Hybrid network compression through tensor decompositions and pruning

Van Tien Pham, Yassine Zniyed, Thanh Phuong Nguyen

Hybrid network compression through tensor decompositions and pruning

Van Tien Pham, Yassine Zniyed, Thanh Phuong Nguyen
Université de Toulon, Aix Marseille Université, CNRS, LIS, UMR 7020, France
van-tien-pham@etud.univ-tln.fr, zniyed@univ-tln.fr, tpnguyen@univ-tln.fr

***Abstract*—The application of network compression methods, combining tensor decompositions and pruning, has demonstrated significant potential in harnessing the benefits of both strategies. This research introduces NORTON (hybrid Network cOmpression thRough Tensor decompOsitions and pruNing), a novel hybrid approach for network compression. The key innovation lies in the introduction of filter decomposition, enhancing the detailed breakdown of the network while preserving the multidimensional properties of filters. Our method incorporates a structured pruning approach, seamlessly integrating the decomposed model. Through a comprehensive set of experiments across various architectures, benchmark datasets, and representative vision tasks, the efficacy of our approach is highlighted.**

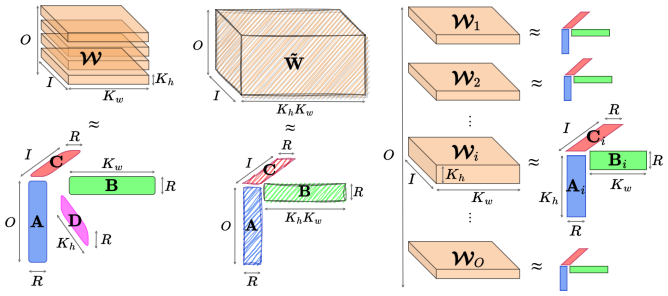
***Index Terms*—tensor decompositions, filter pruning, hybrid network compression, inference acceleration**

I. Introduction

The goal of network compression is to minimize the computational and memory demands of an existing model, making it suitable for deployment in resource-limited environments without compromising performance. Tensor decompositions [1], [2] and structured pruning [3] have proven effective and practical within the realm of model compression. Both strategies operate under the assumption that the original model is over-parameterized, allowing for the removal of redundant information either by optimizing weight representation through low-rank methods or by directly eliminating portions of weights through filter pruning. Beyond their high compression rates, these approaches share the advantage of facilitating the deployment of compressed models on devices with limited resources, without the need for specialized support. However, it is important to note that these techniques have largely evolved independently in the literature, with only a few attempts [4], [5] to explore their combined potential and leverage their individual strengths synergistically. This underscores the necessity for an integrated approach that capitalizes on the advantages offered by both methods.

CNN weights possess both low-rank and sparse characteristics [6], [7], which are partly complementary. Existing decomposition techniques do not completely eliminate all redundant channels [7], and post-pruning might inadvertently ignore low-rank structures. Therefore, there is a natural inclination to integrate these two compression strategies to improve network compression [6]. For instance, in the context of the VGG-16 architecture, RGP, a state-of-the-art filter pruning method [8], achieved a maximal pruning of 90.5% of MACs, while HALOC, representing the state-of-the-art in tensor decompositions [9], compressed a maximum of 86% of MACs. Notably, no existing work has effectively compressed more than 91% of MACs for this architecture. In contrast, NORTON achieves a simultaneous compression of 99% of MACs and parameters by combining both approaches, surpassing previous limitations.

Prior works on tensor decomposition have predominantly centered around the decomposition method itself [1], [2], [10] and the selection of ranks [11]. However, a relatively unexplored aspect in the existing literature revolves around the direct decomposition of the weight tensor. Previous studies have primarily concentrated on decomposing the entire layer as a 4-order tensor [1], [10] or reshaping it into a 3-order tensor before applying decomposition [2]. Yet, limited discussion exists on determining the most effective approach for decomposing the weight tensor. The weight of a convolution layer is a 4-order tensor, presenting various processing formats, each with distinct consequences. In Fig. 1, we illustrate three possible methods for handling this weight tensor, including layer decomposition [1], [10] and reshaped-based decomposition [2], [11], while our proposed approach is referred to as *filters decomposition*. Taking the Canonical Polyadic decomposition (CPD) [12] as a representative example, we compare the differences among these approaches. The approach presented by [1] involves decomposing the entire layer, resulting in 4 factor matrices corresponding to 4 sublayers. Although this method preserves the multidimensional nature of the weight tensor, it processes the tensor at a coarse level, opening up the potential for a more fine-grained treatment. In contrast, in [2], the authors suggest reshaping the 4-order weight tensor into a 3-order tensor, followed by CPD to obtain 3 factor matrices and 3 sublayers. While justified by deeming the kernel size small enough to ignore, this compromises the multidimensionality of the weight tensor, leading to information loss, especially in modern architectures with larger kernel sizes. Our proposed approach operates on the weight tensor in a filter-by-filter manner. The fundamental insight stems from the convolution layer's nature, where the input undergoes convolution independently with each filter, and the resulting outputs are aggregated to generate the final feature map. Hence, it becomes intuitive to decompose each 3-order filter tensor individually. The filters decomposition approach provides a finer granularity compared to its counterpart, layer decomposition. With filters decomposition, not only is the multidimensional property strictly preserved, but the layer's 4-order weight is naturally interpreted as a set of 3-order filters.



