Compressing Neural Networks: Towards Determining the Optimal Layer-wise Decomposition

Anonymous Author(s)
Affiliation
Address
email

Abstract

We present a novel global compression framework for deep neural networks that automatically analyzes each layer to identify the optimal per-layer compression ratio, while simultaneously achieving the desired overall compression. Our algorithm hinges on the idea of compressing each convolutional (or fully-connected) layer by "slicing" its channels into multiple groups and decomposing each group via low-rank decomposition. At the core of our algorithm is the derivation of layer-wise error bounds from the Eckart–Young–Mirsky theorem. We then leverage these bounds to frame the compression problem as an optimization problem where we wish to minimize the maximum compression error across layers and propose an efficient algorithm towards a solution. Our experiments indicate that our method outperforms existing low-rank compression approaches across a wide range of networks and data sets. We believe that our results open up new avenues for future research into the global performance-size trade-offs of modern neural networks.

1 Introduction

Neural network compression entails taking an existing model and reducing its computational and memory footprint in order to enable the deployment of large-scale networks in resource-constrained environments. Beyond inference time efficiency, compression can also yield novel insights into the design (Liu et al., 2019b), training (Liebenwein et al., 2021), and theoretical properties (Arora et al., 2018) of neural networks.

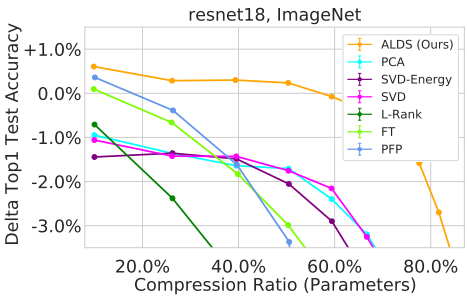


Figure 1: **ALDS**, *Automatic Layer-wise Decomposition Selector*, can compress up to 60% of parameters on a ResNet18 (ImageNet), 3x more compared to baselines. Detailed results are described in Section 3.

Among existing compression techniques – which include quantization (Wu et al., 2016), distillation (Hinton et al., 2015), and pruning (Han et al., 2015) – *low-rank compression* aims at decomposing a layer's weight tensor into a tuple of smaller low-rank tensors. Such compression techniques may build upon the rich literature on low-rank decomposition and its numerous applications outside deep learning such as dimensionality reduction (Laparra et al., 2015) or spectral clustering (Peng et al., 2015). Moreover, low-rank compression can be readily implemented in any machine learning framework by replacing the existing layer with a set of smaller layers without the need for, e.g., sparse linear algebra support.

Within deep learning, we encounter two related, yet distinct challenges when applying low-rank compression. On the one hand, each layer should be efficiently decomposed (the "local step") and, on the other hand, we need to balance the amount of compression in each layer in order to achieve a

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

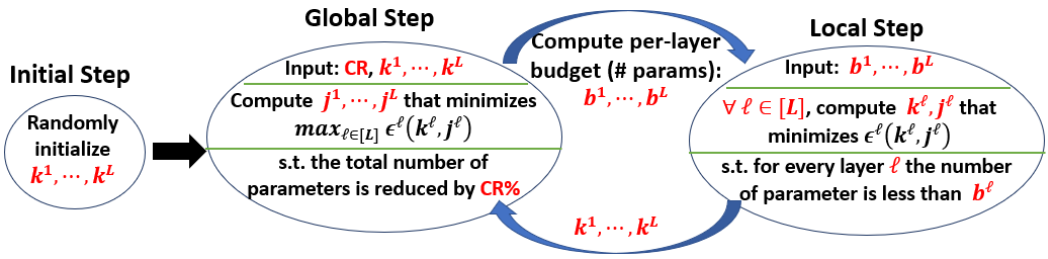


Figure 2: **ALDS Overview**. The framework consists of a global and local step, see Section 2.

desired overall compression ratio with minimal loss in the predictive power of the network (the "global step"). While the "local step", i.e., designing the most efficient layer-wise decomposition method, has traditionally received lots of attention (Denton et al., 2014; Garipov et al., 2016; Jaderberg et al., 2014; Kim et al., 2015b; Lebedev et al., 2015; Novikov et al., 2015), the "global step" has only recently been the focus of attention in research, e.g., see the recent works of Alvarez and Salzmann (2017); Idelbayev and Carreira-Perpinán (2020); Xu et al. (2020).

In this paper, we set out to design a framework that simultaneously accounts for both the local and global step. Our proposed solution, termed *Automatic Layer-wise Decomposition Selector* (ALDS), addresses this challenge by iteratively optimizing for each layer's decomposition method (local step) and the low-rank compression itself while accounting for the maximum error incurred across layers (global step). In Figure 1, we show how ALDS outperforms existing approaches on the common ResNet18 (ImageNet) benchmark (60% compression compared to $\sim$20% for baselines).

Efficient layer-wise decomposition. Our framework relies on a straightforward SVD-based decomposition of each layer. Inspired by Denton et al. (2014); Idelbayev and Carreira-Perpinán (2020); Jaderberg et al. (2014) and others, we decompose each layer by first folding the weight tensor into a matrix before applying SVD and encoding the resulting pair of matrices as two separate layers.

Enhanced decomposition via multiple subsets. A natural generalization of low-rank decomposition methods entails splitting the matrix into multiple subsets (subspaces) before compressing each subset individually. In the context of deep learning, this was investigated before for individual layers (Denton et al., 2014), including embedding layers (Chen et al., 2018; Maalouf et al., 2021). We take this idea further and incorporate it into our layer-wise decomposition method as additional hyper-parameter in terms of the number of subsets. Thus, our local step, i.e., the layer-wise decomposition, constitutes of choosing the number of subsets (k^ℓ) for each layer and the rank (j^ℓ).

