma^{1/2})$    ▷ update $\boldsymbol{U}^{(H)}$ with a least square solution
        $\mathrm{Vec}(\boldsymbol{U}^{(W)}) \leftarrow (\boldsymbol{P}^{(W)})^\dagger\ \mathrm{Vec}(\mathcal{K}\Sigma^{1/2})$    ▷ update $\boldsymbol{U}^{(W)}$ with a least square solution
    return factor matrices $\boldsymbol{U}^{(T)}, \boldsymbol{U}^{(S)}, \boldsymbol{U}^{(W)}, \boldsymbol{U}^{(H)}$
```

4.2.2 Alternating Least Square Algorithm for Tucker2 with Distribution-Aware Norm

The principle of the ALS algorithm for Tucker2 is very similar to the CP case. We derive a quadratic minimization problem for each factor when both others are fixed. Those minimization problems can be solved in closed form using the matrix pseudo-inverse. In the end, this leads to the following algorithm 2. Full details can be found in appendix B.3 and the practical use of the algorithm 2 is provided in the appendix B.4.

5 Experimental Validation

In our experiments, we investigate the convolutional neural network models Resnet18, Resnet50, and GoogLeNet (pretrained on ImageNet 1K) to assess the performance of the proposed algorithms CP-ALS-Sigma (Algorithm 1) and Tucker2-ALS-Sigma (Algorithm 2). We do not compress the first

Algorithm 2 Tucker2-ALS-Sigma

```
function $[\boldsymbol{U}^{(T)}, \boldsymbol{U}^{(S)}, \mathcal{G}, m] = \text{TUCKER2-ALS-SIGMA}$
    Give initializations for matrices $\boldsymbol{U}^{(T)}, \boldsymbol{U}^{(S)}$
    for $n = 1, \ldots, m$ do
        $\text{Vec}(\boldsymbol{U}^{(T)}) \leftarrow (\boldsymbol{P}^{(T)})^{\dagger}\ \text{Vec}(\mathcal{K}\Sigma^{1/2})$    ▷ update $\boldsymbol{U}^{(T)}$ with a least square solution
        $\text{Vec}(\boldsymbol{U}^{(S)}) \leftarrow (\boldsymbol{P}^{(S)})^{\dagger}\ \text{Vec}(\mathcal{K}\Sigma^{1/2})$    ▷ update $\boldsymbol{U}^{(S)}$ with a least square solution
        $\text{Vec}(\mathcal{G}) \leftarrow (\boldsymbol{P}^{(\mathcal{G})})^{\dagger}\ \text{Vec}(\mathcal{K}\Sigma^{1/2})$    ▷ update $\mathcal{G}$ with a least square solution
    return core tensor $\mathcal{G}$ and factor matrices $\boldsymbol{U}^{(T)}, \boldsymbol{U}^{(S)}$
```

convolutional layer for each model, as doing so degrades performance. Additionally, we used the Tensorly package [Kossaifi et al., 2019] to compute the standard CP and Tucker decompositions, referred to as CP-ALS and Tucker2-ALS, respectively. Some parts of the code were also adapted from the MUSCO library [Gusak et al., 2019] to perform the compression on neural networks.

5.1 Evaluation of the CP-ALS and Tensor Deflation with Distribution-Aware Norm

An alternative to the full ALS method is the tensor deflation method. In details, tensor deflation method suggests to update iteratively $W^{(k+1)} \leftarrow W^{(k)} - \alpha_k \otimes \beta_k \otimes \gamma_k \otimes \delta_k$ where $\alpha_k \otimes \beta_k \otimes \gamma_k \otimes \delta_k$ is the rank-one approximation of $W^{(k)}$ such that the low rank approximation is given by $\widetilde{\mathcal{W}} = \sum_{k=1}^{R} \alpha_k \otimes \beta_k \otimes \gamma_k \otimes \delta_k$. We compare this greedy approach with our ALS algorithm that optimize all the ranks at the same time, in the case where we optimize the distribution-aware norm. In Table 1, we show the reconstruction error for the distribution-aware norm ($\left\| \left(\mathcal{K} - \widetilde{\mathcal{K}} \right)_{(1)} \Sigma^{1/2} \right\|_F / \left\| \mathcal{K}_{(1)} \Sigma^{1/2} \right\|_F$ where $\mathcal{K}$ is the original tensor and $\widetilde{\mathcal{K}}$ is the decomposed one) when decomposing kernels of ResNet 18. We observe a clear superiority of the full ALS algorithm in reconstruction error (Table 1) and in accuracy after compression (Table 2).

Table 1: Relative reconstruction errors of Greedy Tensor Deflation(TD) and CP-ALS-Sigma algorithms applied to convolutional layers of ResNet18 where the ranks are estimated through VBMF with the ratio $\alpha = 0.8$.

Conv Layer	Rank	Recons. Err. (Greedy TD)	Recons. Err. (CP-ALS-Sigma)	Improvement (Greedy / ALS)
Layer1.0.conv2	134	0.195	**0.053**	3.7
Layer2.0.conv1	140	0.277	**0.118**	2.3
Layer3.0.conv1	291	0.217	**0.094**	2.3
Layer3.1.conv1	520	0.207	**0.070**	3.0
Layer4.1.conv1	1023	0.159	**0.063**	2.5
Layer4.1.conv2	1167	0.292	**0.051**	5.7

Table 2: Top-1 accuracies of the decomposed models obtained via Greedy Tensor Deflation (TD) and CP-ALS-Sigma algorithms with respect to distribution-aware norm applied to ResNet18 on ImageNet dataset with different compression rates.

VBMF Ratio	Compression Rate	Acc. (Greedy TD)	Acc.(CP-ALS-Sigma)
0.8	2.07	35.2	**67.9**
0.85	2.53	20.9	**66.5**
0.9	3.25	5.1	**63.1**

5.2 Model Compression and Fine-Tuning with Limited Data Access

In this section, we consider the scenario where only a limited amount of data is available, a common situation in many applications. Specifically, we consider the case where only 50,000 images from the ImageNet training set are available to estimate the matrix Σ and to fine-tune the model.

We compare the classification accuracy of the compressed model with Tucker2-ALS-Sigma, Tucker2-ALS algorithms for Frobenius norm and its fine-tuned version. For the network compressed using the standard ALS algorithm under the Frobenius norm, we fine-tune it with the Adam optimizer, selecting the optimal learning rate from the range 10^{-5} to 10^{-10}. We test the performance of our method at different compression rates by varying the parameter α in the rank formula (2). For this, we selected the following values of α: $[0.4, 0.45, 0.5, 0.55]$ for Resnet18, $[0.8, 0.9, 0.95, 1]$ for Resnet50, $[0.6, 0.7, 0.8]$ for GoogLeNet.

As shown in Figure 1, our results indicate that the Tucker2-ALS-Sigma algorithm consistently outperforms both the standard Tucker2-ALS algorithm and the fine-tuned compressed models obtained with Tucker2-ALS across all the neural networks evaluated. For additional results on the CP-ALS-Sigma algorithm, we refer to Appendix C.1.

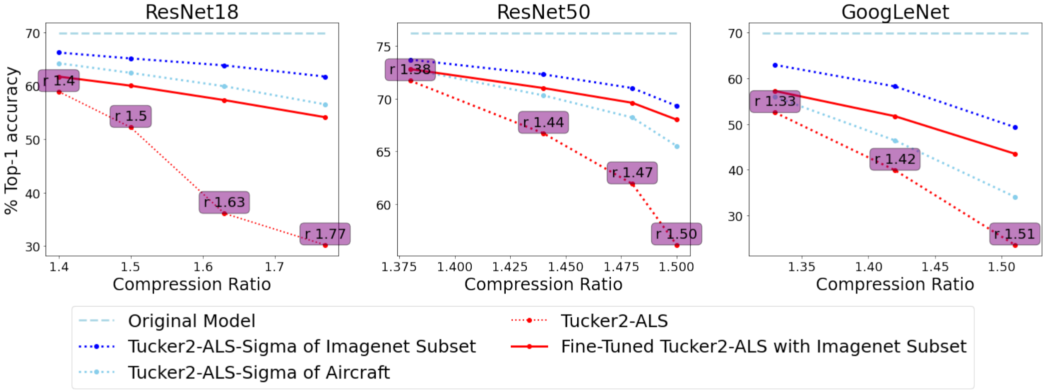

Figure 1: Accuracy comparison of decomposed models obtained with Tucker2-ALS-Sigma and Tucker2-ALS algorithms, also including fine-tuned decomposed model (with Tucker2-ALS algorithm) results where the fine-tuning done on the subset of ImageNet train dataset. (rX denotes the compression ratio, calculated by dividing the number of parameters of the original model by that of the compressed model.)

5.3 Impact of Dataset Changes on Proposed Algorithms

This section aims to assess how variations in the dataset affect the performance of the proposed CP-ALS-Sigma and Tucker2-ALS-Sigma algorithms. In particular, we demonstrate that the distribution-aware norm can be computed from a dataset different from the one used for pretraining, highlighting its transferability. To achieve this, we performed compression on ResNet18, ResNet50, and GoogLeNet models, all trained on the CIFAR-10 dataset, using our proposed ALS-Sigma algorithms (with the Σ matrix obtained from different datasets) and the standard ALS algorithm, applied to both CP and Tucker decompositions. In addition, we fine-tuned the compressed models that are obtained using the standard ALS algorithms to compare their performance with the Sigma-based methods (without fine-tuning). Fine-tuning was conducted on the CIFAR-10 training dataset, employing the Adam optimizer and testing various learning rates from 10^{-3} to 10^{-7}, selecting the best-performing learning rate. Results for the CP-ALS-Sigma algorithm can be found in Appendix C.2.

We compare Tucker2-ALS-Sigma with the standard Tucker2-ALS algorithm and the fine-tuned compressed model obtained from Tucker2-ALS. We chose α from $[0.5, 0.6, 0.7, 0.8, 0.9, 1]$ for Resnet18, $[0.9, 1, 1.1, 1.2, 1.3, 1.4]$ for Resnet50, $[0.6, 0.7, 0.8, 0.9, 0.95, 1]$ for GoogLeNet. Our results, shown in Figure 2, indicate that Tucker2-ALS-Sigma algorithm has better performance than the standard Tucker2-ALS algorithm across all models, even when the Σ matrix is computed using different datasets, such as a subset of the ImageNet training set or the CIFAR-100 training set. Additionally, while Tucker2-ALS experiences rapid performance degradation as the compression rate increases, our method remains much more consistent. Furthermore, the Tucker2-ALS-Sigma algorithm produces results that are close to those of the fine-tuned compressed model obtained using the Tucker2-ALS algorithm. Notably, the performance of the Tucker2-ALS-Sigma algorithm on the CIFAR-100 dataset is comparable to that on the CIFAR-10 dataset, suggesting that our approach is not limited to using the original training dataset for compression with a distribution-aware norm. In fact, as illustrated in Figure 2, the results for the Tucker2-ALS-Sigma algorithm with the CIFAR-100 dataset even outperform those with CIFAR-10 for the ResNet50 model. However, when the Sigma matrix is derived from a subset of the ImageNet training set, the performance of the Tucker2-ALS-Sigma algorithm is somewhat less effective compared to when it is computed on CIFAR-10 or CIFAR-100, likely due to differences in image resolution. We refer to the Appendix G for additional experiments which investigate the effects of image resolution, dataset diversity, and the number of samples chosen for the Σ matrix on the performance of proposed algorithms.