(a) 4-order layer [1]
✗more fine-grained
✓preserved structure
✗simpler sublayers

(b) 3-order reshaped layer [2]
✗more fine-grained
✗preserved structure
✓simpler sublayers

(c) 3-order filters (**ours**)
✓more fine-grained
✓preserved structure
✓simpler sublayers

Fig. 1: Comparison of tensor-based approaches.

Additionally, an intriguing outcome of filters decomposition is a narrower range of ranks, simplifying the subsequent rank selection process, as demonstrated later in subsection II-A. Another notable distinction is that filters decomposition replaces the original layer with 3 sublayers, while layer decomposition necessitates 4 sublayers. This disparity may potentially increase network depth unnecessarily and introduce the risk of gradient vanishing issues.

The second consideration of this study is to synergize filters decomposition and filter pruning, aiming to harness the independent advantages of each method. While prior investigations [4], [7], [13] have employed low-rank representations and network pruning for compression, they have not explored an orthogonal combination of these techniques. In these instances, low-rank representations have been solely utilized in the pruning step without directly contributing to the reduction of the model size. In contrast, our approach takes a divergent approach by sequentially applying low-rank representations and pruning in two distinct phases, facilitating a dual compression process. To the best of our knowledge, this direction has not been extensively explored in the literature.

Two potential arrangements for combination exist: decomposing then pruning and pruning then decomposing. Previous works [4], [5] have favored the pruning-then-decomposing scheme, employing a Taylor expansion-based pruning criterion and Tucker decomposition [12]. However, this approach [4] lacks comprehensive analysis and experiments, leaving room for further investigation. To address this gap, our study delves into the decomposing-then-pruning scheme, wherein the model is initially decomposed using CPD and then subjected to filter pruning, as illustrated in Fig. 2. This order presents greater challenges compared to its counterpart, as the model's architecture becomes more complex after decomposition, imposing specific constraints on the sublayers. To adapt to the decomposed components, we propose using Principal Angles Between Subspaces (PABS) [14] as a suitable filter pruning metric. This work contributes in the following ways:

- Firstly, we introduce a novel filters decomposition method, distinguishing it from existing layer decomposition and reshaped decomposition methods.
- Secondly, we explore the sequential combination of filters decomposition and filter pruning, proposing a novel filter pruning algorithm tailored to address the challenges associated with this integration scheme.
- Third, we evaluate the proposed method on representative vision tasks, benchmarking it against SOTA in low-rank representations, structured pruning, and hybrid domain to demonstrate its efficacy.

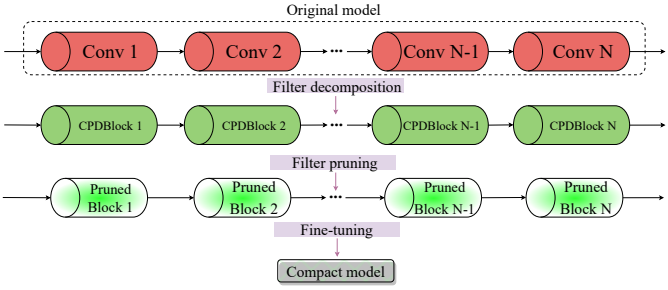


Fig. 2: Graphic illustration of the NORTON approach.