Towards a global solution for low-rank compression. We can describe the optimal solution for low-rank compression as the set of hyperparameters (number of subspaces k^ℓ and rank j^ℓ for each layer in our case) that minimizes the drop in accuracy of the compressed network. While finding the globally optimal solution is NP-complete, we propose ALDS as an efficiently solvable alternative that enables us to search for a locally optimal solution in terms of the maximum relative error incurred across layers. To this end, we derive spectral norm bounds based on the Eckhart-Young-Mirsky Theorem for our layer-wise decomposition method to describe the trade-off between the layer compression and the incurred error. Leveraging our bounds we can then efficiently optimize over the set of possible per-layer decompositions. An overview of ALDS is shown in Figure 2.

2 Method

In this section, we introduce our compression framework consisting of a layer-wise decomposition method (Section 2.1), a global selection mechanism to simultaneously compress all layers of a network (Section 2.2), and an optimization procedure (ALDS) to solve the selection problem (Section 2.3).

2.1 Local Layer Compression

We detail our low-rank compression scheme for convolutional layers below and note that it readily applies to fully-connected layers as well as a special case of convolutions with a 1×1 kernel.

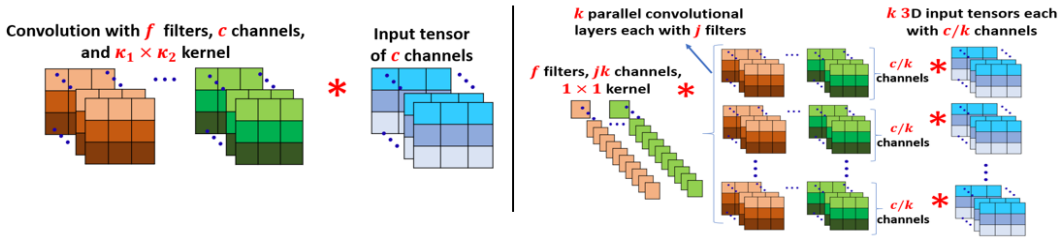


Figure 3: **Left: 2D convolution. right: decomposition used for ALDS.** For a $f \times c \times \kappa_1 \times \kappa_2$ convolution with f filters, c channels, and $\kappa_1 \times \kappa_2$ kernel, our per-layer decomposition consists: (1) k parallel $j \times {}^{c}/_{k} \times \kappa_1 \times \kappa_2$ convolutions; (2) a single $f \times kj \times 1 \times 1$ convolution applied on the first layer's (stacked) output.

Compressing convolutions via SVD. Given a convolutional layer of f filters, c channels, and a $\kappa_1 \times \kappa_2$ kernel we denote the corresponding weight tensor by $\mathcal{W} \in \mathbb{R}^{f \times c \times \kappa_1 \times \kappa_2}$. Following Denton et al. (2014); Idelbayev and Carreira-Perpinán (2020); Wen et al. (2017) and others, we can then interpret the layer as a linear layer of shape $f \times c\kappa_1\kappa_2$ and the corresponding rank j-approximation as two subsequent linear layers of shape $f \times j$ and $j \times c\kappa_1\kappa_2$. Mapped back to convolutions, this corresponds to a $j \times c \times \kappa_1 \times \kappa_2$ convolution followed by a $f \times j \times 1 \times 1$ convolution.

Multiple subspaces. Following the intuition outlined in Section 1 we propose to cluster the columns of the layer's weight matrix into $k \geq 2$ separate subspaces before applying SVD to each subset. To this end, we may consider any clustering method, such as k-means or projective clustering (Chen et al., 2018; Maalouf et al., 2021). However, such methods require expensive approximation algorithms which would limit our ability to incorporate them into an optimization-based compression framework as outlined in Section 2.2. In addition, arbitrary clustering may require re-shuffling the input tensors which could lead to significant slow-downs during inference. We instead opted for a simple clustering method, namely *channel slicing*, where we simply divide the c input channels of the layer into k subsets each containing at most $\lceil c/k \rceil$ consecutive input channels. Unlike other methods, channel slicing is efficiently implementable, e.g., as grouped convolutions in PyTorch (Paszke et al., 2017) and ensures practical speed-ups subsequent to compressing the network.

Overview of per-layer decomposition. In summary, for given integers $j, k \geq 1$ and a 4D tensor $\mathcal{W} \in \mathbb{R}^{f \times c \times \kappa_1 \times \kappa_2}$ representing a convolution the per-layer compression method proceeds as follows:

1. PARTITION the channels of the convolutional layer into k subsets, where each subset has at most $\lceil c/k \rceil$ consecutive channels, resulting in k convolutional tensors $\{\mathcal{W}_i\}_{i=1}^{k}$ where $\mathcal{W}_i \in \mathbb{R}^{f \times c_i \times \kappa_1 \times \kappa_2}$, and $\sum_{i=1}^{k} c_i = c$.
2. DECOMPOSE each tensor $\mathcal{W}_i$, $i \in [k]$, by building the corresponding weight matrix $W_i \in \mathbb{R}^{f \times c_i \kappa_1 \kappa_2}$, c.f. Figure 3, computing its j-rank approximation, and factoring it into a pair of smaller matrices U_i of f rows and j columns and V_i of j rows and $c_i\kappa_1\kappa_2$ columns.
3. REPLACE the original layer in the network by 2 layers. The first consists of k parallel convolutions, where the ith parallel layer, $i \in [k]$, is described by the tensor $\mathcal{V}_i \in \mathbb{R}^{j \times c_i \times \kappa_1 \times \kappa_2}$ which can be constructed from the matrix V_i (j filters, c_i channels, $\kappa_1 \times \kappa_2$ kernel). The second layer is constructed by reshaping each matrix U_i, $i \in [k]$, to obtain the tensor $\mathcal{U}_i \in \mathbb{R}^{f \times j \times 1 \times 1}$, and then channel stacking all k tensors $\mathcal{U}_1, \cdots, \mathcal{U}_k$ to get a single tensor of shape $f \times kj \times 1 \times 1$.