Figure 1 demonstrates that the Tucker2-ALS-Sigma algorithm, when the Sigma matrix is derived from the FGVC-Aircraft training dataset, yields higher classification accuracy than the standard Tucker2-ALS method for all tested ImageNet models. Notably, in the case of ResNet18, the Tucker2-ALS-Sigma variant surpasses even the fine-tuned model obtained with the standard algorithm.

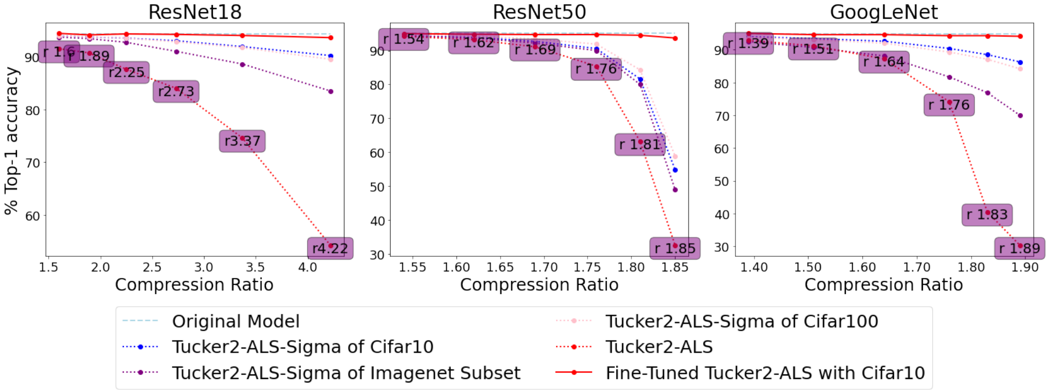

Figure 2: Accuracy comparison of Tucker2-ALS-Sigma and Tucker2-ALS algorithms including the fine-tuned model results after compression with Tucker2-ALS using Cifar10 dataset.

5.4 Evaluation Against Alternative Data-Informed Baselines

To rigorously evaluate our approach, we designed and implemented two strong, data-aware baseline methods inspired by metrics from pruning and activation analysis: Fisher information [Tu et al., 2016] and activation sparsity [Rhu et al., 2018]. Each baseline guides the Tucker2 decomposition by solving a specific Weighted Alternating Least Squares (WALS) problem, enabling a direct comparison against our proposed method.

Fisher-Weighted Low-Rank Approximation (FW-LRA) This baseline adapts Fisher information, a well-established criterion from pruning literature, to guide decomposition. The method solves the weighted objective $\min ||\mathbf{H}_{\text{fisher}} \circledast (\mathcal{K} - \widetilde{\mathcal{K}})||_F^2$, where the weighting tensor $\mathbf{H}_{\text{fisher}}$ is derived from the diagonal of the Fisher Information Matrix. The intuition is to prioritize the preservation of weights that are most sensitive to the network's loss function.

Activation-Guided Low-Rank Approximation (AG-LRA) Inspired by sparsity-based compression techniques, this baseline prioritizes filters based on their activation magnitude. It solves the objective $\min ||\mathbf{H}_{\text{act}} \circledast (\mathcal{K} - \widetilde{\mathcal{K}})||_F^2$, where the weighting tensor $\mathbf{H}_{\text{act}}$ is calculated from the mean absolute activation of each output channel. This heuristic assumes that filters with higher average activation are more critical for the network's predictions.

Comparison with Baselines Tables 3 and 4 summarize the accuracy of our Tucker2-ALS-Sigma method compared to the baseline Tucker2-ALS and the two data-aware variants on GoogLeNet and ResNet18, respectively. Across all compression rates, our method consistently outperforms both FW-LRA and AG-LRA, as well as the vanilla baseline Tucker2-ALS, demonstrating the effectiveness of our distribution-aware norm.

Table 3: Accuracy comparison of different data-aware baseline methods and Tucker2-ALS-Sigma with Tucker2-ALS at various compression rates on GoogLeNet.

Compr. Rate	Tucker2-ALS	AG-LRA	FW-LRA	Our Method
1.24	60.2	60.5	53.0	**66.3**
1.33	52.5	54.9	35.8	**64.2**
1.42	39.9	43.0	6.0	**60.1**
1.51	23.6	25.1	1.1	**52.2**

Table 4: Accuracy comparison of different data-aware baseline methods and Tucker2-ALS-Sigma with Tucker2-ALS at various compression rates on Resnet18.

Compr. Rate	Tucker2-ALS	AG-LRA	FW-LRA	Our Method
1.4	58.9	54.9	36.7	**66.8**
1.5	52.2	45.0	18.9	**66.1**
1.63	36.1	37.5	12.2	**64.9**
1.77	30.2	31.2	4.2	**63.3**

6 Limitations and Future Work

Complete functional norm A first limitation of our work is that when minimizing the reconstruction error in the layer l we only take into account the first part of the network in the functional norm. Indeed, we want to minimize $\|f - f'\|$ where $f = q \circ l_\theta \circ p$ and $f' = q \circ l'_{\theta'} \circ p$. To do so we minimize $\|l_\theta \circ p - l'_{\theta'} \circ p\|_{\mathrm{L}^2}$ as a proxy. This is an important improvement compared to the standard proxy $\|\theta - \theta'\|_F$ but this could be improved by taking the part q of the network into account in the optimization process.

Anisotropic VBMF To choose the rank of the decomposed tensors we use the Variational Bayesian Matrix Factorization (VBMF) algorithm. This algorithm is based on the assumption that the matrix we try to factorize are perturbed by a Gaussian isotropic noise. Similarly to replacing the Frobenius norm by the functional norm, we could use a more general assumption on the noise. Doing so would require to design a new algorithm to compute the VBMF rank but could lead to a better rank choice.

Performance guarantee In data-free compression, we show that our algorithm improves performance for all targeted compression ratios. However, we are not able to give a performance guarantee of the compressed network without testing it on a dataset. Hence to select the rank we rely on a priori heuristic like the VBMF rank. We believe that a possible future work would be to use the reconstruction error in the functional norm to select the rank. Indeed, preliminary experiments shown in the appendix K demonstrate that the reconstruction error in the functional norm is a way better indicator of the performance of the compressed network than the reconstruction error in the Frobenius norm.

7 Conclusion

In summary, we have shown that incorporating *distribution-aware* (functional) norms into tensor-based network compression leads to substantial performance gains. By deriving ALS procedures that directly optimize CP and Tucker decompositions under the Sigma norm, we achieve markedly lower reconstruction error and higher accuracy than with traditional Frobenius-based methods. In addition, CP-ALS-Sigma consistently surpasses greedy tensor deflation optimized with the same norm. The advantage of our approach becomes even more pronounced at higher compression rates, where standard methods degrade sharply. Remarkably, the distribution-aware decompositions recover almost all of the accuracy otherwise obtained by fine-tuning models compressed with conventional ALS, yet they require no additional training when the Σ matrix is estimated from the data distribution. Even when that distribution is learned from a smaller, dataset such as FGVC-Aircraft, the benefits persist. Finally, when the original training data are unavailable, the Σ matrix can be transferred from a related dataset, still delivering significant improvements—highlighting the practicality and robustness of our distribution-aware compression framework.

8 Acknowledgements

We thank Clément Laroudie and Charles Villard for helpful discussions that contributed to this work. This publication was made possible by the use of the FactoryIA supercomputer, financially supported by the Ile-de-France Regional Council. This work is also supported by the PEPR-IA : ANR-23-PEIA-0010 and DeepGreen : ANR-23-DEGR-0001.

References

Marcella Astrid and Seung-Ik Lee. Cp-decomposition with tensor power method for convolutional neural networks compression. In *2017 IEEE International Conference on Big Data and Smart Computing (BigComp)*, pages 115–118. IEEE, 2017.

Marcella Astrid, Seung-Ik Lee, and Beom-Su Seo. Rank selection of CP-decomposed convolutional layers with variational Bayesian matrix factorization. In *2018 20th International Conference on Advanced Communication Technology (ICACT)*, pages 347–350, February 2018. doi: 10.23919/ICACT.2018.8323750. URL `https://ieeexplore.ieee.org/document/8323750/`.

Yaohui Cai, Zhewei Yao, Zhen Dong, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. ZeroQ: A Novel Zero Shot Quantization Framework. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13169–13178, 2020.

Jian Cheng, Pei-song Wang, Gang Li, Qing-hao Hu, and Han-qing Lu. Recent advances in efficient computation of deep convolutional neural networks. *Frontiers of Information Technology & Electronic Engineering*, 19(1):64–77, January 2018. ISSN 2095-9230. doi: 10.1631/FITEE.1700789.

Daria Cherniuk, Stanislav Abukhovich, Anh-Huy Phan, Ivan Oseledets, Andrzej Cichocki, and Julia Gusak. Quantization Aware Factorization for Deep Neural Network Compression. *Journal of Artificial Intelligence Research*, 81:973–988, December 2024. ISSN 1076-9757. doi: 10.1613/jair.1.16167.

Remi 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 27*, pages 1269–1277. Curran Associates, Inc., 2014.

Shupeng Gui, Haotao Wang, Haichuan Yang, Chen Yu, Zhangyang Wang, and Ji Liu. Model compression with adversarial robustness: A unified optimization framework. *Advances in Neural Information Processing Systems*, 32, 2019.

Julia Gusak, Maksym Kholiavchenko, Evgeny Ponomarev, Larisa Markeeva, Philip Blagoveschensky, Andrzej Cichocki, and Ivan Oseledets. Automated multi-stage compression of neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pages 0–0, 2019.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *CVPR*, pages 770–778, 2016.

Frank L. Hitchcock. The Expression of a Tensor or a Polyadic as a Sum of Products. *Journal of Mathematics and Physics*, 6(1-4):164–189, 1927. ISSN 1467-9590. doi: 10.1002/sapm192761164.

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, February 2016.

Jean Kossaifi, Yannis Panagakis, Anima Anandkumar, and Maja Pantic. Tensorly: Tensor learning in python. *Journal of Machine Learning Research (JMLR)*, 20(26), 2019.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, May 2017. ISSN 0001-0782. doi: 10.1145/3065386.

Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan Oseledets, and Victor Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *ICLR*, 2015.