II. NORTON Approach

Figure 2 provides an overview of our approach, encompassing two primary phases: decomposition and pruning. Initially, the original model undergoes decomposition into CPDBlocks (as detailed in Subsection II-A). Subsequently, the decomposed CPDBlocks are subjected to the filter pruning algorithm (outlined in Subsection II-B). This algorithm selectively removes filters from the CPDBlocks, effectively reducing computational and memory requirements. Finally, a fine-tuning process refines the compact model. This approach is simultaneously applied to all convolution layers of the original model. However, for clarity, subsequent sections will concentrate on discussing one layer, as illustrated in Figure 3.

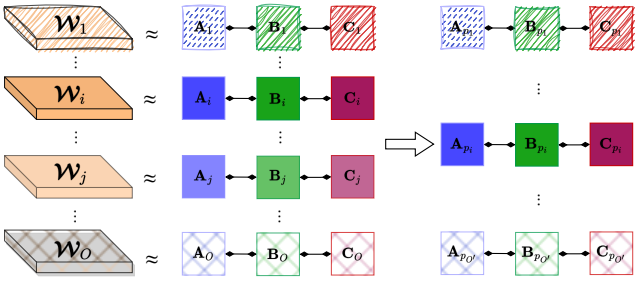


Fig. 3: The decomposition then pruning process for one layer.

A. Filters Decomposition Using the CPD

Consider a convolutional layer with the weight tensor $\mathcal{W} \in \mathbb{R}^{K_h \times K_w \times I \times O}$, where I and O represent the number of input and output channels, and K_h and K_w represent the kernel size. $\mathcal{W}$ can be viewed as a set of O individual 3-order filters denoted as $\{\mathcal{W}_1, \mathcal{W}_2, \ldots, \mathcal{W}_O\}$. These weights operate on an input tensor $\mathcal{I} \in \mathbb{R}^{H_{in} \times W_{in} \times I}$ to produce an

output tensor $\boldsymbol{\mathcal{O}} \in \mathbb{R}^{H_{out} \times W_{out} \times O}$, where H_{in}, W_{in}, H_{out}, and W_{out} denote the height and width of the input and output tensors, respectively. The convolution is given as:

$$\boldsymbol{\mathcal{O}}_k(i,j) = \sum_{m=0}^{K_h-1} \sum_{n=0}^{K_w-1} \sum_{p=0}^{I-1} \boldsymbol{\mathcal{I}}(i+m, j+n, p) \cdot \boldsymbol{\mathcal{W}}_k(m,n,p), \tag{1}$$

where, for $0 \leq k \leq O-1$, $\boldsymbol{\mathcal{O}}_k = \boldsymbol{\mathcal{O}}_{:,:,k}$, and is of size $H_{out} \times W_{out}$. Based on (1) and the CPD definition in [12], we can apply the CPD to each individual filter $\boldsymbol{\mathcal{W}}_k$ in order to obtain a compact representation:

$$\boldsymbol{\mathcal{W}}_k(m,n,p) = \sum_{r=0}^{R-1} \mathbf{A}_k(m,r) \cdot \mathbf{B}_k(n,r) \cdot \mathbf{C}_k(p,r), \tag{2}$$

where $\mathbf{A}_k$, $\mathbf{B}_k$ and $\mathbf{C}_k$ are 3 factor matrices of size $K_h \times R$, $K_w \times R$ and $I \times R$, respectively. This approximation is graphically represented in the left half of Fig. 3.

By substituting (2) into (1), we obtain a new CPD-based approach to compute the convolution. This approach involves a sequence of mappings using the factor matrices instead of high-order tensors, resulting:

$$\begin{aligned} \boldsymbol{\mathcal{O}}_k(i,j) = \sum_{m=0}^{K_h-1} \sum_{n=0}^{K_w-1} \sum_{p=0}^{I-1} \sum_{r=0}^{R-1} \boldsymbol{\mathcal{I}}(i+m, j+n, p) \cdot \\ \mathbf{A}_k(m,r) \cdot \mathbf{B}_k(n,r) \cdot \mathbf{C}_k(p,r). \end{aligned} \tag{3}$$

