# LIGHTWEIGHT CNNS UNDER A UNIFYING TENSOR VIEW

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Despite the decomposition of convolutional kernels for lightweight CNNs being well studied, previous works that relied on tensor network diagrams or higher dimensional abstraction lacked geometry intuition. Our work captures the CNN kernel as a 3D tensor and explores its various decompositions, allowing for a straightforward graphical and analytical perspective between different tensor approximation schemes and efficient CNN components, including pointwise and depthwise convolutions. Extensive experiments are conducted, showing that a pointwise-depthwise-pointwise (PDP) configuration via a canonical polyadic decomposition (CPD) initialization can be a viable starting point for lightweight CNNs. The compression ratio of VGG-16 can reach over $50\%$ while its performance outperforms its randomly initialized counterpart by $> 10\%$ in terms of accuracy. FPGA experiments for the PDP model further demonstrate its hardware efficacy, namely, $2.4\times$ faster and $1.4\times$ more energy efficient than the standard conv2d. Furthermore, our framework offers a unique slice-wise illustration and is the first to ever draw a connection to the shift layer. Such insight inspires a first-of-its-kind pruning method for shift layers, achieving nearly $50\%$ compression with $< 1\%$ drop in accuracy for ShiftResNet-20.

## 1 INTRODUCTION

Lightweight deep neural networks (DNNs) are crucial for edge AI where the DNNs are running on resource-limited hardware, e.g., Lin et al. (2021a). Extra consideration has to be given to implementation constraints such as compute, storage, throughput and power, just to name a few. In this regard, a plethora of efficient convolutional neural network (CNN) architectures have been researched and developed over the years for computer vision application. Prominent examples include MobileNet Sandler et al. (2018), EfficientNet Tan and Le (2019), and later the ShiftNet Wu et al. (2018) and its variants Jeon and Kim (2018); Chen et al. (2019). Even in the current era where Transformers are taking DNN architectures by storm (e.g., Vision Transformer (ViT) Dosovitskiy et al. (2021), MLP-Mixer Tolstikhin et al. (2021), etc.), CNN is still being re-injected into emerging Transformer variants (e.g., ConvMixer Trockman and Kolter (2022)). Consequently, it is always of interest to study and explore lightweight CNNs.

To this end, depthwise separable (DS) convolution Chollet (2017); Sandler et al. (2018); Tan and Le (2019) has proven to be an efficient substitute of the regular convolutional kernels. In particular, it segregates the coupled spatial and channel mixing in a conventional CNN filter into separate depthwise (DW) and pointwise (PW) mixing, respectively. This results in significant savings in storage and computation, with little or even no loss in the output accuracy. This paper systematically and analytically shows that this does not come as a surprise, but follows naturally from the underlying decomposition and approximation schemes of the reshaped CNN kernel tensor.

While tensors and various efficient CNN implementations are not new, it is the first time they are brought under a highly visualizable, unifying tensor umbrella to provide an intuitive illustration to their origin of success. The arithmetic under various decompositions further inspire effective ways to compress the model, and to design accuracy booster when a pretrained or teacher model is available. The main contributions of this work are:

- A unifying reshaped kernel tensor view whose different approximation schemes are naturally tied to different efficient CNN architectures.

- Given an available pretrained network, an analytical way to obtain appropriately tensorized initialization to potentially boost training accuracy.
- A link of canonical polyadic decomposition (CPD) to the bottleneck CNN layer, plus its intrinsic connection to the lately proposed hardware-efficient shift layers. This in turn gives rise to a *first-of-its-kind* channel pruning scheme for shift layers.

## 2 Related Work

DNNs have recently been realized on edge devices thanks to advances in network compression techniques. TinyML takes edge AI one step further, making it possible to run deep learning models on edge-centric FPGA and microcontroller (MCU), such as hls4ml Aarrestad et al. (2021) and MCUNetV2 Lin et al. (2021a). In the context of CNN compression, besides channel pruning, quantization and knowledge distillation, a powerful approach is matrix or tensor decomposition Lebedev et al. (2015); Kim et al. (2016); Astrid et al. (2018); Hayashi et al. (2019); Ran et al. (2021). Lebedev et al. (2015) is among the first that proposed CPD of the regular CONV layer whose nature is a 4-D kernel tensor. Specifically, a CPD of the latter breaks down a CONV layer into two DW and two PW layers. Astrid et al. (2018) also utilized CPD but treated the CONV layer as a 3-D tensor, which further reduced the number of parameters. Tucker decomposition is employed in Kim et al. (2016), which decomposes a CONV layer into two PW layers and a regular CONV layer with smaller input and output channels. Ran et al. (2021) also used Tucker decomposition, adding a regularizer to the loss function during training to dynamically assign the Tucker ranks. It is worth noting that all these works are related in one way or another to efficient CNNs Chollet (2017); Sandler et al. (2018); Tan and Le (2019), which contain DS or bottleneck layers. Compared with Tucker decomposition Kim et al. (2016); Ran et al. (2021), CPD is expected to further shrink the number of weights due to its equivalence to having a diagonal Tucker core. However, the scheme in Lebedev et al. (2015) may result in unstable training and is bound to vector kernels which are not as effective as a square kernel in capturing spatial correlations. Astrid et al. (2018) also tested only with the older-generation AlexNet with few CONV layers, using an ad hoc progressive layer-wise decomposition and fine-tuning scheme that is not scalable. Hayashi et al. (2019) enumerated various CNN decompositions using tensor network diagrams. Nevertheless, the high-dimensional and abstract notations hide the geometry and visual intuitions tied to different DS variants. Prior works have used tensor network diagrams to encapsulate higher dimensional tensors as well as their decompositions into low rank factors Lebedev et al. (2015); Kim et al. (2016); Su et al. (2018); Hayashi et al. (2019). Nonetheless, at 4D and beyond, the geometry view is lost and the underlying arithmetic intuition can become abstract. To this end, we show it is indeed possible to keep everything in 3D and employ only basic geometry visualizations for various tensorized convolutions.

On another front of CNN compaction, the idea of shift operation first appeared in Wu et al. (2018), which is meant to substitute standard spatial convolutions. Ordinary convolutions have a quadratic complexity with kernel size. In contrast, shift operations have zero parameters and zero FLOPs regardless of the receptive field, making them ideal for deployment on edge devices with limited resources. Several variants have been proposed to improve representation power and efficiency of shift operations Jeon and Kim (2018); Chen et al. (2019). Recently, shift operation is also applied to video action recognition tasks Lin et al. (2019); Fan et al. (2020), demonstrating that shift layers, constructed by shift operation and PW convolution, are expressive enough to encode spatio-temporal information. Although the shift layer may look unconventional from regular CNN algebra, we will show below that it can be naturally accommodated in the CPD framework by considering shifts as *one-hot* kernels. Subject to this view, a novel channel pruning scheme can be derived for a shift layer which, to the best of our knowledge, is reported for the first time in literature. In short, we believe such "aggressive" re-implementations of efficient CNN layers, further coupled with recent advances in structured sparse fully-connected (FC) layers Dao et al. (2019; 2020); Lin et al. (2021b), can actively contribute to the success of TinyML in the era of edge AI.

## 3 Kernel Tensor and its Approximation

We begin by instantiating the kernel tensor which is reshaped from 4-way $[c_i, c_o, k, k]$ ($c_i, c_o$ denote the numbers of input and output channels, and $k \times k$ the spatial size) into 3-way $[c_i, c_o, k^2]$ for visualization, see upper left corner of Fig. 1. We then enumerate different views, decompositions and

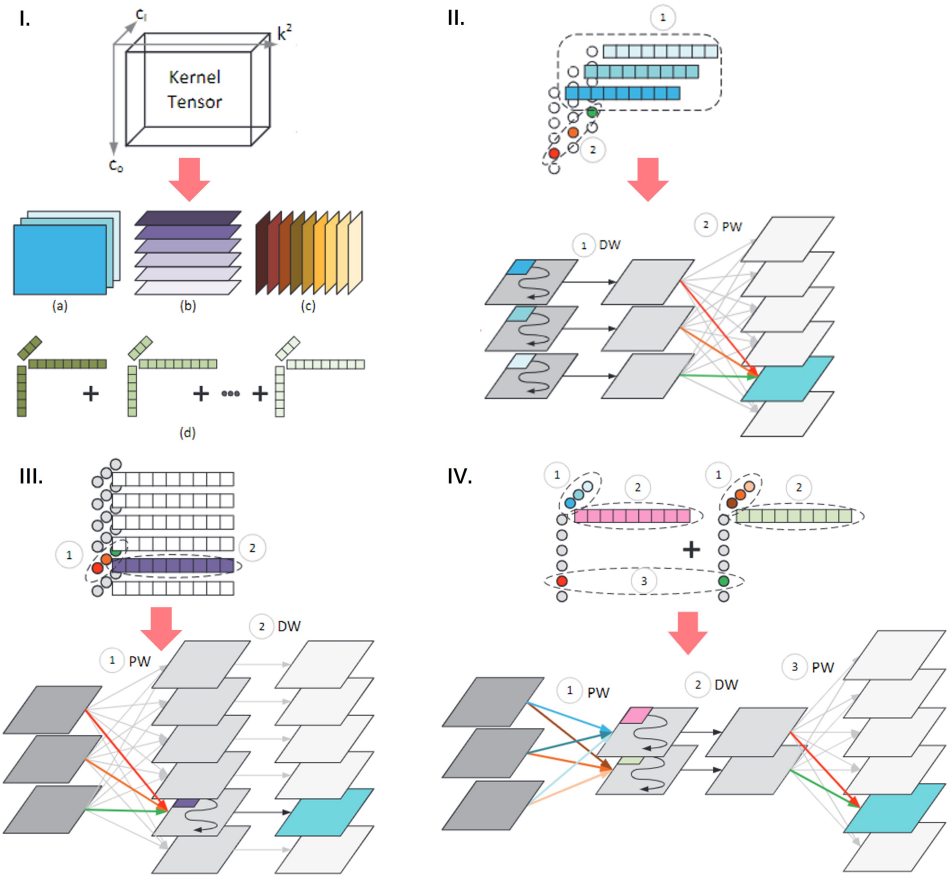



Figure 1: **(I.)** The reshaped kernel tensor and its different views: (a) frontal slices; (b) horizontal slices; (c) lateral slices; (d) CPD. We slightly abuse notations and use $c_o$, $c_i$ and $k^2$ for both the axis names and index labels. This example has dimensions $|c_i| = 3$, $|c_o| = 6$ and $|k^2| = 9$. **(II.)** Approximating front slices with rank-1 terms translates into the standard depthwise separable convolution (viz. depthwise + pointwise, abbreviated as DP in this work). **(III.)** Approximating horizontal slices with rank-1 terms translates into an "inverted" depthwise separable convolution (viz. pointwise + depthwise, called PD in this work). **(IV.)** Approximating the kernel tensor with multiple rank-1 CPD terms (CPD rank $r_{cp} = 2$ here) translates into a linear bottleneck (viz. pointwise + depthwise + pointwise, called PDP in this work).

approximations of this original tensor, which in turn spin off different efficient CNN architectures. We remark such visual links to various CNN implementations is first-of-its-kind, and differs significantly from the notational and diagrammatic views in Guo et al. (2018); Hayashi et al. (2019) which easily overshadow the underlying intuition.