Yann LeCun, John Denker, and Sara Solla. Optimal Brain Damage. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2. Morgan-Kaufmann, 1989. URL `https://proceedings.neurips.cc/paper_files/paper/1989/file/6c9882bbac1c7093bd25041881277658-Paper.pdf`.

Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A ConvNet for the 2020s. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11976–11986, 2022.

Suhas Lohit and Michael Jones. Model Compression Using Optimal Transport. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 2764–2773, 2022.

Raphael Gontijo Lopes, Stefano Fenu, and Thad Starner. Data-Free Knowledge Distillation for Deep Neural Networks, November 2017.

Markus Nagel, Mart van Baalen, Tijmen Blankevoort, and Max Welling. Data-Free Quantization Through Weight Equalization and Bias Correction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1325–1334, 2019.

Shinichi Nakajima, Masashi Sugiyama, and Ryota Tomioka. Global analytic solution for variational Bayesian matrix factorization. *Advances in Neural Information Processing Systems*, 23, 2010.

Alexander Novikov, Dmitrii Podoprikhin, Anton Osokin, and Dmitry P Vetrov. Tensorizing Neural Networks. In *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

Yann Ollivier. True Asymptotic Natural Gradient Optimization, December 2017.

Minsoo Rhu, Mike O'Connor, Niladrish Chatterjee, Jeff Pool, Youngeun Kwon, and Stephen W. Keckler. Compressing DMA Engine: Leveraging Activation Sparsity for Training Deep Neural Networks. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 78–91, February 2018. doi: 10.1109/HPCA.2018.00017.

Nilo Schwencke and Cyril Furtlehner. ANaGRAM: A Natural Gradient Relative to Adapted Model for efficient PINNs learning. In *The Thirteenth International Conference on Learning Representations*, October 2024.

Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition, April 2015.

Suraj Srinivas and R. Venkatesh Babu. Data-free Parameter Pruning for Deep Neural Networks. In *Procedings of the British Machine Vision Conference 2015*, pages 31.1–31.12, Swansea, 2015. British Machine Vision Association. ISBN 978-1-901725-53-7. doi: 10.5244/C.29.31. URL `http://www.bmva.org/bmvc/2015/papers/paper031/index.html`.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going Deeper With Convolutions. In *CVPR*, pages 1–9, 2015.

Hidenori Tanaka, Daniel Kunin, Daniel L Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. In *Advances in Neural Information Processing Systems*, volume 33, pages 6377–6389. Curran Associates, Inc., 2020.

Jialiang Tang, Mingjin Liu, Ning Jiang, Huan Cai, Wenxin Yu, and Jinjia Zhou. Data-Free Network Pruning for Model Compression. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, May 2021. doi: 10.1109/ISCAS51556.2021.9401109.

Ming Tu, Visar Berisha, Yu Cao, and Jae-Sun Seo. Reducing the Model Order of Deep Neural Networks Using Information Theory. In *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 93–98, July 2016. doi: 10.1109/ISVLSI.2016.117.

Ledyard R. Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3): 279–311, September 1966. ISSN 1860-0980. doi: 10.1007/BF02289464.

Manon Verbockhaven, Théo Rudkiewicz, Guillaume Charpiat, and Sylvain Chevallier. Growing Tiny Networks: Spotting Expressivity Bottlenecks and Fixing Them Optimally. *Transactions on Machine Learning Research*, July 2024.

Sanghyun Woo, Shoubhik Debnath, Ronghang Hu, Xinlei Chen, Zhuang Liu, In So Kweon, and Saining Xie. ConvNeXt V2: Co-Designing and Scaling ConvNets With Masked Autoencoders. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16133–16142, 2023.

Qibin Zhao, Masashi Sugiyama, Longhao Yuan, and Andrzej Cichocki. Learning Efficient Tensor Representations with Ring-structured Networks. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8608–8612, May 2019. doi: 10.1109/ICASSP.2019.8682231.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction clearly state the scope and the contributions

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: There is a limitation section stating what we consider as constraints for the use of the methods we introduce.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification: We provide proofs for our proposition in the supplementary material.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We make clear how to reproduce our algorithms.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We provide in the supplementary material the code for the algorithm 2 Tucker2-ALS-Sigma as well as the code for the computation of the sigma matrix. These are the two essential blocks that correspond to our contribution, as a complete compression module would be too complex to include. These two blocks will be made publicly available on GitHub after paper acceptance: `https://github.com/alpkll/Parafac_Sigma`

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: Yes, we tried to put all important settings that help appreciate our experimental results.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: See the section F for the experiments with multiple random seeds.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Yes, we put these details on the supplementary material.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: Yes, all the authors confirm that the research conforms to the Neurips code of ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [No]

    Justification: Since our work focus on the development of theoretical and algorithmic methods on the neural network compression with tensor decomposition, we don't target specific application which can inherent the societal impact.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: We don't release data or models, we propose a method to compress a generic convolutional neural network.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: Yes, in the references.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: We do not release new assets.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: There is no such research in our paper.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: There is no such research in our paper.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.

- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [NA]

    Justification: We didn't use LLMs for any important or original part of our paper.

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.

A Proof on functional norm

Proposition 1. *Consider a distribution $\mathcal{D}$, a partial neural network p, and two convolution $\mathbf{Conv}_{\mathcal{K}}$ and $\mathbf{Conv}_{\widetilde{\mathcal{K}}}$ parametrized by the kernel tensor $\mathcal{K} \in \mathbb{R}^{T\times S\times H\times W}$ and $\widetilde{\mathcal{K}}$. Under reasonable assumptions* [§]*, we can define $\Sigma := \mathbb{E}_{x\sim\mathcal{D}}\big(u(p(x))u(p(x))^\top\big)$ where u is the unfolding operator* [¶] *that transforms the image $p(x)$ into a matrix that can be used to compute the convolution as a matrix product. We can also define $\Sigma^{1/2}$ the square root of Σ such that $\Sigma^{1/2}\left(\Sigma^{1/2}\right)^\top = \Sigma$. Then, we have:*

$$\left\|\mathbf{Conv}_{\mathcal{K}} \circ p - \mathbf{Conv}_{\widetilde{\mathcal{K}}} \circ p\right\|_{\mathsf{L}^2} = \left\|\left(\mathcal{K} - \widetilde{\mathcal{K}}\right)_{(1)} \Sigma^{1/2}\right\|_F \tag{12}$$

Proof. Convolution as a matrix product

We can write the convolution as a matrix product. We first need to unfold the image $p(x)$ of size $S \times H_y \times W_x$ into a matrix $u(p(x))$ of size $(S \times H \times W, H'_y \times W'_x)$. In this matrix each line is a patch of the image $p(x)$ of size $(S \times H \times W)$. We can also reshape the kernel $\mathcal{K}$ of size (T, S, H, W) into a matrix $\mathcal{K}_{(1)}$ of size $(T, S \times H \times W)$. The convolution can then be written as:

$$\mathbf{Conv}_{\mathcal{K}} \circ p(x) = \mathcal{K}_{(1)} u(p(x)) \in (T, H'_y \times W'_x). \tag{13}$$

Technical compuations Let x an input data, we have:

$$\begin{aligned}
\left\|\mathbf{Conv}_{\mathcal{K}} \circ p(x) - \mathbf{Conv}_{\widetilde{\mathcal{K}}} \circ p(x)\right\|_F^2 &= \left\|\mathcal{K}_{(1)} u(p(x)) - \widetilde{\mathcal{K}}_{(1)} u(p(x))\right\|_F^2 &(14)\\
&= \left\|\left(\mathcal{K} - \widetilde{\mathcal{K}}\right)_{(1)} u(p(x))\right\|_F^2 &(15)\\
&= \left\langle \left(\mathcal{K} - \widetilde{\mathcal{K}}\right)_{(1)} u(p(x)), \left(\mathcal{K} - \widetilde{\mathcal{K}}\right)_{(1)} u(p(x))\right\rangle. &(16)
\end{aligned}$$

Using the fact that $\langle u, vw\rangle = \left\langle uw^\top, v\right\rangle$, we have:

$$= \left\langle \left(\mathcal{K} - \widetilde{\mathcal{K}}\right)_{(1)} u(p(x))u(p(x))^\top, \left(\mathcal{K} - \widetilde{\mathcal{K}}\right)_{(1)}\right\rangle. \tag{17}$$

Taking the expectation Now we take the expectation over the data distribution $\mathcal{D}$:

$$\left\|\mathbf{Conv}_{\mathcal{K}} \circ p - \mathbf{Conv}_{\widetilde{\mathcal{K}}} \circ p\right\|_{\mathsf{L}^2}^2 = \mathbb{E}_{x\sim\mathcal{D}}\Big(\left\|\mathbf{Conv}_{\mathcal{K}} \circ p(x) - \mathbf{Conv}_{\widetilde{\mathcal{K}}} \circ p(x)\right\|_F^2\Big). \tag{18}$$

By linearity of the expectation and of the matrix multiplication and using (17), we have:

$$\begin{aligned}
&= \left\langle \left(\mathcal{K} - \widetilde{\mathcal{K}}\right)_{(1)} \mathbb{E}_{x\sim\mathcal{D}}\big(u(p(x))u(p(x))^\top\big), \left(\mathcal{K} - \widetilde{\mathcal{K}}\right)_{(1)}\right\rangle &(19)\\
&= \left\langle \left(\mathcal{K} - \widetilde{\mathcal{K}}\right)_{(1)} \Sigma, \left(\mathcal{K} - \widetilde{\mathcal{K}}\right)_{(1)}\right\rangle &(20)\\
&= \left\langle \left(\mathcal{K} - \widetilde{\mathcal{K}}\right)_{(1)} \Sigma^{1/2}\left(\Sigma^{1/2}\right)^\top, \left(\mathcal{K} - \widetilde{\mathcal{K}}\right)_{(1)}\right\rangle. &(21)
\end{aligned}$$

Using the fact that $\left\langle uw^\top, v\right\rangle = \langle u, vw\rangle$ we have:

$$\begin{aligned}
&= \left\langle \left(\mathcal{K} - \widetilde{\mathcal{K}}\right)_{(1)} \Sigma^{1/2}, \left(\mathcal{K} - \widetilde{\mathcal{K}}\right)_{(1)} \Sigma^{1/2}\right\rangle &(22)\\
&= \left\|\left(\mathcal{K} - \widetilde{\mathcal{K}}\right)_{(1)} \Sigma^{1/2}\right\|_F^2. &(23)
\end{aligned}$$

[§] The reasonable assumptions are that the function p is in L^2 for the distribution $\mathcal{D}$. This is likely the case in our range of applications since p is a neural network that is in most cases continuous and $\mathcal{D}$ can be considered as restricted to a compact set.