The decomposed layer is depicted in Figure 3. The resulting layer pair has $jc\kappa_1\kappa_2$ and jfk parameters, respectively, which implies a parameter reduction from $fc\kappa_1\kappa_2$ to $j(fk + c\kappa_1\kappa_2)$.

2.2 Global Network Compression

In the previous section, we introduced our layer compression scheme. We note that in practice we usually want to compress an entire network consisting of L layers up to a pre-specified relative reduction in parameters ("compression ratio" or **CR**). However, it is generally unclear how much each layer $\ell \in [L]$ should be compressed in order to achieve the desired **CR** while incurring a minimal increase in loss. Unfortunately, this optimization problem is NP-complete as we would have to check every combination of layer compression resulting in the desired **CR** in order to optimally compress each layer. On the other hand, simple heuristics, e.g., constant per-layer compression ratios, may lead to sub-optimal results, see Section 3. To this end, we propose an efficiently solvable global

compression framework based on minimizing the maximum relative error incurred across layers. We describe each component of our optimization procedure in greater detail below.

The layer-wise relative error as proxy for the overall loss. Since the true cost (the additional loss incurred after compression) would result in an NP-complete problem, we replace the true cost by a more efficient proxy. Specifically, we consider the maximum relative error $\varepsilon \coloneqq \max_{\ell \in [L]} \varepsilon^\ell$ across layers, where ε^ℓ denotes the theoretical maximum relative error in the ℓth layer as described in Theorem 1 below. We choose to minimize this particular cost because: (i) minimizing the maximum relative error ensures that no layer incurs an unreasonably large error that might otherwise get propagated or amplified; (ii) relying on a relative instead of an absolute error notion is preferred as scaling between layers may arbitrarily change, e.g., due to batch normalization, and thus the absolute scale of layer errors may not be indicative of the increase in loss; and (iii) the per-layer relative error has been shown to be intrinsically linked to the theoretical compression error, e.g., see the works of Arora et al. (2018) and Baykal et al. (2019a) thus representing a natural proxy for the cost.

Definition of per-layer relative error. Let $\mathcal{W}^\ell \in \mathbb{R}^{f^\ell \times c^\ell \times \kappa_1^\ell \times \kappa_2^\ell}$ and $W^\ell \in \mathbb{R}^{f^\ell \times c^\ell \kappa_1^\ell \kappa_2^\ell}$ denote the weight tensor and corresponding folded matrix of layer ℓ, respectively. The per-layer relative error ε^ℓ is hereby defined as the relative difference in the operator norm between the matrix $\hat{W}^\ell$ (that corresponds to the compressed weight tensor $\hat{\mathcal{W}}^\ell$) and the original weight matrix W^ℓ in layer ℓ, i.e,.

$$\varepsilon^\ell \coloneqq \|\hat{W}^\ell - W^\ell\| / \|W^\ell\|. \tag{1}$$

Note that while in practice our method decomposes the original layer into a set of separate layers (see Section 2.1), for the purpose of deriving the resulting error we re-compose the compressed layers into the overall matrix operator $\hat{W}^\ell$, i.e., $\hat{W}^\ell = [U_1^\ell V_1^\ell \cdots U_{k^\ell}^\ell V_{k^\ell}^\ell]$, where $U_i^\ell V_i^\ell$ is the factorization of the ith cluster (set of columns) in the ℓth layer, for every $\ell \in [L]$ and $i \in [k^\ell]$, see supplementary material for more details. We note that the operator norm $\|\cdot\|$ for a convolutional layer thus signifies the maximum relative error incurred for an individual output patch ("pixel") across all output channels.

Derivation of relative error bounds. We now derive an error bound that enables us to describe the per-layer relative error in terms of the compression hyperparameters j^ℓ and k^ℓ, i.e., $\varepsilon^\ell = \varepsilon^\ell(k^\ell, j^\ell)$. This will prove useful later on as we have to repeatedly query the relative error in our optimization procedure. The error bound is described in the following.

Theorem 1. *Given a layer matrix W^ℓ and the corresponding low-rank approximation $\hat{W}^\ell$, the relative error $\varepsilon^\ell \coloneqq \|\hat{W}^\ell - W^\ell\| / \|W^\ell\|$ is bounded by*

$$\varepsilon^\ell \leq {}^{\sqrt{k}}/_{\alpha_1} \cdot \max_{i \in [k]} \alpha_{i,j+1}, \tag{2}$$

where $\alpha_{i,j+1}$ is the $j+1$ largest singular value of the matrix W_i^ℓ, for every $i \in [k]$, and $\alpha_1 = \|W^\ell\|$ is the largest singular value of W^ℓ.