### 3.1 Frontal Slices

Referring to Fig. 1.I(a), each slice or matrix can be decomposed into a sum of multiple rank-1 terms with decreasing dominance. An obvious way is to use singular value decomposition (SVD), but advanced options like variational Bayesian matrix factorization (VBMF) can also be employed for rank determination Nakajima et al. (2013) in noisy data. Fig. 1.II depicts a rank-1 approximation to each front slice. Here we draw the spatial ($k^2$) axis vectors with rectangular bars to differentiate them from other axes (using balls) to highlight their window sliding nature along the input channels they act on. With reference to Fig. 1.II, now we fix a particular output channel, e.g., $c_o = 5$, for illustration. It can be seen that there is one kernel filter along each $c_i$ index, corresponding to a DW convolution along each input channel. Once the DW convolution is performed, 3 convolved output

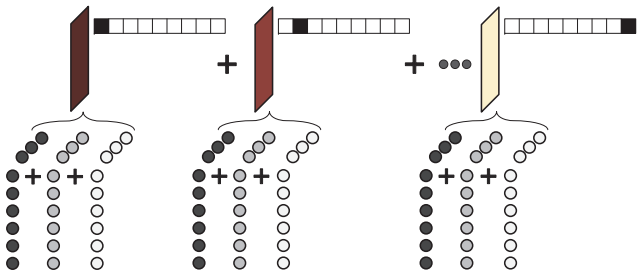


Figure 2: Lateral slices positioned with one-hot kernel filters translates into a ShiftNet (viz. pointwise + shift + pointwise) convolution.

slices are produced which are then contracted by the 3 colored balls along the $c_i$ axis, namely, a PW operation to obtain the $c_o = 5$ channel. Such sequence of operations cannot be swapped because the DW kernels are tied with the $c_i$ indices, and need to be applied on the input channels. The number of CNN weight parameters involved is $(|c_o| + |k^2|)|c_i|$ which equals 45 in this example. We abbreviate this CNN replacement by DW+PW as the DP scheme which coincides with the standard DS scheme.

Similarly, we study Fig. 1.I(b) with its corresponding rank-1 slices shown in Fig. 1.III. As before, we fix a particular output channel $c_o = 5$, then it can be seen that there are six kernels along each $c_o$ index, each corresponding to a DW convolution operating on the PW $(1 \times 1)$-contracted slices along the $c_i$ axis by the colored balls. Again, such sequence of operations cannot be interchanged, since the 6 DW kernels are tied with the $c_o$ indices and need to be applied right before the output channels. The number of weight parameters in this setting is $(|c_i| + |k^2|)|c_o|$ which equals 72 in our example. We call this PW+DW operation the PD scheme in this work.

### 3.2 Canonical Polyadic Decomposition

We detour to the CPD view in Fig. 1.I(d) before getting back to the lateral slices, which turns out to be a more natural order of illustration. Referring to Fig. 1.IV and borrowing from previous sections, we can segregate the contraction/convolution into three stages, namely, PW along $c_i$ to generate a number of slices equal to the CPD rank $r_{cp}$ (i.e. number of CPD terms), followed by DW along these slices and finally PW along the desired $c_o$. This view corresponds to the celebrated bottleneck layer Sandler et al. (2018), whose residual variant is depicted in the upper part of Fig. 3. Apparently, whether it is a bottleneck or an inverted bottleneck is determined by $r_{cp}$, which decides the center DW block being wider or thinner than its 'entrance' and 'exit' PW layers. We name this PW+DW+PW combination the PDP scheme. In this example, the number of weight parameters is $(|c_i| + |c_o| + |k^2|)r_{cp}$ which equals 36. Note that this number depends heavily on the $r_{cp}$ which we can tune for different tradeoffs between complexity and representation capacity.

### 3.3 Lateral Slices

Finally, we turn back to Fig. 1.I(c) for the lateral slices. It is clear we can view the original tensor as a slice-wise aggregation by positioning them at the appropriate $k^2$-axis index via a one-hot kernel vector, as shown in Fig. 2. Like before, each slice can be decomposed into multiple rank-1 terms, all sharing the same one-hot kernel vector. For instance, the leftmost term in Fig. 2 can spin off into three CPD terms, resembling those in Fig. 1.IV. The one-hot nature of the kernel vector effectively performs simple shifting, as advocated in Wu et al. (2018). As such, we draw equivalence of lateral slices to their ShiftNet counterparts Wu et al. (2018); Chen et al. (2019); Jeon and Kim (2018). If we use $r_{cp}$ to denote the total number of 3-way rank-1 terms selected, say, according to their importance characterized by their singular values, then the number of weights equals $(|c_i| + |c_o|)r_{cp}$.

#### 3.3.1 Shift Layer Pruning

To elaborate further, if a pretrained ShiftNet is available, then one way to do pruning of its shifted channels is to collect rank-1 terms corresponding to the same shift, and then add up the $c_o$-$c_i$ mode rank-1 terms into a matrix (cf. lower part of Fig. 3). An SVD is then performed on this $|c_o| \times |c_i|$

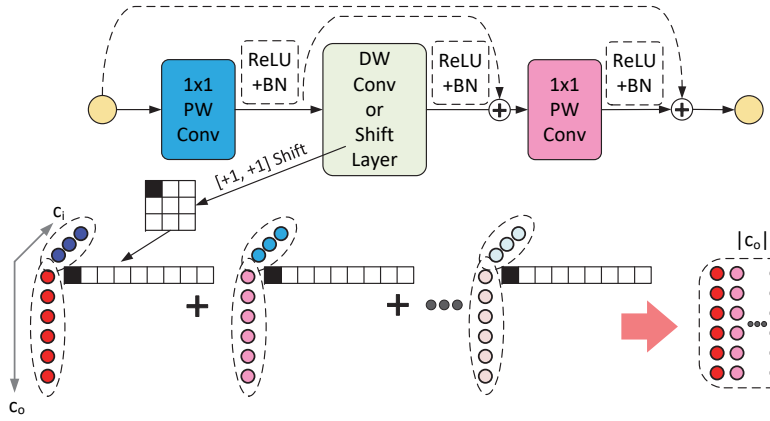



Figure 3: (Upper) A generic PW+DW/Shift+PW block where the dashed-line shortcuts and nonlinearity blocks are optional. (Lower) Assuming a shift layer, one can collect $c_o$-$c_i$ mode rank-1 terms of the same shift and sum them for derivation of principal components.

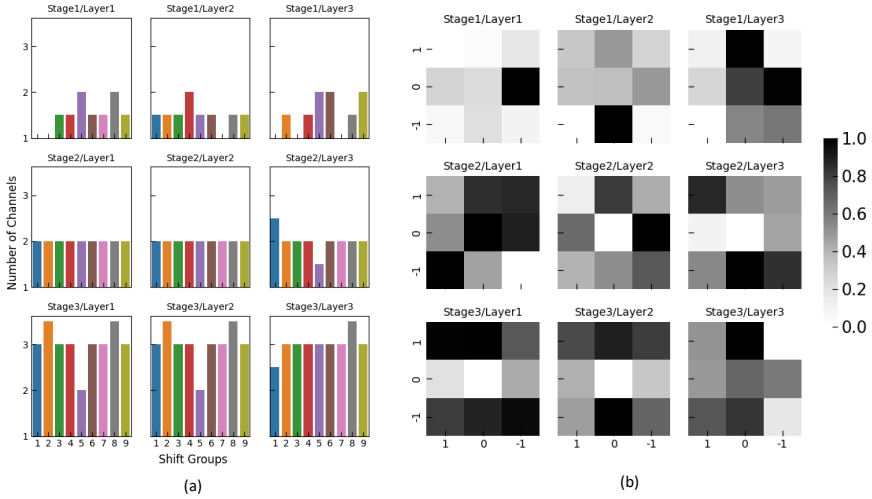



Figure 4: Shift distribution of ShiftResNet-20 trained on CIFAR-10 after uneven shift pruning with $\phi = 16^{-1}$. There are three stages in ShiftResNet-20, each having three shift layers. (a) The number of channels in each shift group; (b) The relative importance of each shift in various layers based on their singular values. Singular values are summed at each position and normalized within each layer.

matrix to decide the number of dominant terms for retention and the number of terms to prune away. As far as we know, this is the first-ever shift layer pruning scheme.

Specifically, under the same shift, the $c_o$-$c_i$ rank-1 terms are summed up into a $|c_o| \times |c_i|$ matrix for principal component analysis, e.g., via SVD. Then, the most dominant terms are retained as the new PW filters on the $c_i$ and $c_o$ modes. As an illustrative example, suppose for two channels with the same shift, the $c_i$-mode $1 \times 1$ filters are $v, \hat{v} \in \mathbb{R}^{|c_i| \times 1}$ whose corresponding $c_o$-mode $1 \times 1$ filters are $u, \hat{u} \in \mathbb{R}^{|c_o| \times 1}$, respectively. Then the summed matrix is $\begin{bmatrix} u & \hat{u} \end{bmatrix} \begin{bmatrix} v^T \\ \hat{v}^T \end{bmatrix}$. Suppose the flattened input channel matrix is $X$ and there exists nonlinearity (say, ReLU) after the first PW layer. Then, if $\hat{v} \approx v$, whether in terms of cosine or Euclidean distance, the mapped channel matrices $\mathrm{ReLU}(v^T X) \approx \mathrm{ReLU}(\hat{v}^T X)$, especially when the feature dimension is high. The same argument also holds when $\hat{u} \approx u$ which, together with $\hat{v} \approx v$, implies the above matrix would be close to rank-1 and its principal $c_i$ and $c_o$ filters can be obtained from an SVD and further fine-tuned through backpropagation. Subsequently, when the entrance and exit PW layers have close PW filters, then they can be consolidated into fewer principal filters. Note that our pruning approach is different from

Chen et al. (2019), where shift operations are sparsified through a loss regularizer, while keeping the size of the intermediate feature map unchanged. In contrast, we treat the one-hot shift kernel vector as an on/off switch carrying information of its associated $c_i$-$c_o$ rank-1 term. Once pruned, the entire 3-way rank-1 term is dropped, so the feature map size (thus also representation capacity) reduces.

Here, we define a new hyperparameter dubbed pruning ratio $\phi = \epsilon_{new}/\epsilon_{old}$, where the notion of expansion ratio $\epsilon$ is borrowed from Wu et al. (2018) which specifies the ratio of bottleneck's channel size to the output channel size of the exit PW. Obviously, $\phi$ controls the degree of compression, allowing for a tradeoff between accuracy and model size. There are two strategies to select the principal filters. The first involves comparing singular values within the same shift and preserving the same amount of dominant terms in each group. Concretely, suppose we have a $3 \times 3$ DW shift layer, then there are 9 shift groups each corresponding to one direction of shift (including zero shift). This method guarantees equal number of channels in each shift group. Another approach is to compare singular values across all shift groups and retain the most dominant terms. Fig. 4 shows the shift distribution under "uneven" shift pruning, where missing bars in Fig. 4(a) illustrate that the whole shift group is pruned. The complete pseudocode for shift layer pruning, as well as additional visuals, can be found in the Appendix C.