[¶] `https://docs.pytorch.org/docs/stable/generated/torch.nn.Unfold.html`

Hence, we have:

$$\left\|\mathbf{Conv}_{\mathcal{K}} \circ p - \mathbf{Conv}_{\widetilde{\mathcal{K}}} \circ p\right\|_{\mathsf{L}^2} = \left\|\left(\mathcal{K} - \widetilde{\mathcal{K}}\right)_{(1)} \Sigma^{1/2}\right\|_F . \tag{24}$$

□

B Details on ALS Algorithm with Distribution-Aware Norm

B.1 Computation of the Square Root of Σ

B.1.1 Definition and Existence

As we stated in the Proposition 1, we have $\Sigma := \mathbb{E}_{x\sim\mathcal{D}}\big(u(p(x))u(p(x))^\top\big)$. In this section, we show that the square root of the matrix Σ can be computed via its Cholesky decomposition. In this end, we need to show that the matrix $\mathbb{E}_{x\sim\mathcal{D}}\big(u(p(x))u(p(x))^\top\big)$ is a symmetric positive definite matrix.

First of all, note that Σ is symmetric:

$$\Sigma^\top = \left[\mathbb{E}_{x\sim\mathcal{D}}\big(u(p(x))u(p(x))^\top\big)\right]^\top = \mathbb{E}_{x\sim\mathcal{D}}\big([u(p(x))u(p(x))^\top]^\top\big) \tag{25}$$

$$= \mathbb{E}_{x\sim\mathcal{D}}\big(u(p(x))u(p(x))^\top\big) = \Sigma. \tag{26}$$

Moreover, Σ is a positive semi-definite matrix, since for any vector $z \in \mathbb{R}^n$, we have

$$z^\top \Sigma z = z^\top \mathbb{E}_{x\sim\mathcal{D}}\big(u(p(x))u(p(x))^\top\big) z \tag{27}$$

$$= \mathbb{E}_{x\sim\mathcal{D}}\big(z^\top u(p(x))u(p(x))^\top z\big) = \mathbb{E}_{x\sim\mathcal{D}}\Big(\left\|z^\top u(p(x))\right\|_F^2\Big) \geq 0. \tag{28}$$

To guarantee that Σ admits a Cholesky decomposition, it must be positive definite. This requires that the columns of the matrix $u(p(x))$ are linearly independent and span the space in which they lie.

Thus, the matrix Σ admits a Cholesky decomposition if and only if it is positive definite, which holds when the covariance of $u(p(x))$ is full rank. However, in practice, the columns of $u(p(x))$ can be linearly dependent, making Σ only positive semi-definite. To ensure numerical stability and enable the Cholesky decomposition, we add a small regularization term $\epsilon\mathrm{Id}$, where $\epsilon > 0$ and Id is the identity matrix, to the diagonal of Σ in cases where it is only positive semi-definite. Alternatively, the square root of Σ can be computed using the singular value decomposition (SVD), which remains applicable even when Σ is not full rank.

B.1.2 Complexity Analysis

Computation of Σ Let recall the notations, for the case where we want to compute Σ at layer l of a network:

- N the number of samples used to estimate Σ
- p the function corresponding to the first part of the network computing the layer l
- C the complexity of one forward pass of the full network
- $H_y[l+1], W_x[l+1]$ the height and width of layer $l+1$
- S the number of channels of layer l
- H, W the height and width of the kernel of the connection from layer l to layer $l+1$

To compute Σ for each x we compute:

- $p(x)$ which has a complexity bounded by C
- For each of the $H_y[l+1]W_x[l+1]$ patches $p_{i,j}$ of size $S \times H \times W$ in $p(x)$ (each patch corresponds to the receptive field of a neuron of layer $l+1$) we compute $(p_{i,j})^\top p_{i,j}$ which has a complexity of $(SHW)^2$ per patch. Then we sum all those matrices to get $u(p(x))\left(u(p(x))\right)^\top$. The complexity of this step is $H_y[l+1]W_x[l+1](SHW)^2$.

In total, the complexity of the computation of Σ is $O(NC + NH_y[l+1]W_x[l+1](SHW)^2)$. In comparison, the complexity of forwarding trought the connection from layer l to layer $l+1$ is $O(H_y[l+1]W_x[l+1]STHW)$ where T is the number of channels of layer $l+1$. Thus the computation of Σ is at most $\frac{SHW}{T}$ times the complexity of a classical forward (whith typical values $S \approx T$ and H, W small, the computation of Σ is comparable to the complexity of a few (9 for 3×3 kernels and 49 for the biggest 7×7 kernels) forward passes).

Computation of the Square Root of Σ Computing the square root of Σ can be done with Cholesky or SVD which require $O((SHW)^3)$ operations in both cases. This cost is independant of N thus as long as $N > SHW$ the cost of computing Σ is higher than the cost of computing its square root.

Conclusion In general, the computation of Σ and its square root is comparable to at most a few (9 for 3×3 kernels and 49 for the biggest 7×7 kernels) forward passes through the network times the number of samples used to estimate Σ.

B.2 Details of the CP-ALS-Sigma Algorithm

Remember that for a given rank R, CP decomposition of a kernel tensor $\mathcal{K} \in \mathbb{R}^{T \times S \times H \times W}$ is given by the following formula:

$$\widetilde{\mathcal{K}} = \sum_{r=1}^{R} \boldsymbol{U}_r^{(T)} \otimes \boldsymbol{U}_r^{(S)} \otimes \boldsymbol{U}_r^{(H)} \otimes \boldsymbol{U}_r^{(W)}. \tag{29}$$

By iterating over the components of the CP decomposition as given in the Section 4.2.1, we obtain the following sequence of minimization problems:

$$\min_{\boldsymbol{U}^{(T)}} \left\| \mathrm{Vec}(\mathcal{K}\Sigma^{1/2}) - \underbrace{\Big((\Sigma^{1/2})^\top (\boldsymbol{U}^{(S)} \odot \boldsymbol{U}^{(H)} \odot \boldsymbol{U}^{(W)}) \otimes \mathrm{Id}(T)\Big)}_{\boldsymbol{P}^{(T)}} \mathrm{Vec}(\boldsymbol{U}^{(T)}) \right\|_F , \tag{30}$$

$$\min_{\boldsymbol{U}^{(S)}} \left\| \mathrm{Vec}(\mathcal{K}\Sigma^{1/2}) - \underbrace{((\Sigma^{1/2})^\top \otimes \mathrm{Id}(T)) \Big[(\boldsymbol{U}^{(T)} \odot \boldsymbol{U}^{(W)} \odot \boldsymbol{U}^{(H)}) \otimes \mathrm{Id}(S)\Big]}_{=:\boldsymbol{P}^{(S)}} \mathrm{Vec}(\boldsymbol{U}^{(S)}) \right\|_F , \tag{31}$$

$$\min_{\boldsymbol{U}^{(H)}} \left\| \mathrm{Vec}(\mathcal{K}\Sigma^{1/2}) - \underbrace{((\Sigma^{1/2})^\top \otimes \mathrm{Id}(T)) \Big[(\boldsymbol{U}^{(T)} \odot \boldsymbol{U}^{(S)} \odot \boldsymbol{U}^{(W)}) \otimes \mathrm{Id}(H)\Big]}_{=:\boldsymbol{P}^{(H)}} \mathrm{Vec}(\boldsymbol{U}^{(H)}) \right\|_F , \tag{32}$$

$$\min_{\boldsymbol{U}^{(W)}} \left\| \mathrm{Vec}(\mathcal{K}\Sigma^{1/2}) - \underbrace{((\Sigma^{1/2})^\top \otimes \mathrm{Id}(T)) \Big[(\boldsymbol{U}^{(T)} \odot \boldsymbol{U}^{(S)} \odot \boldsymbol{U}^{(H)}) \otimes \mathrm{Id}(W)\Big]}_{=:\boldsymbol{P}^{(W)}} \mathrm{Vec}(\boldsymbol{U}^{(W)}) \right\|_F , \tag{33}$$

where we used the property $(\boldsymbol{A} \otimes \boldsymbol{C})(\boldsymbol{B} \otimes \boldsymbol{C}) = (\boldsymbol{AB} \otimes \boldsymbol{C})$ for (30). In consequence, the associated algorithm called CP-ALS-Sigma is described in the algorithm 1.

B.3 Details of the Tucker2-ALS-Sigma Algorithm

Remember that Tucker2 decomposition of $\mathcal{K} \in \mathbb{R}^{T\times S\times H\times W}$ is given by a core tensor $\mathcal{G} \in \mathbb{R}^{R_T\times R_S\times H\times W}$ and two factor matrices $\boldsymbol{U}^{(T)} \in \mathbb{R}^{T\times R_T}$ and $\boldsymbol{U}^{(S)} \in \mathbb{R}^{S\times R_S}$ such that

$$\widetilde{\mathcal{K}} = \mathcal{G} \times_2 \boldsymbol{U}^{(S)} \times_1 \boldsymbol{U}^{(T)}.$$[‖]

Similar to CP decomposition above, using the properties of tensor unfolding and vectorization of matrices, we end up with the following minimization problems:

$$\min_{\boldsymbol{U}^{(T)}} \left\| \mathrm{Vec}(\mathcal{K}\Sigma^{1/2}) - \underbrace{\left[\left((\Sigma^{1/2})^\top(\mathcal{G}\times_2 \boldsymbol{U}^{(S)})^\top_{(1)}\right)\otimes \mathrm{Id}(T)\right]}_{=:\boldsymbol{P}^{(T)}} \mathrm{Vec}(\boldsymbol{U}^{(T)}) \right\|_F , \tag{34}$$

$$\min_{\boldsymbol{U}^{(S)}} \left\| \mathrm{Vec}(\mathcal{K}\Sigma^{1/2}) - \underbrace{\left((\Sigma^{1/2})^\top \otimes \mathrm{Id}(T)\right)\left[(\mathcal{G}\times_1 \boldsymbol{U}^{(T)})^\top_{(2)}\otimes \mathrm{Id}(S)\right]}_{=:\boldsymbol{P}^{(S)}} \mathrm{Vec}(\boldsymbol{U}^{(S)}) \right\|_F , \tag{35}$$

and

$$\min_{\mathcal{G}} \left\| \mathrm{Vec}(\mathcal{K}\Sigma^{1/2}) - \underbrace{\left((\Sigma^{1/2})^\top \otimes \mathrm{Id}(T)\right)\left[\boldsymbol{U}^{(T)}\otimes \boldsymbol{U}^{(S)}\otimes \mathrm{Id}(H)\otimes \mathrm{Id}(W)\right]}_{=:\boldsymbol{P}^{(\mathcal{G})}} \mathrm{Vec}(\mathcal{G}) \right\|_F . \tag{36}$$

Thus, we associate the corresponding factor matrix $\boldsymbol{U}^{(T)}$ (and similarly for the factor matrix $\boldsymbol{U}^{(S)}$ and core tensor $\mathcal{G}$) with

$$\mathrm{Vec}(\boldsymbol{U}^{(T)}) \leftarrow (\boldsymbol{P}^{(T)})^\dagger\, \mathrm{Vec}(\mathcal{K}\Sigma^{1/2}). \tag{37}$$

Consequently, the full algorithm Tucker2-ALS-Sigma is presented in Algorithm 2.