Proof. First, we recall the matrices $W_1^\ell, \cdots, W_k^\ell$ and we denote the SVD factorization for each of them by: $W_i^\ell = \tilde{U}_i^\ell \tilde{\Sigma}_i^\ell \tilde{V}_i^\ell$. Now, observe that for every $i \in [k]$, the matrix $\hat{W}_i^\ell$ is the j-rank approximation of W_i^ℓ. Hence, the SVD factorization of $\hat{W}_i^\ell$ can be writen as $\hat{W}_i^\ell = \tilde{U}_i^\ell \hat{\Sigma}_i^\ell \tilde{V}_i^{\ell^T}$, where $\hat{\Sigma}_i^\ell \in \mathbb{R}^{f \times d}$ is a diagonal matrix such that its first j-diagonal entries are equal to the first j-entries on the diagonal of $\tilde{\Sigma}_i^\ell$, and the rest are zeros. Hence,

$$\begin{aligned} W^\ell - \hat{W}^\ell &= [W_1^\ell - \hat{W}_1^\ell, \cdots, W_k^\ell - \hat{W}_k^\ell] = [\tilde{U}_1^\ell(\tilde{\Sigma}_1^\ell - \hat{\Sigma}_1^\ell)\tilde{V}_1^\ell, \cdots, \tilde{U}_k^\ell(\tilde{\Sigma}_k^\ell - \hat{\Sigma}_k^\ell)\tilde{V}_k^\ell] \\ &= [\tilde{U_1^\ell} \cdots \tilde{U}_k^\ell] \operatorname{diag}\left((\tilde{\Sigma}_1^\ell - \hat{\Sigma}_1^\ell)\tilde{V}_1^\ell, \ldots, (\tilde{\Sigma}_k^\ell - \hat{\Sigma}_k^\ell)\tilde{V}_k^\ell\right). \end{aligned} \tag{3}$$

By (3) and by the triangle inequality, we have that

$$\left\|W^\ell - \hat{W}^\ell\right\| \leq \left\|\left[\tilde{U_1^\ell} \cdots \tilde{U}_k^\ell\right]\right\| \left\|\operatorname{diag}\left((\tilde{\Sigma}_1^\ell - \hat{\Sigma}_1^\ell)\tilde{V}_1^\ell, \ldots, (\tilde{\Sigma}_k^\ell - \hat{\Sigma}_k^\ell)\tilde{V}_k^\ell\right)\right\|. \tag{4}$$

Now, we observe that

$$\left\|\left[\tilde{U_1^\ell} \cdots \tilde{U}_k^\ell\right]\right\|^2 = \left\|\left[\tilde{U_1^\ell} \cdots \tilde{U}_k^\ell\right]\left[\tilde{U_1^\ell} \cdots \tilde{U}_k^\ell\right]^T\right\| = \|\operatorname{diag}(k, \ldots, k)\| = k. \tag{5}$$

Finally, we show that

$$\left\|\operatorname{diag}\left((\tilde{\Sigma}_1^\ell - \hat{\Sigma}_1^\ell)\tilde{V}_1^\ell, \ldots, (\tilde{\Sigma}_k^\ell - \hat{\Sigma}_k^\ell)\tilde{V}_k^\ell\right)\right\| = \max_{i\in[k]} \left\|(\tilde{\Sigma}_i^\ell - \hat{\Sigma}_i^\ell)\tilde{V}_i^\ell\right\| \tag{6}$$

$$= \max_{i\in[k]} \left\|(\tilde{\Sigma}_i^\ell - \hat{\Sigma}_i^\ell)\right\| = \max_{i\in[k]} \alpha_{i,j+1}, \tag{7}$$

where the second equality holds since the columns of V are orthogonal and the last equality holds according to the Eckhart-Young-Mirsky Theorem (Theorem 2.4.8 of Golub and Van Loan (2013)). Plugging (7) and (5) into (4) concludes the proof. □

Resulting network size. Let $\theta = \{\mathcal{W}^\ell\}_{\ell=1}^L$ denote the set of weights for the L layers and note that the number of parameters in layer ℓ is given by $|\mathcal{W}^\ell| = f^\ell c^\ell \kappa_1^\ell \kappa_2^\ell$ and $|\theta| = \sum_{\ell\in[L]} |\mathcal{W}^\ell|$. Moreover, note that $|\hat{\mathcal{W}}^\ell| = j^\ell(k^\ell f^\ell + c^\ell \kappa_1^\ell \kappa_2^\ell)$ if decomposed, $\hat{\theta} = \{\hat{\mathcal{W}}^\ell\}_{\ell=1}^L$, and $|\hat{\theta}| = \sum_{\ell\in[L]} |\hat{\mathcal{W}}^\ell|$. The overall compression ratio is thus given by $1 - {|\hat{\theta}|}/{|\theta|}$ where we neglected other parameters for ease of exposition. Observe that the layer budget $|\hat{\mathcal{W}}^\ell|$ is fully determined by k^ℓ, j^ℓ just like the error bound.

Global Network Compression. Putting everything together we obtain the following formulation for the optimal per-layer budget:

$$\begin{aligned} \varepsilon_{opt} = \min_{\{j^\ell, k^\ell\}_{\ell=1}^L} \quad & \max_{\ell\in[L]} \varepsilon^\ell(k^\ell, j^\ell) \\ \text{subject to} \quad & 1 - {|\hat{\theta}(k^1, j^1, \ldots, k^L, j^L)|}/{|\theta|} \leq \mathbf{CR}, \end{aligned} \tag{8}$$

where $\mathbf{CR}$ denotes the desired overall compression ratio. Thus optimally allocating a per-layer budget entails finding the optimal number of subspaces k^ℓ and ranks j^ℓ for each layer constrained by the desired overall compression ratio $\mathbf{CR}$.