## 4 EXPERIMENTS

To showcase the power of the proposed tensor machinery, extensive experiments are conducted on image classification tasks, using CIFAR Krizhevsky et al. (2009), TinyImageNet and ImageNet2012 Deng et al. (2009) as datasets. The popular VGG-16 Simonyan and Zisserman (2014) is chosen to demonstrate the power of CPD initialization as a performance booster. Then, we select the latest Transformer-like ConvMixer Trockman and Kolter (2022) to articulate the advantages of CPD-induced PDP over the heavier DS convolution. We further compare our advocated PDP decomposed kernels to other tensor decomposition methods, using ResNet He et al. (2016) as the backbone. To verify the newly proposed shift layer pruning, we employ ShiftResNet Wu et al. (2018) as a backbone and evaluate on CIFAR datasets. Finally, FPGA benchmarking showcases the hardware advantages of lightweight bottlenecks. All models are trained on a single NVIDIA GeForce RTX 3090 Graphics Card with 24GB frame buffer. We refer to the appendix for further implementation details and results.

Table 1: VGG-16 with SVD vs randomly initialized DP/PD kernels on CIFAR-10.

| Method | CR (%) | #Params(M) | Acc (%) Random/SVD |
|---|---|---|---|
| DP | 88.53 | 1.69 | **92.81**/92.70 |
| PD | 88.49 | 1.69 | **93.72**/93.54 |

Table 2: VGG-16 with CPD vs randomly initialized PDP kernels on CIFAR-10.

| Rank $r_{cp}$ | CR (%) | #Params(M) | Acc (%) Random/CPD |
|---|---|---|---|
| 4 | 99.65 | 0.05 | 51.31/**62.48** |
| 8 | 99.44 | 0.08 | 61.63/**72.27** |
| 16 | 99.00 | 0.15 | 65.06/**86.09** |
| 32 | 98.12 | 0.28 | 75.19/**89.73** |

### 4.1 VGG-16

Here approximations with different tensor views discussed in Section 3 are realized by replacing *every* CONV layers of a VGG-16. A baseline VGG-16 is first trained on CIFAR-10 for 200 epochs. The pretrained CONV kernels are then decomposed into various configurations followed by 300 epochs of fine-tuning. Table 1 shows the results of using DP and PD schemes to decompose the kernel tensors. The PD configuration has a slightly better performance as compared to DP with the same degree of compression. In this paper, compression is measured by the model-wise compression rate (CR) which will be used throughout. Surprisingly, random initialization leads to a higher test accuracy in both cases as compared to SVD initialization. We believe that the reason is approximation using a single rank-1 term for each tensor slice is insufficient.

Table 2, on the other hand, highlights the significance of CPD initialization when every CONV layer in the VGG-16 is turned into a PDP configuration comprising $r_{cp}$ DW filters. VGG-16 with CPD-initialized kernels outperforms its random-initialized counterpart by a significant margin, ranging from $11\%$ to $14\%$. By gradually raising the CPD rank ($r_{cp}$) from $4$ to 32, classification accuracy

rises from $62.48\%$ to $89.73\%$ with only a slight decrease in compression ratio ($99.65\%$ vs $98.12\%$). Indeed, PDP slimming of a dense network has a distinct advantage over DP and PD, which achieves an additional $\approx 6\times$ parametric reduction than the latter two. Utilizing CPD initialization for PDP, the final accuracy is mostly regained.

We highlight that CPD-initialized PDP kernels can be used in conjunction with existing compression methods. Here, we apply, separately, three compression techniques on top of a PDP VGG-16, including quantization-aware training (QAT), knowledge distillation (KD) Hinton et al. (2015) and HRank Lin et al. (2020) filter pruning. Table 3 shows classification results on CIFAR-10 as well as their corresponding model sizes in terms of parameters and storage (MB). The PDP-based model can be further compressed to $> 100\times$ smaller than the original baseline by using QAT with 8-bit weights and activations (i.e. 8W8A). KD, on the other hand, can be utilized to boost the performance ($91.25\%$ vs $89.73\%$) of a CNN with PDP decomposed kernels. Besides, we can apply PDP to a CNN with pruned channels using HRank to further obtain a more compact model. The pruned model has $1.87$M parameters, which shrinks to $0.11$M after being decomposed into PDP kernels at $r_{cp} = 32$. Using CPD initialization, the accuracy of this extremely compact model is $90.00\%$, which soars even higher than applying CPD directly on the original model, with a size even smaller than the $r_{cp} = 16$ case in Table 2. The above results show CPD-induced PDP kernels are highly versatile and can effectively supplement other compression techniques in an orthogonal manner.

Table 3: Effects of different compression methods on PDP VGG-16 evaluated using CIFAR-10.

| Method | #Params(M) | Size(MB) | Acc (%) |
|---|---|---|---|
| Baseline | 14.73 | 56.25 | 94.13 |
| PDP ($r_{cp} = 32$) | 0.28 | 1.19 | 89.73 |
| + QAT (8W8A) | 0.28 | 0.52 | 89.16 |
| + KD | 0.28 | 1.19 | 91.25 |
| + HRank | 0.11 | 0.54 | 90.00 |

### 4.2 TRANSFORMER

Next, we apply PDP to a transformer-like architecture named ConvMixer Trockman and Kolter (2022). It is derived from the simplistic MLP-Mixer Tolstikhin et al. (2021) featuring repetitive square (viz. same I/O channels) CNN mixer blocks with an entrance DW layer for spatial mixing and an exit PW layer for channel mixing, each having an equal number of channels and is succeeded by a Gaussian error linear unit (GELU) Hendrycks and Gimpel (2016) and batch normalization (BN). While it is generally believed an inverted bottleneck (i.e., an expanded number of DW channels) is needed for intact information flow, we demonstrate a counter-intuition that using a relatively much narrower DW layer can as well handle the task without much sacrifice in accuracy, while enabling a remarkable compression in the number of parameters.

Table 4: Classification result on different datasets. Replacing DS in ConvMixer-256/8 with Conv2d.

| Method | CIFAR-10 | | CIFAR-100 | | TinyImageNet | |
|---|---|---|---|---|---|---|
| | #Params(M) | Acc (%) | #Params(M) | Acc (%) | #Params(M) | Top-1/Top-5 (%) |
| Baseline (DS) | 0.59 | 94.80 | 0.62 | 75.59 | 0.65 | 59.44/81.14 |
| Conv2d remake | 13.12 | 94.23 | 13.14 | 75.23 | 13.18 | 57.76/78.38 |

Table 5: Ablation of ConvMixer with CPD initialized PDP kernels on CIFAR. $--$ indicates training diverges. **ACT**ivation : **G**ELU or **R**eLU.

| Method | CIFAR-10 Acc(%) (G/R) | | | CIFAR-100 Acc(%) (G/R) | | |
|---|---|---|---|---|---|---|
| | $r_{cp} = 8$ | $r_{cp} = 16$ | $r_{cp} = 32$ | $r_{cp} = 8$ | $r_{cp} = 16$ | $r_{cp} = 32$ |
| CPD | 80.86/79.35 | 88.10/88.39 | $--$ | 50.57/47.88 | 64.45/64.53 | 70.51/69.36 |
| + Residual 2 | 79.72/80.11 | 88.71/88.64 | 92.54/92.26 | 53.76/55.18 | 63.70/64.81 | 69.58/69.40 |
| + Residual 1 | 86.77/86.33 | 90.84/90.69 | 92.87/92.90 | 61.51/**62.25** | **69.19**/67.60 | 70.89/**71.15** |
| + Residual 1 w/o ACT after DW | **86.89**/**86.99** | **91.22**/**91.47** | **93.42**/**93.36** | **62.44**/62.2 | 68.83/**67.75** | **72.26**/70.90 |

In the following, ConvMixer-256/8 is selected as the baseline, where 256 and 8 denote the hidden dimension and depth, respectively. To acquire the CPD-initialized PDP kernels, DS layers in the original mixer are first replaced by a standard CONV layer (Conv2d), with GELU and BN after PW remaining unchanged. Models are first trained on various datasets shown in Table 4, which shows that redoing ConvMixer with Conv2d results in slight performance degradation. Afterwards, we apply CPD on the Conv2d kernel tensors to initialize the PDP configuration and fine-tune for another 200

epochs. An ablation study on CIFAR datasets is conducted to find the optimal design using various nonlinearities and residual connections He et al. (2016), depicted in Table 5. Residual 1 and Residual 2 refer to the long and short dashed line arrows in Fig. 3, respectively. Residual 1 connecting two PW layers allows communication via more channels as compared to Residual 2 that wires across the relative narrow DW filters, leading to better performance. Motivated by Sandler et al. (2018), we further remove nonlinearity after the DW layer to prevent information loss, yielding the best result.

Now we focus on TinyImageNet which is a subset of ImageNet Deng et al. (2009) with 100k training and 10k test images. Each image is classified into one of 200 categories. The goal of evaluating on TinyImageNet is to demonstrate that CPD-initialized narrow bottleneck can handle complicated tasks with performance on par with the heavier, original DS in ConvMixer. Based on the ablation study on CIFAR datasets, we select the configuration in the bottom row of Table 5 with GELU activation as our building block. Table 6 shows that the narrow bottleneck with $r_{cp} = 64$ can achieve comparable accuracies at Top-1 ($58.26\%$ vs $59.44\%$) and Top-5 ($80.18\%$ vs $81.14\%$), with $1.9\times$ reduction in number of parameters compared to the baseline in Table 4. This reveals isotropic design (i.e. uniform width) may introduce significant channel redundancy that can be exploited for compression.

Table 6: Classification result on TinyImageNet. CR is computed with respect to Baseline (DS) in Table 4.

| Rank $r_{cp}$ | CR (%) | #Params (M) | Top-1(%) | Top-5(%) |
|---|---|---|---|---|
| 8 | 83.85 | 0.11 | 46.82 | 72.46 |
| 16 | 78.52 | 0.14 | 52.98 | 77.60 |
| 32 | 67.86 | 0.21 | 55.96 | 79.16 |
| 64 | 46.55 | 0.35 | **58.26** | **80.18** |

### 4.3 RESNET

In this section, we conduct extensive experiments to compare the lightweight PDP kernels with other tensorized CONV kernels. Two sets of experiments are carried out. First, we demonstrate the PDP on large-scale ImageNet2012 dataset using ResNet-34 as backbone and compare with Tucker-2 decomposition Kim et al. (2016). Table 7 shows that CPD initialized PDP kernels outperform Tucker counterpart by a significant margin ($67.72\%$ vs. $62.73\%$) under approximately the same compression ratio ($\approx 92\%$). We then compare PDP with CP layers proposed in Lebedev et al. (2015) and an enhanced version of Tensor Train (mTT) in Su et al. (2018) on CIFAR-10 with ResNet-32 as backbone. According to the Table 8, the more sophisticated Tensor Train decomposition does not show superior performance versus the simpler CPD in terms of accuracy. Furthermore, because of the intuitive connection between CPD decomposition and PDP bottleneck, highly optimized cuDNN convolutional operations can be used in our framework, whereas mTT requires customized CUDA kernels or the use of less optimized tensor contractions, resulting in a 1.7X slower inference speed than ours. Furthermore, PDP exhibits more stable training than the CP layers suggested in Lebedev et al. (2015), wherein CPD is performed along relatively short spatial dimensions, resulting in diverging training.