B.4 Pseudo inverse computation for CP-ALS-Sigma and Tucker2-ALS-Sigma Algorithms

We refer the section 4.2.1 implementing Sigma norm on the CP-ALS algorithm, we obtain the closed form solution for the factor matrix $\boldsymbol{U}^{(T)}$ (and similarly for the other factors) with the following formula:

$$\mathrm{Vec}(\boldsymbol{U}^{(T)}) \leftarrow (\boldsymbol{P}^{(T)})^\dagger \mathrm{Vec}(\mathcal{K}\Sigma^{1/2}). \tag{38}$$

Since $\boldsymbol{P}^{(T)}$ is a matrix of size $TSHW \times RT$, we apply the identity $\boldsymbol{A}^\dagger = (\boldsymbol{A}^\top\boldsymbol{A})^\dagger\boldsymbol{A}^\top$ to avoid explicitly forming and storing large intermediate matrices of size $TSHW \times RT$, $TSHW \times RS$, $TSHW \times RH$, and $TSHW \times RW$, which would otherwise be required for computing the factor matrices $\boldsymbol{U}^{(T)}$, $\boldsymbol{U}^{(S)}$, $\boldsymbol{U}^{(H)}$, and $\boldsymbol{U}^{(W)}$, respectively.

Then, we can rewrite the above equation (38) as:

$$\mathrm{Vec}(\boldsymbol{U}^{(T)}) \leftarrow \left((\boldsymbol{P}^{(T)})^\top\boldsymbol{P}^{(T)}\right)^\dagger (\boldsymbol{P}^{(T)})^\top \mathrm{Vec}(\mathcal{K}\Sigma^{1/2}), \tag{39}$$

such that $(\boldsymbol{P}^{(T)})^\top\mathrm{Vec}(\mathcal{K}\Sigma^{1/2})$ is a vector of size RT and $(\boldsymbol{P}^{(T)})^\top\boldsymbol{P}^{(T)}$ is a matrix of size $RT\times RT$. However, in practice, we observed that the matrix $(\boldsymbol{P}^{(T)})^\top\boldsymbol{P}^{(T)}$ can be ill-conditioned, making the computation of its pseudoinverse numerically unstable. To mitigate this issue, we reformulate the problem as a linear system and solve it using the MINRES (Minimum Residual) method, which is particularly effective for solving symmetric but large-scale systems. Specifically, we define

$$b = (\boldsymbol{P}^{(T)})^\top\, \mathrm{Vec}(\mathcal{K}\Sigma^{1/2}), \quad A = (\boldsymbol{P}^{(T)})^\top\boldsymbol{P}^{(T)},$$

and solve the system

$$A\boldsymbol{U} = b,$$

[‖] $\times_i$ indicates a product along the ith axis of the tensor $\mathcal{G}$

where the solution vector $\boldsymbol{U}$ corresponds to the desired factor matrix. This system is solved efficiently on the GPU using the `minres` function provided by the CuPy library.

Likewise to CP decomposition, we solve the factor matrices $\boldsymbol{U}^{(T)}$ and $\boldsymbol{U}^{(S)}$, and the core tensor $\mathcal{G}$ of the Tucker2 decomposition using the the `minres` function provided by the CuPy library.

C Experiments with CP-ALS-Sigma Algorithm

C.1 Model Compression and Fine-Tuning with Limited Data Access

Refering the Section 5.2, we consider the case when only a limited amount of data is available, a common situation in many applications. Specifically, we consider the case where only 50,000 images from the ImageNet training set are available to estimate the matrix Σ and to fine-tune the model.

Here, we conducted the compression algorithms on ResNet18 and GoogLeNet, and we compare the classification accuracy of the compressed model with CP-ALS-Sigma, CP-ALS algorithms for Frobenius norm and its fine-tuned version. For the network compressed using the standard ALS algorithm under the Frobenius norm, we fine-tune it with the Adam optimizer, selecting the optimal learning rate from the range 10^{-5} to 10^{-10}. In addition, we selected the following values of α: $[0.8, 0.85, 0.9]$ for Resnet18, and $[0.85, 0.9, 0.95]$ for GoogLeNet.

According to the results on ResNet18, presented in Figure 3, the CP-ALS-Sigma algorithm consistently outperforms both the standard CP-ALS algorithm and the fine-tuned compressed models obtained with CP-ALS. In addition, our experiments on GoogLeNet (see Figure 3) show that the CP-ALS-Sigma algorithm achieves higher accuracy than the standard CP-ALS while yielding close results to the fine-tuned compressed model obtained via CP-ALS.

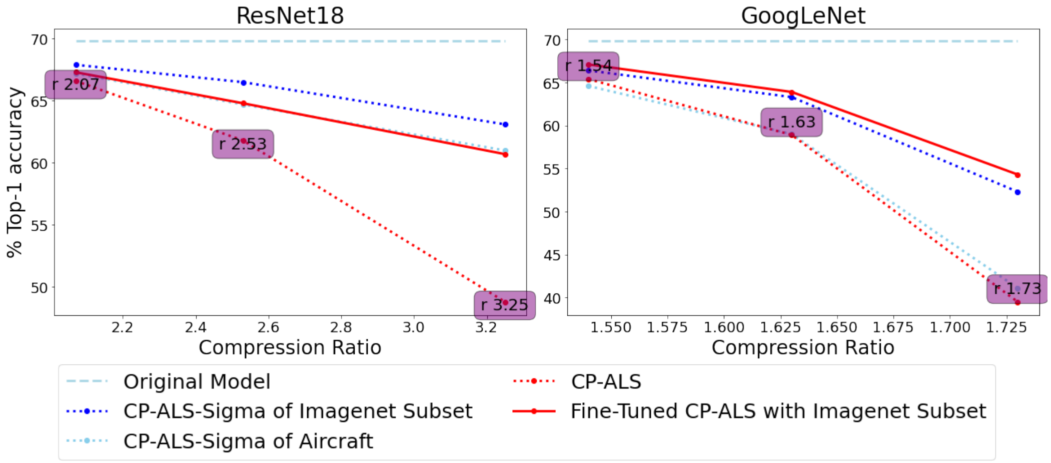

Figure 3: Accuracy comparison of decomposed models obtained with CP-ALS-Sigma and CP-ALS algorithms, also including fine-tuned decomposed model (with CP-ALS algorithm) results where fine-tuning done on the subset of ImageNet train dataset.

C.2 Impact of Dataset Changes on Proposed Algorithms

Here, we provide the detailed results for the CP-ALS-Sigma algorithm in the dataset transferability experiments described in Section 5.3. Specifically, we evaluate the compression performance of CP-ALS-Sigma on ResNet18, and GoogLeNet models trained on CIFAR-10, including Σ matrices derived from datasets different than the original training set. Our goal is to assess how the choice of dataset for computing the distribution-aware norm affects the final accuracy of the compressed models.

We compare CP-ALS-Sigma against the standard CP-ALS algorithm, and the fine-tuned compressed model obtained with CP-ALS. Fine-tuning was conducted on the CIFAR-10 training set using the Adam optimizer and a range of learning rates from 10^{-3} to 10^{-7}, selecting the best performing rate per model. In addition, we selected parameters α from $[0.85, 0.9, 0.95, 1]$ for Resnet18, and $[0.9, 0.95, 1]$ for GoogLeNet.

The results shown in Figure 4 indicate that the CP-ALS-Sigma algorithm consistently outperforms the standard CP-ALS algorithm across all models, even when the Σ matrix is computed from different datasets, such as a subset of the ImageNet training set or the CIFAR-100 training set. Additionally, as for Tucker2-ALS experiences, while CP-ALS experiences rapid performance degradation as the compression rate increases, our method remains much more consistent. Notably, the CP-ALS-Sigma algorithm achieves similar performance on the CIFAR-100 dataset as it does on CIFAR-10, indicating that the use of a distribution-aware norm for compression is not restricted to the original training dataset. However, when the Sigma matrix is derived from a subset of the ImageNet training set, the performance of the CP-ALS-Sigma algorithm is somewhat less effective compared to when it is computed on CIFAR-10 or CIFAR-100, due to differences in image resolution as observed in Tucker2-ALS-Sigma experiments, see the appendix G for further investigation of dataset features that affect algorithm performance. Moreover, as illustrated in Figure 4, using the CP-ALS-Sigma algorithm on ResNet18 results in only a 0.8% accuracy drop at a compression rate of 3.53, and a 1.7% drop at a compression rate of 5.22, relative to the original model accuracy of 94.3% where Σ is computed using the CIFAR-10 train dataset.

Furthermore, as illustrated in Figure 3, the compressed model obtained using the CP-ALS-Sigma algorithm, with the Sigma matrix computed from the FGVC-Aircraft training dataset, achieves close classification accuracy to the fine-tuned compressed model obtained through the standard CP-ALS algorithm applied to the original ResNet18.

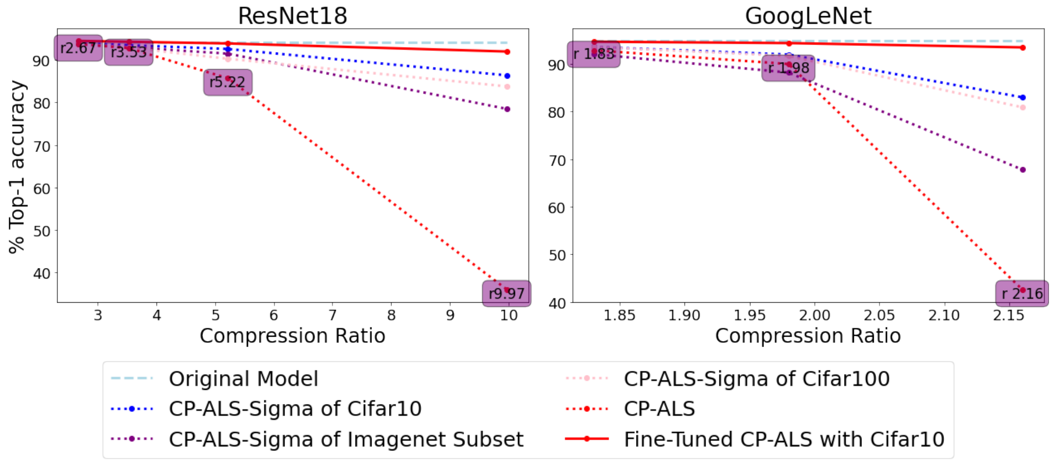

Figure 4: Comparison of CP-ALS-Sigma and CP-ALS algorithms including with fine-tuned model after compression with CP-ALS using CIFAR-10 dataset.