2.3 Automatic Layer-wise Decomposition Selector (ALDS)

We propose to solve (8) by iteratively optimizing $k^1, \ldots, k^L$ and $j^1, \ldots, j^L$ until convergence akin of an EM-like algorithm as shown in Algorithm 1 and Figure 2. Specifically, for a given set of weights θ and desired compression ratio $\mathbf{CR}$ we first randomly initialize the number of subspaces $k^1, \ldots, k^L$ for each layer (Line 2). Based on given values for each k^ℓ we then solve for the optimal ranks $j^1, \ldots, j^L$ such that the overall compression ratio is satisfied (Line 4). Note that the maximum error ε is minimized if all errors are equal. Thus solving for the ranks in Line 4 entails guessing a value for ε, computing the resulting network size, and repeating the process until the desired $\mathbf{CR}$ is satisfied, e.g. via binary search. Subsequently, we re-assign the number of subspaces k^ℓ for each layer by iterating through the finite set of possible values for k^ℓ (Line 7) and choosing the one that minimizes the relative error for the current layer budget b^ℓ (computed in Line 6). We then iteratively repeat both steps until convergence (Lines 3-8). To improve the quality of the local optimum we

Algorithm 1 ALDS(θ, $\mathbf{CR}$, n_{seed})

Input: θ: network parameters; $\mathbf{CR}$: overall compression ratio; n_{seed}: number of random seeds to initialize
Output: $k^1, \ldots, k^L$: number of subspaces for each layer; $j_1, \ldots, j^L$: desired rank per subspace for each layer

1: **for** $i \in [n_{\text{seed}}]$ **do**
2: $\quad k^1, \ldots, k^L \leftarrow$ RANDOMINIT()
3: $\quad$ **while** not converged **do**
4: $\quad\quad j^1, \ldots, j^L \leftarrow$ OPTIMALRANKS($\mathbf{CR}, k^1, \ldots, k^L$) ▷ Global step: choose such that $\varepsilon^1 = \ldots = \varepsilon^L$
5: $\quad\quad$ **for** $\ell \in [L]$ **do**
6: $\quad\quad\quad b^\ell \leftarrow j^\ell(k^\ell f^\ell + c^\ell \kappa_1^\ell \kappa_2^\ell)$ ▷ resulting layer budget
7: $\quad\quad\quad k^\ell \leftarrow$ OPTIMALSUBSPACES(b^ℓ) ▷ Local step: minimize error bound for a given layer budget
8: $\quad\quad$ **end for**
9: $\quad$ **end while**
10: $\quad \varepsilon_i =$ RECORDERROR($k^1, \ldots, k^L, j^1, \ldots, j^L$)
11: **end for**
12: **return** $k^1, \ldots, k^L, j^1, \ldots, j^L$ from $i_{\text{best}} = \operatorname{argmin}_i \varepsilon_i$

initialize the procedure with multiple random seeds (Lines 1-11) and pick the allocation with the lowest error (Line 12). We note that we make repeated calls to our decomposition subroutine (i.e. SVD; Lines 4, 7) highlighting the necessity for it to be efficient and cheap to evaluate.

Extensions. Here, we use SVD with multiple subspaces as per-layer compression method. However, we note that ALDS can be readily extended to any desired *set* of low-rank compression techniques. Specifically, we can replace the local step of Line 7 by a search over different methods, e.g., Tucker decomposition, PCA, or other SVD compression schemes, and return the best method for a given budget. In general, we may combine ALDS with any low-rank compression as long as we can efficiently evaluate the per-layer error of the compression scheme. In the supplementary material, we discuss some preliminary results that highlight the promising performance of such extensions.

3 Experiments

Networks and datasets. We study various standard network architectures and data sets. Particularly, we test our compression framework on ResNet20 (He et al., 2016), DenseNet22 (Huang et al., 2017), WRN16-8 (Zagoruyko and Komodakis, 2016), and VGG16 (Simonyan and Zisserman, 2015) on CIFAR10 (Torralba et al., 2008); ResNet18 (He et al., 2016), and AlexNet (Krizhevsky et al., 2012) on ImageNet (Russakovsky et al., 2015); and on Deeplab-V3 (Chen et al., 2017) with a ResNet50 backbone on Pascal VOC segmentation data (Everingham et al., 2015).

Baselines. We compare ALDS to a diverse set of low-rank compression techniques. Specifically, we have implemented PCA (Zhang et al., 2015a), SVD with energy-based layer allocation (SVD-Energy) following Alvarez and Salzmann (2017); Wen et al. (2017); Xu et al. (2018), and simple SVD with constant per-layer compression (Denton et al., 2014). Additionally, we also implemented the recent learned rank selection mechanism (L-Rank) of Idelbayev and Carreira-Perpinán (2020). Finally, we implemented two recent filter pruning methods, i.e., FT of Li et al. (2016) and PFP of Liebenwein et al. (2020), as alternative compression techniques for densely compressed networks. Additional comparisons on ImageNet are provided in Section 3.2.

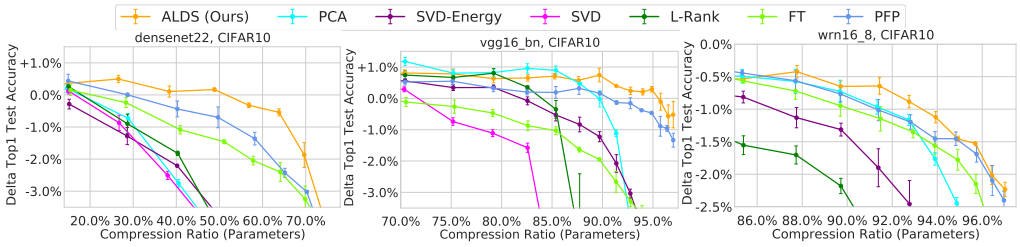
Retraining. For our experiments, we study one-shot and iterative learning rate rewinding inspired by Renda et al. (2020) for various amounts of retraining. In particular, we consider the following unified compress-retrain pipeline across all methods:

1. TRAIN for e epochs according to the standard training schedule for the respective network.
2. COMPRESS the network according to the chosen method.
3. RETRAIN the network for r epochs using the training hyperparameters from epochs $[e - r, e]$.
4. ITERATIVELY repeat 1.-3. after projecting the decomposed layers back (optional).