Table 7: Comparison of PDP with Tucker-2 on ImageNet2012.

| Method | MC (%) | Acc@1 (%) | Acc@5 (%) |
|---|---|---|---|
| Baseline (ResNet-34) | $--$ | 73.30 | 91.42 |
| PDP | 92.35 | **67.72** | **88.39** |
| Tucker-2 | 92.49 | 62.73 | 84.70 |

Table 8: Comparison of PDP with other tensor decomposition methods on CIFAR-10. Latency is evaluated on single RTX3090 GPU.

| Method | MC (%) | Acc (%) | Latency (ms) |
|---|---|---|---|
| PDP-ResNet-32 | 90.79 | **90.30** | **5.20** |
| CP-ResNet-32 | 89.23 | $--$ | 10.63 |
| mTT-ResNet-32 | 89.67 | 90.21 | 8.65 |

### 4.4 SHIFT LAYER PRUNING

To demonstrate the idea of shift layer pruning, ShiftResNet-20 Wu et al. (2018) with an expansion ratio $\epsilon = 9$ is chosen as our backbone. A shift module in the ShiftResNet consists of a $3 \times 3$ DW shift layer surrounded by two PW layers, interleaved with BN and nonlinearity. By varying $\epsilon_{new}$, we obtain a set of pruning ratios $\phi \in \{\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}\}$. Baseline models are first obtained by training on target datasets for 200 epochs. Shift layer pruning as described in Section 3.3.1 is then applied to every shift module in the pretrained network, followed by another 200 epochs of fine-tuning. The result is shown in Table 9. Shift layer pruning achieves nearly $50\%$ compression with $< 1\%$ accuracy

drop for both CIFAR-10 (92.88% vs 93.37%) and CIFAR-100 (70.58% vs 71.47%). The results also suggest that uneven shift pruning exhibits higher accuracies than even pruning in most circumstances.

Table 9: Shift Layer Pruning. Baseline accuracies are 93.37%@CIFAR-10, 71.47%@CIFAR-100.

| $\phi$ | CIFAR-10 | | | CIFAR-100 | | |
|---|---|---|---|---|---|---|
| | CR(%) | Acc (%) (Even) | Acc (%) (Uneven) | CR(%) | Acc (%) (Even) | Acc (%) (Uneven) |
| $\frac{1}{2}$ | 49.18 | 92.81 | **92.88** | 48.17 | 70.48 | **70.58** |
| $\frac{1}{4}$ | 73.78 | 91.41 | **91.50** | 72.25 | 67.16 | **67.62** |
| $\frac{1}{8}$ | 86.07 | 89.09 | **89.11** | 84.30 | 62.21 | **63.13** |
| $\frac{1}{16}$ | 92.22 | 85.57 | **85.84** | 90.32 | **57.25** | 56.83 |

### 4.5 Hardware Benchmarking

Since an obvious niche of our compressed models lies in their extreme light weight, we further realize them on a field-programmable gate array (FPGA) prototyping platform to contrast against standard convolution Aarrestad et al. (2021) by a customized hardware architecture. All models are compiled and synthesized with Vivado 2020.2 for deployment onto a Xilinx Ultra96 v2 FPGA board featuring a Zynq UltraScale+ MPSoC device(ZU3EG) clocked at 250MHz. Ultra96-v2 contains 71k LUTs, 7.6Mb BRAMs and 360 DSPs.Our accelerator supports both $32 \times 32$ and $224 \times 224$ input formats with 8-bit weights and activations. Here, we employ the $224 \times 224$ format which is widely adopted in various datasets such as ImageNet2012. For fair comparison, we compare our PDP accelerator to prior works, all of which realize VGG-16, including Aarrestad et al. (2021), Kim Bjerge (2020) and Guo et al. (2017). The standard implementation of convolutional layers from hls4ml library Aarrestad et al. (2021) is latency oriented. However, such implementation comes at the high expense of resource usage. CNNA Kim Bjerge (2020), on the other hand, uses a scalable architecture with an auto-scaled fixed-point format, leading to a reduction in energy consumption but an increase in latency. Among all the baselines, an efficient software-hardware co-design accelerator tailored for VGG-16 proposed in Guo et al. (2017) performs the best in terms of energy consumption and latency. To strike a balance between resource consumption and latency, our implementation uses an optimized semi-parallel computation algorithm, which parallelizes the majority of channels while pipelineing the remaining. As is obvious from the Table 10, with our custom accelerator, the PDP-compressed VGG overtakes all its competitors, achieving a $> 2.4\times$ speedup and $> 1.4\times$ reduction in energy consumption as compared to the standard conv2d Aarrestad et al. (2021) implementation. We remark that such speedup is the result of an adaptively parallel dataflow stream without any bells and whistles. This provides strong evidence that PDP compression scheme is highly practical and valuable for edge AI running on resource-limited devices.

Table 10: Energy consumption and latency on FPGA using VGG-16 as backbone. PDP-VGG-16 refers to $r_{cp} = 32$ of Table 2.

| Method | Power (W) | Latency(ms) |
|---|---|---|
| PDP-VGG-16 | **3.7** | **96.58** |
| standard conv2d Aarrestad et al. (2021) | 5.01 | 234.37 |
| CNNA Kim Bjerge (2020) | 4.74 | 2200 |
| VGG-16 Guo et al. (2017) | $\mathbf{3-4}$ | 175.43 |

## 5 Conclusion

This work first presents an elegant tensor exposition of how different approximations of the CNN kernel tensor are intimately connected to separable pointwise and depthwise convolutions. Such perspective allows an analytical lens for characterizing the importance of the associated pointwise and depthwise filters, plus the development of a first-ever channel pruning scheme for zero-flop shift layers. Using these basic components and pushing them to their limits, we demonstrate through extensive experiments an astonishing degree of compression and high performance. The proposed schemes thereby constitute a set of powerful tools for designing lightweight CNNs favorable for edge AI running on resource-limited hardware. Such unifying tensor view for various CNN modules also serves as a neat starter to CNN approximation theory, as well as a basis to discover new architectures.

## REFERENCES

Thea Aarrestad, Vladimir Loncar, Nicolò Ghielmetti, Maurizio Pierini, Sioni Summers, Jennifer Ngadiuba, Christoffer Petersson, Hampus Linander, Yutaro Iiyama, Giuseppe Di Guglielmo, Javier Duarte, Philip Harris, Dylan Rankin, Sergo Jindariani, Kevin Pedro, Nhan Tran, Edward Kreinar Mia Liu, Zhenbin Wu, and Duc Hoang. Fast convolutional neural networks on fpgas with hls4ml. arXiv, 2021. URL https://doi.org/10.1088/2632-2153/ac0ea1.

Genevera Allen. Sparse higher-order principal components analysis. In *Artificial Intelligence and Statistics*, pages 27–36. PMLR, 2012.

M. Astrid, Seung-Ik Lee, and Beom-Su Seo. Rank selection of CP-decomposed convolutional layers with variational Bayesian matrix factorization. *International Conference on Advanced Communication Technology (ICACT)*, pages 347–350, 2018.

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.

Weijie Chen, Di Xie, Yuan Zhang, and Shiliang Pu. All you need is a few shifts: Designing efficient convolutional neural networks for image classification. pages 7234–7243, 2019.

Francois Chollet. Xception: Deep learning with depthwise separable convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

Tri Dao, Albert Gu, Matthew Eichhorn, Atri Rudra, and Christopher Ré. Learning fast algorithms for linear transforms using butterfly factorizations. *Proceedings of machine learning research*, pages 1517–1527, 2019.

Tri Dao, Nimit Sohoni, Albert Gu, Matthew Eichhorn, Amit Blonder, Megan Leszczynski, Atri Rudra, and Christopher Ré. Kaleidoscope: An efficient, learnable representation for all structured linear maps. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=BkgrBgSYDS.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=YicbFdNTTy.

Linxi Fan, Shyamal Buch, Guanzhi Wang, Ryan Cao, Yuke Zhu, Juan Carlos Niebles, and Li Fei-Fei. Rubiksnet: Learnable 3d-shift for efficient video action recognition. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020.

Jianbo Guo, Yuxi Li, Weiyao Lin, Yurong Chen, and Jianguo Li. Network decoupling: From regular to depthwise separable convolutions. *arXiv preprint arXiv:1808.05517*, 2018.

Kaiyuan Guo, Song Han, Song Yao, Yu Wang, Yuan Xie, and Huazhong Yang. Software-hardware codesign for efficient neural network acceleration. IEEE Micro, 2017. URL https://doi.org/10.1109/MM.2017.39.

Kohei Hayashi, Taiki Yamaguchi, Yohei Sugawara, and Shin ichi Maeda. Einconv: Exploring unexplored tensor network decompositions for convolutional neural networks. In *Advances in Neural Information Processing Systems*, volume 32, pages 5552–5562. Curran Associates, Inc., 2019.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2(7), 2015.

Yunho Jeon and Junmo Kim. Constructing fast network through deconstruction of convolution. In *Advances in Neural Information Processing Systems*, page 5955–5965, 2018.

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. In *International Conference on Learning Representations*, 2016. URL http://arxiv.org/abs/1511.06530.

Daniel Ejnar Larsen Kim Bjerge, Jonathan Horsted Schougaard. A scalable and efficient convolutional neural network accelerator using hls for a system on chip design. arXiv, 2020. URL https://doi.org/10.48550/arXiv.2004.13075.

Jean Kossaifi, Yannis Panagakis, Anima Anandkumar, and Maja Pantic. Tensorly: Tensor learning in python. *arXiv preprint arXiv:1610.09555*, 2016.

Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342*, 2018.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan V. Oseledets, and Victor S. Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. In *International Conference on Learning Representations*, 2015. URL http://arxiv.org/abs/1412.6553.

Ji Lin, Chuang Gan, and Song Han. Tsm: Temporal shift module for efficient video understanding. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7083–7093, 2019.

Ji Lin, Wei-Ming Chen, Han Cai, Chuang Gan, and Song Han. Mcunetv2: Memory-efficient patch-based inference for tiny deep learning. In *Advances in Neural Information Processing Systems*, 2021a.

Mingbao Lin, Rongrong Ji, Yan Wang, Yichen Zhang, Baochang Zhang, Yonghong Tian, and Ling Shao. Hrank: Filter pruning using high-rank feature map. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1529–1538, 2020.