D Additional Experiments on the Impact of Dataset Changes Using CIFAR-100 Trained Models

Building on our previous experiments with CIFAR-10 trained neural network architectures (see the Section 5.3), we continue to assess how variations in the dataset influence compression outcomes and further demonstrate that the distribution-aware norm retains its transferability—even when computed from a dataset different from the one used for pretraining. Specifically, we compress ResNet18, ResNet50, and GoogLeNet models trained on CIFAR-100 using our ALS-Sigma methods—with the Σ matrix derived from datasets different from the original train dataset—and compare the results against those of the standard ALS approach under both CP and Tucker decomposition. In addition, we fine-tuned the compressed models that are obtained using the standard ALS algorithms to compare their performance with the Sigma-based methods (without fine-tuning). Fine-tuning was performed on the CIFAR-100 training dataset, employing the Adam optimizer and testing various learning rates from 10^{-3} to 10^{-7}, selecting the best-performing learning rate.

Tucker2-ALS-Sigma Firstly, we compare the results of Tucker2-ALS-Sigma with those of the standard Tucker2-ALS algorithm and fine-tuned compressed model obtained with Tucker2-ALS algorithm. We selected α from $[0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]$ for Resnet18, $[0.9, 1, 1.1, 1.2, 1.3]$ for Resnet50, $[0.6, 0.7, 0.8, 0.9, 0.95, 1]$ for GoogLeNet. The results, presented in Figure 5, confirm the earlier observations: the Tucker2-ALS-Sigma algorithm consistently outperforms the standard Tucker2-ALS method across all evaluated models. The observed performance difference remains evident even when the Σ matrix is derived from datasets other than the one used for training, such as a subset of ImageNet or the CIFAR-10 dataset.

Moreover, we verified again that while Tucker2-ALS causes to significant accuracy degradation as compression rate increases, the Tucker2-ALS-Sigma algorithm maintains significantly more consistent performance. Importantly, Tucker2-ALS-Sigma remains comparable in performance to fine-tuned Tucker2-ALS models—despite not involving any additional fine-tuning. These results reaffirm the transferability and effectiveness of our distribution-aware norm implementation under varying dataset conditions.

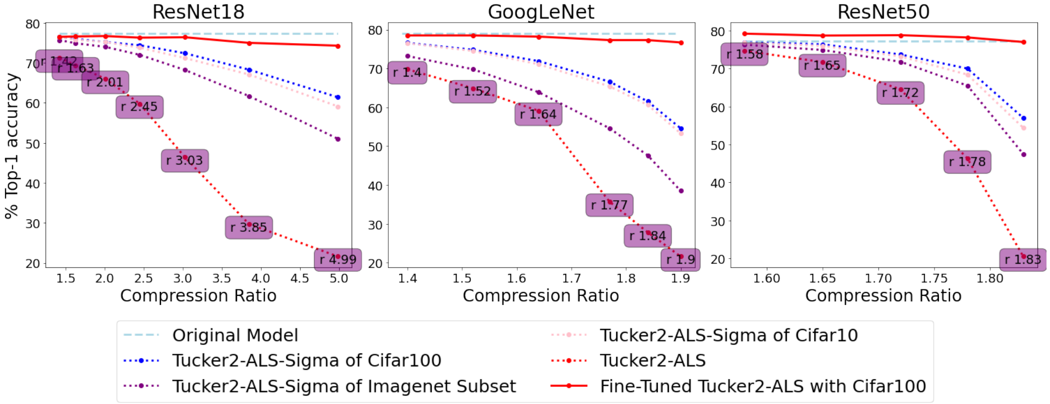

Figure 5: Accuracy comparison of decomposed models obtained using the Tucker2-ALS-Sigma and Tucker2-ALS algorithms across ResNet18, GoogLeNet, and ResNet50 architectures. The results also include fine-tuned decomposed models obtained via Tucker2-ALS, where fine-tuning was performed on the CIFAR-100 training dataset.

CP-ALS-Sigma Here, we present an accuracy comparison of decomposed models obtained using CP-ALS-Sigma, and the standard CP-ALS algorithm, and the fine-tuned compressed model obtained with CP-ALS. We selected α from $[0.85, 0.9, 0.95, 1]$ both for Resnet18, and GoogLeNet. The results shown in Figure 4 indicate that the CP-ALS-Sigma algorithm consistently outperforms the standard CP-ALS algorithm across all models, even when the Σ matrix is computed from different datasets, such as a subset of the ImageNet training set or the CIFAR-10 training set. Additionally, as for the Tucker2-ALS experiences, while CP-ALS experiences rapid performance degradation as the compression rate increases, our method remains much more consistent. Notably, we verified again that since the CP-ALS-Sigma algorithm achieves similar performance on the CIFAR-10 dataset as it does on CIFAR-100, the use of a distribution-aware norm for compression is not restricted to the original training dataset.

In addition, when the Sigma matrix is derived from a subset of the ImageNet training set, the performance of the ALS-Sigma algorithms are somewhat less effective compared to when it is computed on CIFAR-10 or CIFAR-100, likely due to differences in image resolution as observed in experiments performed on the CIFAR-10 trained models (see the Sections 5.3 and C.2).

E Additional Experiments on AlexNet

In this section, we evaluate our proposed algorithms on the AlexNet architecture, extending the compression to the fully-connected layers using Singular Value Decomposition (SVD).

First, we introduce **SVD-Sigma**, a method that adapts SVD to minimize error with respect to the Sigma norm. The goal is to find a low-rank approximation $\widetilde{\mathcal{W}}$ for a given weight matrix $\mathcal{W}$ by solving

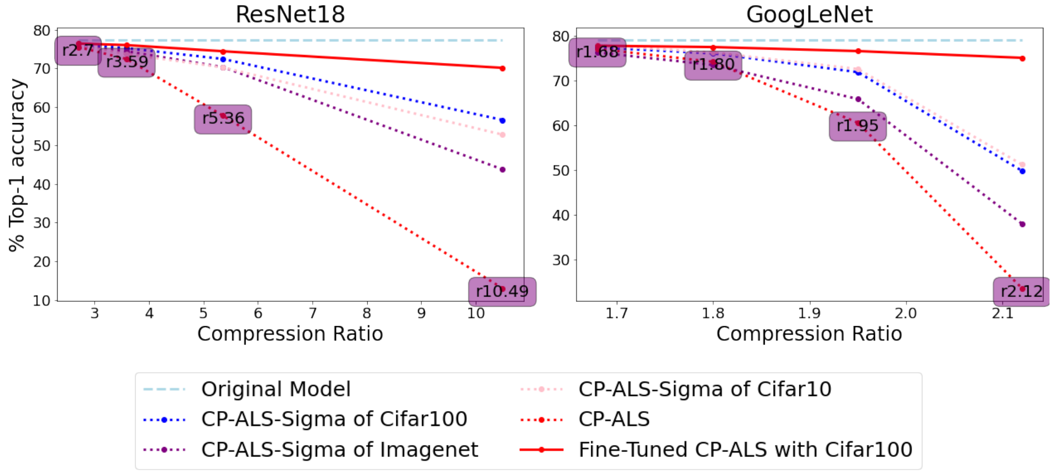

Figure 6: Accuracy comparison of decomposed models obtained using the CP-ALS-Sigma and CP-ALS algorithms across ResNet18 and GoogLeNet architectures. The results also include fine-tuned decomposed models obtained via CP-ALS, where fine-tuning was performed on the CIFAR-100 training dataset.

the following minimization problem:

$$\min_{\widetilde{\mathcal{W}}} \|(\mathcal{W} - \widetilde{\mathcal{W}})\Sigma^{1/2}\|_F. \tag{40}$$

We then compare the performance of two hybrid compression schemes in Table 5:

1. **Baseline Hybrid:** The convolutional layers are compressed with the standard **Tucker2-ALS** algorithm, while the linear layers are compressed with conventional **SVD**.
2. **Proposed Hybrid:** The convolutional layers are compressed with our **Tucker2-ALS-Sigma** algorithm, while the linear layers are compressed with the proposed **SVD-Sigma** method.

Table 5: Top-1 accuracies of the compressed AlexNet architecture using Tucker2-ALS followed by SVD, and Tucker2-ALS-Sigma followed by SVD-Sigma, where the Sigma matrix is computed using 5,000 images from the ImageNet training dataset. AlexNet original Top-1 accuracy is % 56.

VBMF Ratio	Compression Rate	Tucker2-ALS-Sigma + SVD-Sigma	Tucker2-ALS + SVD
0.6	1.46	49.8	42.4
0.65	1.63	48.1	40.1
0.7	1.85	46.1	34.4
0.8	2.54	40.9	25.8

F Stability of the Tensor Decomposition Methods

We examine the stability of our proposed CP-ALS-Sigma algorithm relative to the standard CP-ALS baseline. Experiments are performed on GoogLeNet trained on ImageNet, CIFAR-10, and CIFAR-100, each evaluated across multiple random seeds to measure the sensitivity of the decompositions to initialization. As shown in Tables 6, 7 and 8, CP-ALS is more sensitive to random initialization compared to CP-ALS-Sigma, particularly under high compression rates.

For Tucker2-ALS and Tucker2-ALS-Sigma, we adopt SVD-based initialization, which yields deterministic and thus fully reproducible results.

Table 6: Performance of CP-ALS and CP-ALS-Sigma on GoogLeNet (ImageNet) with the Σ matrix computed with a subset of ImageNet training dataset. Top-1 accuracy results are reported as mean $\pm$ standard deviation, obtained using 5 different random seeds.

VBMF Ratio	**Compr. Rate**	**CP-ALS**	**Σ of 50k-ImageNet**
0.85	1.54	65.85 ± 0.45	66.4 ± 0.07
0.9	1.63	59.5 ± 0.58	63.25 ± 0.05
0.95	1.73	37.43 ± 2	52.45 ± 0.12

Table 7: Performance of CP-ALS and CP-ALS-Sigma on GoogLeNet (trained on CIFAR-10) with the Σ matrix computed with CIFAR-10, CIFAR-100, and a subset of ImageNet training datasets. Top-1 accuracy results are reported as mean $\pm$ standard deviation, obtained using 5 different random seeds.

VBMF Ratio	**Compr. Rate**	**CP-ALS**	**Σ of 5k-ImageNet**	**Σ of CIFAR-100**	**Σ of CIFAR-10**
0.9	1.54	93.06 ± 0.15	92.58 ± 0.12	93.44 ± 0.09	93.53 ± 0.08
0.95	1.63	89.14 ± 0.62	88.31 ± 0.24	91.47 ± 0.06	91.88 ± 0.04
1	1.73	37.71 ± 3.0	69.86 ± 0.31	79.94 ± 0.24	82.22 ± 0.5

Table 8: Performance of CP-ALS and CP-ALS-Sigma on GoogLeNet (trained on CIFAR-100) with the Σ matrix computed with CIFAR-10, CIFAR-100, and a subset of ImageNet training datasets. Top-1 accuracy results are reported as mean $\pm$ standard deviation, obtained using 5 different random seeds.