Reporting metrics. We report Top-1, Top-5, and IoU test accuracy as applicable for the respective task. For each compressed network we also report the compression ratio, i.e., relative reduction, in terms of parameters and floating point operations denoted by CR-P and CR-F, respectively. Each experiment was repeated 3 times and we report mean and standard deviation.

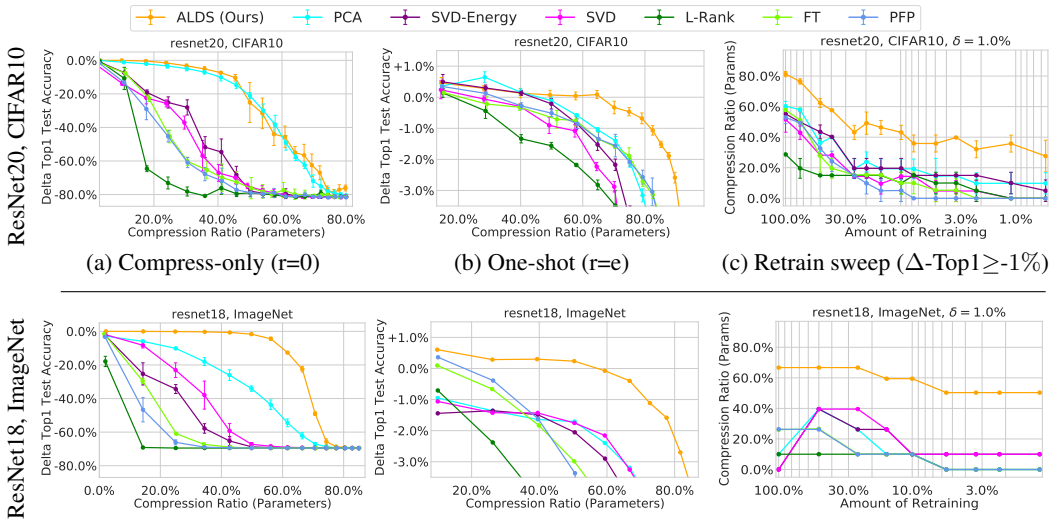
3.1 One-shot Compression on CIFAR10, ImageNet, and VOC with Baselines

We train reference networks on CIFAR10, ImageNet, and VOC, and then compress and retrain the networks *once* with $r = e$ for various baseline comparisons and compression ratios. In Figure 4, we provide results for DenseNet22, VGG16, and WRN16-8 on CIFAR10. Notably, our approach is able to outperform existing baselines approaches across a wide range of tested compression ratios.



(a) DenseNet22 (b) VGG16 (c) WRN16-8

Figure 4: One-shot compress+retrain experiments on CIFAR10 with baseline comparisons.



(a) Compress-only (r=0) (b) One-shot (r=e) (c) Retrain sweep (Δ-Top1$\geq$-1%)

(d) Compress-only (r=0) (e) One-shot (r=e) (f) Retrain sweep (Δ-Top1$\geq$-1%)

Figure 5: The size-accuracy trade-off for various compression ratios, methods, and networks. Compression was performed after training and networks were re-trained once for the indicated amount (**one-shot**). (a, b, d, e): the difference in test accuracy for fixed amounts of retraining. (c, f): the maximal compression ratio with less-than-1% accuracy drop for variable amounts of retraining.

Specifically, in the region where the networks incur only minimal drop in accuracy (Δ-Top1$\geq -1\%$) ALDS is particularly effective.

Moreover, we tested ALDS on ResNet20 (CIFAR10) and ResNet18 (ImageNet) as shown in Figure 5. For these experiments, we performed a grid search over both multiple compression ratios and amounts of retraining. Here, we highlight that ALDS outperforms baseline approaches even with significantly less retraining. On Resnet 18 (ImageNet) ALDS can compress over 50% of the parameters with minimal retraining (1% retraining) and a less-than-1% accuracy drop compared to the best comparison methods (40% compression with 50% retraining). Finally, we tested the same setup on a DeeplabV3 with a ResNet50 backbone trained on Pascal VOC 2012 segmentation data, see Figure 6. We note that ALDS consistently outperforms other baselines methods in this setting as well (60% CR-P vs. 20% without accuracy drop).

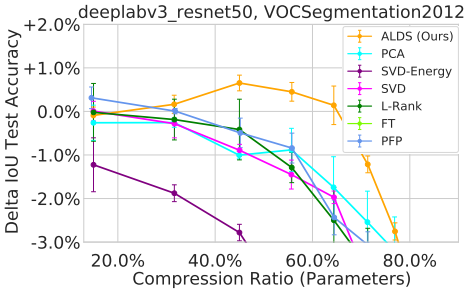


Figure 6: One-shot compress+retrain for DeeplabV3-ResNet50 on VOC.

Our one-shot results are again summarized in Table 1 where we report CR-P and CR-F for Δ-Top1$\geq -0.5\%$. We note that pruning usually takes on the order of seconds and minutes for CIFAR and ImageNet, respectively, which is usually faster than even a single training epoch.

Table 1: Baseline results for Δ-Top1$\geq -0.5\%$ for one-shot. Results coincide with Figures 4, 5, 6.