Starting from (3), we observe that the CPD-based convolution involves element-wise multiplications between the input tensor $\boldsymbol{\mathcal{I}}$ and the factor matrices $\mathbf{A}_k$, $\mathbf{B}_k$, and $\mathbf{C}_k$. It is important to note that the order of the convolutions can be rearranged without affecting the final result. This flexibility allows us to describe the computation as a sequential block of convolutions with smaller kernels, followed by a summation:

$$\boldsymbol{\mathcal{O}}^{\mathbf{C}}_k(i+m, j+n, r) = \sum_{p=0}^{I-1} \boldsymbol{\mathcal{I}}(i+m, j+n, p) \cdot \mathbf{C}_k(p,r), \tag{4}$$

$$\boldsymbol{\mathcal{O}}^{\mathbf{B}}_k(i+m, j, r) = \sum_{n=0}^{K_w-1} \boldsymbol{\mathcal{O}}^{\mathbf{C}}_k(i+m, j+n, r) \cdot \mathbf{B}_k(n,r), \tag{5}$$

$$\boldsymbol{\mathcal{O}}^{\mathbf{A}}_k(i,j,r) = \sum_{m=0}^{K_h-1} \boldsymbol{\mathcal{O}}^{\mathbf{B}}_k(i+m, j, r) \cdot \mathbf{A}_k(m,r), \tag{6}$$

$$\boldsymbol{\mathcal{O}}_k(i,j) = \sum_{r=0}^{R-1} \boldsymbol{\mathcal{O}}^{\mathbf{A}}_k(i,j,r), \tag{7}$$

where $\boldsymbol{\mathcal{O}}^{\mathbf{C}}_k \in \mathbb{R}^{H_{in} \times W_{in} \times R}$, $\boldsymbol{\mathcal{O}}^{\mathbf{B}}_k \in \mathbb{R}^{H_{in} \times W_{out} \times R}$, and $\boldsymbol{\mathcal{O}}^{\mathbf{A}}_k \in \mathbb{R}^{H_{out} \times W_{out} \times R}$.

One should note that equations (4), (5), and (6) can be seen as convolutions and can be implemented using common deep learning frameworks. Specifically, equation (4) can be computed using a classical 2D convolution operation, while equations (5) and (6) can be computed via group convolutions. One can refer to Fig. 4, which illustrates the structure of the CPDBlock. To ensure compatibility with classical frameworks, certain preprocessing operations including reshaping and mode permutations are required for adapting the kernel. Specifically, for the O factors $\mathbf{C}_k$ of dimensions $I \times R$, they need to be reshaped into a kernel of size $1 \times 1 \times I \times (R \cdot O)$. Additionally, for the group convolutions in equations (5) and (6), the kernels should be remodeled as $1 \times K_w \times 1 \times (R \cdot O)$ and $K_h \times 1 \times 1 \times (R \cdot O)$, respectively. By this preprocessing, the CPDBlock can be seamlessly integrated into existing deep learning frameworks.

The choice of rank R in CPD plays a crucial role in balancing model compression and accuracy. Kruskal's theory provides a weak upper bound on the maximum rank [12], expressed as:

$$R \leq \min\{I \cdot K_h, I \cdot K_w, K_h \cdot K_w\}. \tag{8}$$

This upper bound offers a guideline for selecting an appropriate rank, ensuring a reasonable trade-off between model compression and preservation of critical features. By fixing a rank R, the CPDBlock achieves a significant reduction in the number of parameters compared to the original layer, from $O \cdot I \cdot K_h \cdot K_w$ to $O \cdot R \cdot (I + K_h + K_w)$. It also notably reduces the computational complexity from $\mathcal{O}(I \cdot O \cdot K_h \cdot K_w \cdot H \cdot W)$ to $\mathcal{O}(R \cdot O \cdot (I + K_h + K_w) \cdot H \cdot W)$.

B. CPDBlock Pruning

First, it's crucial to recognize that each output $\boldsymbol{\mathcal{O}}_k$ is computed based on three factor matrices: $\mathbf{A}_k$, $\mathbf{B}_k$, and $\mathbf{C}_k$. Hence, when pruning the kernel associated with a specific output, the removal of all three matrices must be considered. This requires employing a pruning criterion that accounts for the interdependencies among these three matrices. Second, it's noteworthy that when the CPD is unique, it is unique up to scaling and permutation ambiguities. In other words, if two 3-order filters, $\boldsymbol{\mathcal{W}}_i$ and $\boldsymbol{\mathcal{W}}_j$, are strictly similar and satisfy the uniqueness conditions of the CPD [12], then

$$\begin{cases} \boldsymbol{\mathcal{W}}_i = [\![\mathbf{A}_i, \mathbf{B}_i, \mathbf{C}_i]\!], \\ \boldsymbol{\mathcal{W}}_j = [\![\mathbf{A}_j, \mathbf{B}_j, \mathbf{C}_j]\!], \\ \boldsymbol{\mathcal{W}}_i = \boldsymbol{\mathcal{W}}_j. \end{cases} \nRightarrow \begin{cases} \mathbf{A}_i = \mathbf{A}_j, \\ \mathbf{B}_i = \mathbf{B}_j, \\ \mathbf{C}_i = \mathbf{C}_j, \end{cases} \tag{9}$$