VBMF Ratio	**Compr. Rate**	**CP-ALS**	**Σ of 5k-ImageNet**	**Σ of CIFAR-10**	**Σ of CIFAR-100**
0.85	1.68	76.95 ± 0.32	77.1 ± 0.08	77.7 ± 0.19	77.64 ± 0.19
0.9	1.80	73.46 ± 0.48	74.97 ± 0.11	76.41 ± 0.15	76.36 ± 0.10
0.95	1.95	61.4 ± 1.22	67.3 ± 0.38	72.54 ± 0.09	72.4 ± 0.17
1	2.12	21.25 ± 1.84	38.59 ± 0.45	51.28 ± 0.11	50.74 ± 0.17

G Impact of Dataset Properties on Algorithm Performance

G.1 Image Resolution

This section investigates the hypothesis that differences in image resolution are a primary factor in cross-dataset performance degradation. Our experiments confirm that algorithm performance is sensitive to resolution matching and the quality of the image scaling process.

First, we found that matching the proxy dataset's resolution (ImageNet) to the target dataset's resolution (CIFAR-10) via downsampling leads to a significant performance recovery. The results in Tables 9 and 10 show that using a Σ matrix computed on downsampled ImageNet images consistently outperforms one computed on original-resolution images.

Second, our method is sensitive to the choice of interpolation used during preprocessing. As shown in Table 11, replacing the default bilinear interpolation (used in other experiments) with bicubic interpolation during the image resizing operation yields better results, underscoring the importance of preserving feature quality during upsampling.

G.2 Dataset Diversity

This section evaluates the impact of dataset diversity on the performance of our proposed algorithms. Our findings indicate that the method is remarkably flexible regarding the specific semantic composition of the data used to compute the Σ matrix.

First, we found that broad dataset diversity is beneficial but not essential. The results in Table 12 confirm that a large-scale dataset like ImageNet yields the best performance. Nevertheless, performance remains strong even with smaller, more specialized datasets like FGVC-Aircraft or Oxford Flowers-102.

Table 9: Accuracy comparison of the Tucker2-ALS-Sigma algorithm using Sigma matrices computed from different datasets on a ResNet-18 model trained with CIFAR-10. The Sigma matrices were derived from the full CIFAR-10 and CIFAR-100 training sets, as well as 50,000 ImageNet training images. For the ImageNet (Down) and Aircraft (Down) variants, images were first downsampled to CIFAR-10 resolution ($3 \times 32 \times 32$) and then upsampled to ImageNet resolution ($3 \times 224 \times 224$) before computing the Sigma matrix, whereas for the other variants, images were only upsampled to ImageNet resolution.

VBMF Ratio	**Compr. Rate**	**Σ of CIFAR-10**	**Σ of CIFAR-100**	**Σ of ImageNet (Orig.)**	**Σ of ImageNet (Down)**	**Σ of Aircraft (Orig.)**	**Σ of Aircraft (Down)**
0.5	1.6	93.9	93.9	93.7	94	93.1	93.6
0.6	1.89	93.7	93.7	93.4	93.7	92.1	93.3
0.7	2.25	93.5	93.5	92.7	93.2	89.8	92.4
0.8	2.73	93	92.8	91	92.5	86.9	91.1
0.9	3.37	91.9	91.7	88.6	91	81.7	88.6
1	4.22	90.2	89.5	83.5	87.9	72	82.7

Table 10: Accuracy comparison of the Tucker2-ALS-Sigma algorithm using Sigma matrices computed from different datasets on a GoogleNet model trained with CIFAR-10. The Sigma matrices were derived from the full CIFAR-10 and CIFAR-100 training sets, as well as 5000 ImageNet training images.

VBMF Ratio	**Compr. Rate**	**Σ of CIFAR-10**	**Σ of CIFAR-100**	**Σ of 5k-ImageNet (Orig.)**	**Σ of 5k-ImageNet (Down)**
0.6	1.39	94	93.8	92.5	93.4
0.7	1.51	93.4	93.1	90.7	92.3
0.8	1.64	92.6	92	88.1	90.4
0.9	1.76	90.4	89.3	81.7	87
0.95	1.83	88.5	87.1	76.9	83.3
1	1.89	86.3	84.2	70	78.6

Table 11: Accuracy comparison of the Tucker2-ALS-Sigma algorithm showing the impact of upsampling quality when using low-resolution datasets for the computation of Sigma matrix on ResNet18 trained with ImageNet. The Sigma matrices were derived from the full CIFAR-10 and CIFAR-100 training sets.

VBMF Ratio	**Compr. Rate**	**Σ of CIFAR-10 (Bilinear)**	**Σ of CIFAR-10 (Bicubic)**	**Σ of CIFAR-100 (Bilinear)**	**Σ of CIFAR-100 (Bicubic)**
0.4	1.4	55.4	56.3	55.6	56.7
0.45	1.5	52.3	53.8	52.4	53.9
0.5	1.63	47.2	50.5	48.3	50.8
0.55	1.77	41.7	46.5	42	47

Furthermore, we investigated whether performance is sensitive to the specific theme of the proxy dataset. As detailed in Table 13, computing Σ matrix from a single broad category (e.g., 'dogs', 'reptiles', or 'aircraft') or from mix of diverse categories has minor impact on the performance, highlighting the method's robustness to the semantic composition of the proxy data.

Table 12: Accuracy comparison of the Tucker2-ALS-Sigma algorithm using Sigma matrices computed from different datasets on the original ResNet18 architecture. The Sigma matrices were derived from the FGVC-Aircraft and Oxford Flowers-102 training sets, and the 50.000 images of ImageNet.

VBMF Ratio	**Compr. Rate**	**Σ of 50k-ImageNet**	**Σ of Aircraft**	**Σ of Flowers-102**
0.4	1.4	66.3	64.2	64.6
0.45	1.5	65.2	62.4	63.2
0.5	1.63	63.9	59.9	61.6
0.55	1.77	61.9	56.5	59.1

Table 13: Accuracy comparison of the Tucker2-ALS-Sigma algorithm using Sigma matrices computed depending on the categories of ImageNet dataset on the original ResNet18 architecture. We selected the categories Aircraft, Dog, and Reptiles from ImageNet to compute the Sigma matrices and called by ImageNet-Aircraft, ImageNet-Dog, and ImageNet-Reptiles. For ImageNet-MixCateg, we have chosen images from 10 different categories of ImageNet.

VBMF Ratio	**Compr. Rate**	**Σ of ImageNet-Aircraft**	**Σ of ImageNet-Dog**	**Σ of ImageNet-Reptiles**	**Σ of ImageNet-MixCateg**
0.4	1.4	65.6	66.1	65.3	66.6
0.45	1.5	64.3	65	64.2	65.7
0.5	1.63	62.6	63.7	62.3	64.4
0.55	1.77	60	61.7	60.2	62.6

G.3 Sample Size

We next investigate the sensitivity of our method to the number of samples used to compute the Σ matrix. The results, presented in Tables 14 and 15, indicate that performance is remarkably stable across different sample sizes. This robustness is a key practical advantage, as it shows our algorithm does not require an excessively large dataset for effective covariance estimation.

Table 14: Performance of Tucker2-ALS-Sigma on ResNet-18 with the Σ matrix computed using varying sample sizes. Top-1 accuracy results are reported as mean ± standard deviation, with each sample subset chosen randomly over 5 different seeds.

VBMF Ratio	**Compr. Rate**	**Σ of 5k-ImageNet**	**Σ of 10k-ImageNet**	**Σ of 20k-ImageNet**	**Σ of 50k-ImageNet**
0.4	1.4	66.81 ± 0.02	66.80 ± 0.03	66.80 ± 0.02	66.80 ± 0.02
0.45	1.5	66.09 ± 0.03	66.11 ± 0.06	66.10 ± 0.02	66.08 ± 0.02
0.5	1.63	64.88 ± 0.03	64.90 ± 0.05	64.91 ± 0.02	64.89 ± 0.01
0.55	1.77	63.25 ± 0.05	63.29 ± 0.07	63.31 ± 0.03	63.30 ± 0.01

Table 15: Performance of Tucker2-ALS-Sigma on GoogLeNet with the Σ matrix computed using varying sample sizes. Top-1 accuracy results are reported as mean ± standard deviation, with each sample subset chosen randomly over 5 different seeds.

VBMF Ratio	**Compr. Rate**	**Σ of 5k-ImageNet**	**Σ of 10k-ImageNet**	**Σ of 20k-ImageNet**	**Σ of 50k-ImageNet**
0.6	1.33	64.17 ± 0.04	64.13 ± 0.05	64.16 ± 0.07	64.17 ± 0.03
0.7	1.42	60.04 ± 0.03	60.11 ± 0.06	60.13 ± 0.04	60.11 ± 0.04
0.8	1.51	52.21 ± 0.09	52.18 ± 0.08	52.20 ± 0.08	52.17 ± 0.05

H Synergy of Tensor Decomposition and Quantization

Tensor decomposition and quantization are two principal and complementary model compression techniques. While our factorization approach reduces model size by exploiting structural redundancy, quantization enhances efficiency by reducing numerical precision (e.g., to INT8) of model weights. Since these methods target different aspects of compression, combining them is expected to provide additive benefits, as shown in previous work [Gui et al., 2019].

To validate this idea further, we applied post-training FP16 and INT8 quantization to our compressed GoogLeNet (trained on CIFAR-10) models. The results in Table 16 confirm this strong synergy, particularly for our **Tucker2-ALS-Sigma** algorithm. Across all compression ratios, applying FP16 quantization to the factorized model results in a negligible change in accuracy. Even with more aggressive INT8 quantization, the performance remains remarkably high, demonstrating that our method is robust to a subsequent reduction in precision.

Table 16: Top-1 accuracies of GoogLeNet(trained on CIFAR-10) compressed with **Tucker2-ALS-Sigma** and **Tucker2-ALS**. The table compares the performance without quantization (FP32) against post-training quantization (FP16 and INT8) at various compression rates.

	Tucker2-ALS-Sigma			**Tucker2-ALS**		
Compr. Rate	**WO Quant.**	**W Quant.**		**WO Quant.**	**W Quant.**	
	FP32	**FP16**	**INT8**	**FP32**	**FP16**	**INT8**
1.39	93.97	93.94	93.65	92.81	92.82	92.47
1.51	93.43	93.39	93.19	91.16	91.18	91.24
1.64	92.69	92.68	92.29	87.66	87.68	88.38
1.76	90.65	90.68	90.02	76.43	76.34	77.03
1.83	88.77	88.79	88.19	43.57	43.56	44.31
1.89	86.73	86.73	85.91	33.73	33.71	34.33

While Tucker decomposition is generally robust to post-factorization quantization, the high sensitivity of Canonical Polyadic (CP) decomposition can lead to significant accuracy degradation. To address this challenge, Cherniuk et al. [2024] recently proposed a quantization-aware framework called ADMM-EPC. Their method uses a novel CP-EPC initialization to produce low-rank factors that are inherently robust to quantization, enabling the successful combination of CP decomposition and numerical precision reduction.