Rui Lin, Jie Ran, King Hung Chiu, Graziano Chesi, and Ngai Wong. Deformable butterfly: A highly structured and sparse linear transform. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 16145–16157. Curran Associates, Inc., 2021b. URL https://proceedings.neurips.cc/paper/2021/file/86b122d4358357d834a87ce618a55de0-Paper.pdf.

Shinichi Nakajima, Masashi Sugiyama, S. Derin Babacan, and Ryota Tomioka. Global analytic solution of fully-observed variational bayesian matrix factorization. *J. Mach. Learn. Res.*, 14(1): 1–37, jan 2013. ISSN 1532-4435.

J. Ran, R. Lin, H. H. So, G. Chesi, and N. Wong. Exploiting elasticity in tensor ranks for compressing neural networks. In *International Conference on Pattern Recognition (ICPR)*, pages 9866–9873, Los Alamitos, CA, USA, jan 2021. IEEE Computer Society. doi: 10.1109/ICPR48806.2021.9412765. URL https://doi.ieeecomputersociety.org/10.1109/ICPR48806.2021.9412765.

Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. pages 4510–4520, 2018.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Jiahao Su, Jingling Li, Bobby Bhattacharjee, and Furong Huang. Tensorial neural networks: Generalization of neural networks and application to model compression. *arXiv preprint arXiv:1805.10352*, 2018.

Mingxing Tan and Quoc Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/tan19a.html.

Ilya Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, and Alexey Dosovitskiy. Mlp-mixer: An all-mlp architecture for vision. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 24261–24272. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper/2021/file/cba0a4ee5ccd02fda0fe3f9a3e7b89fe-Paper.pdf.

Asher Trockman and J. Zico Kolter. Patches are all you need?, 2022. URL https://arxiv.org/abs/2201.09792.

Bichen Wu, Alvin Wan, Xiangyu Yue, Peter H. Jin, Sicheng Zhao, Noah Golmant, Amir Gholaminejad, Joseph E. Gonzalez, and Kurt Keutzer. Shift: A zero flop, zero parameter alternative to spatial convolutions. pages 9127–9135, 2018.

# APPENDIX

The Appendix covers the following contents:

- PDP-ResNet on CIFAR-10. (Appendix A)
- Ablation study of VGG-16 on CIFAR-10. (Appendix B)
- The pseudocode and additional visualizations for shift layer pruning. (Appendix C)
- Implementation details and experimental settings. (Appendix D)
- A comparison between SVD and VBMF initialization. (Appendix E)
- Iterative fine-tuning. (Appendix F)

## A PDP-RESNET ON CIFAR-10

To emphasize the significance of CPD initialised PDP kernels, we repeat the experiment described in Section 4.1, except this time we use ResNet-34 He et al. (2016) as the backbone. Similarly, we convert every CONV layers of a ResNet-34 to PDP kernels, excluding those PW convolutions employed in shortcut connections. In PDP-ResNet, the performance gap between CPD and randomly initialization is much narrower than in PDP-VGG. CPD initialization has a greater influence on performance when the CPD rank $r_{cp}$ is higher, as seen in Table 11. When $r_{cp} = 4$, PDP-ResNet with CPD initialization performs similarly to the randomly initialized one. When the $r_{cp}$ grows, however, the effect of CPD initialization becomes considerably more important and exceeds the randomly initialized counterpart by a large margin ($93.47\%$ vs. $90.86\%$). This is because the higher the $r_{cp}$, the more accurate of the CPD initialized PDP kernels are at approximating the original CONV kernel, resulting in a higher chance of recovering performance.

Table 11: ResNet-34 with CPD vs randomly initialized PDP kernels on CIFAR-10. Baseline has $21.28$M parameters with accuracy of $95.71\%$.

| Rank $r_{cp}$ | CR (%) | #Params(M) | Acc (%) Random/CPD |
|---|---|---|---|
| 4 | 98.81 | 0.25 | 87.02/**87.13** |
| 8 | 98.52 | 0.31 | 89.65/**90.28** |
| 16 | 97.96 | 0.44 | 90.11/**91.88** |
| 32 | 96.83 | 0.68 | 90.86/**93.47** |

## B ABLATION STUDY OF VGG-16 ON CIFAR-10

CP decomposition is an optimisation problem which tries to minimize the difference between the decomposed tensor and the target tensor. The Alternating Least Squares (ALS) is the most extensively used approach, whereas the Tensor Power Method (TPM) is another method which is known to explain the same variance with less rank than ALS Allen (2012); Astrid and Lee (2017). In this ablation study, we explore the impact of these two initialization schemes, whereas, in the paper as well as other parts of the supplemental materials, we select the best result amongst the two for simplicity. Table 12 illustrates that TPM initialization tends to perform better at lower ranks ($r_{cp} = 4, 8$) while ALS performs better at higher ranks ($r_{cp} = 16, 32$).

Furthermore, an ablation study similar to that in Section 4.2 is carried out to better the design of the PDP-VGG building block. ReLU and BN are added after the entrance PW and DW, whereas the nonlinearity and BN in the baseline VGG-16 remain intact for the exit PW. Previous research Sandler et al. (2018) suggests that ReLU may lead to information loss when the number of channels in the bottleneck is too small. As a result, ReLU after DW is removed to examine the effect. In Table 13, it is shown that ReLU and BN makes training harder if we have a very narrow bottleneck (i.e. $r_{cp} = 4$) and even leads to diverge training. By removing the ReLU after DW stabilizes the training and achieves better result. Comparing to Table 12, accuracy is up by more than $5\%$ at $r_{cp} = 8$ ($77.57\%$ vs. $72.27\%$).

Table 12: PDP-VGG using CPD with ALS vs. TPM initialization schemes on CIFAR-10.

| Setting | Initialization | | Rank | | | | Optimisation | | CR (%) | Acc (%) |
|---|---|---|---|---|---|---|---|---|---|---|
| | Random | Pretrained | 4 | 8 | 16 | 32 | ALS | TPM | | |
| 1 | √ | | √ | | | | | | 99.65 | 51.34 |
| 2 | | √ | √ | | | | √ | | | 58.76 |
| 3 | | √ | √ | | | | | √ | | **62.48** |
| 4 | √ | | | √ | | | | | 99.44 | 61.63 |
| 5 | | √ | | √ | | | √ | | | 67.23 |
| 6 | | √ | | √ | | | | √ | | **72.27** |
| 7 | √ | | | | √ | | | | 99.00 | 65.06 |
| 8 | | √ | | | √ | | √ | | | **86.09** |
| 9 | | √ | | | √ | | | √ | | 80.87 |
| 10 | √ | | | | | √ | | | 98.12 | 75.19 |
| 11 | | √ | | | | √ | √ | | | **89.73** |
| 12 | | √ | | | | √ | | √ | | 86.73 |

Table 13: Ablation of VGG-16 with CPD initialised kernels on CIFAR-10. $--$ indicates that the training diverges.

| Setting | ReLU after DW | | Rank | | | | Optimisation | | CR (%) | Acc (%) |
|---|---|---|---|---|---|---|---|---|---|---|
| | True | False | 4 | 8 | 16 | 32 | ALS | TPM | | |
| 1 | √ | | √ | | | | √ | | 99.65 | $--$ |
| 2 | √ | | √ | | | | | √ | | $--$ |
| 3 | | √ | √ | | | | √ | | | $--$ |
| 4 | | √ | √ | | | | | √ | | **55.43** |
| 5 | √ | | | √ | | | √ | | 99.44 | 54.15 |
| 6 | √ | | | √ | | | | √ | | $--$ |
| 7 | | √ | | √ | | | √ | | | 76.94 |
| 8 | | √ | | √ | | | | √ | | **77.57** |
| 9 | √ | | | | √ | | √ | | 99.00 | 81.83 |
| 10 | √ | | | | √ | | | √ | | 81.65 |
| 11 | | √ | | | √ | | √ | | | 85.31 |
| 12 | | √ | | | √ | | | √ | | **86.27** |
| 13 | √ | | | | | √ | √ | | 98.12 | 88.49 |
| 14 | √ | | | | | √ | | √ | | 88.22 |
| 15 | | √ | | | | √ | √ | | | **89.94** |
| 16 | | √ | | | | √ | | √ | | 89.2 |

## C SHIFT LAYER PRUNING

A shift module, or the building block of ShiftNet, is usually composed of one DW shift surrounded by two PW layers, interleaved with BN and nonlinearity. In additional to uneven shift pruning as described in Section 3.3.1, BN folding is also adopted to further boost the performance. BN folding Krishnamoorthi (2018) is a standard procedure to be applied preparatory to neural network quantization, wherein kernel weights $W$ are scaled using BN parameters $\gamma$ and $\sigma^2$. The BN parameters contain information summarising mini-batch statistics. Applying BN folding before shift layer pruning allows those valuable information to be captured by the principal filters. Employing a PyTorch-like notation, the detailed steps in shift layer pruning are shown in Algorithm 1.

Fig. 5- 7(a) visualize the shift distributions of ShiftResNet-20 trained on CIFAR-10 under uneven shift pruning with different pruning ratios $\phi \in \{\frac{1}{2}, \frac{1}{4}, \frac{1}{8}\}$. When the pruning ratio is low ($\phi = \frac{1}{2}$), there are plenty channels in each shift group, particularly in the last few layers. In such situation, we can observe that the channels are divided rather evenly. However, when the pruning ratio slowly increases from $\phi = \frac{1}{2}$ to $\frac{1}{8}$, the algorithm tends to assign channels in a more uneven manner, as clearly shown in first three layers (i.e. top rows of figures). Fig. 4(a) demonstrates that certain shift groups can be totally trimmed when the pruning ratio is sufficiently high (i.e. $\phi = \frac{1}{16}$ in our case).

On the other hand, Fig. 5- 7(b) describes the relative importance of each shift under different pruning ratios. Interestingly, certain shift groups appear much darker in color (i.e. larger sum) (cf. the lower right corner of Fig. 5(b)) while the channels are spread relatively uniformly (cf. the lower right corner of Fig. 5(a)). This is due to the fact that there are a few dominating terms in that shift group with very large singular values. More study is required to evaluate the impact of those highly dominant terms.

**Algorithm 1** Pseudocode of Shift Layer Pruning with PyTorch-like Style Notation.

**Input:** pruning ratio $\phi$; kernel size $k$; 4-D kernel tensors $\boldsymbol{\mathcal{W}}_1$ and $\boldsymbol{\mathcal{W}}_2$ for the entrance and exit PW of a shift module, respectively; batch normalization layers $BN_1$ and $BN_2$ following the entrance and exit PW, respectively.