	Model	Metric	ALDS (Ours)	PCA	SVD-Energy	SVD	L-Rank	FT	PFP
CIFAR10	ResNet20	Δ-Top1	-0.47	-0.11	-0.21	-0.29	-0.44	-0.32	-0.28
	Top1: 91.39	CR-P, CR-F	**74.91**, **67.86**	49.88, 48.67	49.88, 49.08	39.81, 38.95	28.71, 54.89	39.69, 39.57	40.28, 30.06
	VGG16	Δ-Top1	-0.11	-0.02	-0.08	+0.29	-0.35	-0.47	-0.47
	Top1: 92.78	CR-P, CR-F	**95.77**, **86.23**	89.72, 85.84	82.57, 81.32	70.35, 70.13	85.38, 75.86	79.13, 78.44	94.87, 84.76
	DenseNet22	Δ-Top1	-0.32	+0.20	-0.29	+0.13	+0.26	-0.24	-0.44
	Top1: 89.88	CR-P, CR-F	**56.84**, **61.98**	14.67, 34.55	15.16, 19.34	15.00, 15.33	14.98, 35.21	28.33, 29.50	40.24, 43.37
	WRN16-8	Δ-Top1	-0.42	-0.49	-0.41	-0.96	-0.45	-0.32	-0.44
	Top1: 89.88	CR-P, CR-F	**87.77**, 79.90	85.33, **83.45**	64.75, 60.94	40.20, 39.97	49.86, 58.00	82.33, 75.97	85.33, 80.68
ImageNet	ResNet18	Δ-Top1, Top5	-0.40, -0.05	-0.95,-0.37	-1.49, -0.64	-1.75, -0.72	-0.71, -0.23	+0.10, +0.42	-0.39, -0.08
	Top1: 69.62, Top5: 89.08	CR-P, CR-F	**66.70**, **43.51**	9.99, 12.78	39.56, 40.99	50.38, 50.37	10.01, 32.64	9.86, 11.17	26.35, 17.96
VOC	DeeplabV3	Δ-IoU, Top1	+0.14, -0.15	-0.26, -0.02	-1.88, -0.47	-0.28, -0.18	-0.42, -0.09	-4.30, -0.91	-0.49, -0.21
	IoU: 91.39 Top1: 99.34	CR-P, CR-F	**64.38**, **64.11**	55.68, 55.82	31.61, 32.27	31.64, 31.51	44.99, 45.02	15.00, 15.06	45.17, 43.93

3.2 ImageNet Benchmarks

Next, we test our framework on two common ImageNet benchmarks, ResNet18 and AlexNet. We follow the compress-retrain pipeline outlined in the beginning of the section and repeat it iteratively to obtain higher compression ratios. Specifically, after retraining and before the next compression step we project the decomposed layers back to the original layer. This way, we avoid recursing on the decomposed layers. Our results are reported in Table 2 where we compare to a wide variety of available compression benchmarks (results were adapted directly from the respective papers). The middle part and bottom part for each network are organized into low-rank compression and filter pruning approaches, respectively. Note that the reported differences in accuracy (Δ-Top1 and Δ-Top5) are relative to our baseline accuracies. On ResNet18 we can reduce the number of FLOPs by 65% with minimal drop in accuracy compared to the best competing method (MUSCO, 58.67%). With a slightly higher drop in accuracy (-1.37%) we can even compress 76% of FLOPs. On AlexNet, our framework finds networks with -0.21% and -0.41% difference in accuracy with over 77% and 81% fewer FLOPs. This constitutes a more-than-10% improvement in terms of FLOPs compared to current state-of-the-art (L-Rank) for similar accuracy drops.

Table 2: AlexNet and ResNet18 Benchmarks on ImageNet. We report Top-1, Top-5 accuracy and percentage reduction of FLOPs (CR-F). Best results with less than 0.5% accuracy drop are bolded.

	Method	Δ-Top1	Δ-Top5	CR-F (%)
ResNet18, Top1, 5: 69.64%, 88.98%	ALDS (Ours)	**-0.38**	**+0.04**	**64.5**
	ALDS (Ours)	-1.37	-0.56	76.3
	MUSCO (Gusak et al., 2019)	**-0.37**	**-0.20**	58.67
	TRP1 (Xu et al., 2020)	-4.18	-2.5	44.70
	TRP1+Nu (Xu et al., 2020)	-4.25	-2.61	55.15
	TRP2+Nu (Xu et al., 2020)	-4.3	-2.37	68.55
	PCA (Zhang et al., 2015a)	-6.54	-4.54	29.07
	Expand (Jaderberg et al., 2014)	-6.84	-5.26	50.00
	PFP (Liebenwein et al., 2020)	-2.26	-1.07	29.30
	SoftNet (He et al., 2018)	-2.54	-1.2	41.80
	Median (He et al., 2019)	-1.23	-0.5	41.80
	Slimming (Liu et al., 2017)	-1.77	-1.19	28.05
	Low-cost (Dong et al., 2017)	-3.55	-2.2	34.64
	Gating (Hua et al., 2018)	-1.52	-0.93	37.88
	FT (He et al., 2017)	-3.08	-1.75	41.86
	DCP (Zhuang et al., 2018)	-2.19	-1.28	47.08
	FBS (Gao et al., 2018)	-2.44	-1.36	49.49
AlexNet, Top1, 5: 57.30%, 80.20%	ALDS (Ours)	-0.21	-0.36	77.9
	ALDS (Ours)	**-0.41**	**-0.54**	**81.4**
	Tucker (Kim et al., 2015a)	N/A	-1.87	62.40
	Regularize (Tai et al., 2015)	N/A	-0.54	74.35
	Coordinate (Wen et al., 2017)	N/A	-0.34	62.82
	Efficient (Kim et al., 2019)	-0.7	-0.3	62.40
	L-Rank (Idelbayev et al., 2020)	**-0.13**	**-0.13**	66.77
	NISP (Yu et al., 2018)	-1.43	N/A	67.94
	OICSR (Li et al., 2019a)	-0.47	N/A	53.70
	Oracle (Ding et al., 2019)	-1.13	-0.67	31.97

3.3 Ablation Study

To investigate the different features of our method we ran compression experiments using multiple variations derived from our method, see Figure 7. For the simplest version of our method we consider a constant per-layer compression ratio and fix the value of k to either 3 or 5 for all layers denoted by ALDS-Simple3 and ALDS-Simple5, respectively. Note that ALDS-Simple with $k = 1$ corresponds to the SVD comparison method. For the version denoted by ALDS-Error3 we fix the number of subspaces per layer ($k = 3$) and only run the global step of ALDS (Line 4 of Algorithm 1) to determine the optimal per-layer compression ratio. The results of our ablation study in Figure 7 indicate that our method clearly benefits from the combination of both the global and local step in terms of the number of subspaces (k) and the rank per subspace (j).