I Compute Resources

To ensure reproducibility, we detail the computational resources used for all experiments:

- **Hardware:** All CP-ALS and Tucker2-ALS experiments were executed exclusively on CPU, using a machine with an `x86_64` architecture and 1.5TB of RAM. In contrast, all CP-ALS-Sigma and Tucker2-ALS-Sigma experiments were performed on systems equipped with either an NVIDIA A100 GPU (80GB) or an NVIDIA H100 GPU (100GB), alongside the same CPU and memory configuration.
- **Software Environment:** We used PyTorch 2.6.0 and CUDA 12.4. Experiments were run on a Linux system with Python 3.10.14.
- **Execution Time:** The runtime of Tucker2 and CP decompositions varied depending on model size and compression ratio. Fine-tuning phases were run for 30 epochs, taking approximately 1–6 hours per model.

J Extended Background on Compression by Tensor Decomposition

We detail the implementation of CP and Tucker decompositions for convolutional layers, as introduced in Section 3. We recall the convolution $\mathbf{Conv}_{\mathcal{K}}$ parameterized by a tensor $\mathcal{K}$ of size (T, S, H, W), which defines a mapping from the space of images $\mathcal{X} \in \mathbb{R}^{S\times H_y\times W_x}$ to the space of images $\mathcal{Y} \in \mathbb{R}^{T\times H'_y\times W'_x}$. Assuming stride 1 and no padding, the output of the convolution at a given spatial location can be written as

$$\mathcal{Y}[t,y,x] = \sum_{s=1}^{S}\sum_{h=-h_d}^{h_d}\sum_{w=-w_d}^{w_d}\mathcal{K}[t,s,h,w]\mathcal{X}[s,y+h,x+w] \tag{41}$$

where $2h_d + 1 = H$ and $2w_d + 1 = W$, and the output dimensions are reduced accordingly: $H'_y = H_y - H + 1$ and $W'_x = W_x - W + 1$. For a detailed introduction to the CP and Tucker decomposition methods, we refer to [Lebedev et al., 2015]. We reproduce some of the key equations from that paper below.

J.1 CP Decomposition for Convolutional Layer Compression

Recall that for a rank R, CP decomposition of a kernel tensor $\mathcal{K} \in \mathbb{R}^{T\times S\times H\times W}$ is given by the following formula:

$$\widetilde{\mathcal{K}} = \sum_{r=1}^{R}\boldsymbol{U}_r^{(T)}\otimes\boldsymbol{U}_r^{(S)}\otimes\boldsymbol{U}_r^{(H)}\otimes\boldsymbol{U}_r^{(W)} \approx \mathcal{K} \tag{42}$$

such that it can be written as:

$$\widetilde{\mathcal{K}}[t,s,h,w] = \sum_{r=1}^{R}\boldsymbol{U}_r^{(T)}[t]\boldsymbol{U}_r^{(W)}[w]\boldsymbol{U}_r^{(H)}[h]\boldsymbol{U}_r^{(S)}[s]. \tag{43}$$

As proposed by Lebedev et al. [2015], we replace the original kernel tensor $\mathcal{K}$ with its CP decomposition $\widetilde{\mathcal{K}}$, thereby expressing the convolutional layer as a sequence of 4 successive layers we obtain the followings:

$$\mathcal{Y}[t,y,x] = \sum_{r=1}^{R}\sum_{w=-w_d}^{w_d}\sum_{h=-h_d}^{h_d}\sum_{s=1}^{S}U_r^{(T)}[t]U_r^{(W)}[w]U_r^{(H)}[h]U_r^{(S)}[s]\mathcal{X}[s,y+h,x+w] \tag{44}$$

$$= \underbrace{\sum_{r=1}^{R}U_r^{(T)}[t]\underbrace{\left[\sum_{w=-w_d}^{w_d}U_r^{(W)}[w]\underbrace{\left[\sum_{h=-h_d}^{h_d}U_r^{(H)}[h]\underbrace{\left[\sum_{s=1}^{S}U_r^{(S)}[s]\mathcal{X}[s,y+h,x+w]\right]}_{1\times1\text{ conv}}\right]}_{\text{depthwise conv}}\right]}_{\text{depthwise conv}}}_{1\times1\text{ convolution}}. \tag{45}$$

J.2 Tucker Decomposition for Convolutional Layer Compression

Remember that given the ranks R_T and R_S corresponding to the mode-1 and mode-2 unfoldings (i.e., the first and second axes) of the kernel tensor $\mathcal{K} \in \mathbb{R}^{T\times S\times H\times W}$, the Tucker2 decomposition of $\mathcal{K}$ is

$$\widetilde{\mathcal{K}} = \mathcal{G}\times_1\boldsymbol{U}^{(T)}\times_2\boldsymbol{U}^{(S)}. \tag{46}$$

Accordingly, the elementwise representation of the decomposed kernel at a given spatial location is given by:

$$\widetilde{\mathcal{K}}[t,s,h,w] = \sum_{r_s=1}^{R_S}\sum_{r_t=1}^{R_T}\mathcal{G}[r_t,r_s,h,w]\boldsymbol{U}^{(T)}[r_t,t]\boldsymbol{U}^{(S)}[r_s,s]. \tag{47}$$

Similarly, as suggested by Kim et al. [2016], this leads to replace the convolution by a series of the three layers: a 1×1 convolution parameterized by $\boldsymbol{U}^{(T)}$, a full convolution parameterized by $\mathcal{G}$ and a 1×1 convolution parameterized by $\boldsymbol{U}^{(S)}$. Specifically, this can be expressed as:

$$\mathcal{Y}[t,y,x] = \sum_{s=1}^{S}\sum_{h=-h_d}^{h_d}\sum_{w=-w_d}^{w_d}\sum_{r_s=1}^{R_S}\sum_{r_t=1}^{R_T}\mathcal{G}[r_t,r_s,h,w]\boldsymbol{U}^{(T)}[r_t,t]\boldsymbol{U}^{(S)}[r_s,s]\mathcal{X}[s,y+h,x+w]$$

$$= \underbrace{\sum_{r_t=1}^{R_T}\boldsymbol{U}^{(T)}[t,r_t]\left[\underbrace{\sum_{h=-h_d}^{h_d}\sum_{w=-w_d}^{w_d}\sum_{r_s=1}^{R_S}\mathcal{G}[r_t,r_s,h,w]\left[\underbrace{\sum_{s=1}^{S}\boldsymbol{U}^{(S)}[s,r_s]\mathcal{X}[s,h+y,w+x]}_{1\times 1\text{ conv}}\right]}_{H\times W\text{ conv}}\right]}_{1\times 1\text{ conv}}.$$

K Correlation Between the Reconstruction Error and Accuracy

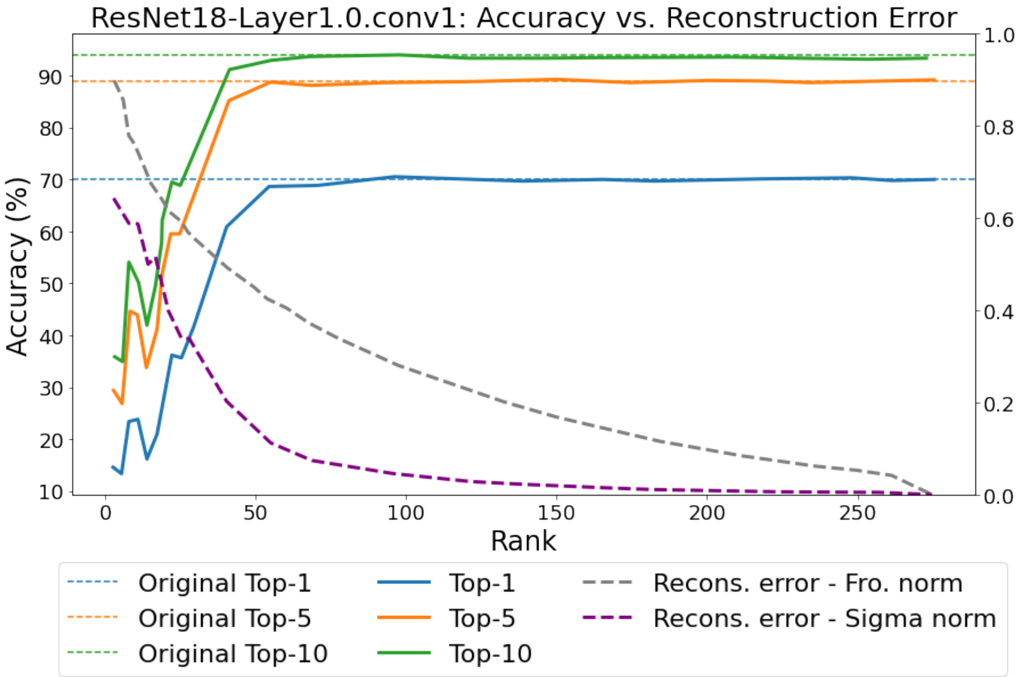

Figure 7: Relative reconstruction error and accuracy with respect to rank of the CP-decomposition of the second convolution of ResNet18.

In this section we explore the link between reconstruction error and accuracy. To do this we took a neural network trained on ImageNet and we decomposed one layer with the CP decomposition at different ranks. For each compression rank we computed the relative reconstruction error in Frobenius norm and distribution-aware norm. We also computed the accuracy of the model with the decomposed layer. The results are shown in Figure 7. We see that the accuracy of the model is not changing linearly with the rank, instead it first increases rapidly and then saturates. The Frobenius reconstruction error is more smooth and does not show a clear drop in reconstruction error when the accuracy increases. The distribution-aware reconstruction error is a bit chaotic for low-ranks. This is due to the fact that the optimisation is not done for this norm but for the Frobenius norm, hence we don't have the guarantee that the distribution-aware reconstruction error will decrease when the rank increases. We observe that the distribution-aware reconstruction error is inversely correlated with the accuracy of the model. Indeed, for the low ranks, the distribution-aware reconstruction error rapidly decreases until the accuracy saturates. After that, the distribution-aware reconstruction error is still decreasing slowly while the accuracy is not changing. This shows that the distribution-aware reconstruction error is a good metric to measure the quality of the decomposition and that we could use it to choose the rank of the decomposition without having to compute the accuracy of the model.

The behavior observed in Figure 7 is representative of what we consistently observed for the first layers across all the networks tested. In addition, for the last layers we still have similar structure for the accuracy and the reconstruction error in sigma-norm but the Frobenius norm tends to be way more aligned with the sigma-norm.