where $[\![\mathbf{A}_i, \mathbf{B}_i, \mathbf{C}_i]\!]$ is the compact representation of the CPD of $\boldsymbol{\mathcal{W}}_i$. Instead, we have $\mathbf{A}_i = \mathbf{A}_j \boldsymbol{\Pi} \boldsymbol{\Lambda}^A$, $\mathbf{B}_i = \mathbf{B}_j \boldsymbol{\Pi} \boldsymbol{\Lambda}^B$, and $\mathbf{C}_i = \mathbf{C}_j \boldsymbol{\Pi} \boldsymbol{\Lambda}^C$, where $\boldsymbol{\Pi}$ is a permutation matrix, and the diagonal scaling matrices satisfy $\boldsymbol{\Lambda}^A \boldsymbol{\Lambda}^B \boldsymbol{\Lambda}^C = \mathbf{I}$, where $\mathbf{I}$ is the identity matrix. For these reasons, we opted for PABS [14] as a metric to measure the distance between two CPDs. The use of PABS is justified in both unique and non-unique cases of CPD. In the unique cases, PABS enables the capture of distances between factor matrices, facilitating the identification of redundant filters based on their distance patterns while addressing scaling and permutation ambiguities. Let $\phi(.,.)$ be a function that computes the PABS between two factor matrices [14]. Reconsider the example in (9) in the case of unique CPDs, we have:

$$\begin{cases} \boldsymbol{\mathcal{W}}_i = [\![\mathbf{A}_i, \mathbf{B}_i, \mathbf{C}_i]\!], \\ \boldsymbol{\mathcal{W}}_j = [\![\mathbf{A}_j, \mathbf{B}_j, \mathbf{C}_j]\!], \\ \boldsymbol{\mathcal{W}}_i = \boldsymbol{\mathcal{W}}_j. \end{cases} \Rightarrow \begin{cases} \phi(\mathbf{A}_i, \mathbf{A}_j) = 0, \\ \phi(\mathbf{B}_i, \mathbf{B}_j) = 0, \\ \phi(\mathbf{C}_i, \mathbf{C}_j) = 0. \end{cases} \tag{10}$$

Even in non-unique cases, PABS remains effective in identifying redundancies. It captures the distance between different sets of factor matrices representing the same tensor, as will be confirmed in the simulations. This capability empowers the pruning process to eliminate filters that contribute minimally to model performance or display high similarity to other filters. The outcome is a more compact model that preserves critical features and maintains performance.

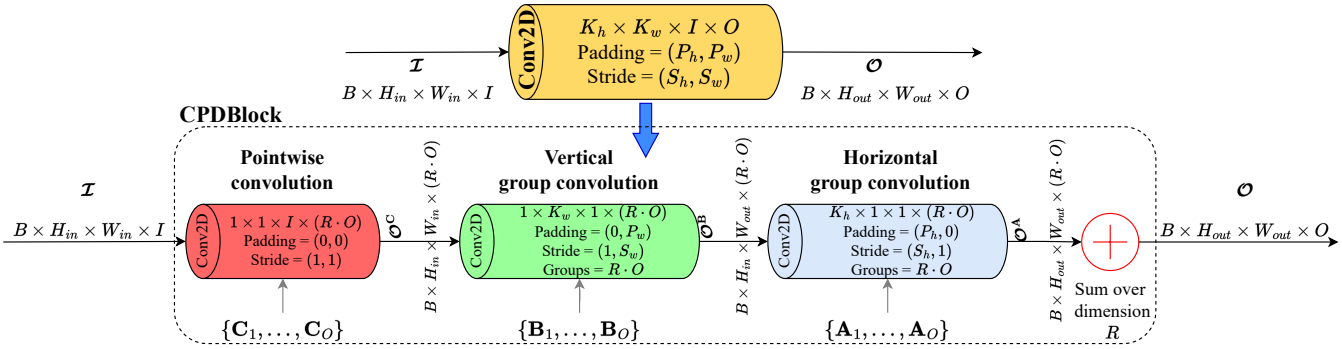


Fig. 4: Visualization depicting the structure of CPDBlock in popular deep learning frameworks.