**Output:** the pruned 4-D kernel tensors $\boldsymbol{\mathcal{W}}'_1$ and $\boldsymbol{\mathcal{W}}'_2$.

$c_e, c_{in}, \_, \_ = \boldsymbol{\mathcal{W}}_1.shape$ ◁ Get the shape of the 4-D kernel tensor.
$c_{out}, \_, \_, \_ = \boldsymbol{\mathcal{W}}_2.shape$
**if** fold BN **then**
  **for** $i \in \{1, 2\}$ **do**
    $\gamma, \sigma^2, \epsilon = \text{GetParams}(\text{BN}_\text{i})$ ◁ Get BN parameters.
    $\boldsymbol{\mathcal{W}}_i = \frac{\gamma}{\sqrt{\sigma^2+\epsilon}}\boldsymbol{\mathcal{W}}_i$ ◁ Scale $\boldsymbol{\mathcal{W}}_i$ with BN statistics.
  **end for**
**end if**
$n_c = c_e // k^2$ ◁ $n_c$ is the number of channels per shift before pruning.
$v^T = \boldsymbol{\mathcal{W}}_1.reshape(k^2, n_c, c_{in})$ ◁ Collect $c_i$-mode $1 \times 1$ filters.
$u = \boldsymbol{\mathcal{W}}_2.transpose(0, 1).reshape(k^2, n_c, c_{out}).transpose(2, 1)$ ◁ Collect $c_o$-mode $1 \times 1$ filters.
$A = u@v^T$ ◁ Compute the batch matrix-matrix product where $A \in \mathbb{R}^{k^2 \times c_{out} \times c_{in}}$.
$n'_c = \text{floor}(\text{n}_\text{c} * \phi)$ ◁ $n'_c$ is the number of channels per shift after pruning.
$c'_e = \text{floor}(\text{n}'_\text{c} * \text{k}^2)$ ◁ $c'_e$ is the number of channels in the bottleneck after pruning.
$U, S, V^T = \text{SVD(A)}$ ◁ Perform SVD on $A$
**if** even pruning **then**
  $U', S', V^{T'} = U[:, :, : n'_c], S[:, : n'_c], U[:, : n'_c, :]$ ◁ Select $n'_c$ evenly from each shift group.
  $\boldsymbol{\mathcal{W}}'_1 = (\text{diag(S}')^{\circ\frac{1}{2}}@\text{V}^{\text{T}'}).\text{reshape}(\text{c}'_\text{e}, \text{c}_\text{in})$ ◁ Equally split the SVs for $c_i$ and $c_o$-modes.
  $\boldsymbol{\mathcal{W}}'_2 = (U'@\text{diag(S}')^{\circ\frac{1}{2}}).\text{transpose}(0, 1).\text{reshape}(\text{c}_\text{out}, \text{c}'_\text{e})$
**else**
  $S' = \text{SelectTopkSV(S}, \text{c}'_\text{e})$ ◁ Mask all except the top $c'_e$ SVs across all shifts, $\dim(\text{S}') = \dim(\text{S})$.
  $\boldsymbol{\mathcal{W}}_1 = (\text{diag(S}')^{\circ\frac{1}{2}}@\text{V}^{\text{T}'}).\text{reshape}(\text{c}_\text{e}, \text{c}_\text{in})$ ◁
  $\boldsymbol{\mathcal{W}}_2 = (U@\text{diag(S}')^{\circ\frac{1}{2}}).\text{transpose}(0, 1).\text{reshape}(\text{c}_\text{out}, \text{c}_\text{e})$
  $\boldsymbol{\mathcal{W}}'_1 = \text{RemoveZeroRows}(\boldsymbol{\mathcal{W}}_1)$ ◁ Remove $c_o$-mode filters with SV equal to zero.
  $\boldsymbol{\mathcal{W}}'_2 = \text{RemoveZeroColumns}(\boldsymbol{\mathcal{W}}_2)$ ◁ Remove $c_i$-mode filters with SV equal to zero.
**end if**
$\boldsymbol{\mathcal{W}}'_1 = \boldsymbol{\mathcal{W}}'_1.reshape(c'_e, c_{in}, 1, 1)$
$\boldsymbol{\mathcal{W}}'_2 = \boldsymbol{\mathcal{W}}'_2.reshape(c_{out}, c'_e, 1, 1)$
**return** $\boldsymbol{\mathcal{W}}'_1, \boldsymbol{\mathcal{W}}'_2$

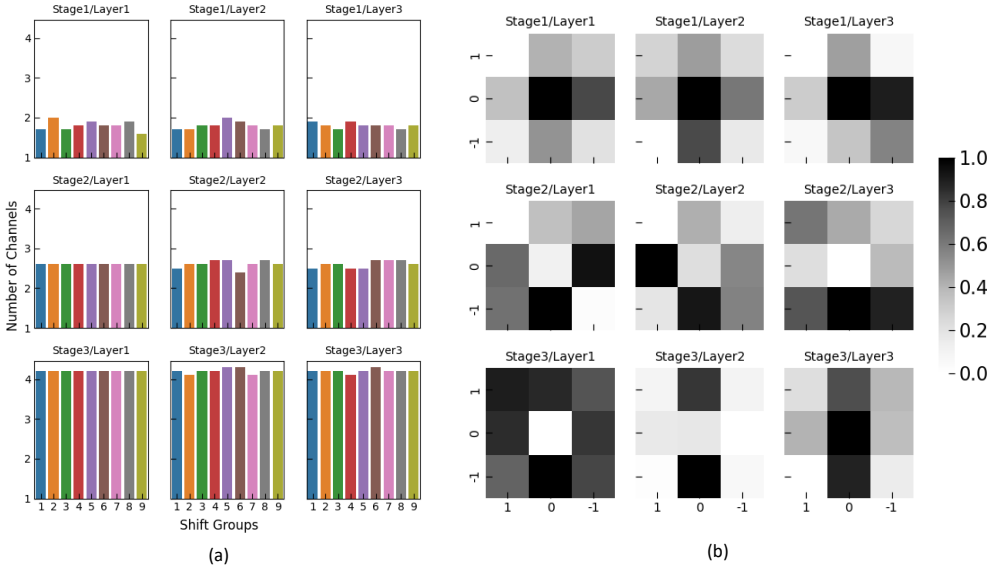



Figure 5: $\phi = \frac{1}{2}$. (a) The number of channels in each shift group; (b) The relative importance of each shift in various layers based on their singular values. Singular values are first summed at each position and then normalized within each layer.

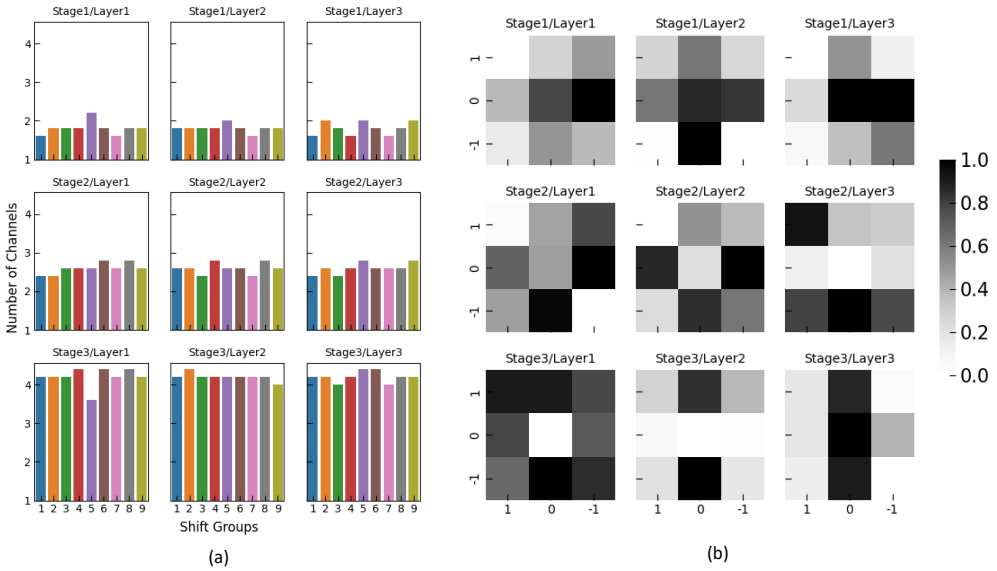



Figure 6: $\phi = \frac{1}{4}$. (a) The number of channels in each shift group; (b) The relative importance of each shift in various layers based on their singular values. Singular values are first summed at each position and then normalized within each layer.

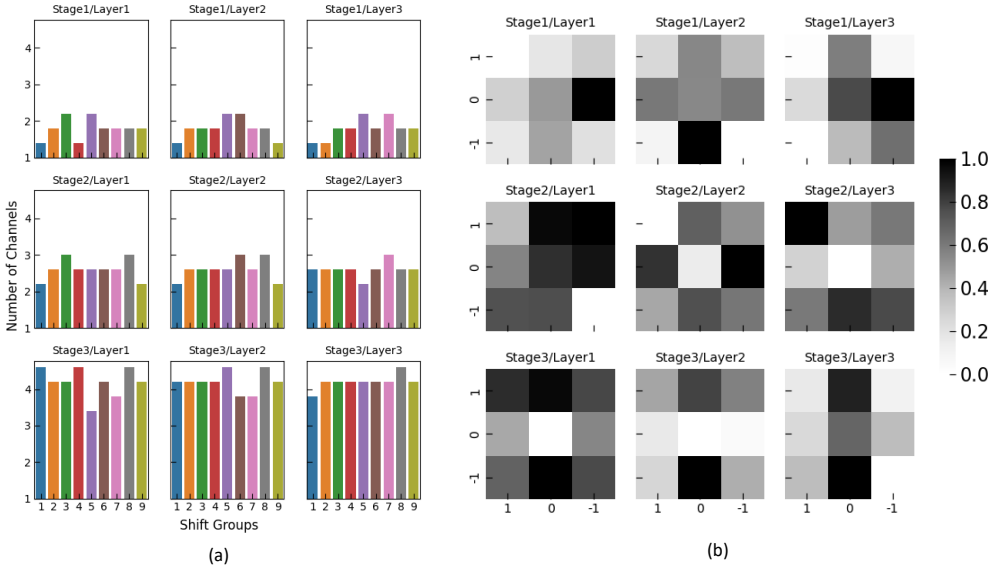



Figure 7: $\phi = \frac{1}{8}$. (a) The number of channels in each shift group; (b) The relative importance of each shift in various layers based on their singular values. Singular values are first summed at each position and then normalized within each layer.

# D IMPLEMENTATION DETAILS AND EXPERIMENTAL SETTINGS

## D.1 VGG-16 AND RESNET

We follow the setting from a popular public-domain PyTorch training framework[1] for CIFAR-10 experiments, wherein models are trained with SGD with momentum $0.9$ and weight decay $5e^{-4}$ using a batch size of $128$. Learning rate is set to $0.1$ with a cosine annealing scheduler. Random crop and random horizontal flip are employed during training. Baseline models are obtained by training from scratch for $200$ epochs. After that, pretrained CONV layers are decomposed to different configurations (i.e. PD, DP and PDP) while parameters in FC layers are reset. For ResNet-34, we also reinitialize the PW convolutions in the shortcut connections. We use TensorLy Kossaifi et al. (2016) to implement CPD for PDP configurations and PyTorch built-in SVD for PD and DP configurations. All models are fine-tuned under the same setting for another $200$ epochs.