The fundamental concept behind CPDBlock pruning is to create a distance matrix $\mathbf{D}$, where each element $\mathbf{D}_{ij}$ corresponds to the distance between the factor matrices $[\![\mathbf{A}_i, \mathbf{B}_i, \mathbf{C}_i]\!]$ and $[\![\mathbf{A}_j, \mathbf{B}_j, \mathbf{C}_j]\!]$. The pruning process involves iteratively identifying the pair of decompositions, i and j, with the minimum value of $\mathbf{D}_{ij}$ and removing one of them. In our strategy, the decompositions between i and j are iteratively assessed, and the one that most closely resembles the rest of the decompositions is removed. This ensures that the pruned one retains a representation most similar to the remaining ones. The distance matrix $\mathbf{D} \in \mathbb{R}^{O \times O}$ can be expressed as

$$\mathbf{D}_{ij} = \alpha \mathbf{D}^{\mathbf{A}}_{ij} + \beta \mathbf{D}^{\mathbf{B}}_{ij} + \gamma \mathbf{D}^{\mathbf{C}}_{ij}, \tag{11}$$

where $\mathbf{D}^A_{ij} = \phi(\mathbf{A}_i, \mathbf{A}_j)$ (similarly for $\mathbf{D}^B_{ij}$ and $\mathbf{D}^C_{ij}$), and α, β, and γ are weight parameters whose sum is equal to 1. A straightforward configuration is to set $\alpha = \beta = \gamma = \frac{1}{3}$. The pruning strategy is outlined in Algorithm 1, complemented by a visual representation in the right half of Figure 3.

Algorithm 1 CPDBlock Pruning

Require: The decompositions of O filters $\{\mathbf{A}_1, \mathbf{B}_1, \mathbf{C}_1\}, \ldots, \{\mathbf{A}_O, \mathbf{B}_O, \mathbf{C}_O\}$ and the number of filters after pruning O'.
Ensure: Selected factors $\{\mathbf{A}_{p_1}, \mathbf{B}_{p_1}, \mathbf{C}_{p_1}\}, \ldots, \{\mathbf{A}_{p_{O'}}, \mathbf{B}_{p_{O'}}, \mathbf{C}_{p_{O'}}\}$.

```
Compute distance matrix D following (11).
for t = 1 to O − O′ do
    Find the shortest distance: (i, j) = argmin_{(x,y), x≠y} D_{x,y}
    if Σ_{k=1, k≠i}^{O} D_{i,k} ≤ Σ_{k=1, k≠j}^{O} D_{j,k} then
        Delete factors of decomposition i.
    else
        Delete factors of decomposition j.
    end if
    Delete row/column of the deleted decomposition in D.
end for
```

III. Experiments

NORTON is compared with SOTAs in the fields of low-rank decompositions (D), structured pruning (P), and hybrid methods (H). The model is assessed via accuracy, required Multiply Accumulate Operations (MACs), and the number of parameters (Params). The compression ratio (CR) is defined as the percentage reduction in MACs/Params when compared to the original model. Top-1/top-5 accuracy is employed for classification tasks, while mean average precision (AP) and recall (AR) are used on detection/segmentation tasks.

TABLE I: Compression results of VGG-16-BN on CIFAR

Method	Type	Top-1	MACs (CR)	Params (CR)
VGG-16-BN		93.96	313.73M (00)	14.98M (00)
DECORE-500 [3]	P	94.02	203.08M (35)	5.54M (63)
RGP-64_16 [8]	P	92.76	78.78M (75)	3.81M (75)
NORTON (Ours)	H	**94.11**	**74.14M (77)**	**3.60M (76)**
DECORE-100 [3]	P	92.44	51.20M (82)	0.51M (96)
ALDS [11]	D	92.67	66.95M (86)	1.90M (96)
Dai *et al.* [5]	H	93.03	37.76M (87)	0.43M (97)
Lebedev *et al.* [1]	D	93.07	68.53M (78)	3.22M (78)
HALOC [9]	D	93.16	43.92M (86)	0.30M (98)
EDP [7]	H	93.52	62.40M (80)	0.66M (96)
NORTON (Ours)	H	**93.84**	**37.68M (88)**	1.94M (87)
RGP-64_6 [8]	P	91.45	31.37M (90)	1.43M (90)
DECORE-50 [3]	P	91.68	36.85M (88)	0.26M (98)
NORTON (Ours)	H	**92.54**	**13.54M (96)**	**0.24M (98)**
NORTON (Ours)	H	**90.32**	**4.58M (99)**	**0.14M (99)**

Tab. I shows compression results of VGG-16-BN on CIFAR-10. In all compression levels, compared with other methods, NORTON consistently achieves the highest accuracy while reducing much more computation costs and enjoying a similar number of parameters. Notably robust at high compression rates, NORTON can reduce 88% of FLOPs and 87% of parameters with just 0.12% loss, or 96% of MACs and 98% of parameters with a modest 1.42% loss. Even at an ultra-high 99% reduction in MACs and parameters, NORTON remains resilient, experiencing only a modest loss.