## D.2 CONVMIXER-256/8

Our implementation is adapted from the code[2] provided by the authors Trockman and Kolter (2022), where AdamW optimizer with weight decay $0.05$ is used. After a simple grid search, we set the learning rate to $0.05$. A Slanted Triangular Learning Rates (STLR) scheduler is adopted. As described in Section 4.2, we redo the ConvMixer-256/8 by replacing the DS layers in repeating modules with standard CONV layers (Conv2d) and train it from scratch for $200$ epochs. After that, pretrained Conv2d layers are replaced with CPD initialized PDP kernels while other layers, including the stem and classifier layers, are reinitialized. The resultant models are fine-tuned for another $300$ epochs. The same set of data augmentations is used for CIFAR-10, CIFAR-100, and TinyImageNet, with the exception of TinyImageNet, where the resolution after random resized crop is $64 \times 64$ instead of $32 \times 32$. A batch size of $512$ is used for all ConvMixer experiments.

## D.3 SHIFTRESNET-20

We use the PyTorch implementation of ShiftResNet-20[3] given by the authors Wu et al. (2018), wherein we reimplement the $3 \times 3$ DW shift operation with PyTorch. Instead of adopting the training settings provided by the authors, we employ a similar setting as Appendix D.2 with minor modifications. We set both learning rate and weight decay to $0.01$ and use a batch size of $1024$ instead. After training for $200$ epochs, our baseline achieves a slightly higher accuracy in both CIFAR-10 ($93.37$ vs. $91.79$) and CIFAR-100 ($71.47$ vs. $70.77$) than the those reported by the authors. Shift layer pruning is then applied to every shift module in the pretrained network. Similar to previous experiments, parameters in all other layers are reset.

# E COMPARISON BETWEEN SVD AND VBMF INITIALIZATION

In Fig. 8, we show how ranks affect the approximation error of a given tensor using CP decomposition. The approximation error is defined as $\frac{||X-X'||_2}{||X||_2}$, where $X$ is the original tensor, and $X'$ is the reconstructed tensor. The original tensor is CONV8 (model.features[24]) of the pretrained VGG-16 on CIFAR-10. When employing VBMF to select the rank, the determined value is 270. For the ranks we considered for CPD_svd are: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50, 100, 150, 200, 270. It can be observed that the approximation error decreases as the rank increases. When the rank is fixed to $270$, the approximation error obtained by CPD_svd and CPD_vbmf are very similar.

In Fig. 9, we compare the test accuracy of VGG-16 after the selected layer being replaced at different ranks. Taking Fig. 8 into account, it is seen that a smaller approximation error does not guarantee a higher accuracy. Besides, under the same rank setting, SVD obtains higher accuracy than VBMF ($93.99\%$ vs. $93.90\%$), which reflects that VBMF is a good tool to determine the ranks but not that good to do initialization.

[1] https://github.com/kuangliu/pytorch-cifar

[2] https://github.com/locuslab/convmixer-cifar10

[3] https://github.com/alvinwan/shiftresnet-cifar

Figure 8: Approximation errors vs. ranks.

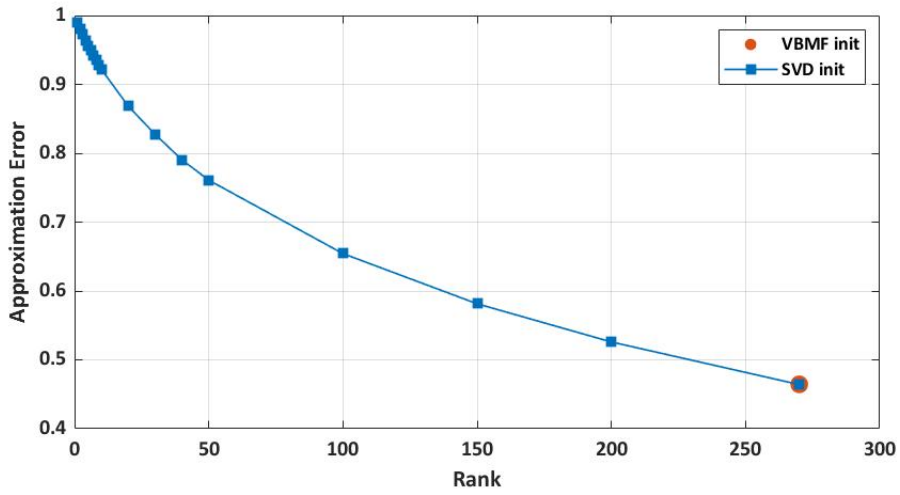
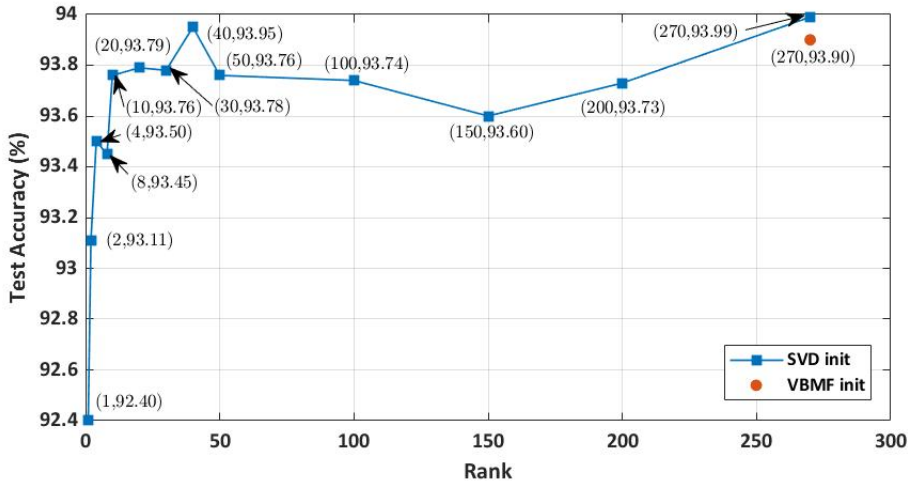



Figure 9: Test accuracy under different ranks.




## F ITERATIVE FINE-TUNING

**Implementation Details**

1. Conduct CPD individually for each layer by setting CPD_rank = 4, 8, 16 and using CPD_svd to initialize the VGG-16. Choose the model having the highest test accuracy as the baseline structure for the next round experiment.
2. Perform task 1 individually for each layer (excepting the layer already decomposed in the form step) by setting CPD_rank = 4, 8, 16 and using CPD_svd to initialize the model. Choose the model having the highest accuracy as the baseline structure for the next round.
3. Repeat step (b) until all CONV layers are decomposed.
4. The number of epochs for fine-tuning is set to be 200.

### ROUND 1

In round 1, no layer is decomposed in advance. In this round, we obtain 39 different models, each decomposes only one layer at different ranks (cf. Table 14).

| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Acc.(%) | rank = 4 | 94.04 | 92.47 | 92.08 | 91.30 | 91.75 | 92.36 | 93.03 | 93.50 | 94.02 | 94.16 | 94.08 | 94.08 | 94.08 |
| | rank = 8 | 94.12 | 93.54 | 93.43 | 92.38 | 92.63 | 92.15 | 93.24 | 93.45 | 93.90 | 94.00 | 94.23 | 93.98 | 93.91 |
| | rank = 16 | 93.79 | 93.50 | 93.63 | 93.23 | 93.31 | 92.58 | 92.98 | 93.84 | 94.17 | 94.11 | 94.07 | 93.94 | 94.30 |

Table 14: Acc. means the test accuracy, and the baseline is 94.13%. (Red) The highest test accuracy. (Blue) The performance that is comparable or even higher than the baseline.

### ROUND 2

According to the results in round 1:

1. the 13-th CONV layer is selected to be first decomposed with rank = 16.

Therefore, we obtain 36 different models, each decomposes two layers, and layer 13 is fixed to be decomposed at rank 16 (cf. Table 15). For following rounds, the tables can be interpreted similarly.

| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Acc.(%) | rank = 4 | 93.87 | 92.50 | 92.03 | 91.29 | 91.24 | 91.95 | 92.54 | 93.34 | 93.98 | 93.91 | 93.77 | 93.94 | —— |
| | rank = 8 | 93.24 | 93.01 | 92.91 | 92.14 | 92.70 | 92.48 | 92.67 | 93.79 | 93.92 | 93.98 | 94.07 | 93.90 | —— |
| | rank = 16 | 93.48 | 93.47 | 93.24 | 92.83 | 92.78 | 92.80 | 93.00 | 93.21 | 93.94 | 94.02 | 93.92 | 93.98 | √ |

Table 15: Acc. means the test accuracy, and the baseline is 94.13%. (Red) The highest test accuracy. (Blue) The performance that is comparable or even higher than the accuracy *in the last round*.

### ROUND 3

According to the results in round 2:

1. the 13-th CONV layer is selected to be first decomposed with rank = 16.
2. the 11-th CONV layer is selected to be decomposed with rank = 8.

| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Acc.(%) | rank = 4 | 93.64 | 92.21 | 91.81 | 90.99 | 90.93 | 91.73 | 92.30 | 93.21 | 93.56 | 93.85 | —— | 93.96 | —— |
| | rank = 8 | 94.01 | 92.97 | 92.67 | 91.69 | 92.12 | 91.96 | 92.52 | 93.26 | 93.76 | 93.89 | √ | 93.76 | —— |
| | rank = 16 | 93.30 | 93.46 | 93.08 | 93.05 | 92.75 | 92.21 | 92.59 | 93.60 | 93.75 | 94.19 | —— | 93.94 | √ |

Table 16: Acc. means the test accuracy, and the baseline is 94.13%. (Red) The highest test accuracy. (Blue) The performance that is comparable or even higher than the accuracy *in the last round*.

ROUND 4

According to the results in round 3:

1. the 13-th CONV layer is selected to be first decomposed with rank = 16.
2. the 11-th CONV layer is selected to be decomposed with rank = 8.
3. the 10-th CONV layer is selected to be decomposed with rank = 16.

| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Acc.(%) | rank = 4 | 93.36 | 91.95 | 91.90 | 90.58 | 90.36 | 91.13 | 92.03 | 92.51 | 93.77 | — | — | 93.72 | — |
| | rank = 8 | 93.61 | 92.78 | 92.42 | 92.24 | 91.76 | 91.87 | 92.68 | 93.55 | 93.85 | — | √ | 93.77 | — |
| | rank = 16 | 93.47 | 90.16 | 92.57 | 92.63 | 92.52 | 91.92 | 92.56 | 93.01 | 93.82 | √ | — | 94.00 | √ |

Table 17: Acc. means the test accuracy, and the baseline is 94.13%. (Red) The highest test accuracy. (Blue) The performance that is comparable or even higher than the baseline.