Tab. II shows compression results on ImageNet using ResNet-50. Across all evaluated scenarios, NORTON consistently outperforms other approaches in terms of both performance and complexity reduction. Our method can reduce 50% MACs while still enjoying an accuracy increment of 1.88% compared to Hinge [13], another hybrid method. At a 78% reduction in MACs, NORTON exhibited a 1.59% and 0.97% higher accuracy than DECORE [3] and RGP [8], respectively.

TABLE II: Compression results of ResNet-50 on ImageNet

Method	Type	Top-1	Top-5	MACs (CR)	Params (CR)
ResNet-50		76.15	92.87	4.09G (00)	25.50M (00)
Kim *et al.* [10]	D	75.34	92.68	N/A	17.60M (31)
DECORE-8 [3]	P	76.31	93.02	3.54G (13)	22.69M (11)
Hinge [13]	H	74.70	N/A	2.17G (47)	N/A
NORTON (Ours)	H	**76.58**	**93.43**	**2.08G (50)**	**13.51M (47)**
CC-0.6 [6]	H	74.54	92.25	1.53G (63)	10.58M (59)
RGP-64_30 [8]	P	74.58	92.09	1.92G (53)	11.99M (53)
Phan *et al.* [2]	D	74.68	92.16	1.56G (62)	N/A
C-SGD-60 [15]	P	75.29	92.39	1.82G (55)	12.37M (52)
EDP [7]	H	75.34	92.43	1.92G (53)	14.28M (44)
NORTON (Ours)	H	**75.95**	**92.91**	**1.49G (64)**	**10.52M (59)**
DECORE-5 [3]	P	72.06	90.82	1.60G (61)	8.87M (65)
RGP-64_16 [8]	P	72.68	91.06	1.02G (75)	6.38M (75)
NORTON (Ours)	H	**73.65**	**91.64**	**0.92G (78)**	**5.88M (77)**

TABLE III: Results of RCNN on COCO-2017

Model	AP	AR	MACs (CR)	Params (CR)	FPS	Δ(ms)
FasterRCNN	0.37	0.51	134.85G (00)	41.81M (00)	12	85
NORTON	0.32	0.48	**93.39G (31)**	**22.01M (47)**	**25**	**41**
MaskRCNN	0.34	0.47	134.85G (00)	44.46M (00)	9	111
NORTON	0.32	0.46	**93.39G (31)**	**24.65M (45)**	**20**	**50**
KeypointRCNN	0.65	0.77	137.42G (00)	59.19M (00)	8	125
NORTON	0.63	0.75	**95.97G (30)**	**39.39M (34)**	**17**	**59**

Using our compressed ResNet-50/Imagenet as the backbone for training Faster/Mask/Keypoint-RCNN on COCO (see Tab. III), NORTON significantly enhances inference throughput, achieving over a 2× FPS improvement compared to baseline models. FasterRCNN experiences a drop in end-to-end latency Δ from 85 ms to 41 ms, achieving a real-time framerate of 25 FPS. These evaluations, conducted on an RTX 3060, strongly demonstrate the real-world utility of NORTON in demanding computer vision tasks including detection and segmentation.

To investigate the impact of rank selection, which directly relates to the approximation error and compression gain, additional experiments are conducted on CIFAR-10 using VGG-16. Table IV presents the complexity reduction, approximation error (Normalized Mean Square Error), and accuracy before and after fine-tuning. Without fine-tuning, higher ranks lead to better weight approximations but result in less compression. With $5 \leq R$, our filter decomposition method produces comparable accuracies (maximum 1.46% drop) to the original model without fine-tuning. This indicates that our decomposition step performs well even without fine-tuning.