ROUND 5

According to the results in round 4:

1. the 13-th CONV layer is selected to be first decomposed with rank = 16.
2. the 11-th CONV layer is selected to be decomposed with rank = 8.
3. the 10-th CONV layer is selected to be decomposed with rank = 16.
4. the 12-th CONV layer is selected to be decomposed with rank = 16.

| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Acc.(%) | rank = 4 | 92.99 | 91.77 | 90.57 | 90.46 | 89.33 | 90.51 | 10.00 | 92.59 | 93.47 | — | — | — | — |
| | rank = 8 | 93.14 | 92.47 | 92.13 | 91.23 | 91.72 | 91.78 | 91.78 | 93.50 | 93.39 | — | √ | — | — |
| | rank = 16 | 93.10 | 90.82 | 92.35 | 91.88 | 92.16 | 10.00 | 92.38 | 92.64 | 93.44 | √ | — | √ | √ |

Table 18: Acc. means the test accuracy, and the baseline is 94.13%. (Red) The highest test accuracy. (Blue) The performance that is comparable or even higher than the baseline.

ROUND 6

According to the results in round 5:

1. the 13-th CONV layer is selected to be first decomposed with rank = 16.
2. the 11-th CONV layer is selected to be decomposed with rank = 8.
3. the 10-th CONV layer is selected to be decomposed with rank = 16.
4. the 12-th CONV layer is selected to be decomposed with rank = 16.
5. the 8-th CONV layer is selected to be decomposed with rank = 8.

| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Acc.(%) | rank = 4 | 92.74 | 10.00 | 89.46 | 89.13 | 88.19 | 89.63 | 91.97 | — | 92.39 | — | — | — | — |
| | rank = 8 | 92.71 | 91.80 | 90.95 | 90.56 | 89.69 | 90.90 | 91.97 | √ | 92.82 | — | √ | — | — |
| | rank = 16 | 92.85 | 10.0 | 91.06 | 90.95 | 90.53 | 91.37 | 91.87 | — | 92.88 | √ | — | √ | √ |

Table 19: Acc. means the test accuracy, and the baseline is 94.13%. (Red) The highest test accuracy. (Blue) The performance that is comparable or even higher than the baseline.

ROUND 7

According to the results in round 5:

1. the 13-th CONV layer is selected to be first decomposed with rank = 16.
2. the 11-th CONV layer is selected to be decomposed with rank = 8.
3. the 10-th CONV layer is selected to be decomposed with rank = 16.
4. the 12-th CONV layer is selected to be decomposed with rank = 16.
5. the 8-th CONV layer is selected to be decomposed with rank = 8.
6. the 9-th CONV layer is selected to be decomposed with rank = 16.

| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Acc.(%) | rank = 4 | 91.92 | 90.33 | 89.14 | 88.56 | 85.21 | 89.50 | 91.06 | — | — | — | — | — | — |
| | rank = 8 | 91.45 | 91.66 | 10.00 | 90.51 | 88.95 | 90.07 | 91.80 | √ | — | — | √ | — | — |
| | rank = 16 | 91.94 | 89.07 | 90.41 | 89.05 | 90.25 | 81.19 | 91.62 | — | √ | √ | — | √ | √ |

Table 20: Acc. means the test accuracy, and the baseline is 94.13%. (Red) The highest test accuracy. (Blue) The performance that is comparable or even higher than the baseline.

ROUND 8

According to the results in round 5:

1. the 13-th CONV layer is selected to be first decomposed with rank = 16.
2. the 11-th CONV layer is selected to be decomposed with rank = 8.
3. the 10-th CONV layer is selected to be decomposed with rank = 16.
4. the 12-th CONV layer is selected to be decomposed with rank = 16.
5. the 8-th CONV layer is selected to be decomposed with rank = 8.
6. the 9-th CONV layer is selected to be decomposed with rank = 16.
7. the 1-th CONV layer is selected to be decomposed with rank = 16.

| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Acc.(%) | rank = 4 | — | 89.36 | 89.12 | 10.0 | 86.09 | 87.63 | 10.0 | — | — | — | — | — | — |
| | rank = 8 | — | 10.0 | 89.95 | 88.98 | 88.69 | 89.21 | 91.23 | √ | — | — | √ | — | — |
| | rank = 16 | √ | 10.0 | 90.62 | 89.73 | 88.85 | 90.26 | 10.0 | — | √ | √ | — | √ | √ |

Table 21: Acc. means the test accuracy, and the baseline is 94.13%. (Red) The highest test accuracy. (Blue) The performance that is comparable or even higher than the baseline.

ROUND 9

According to the results in round 5:

1. the 13-th CONV layer is selected to be first decomposed with rank = 16.
2. the 11-th CONV layer is selected to be decomposed with rank = 8.
3. the 10-th CONV layer is selected to be decomposed with rank = 16.
4. the 12-th CONV layer is selected to be decomposed with rank = 16.
5. the 8-th CONV layer is selected to be decomposed with rank = 8.
6. the 9-th CONV layer is selected to be decomposed with rank = 16.
7. the 1-th CONV layer is selected to be decomposed with rank = 16.
8. the 7-th CONV layer is selected to be decomposed with rank = 8.

| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Acc.(%) | rank = 4 | —— | 84.84 | 86.51 | 85.05 | 65.69 | 79.19 | —— | —— | —— | —— | —— | —— | —— |
| | rank = 8 | —— | 89.46 | 89.26 | 88.42 | 85.70 | 89.47 | √ | √ | —— | —— | √ | —— | —— |
| | rank = 16 | √ | 88.79 | 89.44 | 87.75 | 87.43 | 90.15 | —— | —— | √ | √ | —— | √ | √ |

Table 22: Acc. means the test accuracy, and the baseline is 94.13%. (Red) The highest test accuracy. (Blue) The performance that is comparable or even higher than the baseline.

### ROUND 10

According to the results in round 5:

1. the 13-th CONV layer is selected to be first decomposed with rank = 16.
2. the 11-th CONV layer is selected to be decomposed with rank = 8.
3. the 10-th CONV layer is selected to be decomposed with rank = 16.
4. the 12-th CONV layer is selected to be decomposed with rank = 16.
5. the 8-th CONV layer is selected to be decomposed with rank = 8.
6. the 9-th CONV layer is selected to be decomposed with rank = 16.
7. the 1-th CONV layer is selected to be decomposed with rank = 16.
8. the 7-th CONV layer is selected to be decomposed with rank = 8.
9. the 6-th CONV layer is selected to be decomposed with rank = 16.

| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Acc.(%) | rank = 4 | —— | 10.00 | 10.00 | 10.00 | 81.95 | —— | —— | —— | —— | —— | —— | —— | —— |
| | rank = 8 | —— | 85.61 | 85.42 | 10.00 | 10.00 | —— | √ | √ | —— | —— | √ | —— | —— |
| | rank = 16 | √ | 82.50 | 87.64 | 84.68 | 10.00 | √ | —— | —— | √ | √ | —— | √ | √ |

Table 23: Acc. means the test accuracy, and the baseline is 94.13%. (Red) The highest test accuracy. (Blue) The performance that is comparable or even higher than the baseline.

### ROUND 11

According to the results in round 5:

1. the 13-th CONV layer is selected to be first decomposed with rank = 16.
2. the 11-th CONV layer is selected to be decomposed with rank = 8.
3. the 10-th CONV layer is selected to be decomposed with rank = 16.
4. the 12-th CONV layer is selected to be decomposed with rank = 16.
5. the 8-th CONV layer is selected to be decomposed with rank = 8.
6. the 9-th CONV layer is selected to be decomposed with rank = 16.
7. the 1-th CONV layer is selected to be decomposed with rank = 16.
8. the 7-th CONV layer is selected to be decomposed with rank = 8.
9. the 6-th CONV layer is selected to be decomposed with rank = 16.
10. the 3-th CONV layer is selected to be decomposed with rank = 16.

### ROUND 12

According to the results in round 5:

1. the 13-th CONV layer is selected to be first decomposed with rank = 16.
2. the 11-th CONV layer is selected to be decomposed with rank = 8.

| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Acc.(%) | rank = 4 | — | 80.02 | — | 10.00 | 10.00 | — | — | — | — | — | — | — | — |
| | rank = 8 | — | 84.53 | — | 10.00 | 65.53 | — | √ | √ | — | — | √ | — | — |
| | rank = 16 | √ | 10.00 | √ | 82.14 | 82.92 | √ | — | — | √ | √ | — | √ | √ |

Table 24: Acc. means the test accuracy, and the baseline is 94.13%. (Red) The highest test accuracy. (Blue) The performance that is comparable or even higher than the baseline.

3. the 10-th CONV layer is selected to be decomposed with rank = 16.
4. the 12-th CONV layer is selected to be decomposed with rank = 16.
5. the 8-th CONV layer is selected to be decomposed with rank = 8.
6. the 9-th CONV layer is selected to be decomposed with rank = 16.
7. the 1-th CONV layer is selected to be decomposed with rank = 16.
8. the 7-th CONV layer is selected to be decomposed with rank = 8.
9. the 6-th CONV layer is selected to be decomposed with rank = 16.
10. the 3-th CONV layer is selected to be decomposed with rank = 16.
11. the 2-th CONV layer is selected to be decomposed with rank = 8.

| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Acc.(%) | rank = 4 | — | — | — | 10.00 | 73.89 | — | — | — | — | — | — | — | — |
| | rank = 8 | — | √ | — | 81.85 | 73.50 | — | √ | √ | — | — | √ | — | — |
| | rank = 16 | √ | — | √ | 79.63 | 80.44 | √ | — | — | √ | √ | — | √ | √ |

Table 25: Acc. means the test accuracy, and the baseline is 94.13%. (Red) The highest test accuracy. (Blue) The performance that is comparable or even higher than the baseline.

ROUND 13

According to the results in round 5:

1. the 13-th CONV layer is selected to be first decomposed with rank = 16.
2. the 11-th CONV layer is selected to be decomposed with rank = 8.
3. the 10-th CONV layer is selected to be decomposed with rank = 16.
4. the 12-th CONV layer is selected to be decomposed with rank = 16.
5. the 8-th CONV layer is selected to be decomposed with rank = 8.
6. the 9-th CONV layer is selected to be decomposed with rank = 16.
7. the 1-th CONV layer is selected to be decomposed with rank = 16.
8. the 7-th CONV layer is selected to be decomposed with rank = 8.
9. the 6-th CONV layer is selected to be decomposed with rank = 16.
10. the 3-th CONV layer is selected to be decomposed with rank = 16.
11. the 2-th CONV layer is selected to be decomposed with rank = 8.
12. the 4-th CONV layer is selected to be decomposed with rank = 8.

| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Acc.(%) | rank = 4 | — | — | — | — | 10.00 | — | — | — | — | — | — | — | — |
| | rank = 8 | — | √ | — | √ | 69.50 | — | √ | √ | — | — | √ | — | — |
| | rank = 16 | √ | — | √ | — | 10.00 | √ | — | — | √ | √ | — | √ | √ |

Table 26: Acc. means the test accuracy, and the baseline is 94.13%. (Red) The highest test accuracy. (Blue) The performance that is comparable or even higher than the baseline.

According to the results shown in rounds 1 - 13, it is seen that the accuracy decreases significantly since round 10. It is also worth noting that the accuracy obtained by adaptive substitution is worse than that of compressing all CONV layers at one time with pretrained parameters.