Moreover, we demonstrate that our decomposition step effectively works when followed by a fine-tuning step. The results reveal that after fine-tuning, accuracy is completely restored for all cases. However, it should be noted that the achieved compression ratios are not as favorable as those obtained with the proposed hybrid strategy. Importantly, our proposed method has not encountered the degeneracy problem (*i.e.*, instability issue when training a CNN with decomposed layers in the CP format), as seen in previous decomposition approaches [1], [2]. We suspect that the filter decomposition method is more fine-grained than the reshaped one. The trend of NMSE demonstrates a strong correlation between the approximation error and accuracy without fine-tuning.

TABLE IV: Complexity reduction, approximation error, and accuracy with and without fine-tuning with respect to the rank

Rank	MACs CR	Params CR	NMSE	Accuracy (%) Without FT	With FT
1	88.03	87.06	0.6265	10.00	93.84
2	76.44	75.98	0.4114	10.00	94.11
3	64.85	64.91	0.2760	68.37	94.18
4	53.27	53.84	0.1837	88.30	94.07
5	41.69	42.76	0.1173	92.44	94.22
6	30.10	31.69	0.0698	93.45	94.15
7	18.51	20.61	0.0372	93.90	94.11
8	6.93	9.54	0.0137	94.03	94.03

IV. Conclusion

This work introduces NORTON, a hybrid compression method combining tensor decompositions and structured pruning. NORTON excels in reducing model complexity and parameters, extending the boundaries of network compression. The proposed CP filter decomposition offers fine-grained control while PABS adapts to scaling and permutation ambiguities. Evaluation across architectures and datasets highlights NORTON's scalability, generalizability, and effectiveness.

References

[1] V. Lebedev, Y. Ganin, M. Rakhuba, I. V. Oseledets, and V. S. Lempitsky, "Speeding-up convolutional neural networks using fine-tuned cp-decomposition," in *ICLR*, 2015.

[2] A. Phan, K. Sobolev, K. Sozykin, D. Ermilov, J. Gusak, P. Tichavský, V. Glukhov, I. Oseledets, and A. Cichocki, "Stable low-rank tensor decomposition for compression of convolutional neural network," in *ECCV*, 2020.

[3] M. Alwani, V. Madhavan, and Y. Wang, "Decore: Deep compression with reinforcement learning," *CVPR*, 2022.

[4] S. Goyal, A. Roy Choudhury, and V. Sharma, "Compression of deep neural networks by combining pruning and low rank decomposition," in *IPDPSW*, 2019.

[5] C. Dai, X. Liu, H. Cheng, L. T. Yang, and M. J. Deen, "Compressing deep model with pruning and tucker decomposition for smart embedded systems," *IoT-J*, 2021.

[6] Y. Li, S. Lin, J. Liu, Q. Ye, M. Wang, F. Chao, F. Yang, J. Ma, Q. Tian, and R. Ji, "Towards compact cnns via collaborative compression," in *CVPR*, 2021.

[7] X. Ruan, Y. Liu, C. Yuan, B. Li, W. Hu, Y. Li, and S. Maybank, "Edp: An efficient decomposition and pruning scheme for convolutional neural network compression," *TNNLS*, 2021.

[8] Z. Chen, J. Xiang, Y. Lu, Q. Xuan, Z. Wang, G. Chen, and X. Yang, "Rgp: Neural network pruning through regular graph with edges swapping," *TNNLS*, 2023.

[9] J. Xiao, C. Zhang, Y. Gong, M. Yin, Y. Sui, L. Xiang, D. Tao, and B. Yuan, "Haloc: Hardware-aware automatic low-rank compression for compact neural networks," in *AAAI*, 2023.

[10] Y. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, "Compression of deep convolutional neural networks for fast and low power mobile applications," in *ICLR*, 2016.

[11] L. Liebenwein, A. Maalouf, O. Gal, D. Feldman, and D. Rus, "Compressing neural networks: Towards determining the optimal layer-wise decomposition," in *NIPS*, 2021.

[12] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM Review*, vol. 51, no. 3, pp. 455–500, 2009.

[13] Y. Li, S. Gu, C. Mayer, L. Van Gool, and R. Timofte, "Group sparsity: The hinge between filter pruning and decomposition for network compression," in *CVPR*, 2020.

[14] Åke Björck and G. H. Golub, "Numerical methods for computing angles between linear subspaces," *Mathematics of Computation*, 1973.

[15] T. Hao, X. Ding, J. Han, Y. Guo, and G. Ding, "Manipulating identical filter redundancy for efficient pruning on deep and complicated cnn," *TNNLS*, 2023.