We also compare our subspace clustering (channel slicing) to the clustering technique of Maalouf et al. (2021), which clusters the matrix columns using projective clustering. Specifically, we replace the channel slicing of ALDS-Simple3 with projective clustering (Messi3 in Figure 7). As expected Messi improves the performance over ALDS-Simple but only slightly and the difference is essentially negligible. Together with the computational disadvantages of Messi-like clustering methods (unstructured, NP-hard; see Section 2.1) ALDS-based simple channel slicing is therefore the preferred choice in our context.

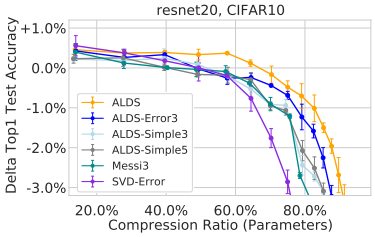


Figure 7: One-shot ablation study of ALDS for Resnet20 (CIFAR10).

4 Related Work

Our work builds upon prior work in neural network compression. We discuss related work focusing on pruning, low-rank compression, and global aspects of compression.

Unstructured pruning. Weight pruning (Lin et al., 2020b; Molchanov et al., 2016, 2019; Singh and Alistarh, 2020; Wang et al., 2021; Yu et al., 2018) techniques aim to reduce the number of individual

weights, e.g., by removing weights with absolute values below a threshold (Han et al., 2015; Renda et al., 2020), or by using a mini-batch of data points to approximate the influence of each parameter on the loss function (Baykal et al., 2019a,b). However, since these approaches generate sparse instead of smaller models they require some form of sparse linear algebra support for runtime speed-ups.

Structured pruning. Pruning structures such as filters directly shrinks the network (Chen et al., 2020; Li et al., 2019b; Lin et al., 2020a; Liu et al., 2019a; Luo and Wu, 2020; Ye et al., 2018). Filters can be pruned using a score for each filter, e.g., weight-based (He et al., 2018, 2017) or data-informed (Liebenwein et al., 2020; Yu et al., 2018), and removing those with a score below a threshold. It is worth noting that filter pruning is complimentary to low-rank compression.

Low-rank compression (local step). A common approach to low-rank compression entails tensor decomposition including Tucker-decomposition (Kim et al., 2015b), CP-decomposition (Lebedev et al., 2015), Tensor-Train (Garipov et al., 2016; Novikov et al., 2015) and others (Denil et al., 2013; Ioannou et al., 2017; Jaderberg et al., 2014). Other decomposition-like approaches include weight sharing, random projections, and feature hashing (Arora et al., 2018; Chen et al., 2015a,b; Shi et al., 2009; Ullrich et al., 2017; Weinberger et al., 2009). Alternatively, low-rank compression can be performed via matrix decomposition (e.g., SVD) on flattened tensors as done by Denton et al. (2014); Sainath et al. (2013); Tukan et al. (2020); Xue et al. (2013); Yu et al. (2017) among others. Chen et al. (2018); Denton et al. (2014); Maalouf et al. (2021) also explores the use of subspace clustering before applying low-rank compression to each cluster to improve the approximation error. Notably, most prior work relies on some form of expensive approximation algorithm – even to just solve the per-layer low-rank compression, e.g., clustering or tensor decomposition. In this paper, we instead focus on the global compression problem and show that simple compression techniques (SVD with channel slicing) are advantageous in this context as we can use them as efficient subroutines. We note that we can even extend our algorithm to multiple, different types of per-layer decomposition.

Network-aware compression (global step). To determine the rank (or the compression ratio) of each layer, prior work suggests to account for compression during training (Alvarez and Salzmann, 2017; Ioannou et al., 2016, 2015; Wen et al., 2017; Xu et al., 2020), e.g, by training the network with a penalty that encourages the weight matrices to be low-rank. Others suggest to select the ranks using variational Bayesian matrix factorization (Kim et al., 2015b). In their recent paper, Chin et al. (2020) suggest to produce an entire set of compressed networks with different accuracy/speed trade-offs. Our paper was also inspired by a recent line of work towards automatically choosing or learning the rank of each layer (Gusak et al., 2019; Idelbayev and Carreira-Perpinán, 2020; Li and Shi, 2018; Tiwari et al., 2021; Zhang et al., 2015a,b). We take such approaches further and suggest a global compression framework that incorporates multiple decomposition techniques with more than one hyper-parameter per layer (number of subspaces and ranks of each layer). This approach increases the number of local minima in theory and helps improving the performance in practice.

5 Discussion and Conclusion

Practical benefits. By conducting a wide variety of experiments across multiple data sets and networks we have shown the effectiveness and versatility of our compression framework compared to existing methods. The runtime of ALDS is negligible compared to retraining and it can thus be efficiently incorporated into compress-retrain pipelines.

ALDS as modular compression framework. By separately considering the low-rank compression scheme for each layer (local step) and the actual low-rank compression (global step) we have provided a framework that can efficiently search over a set of desired hyperparameters that describe the low-rank compression. Naturally, our framework can thus be generalized to other compression schemes (such as tensor decomposition) and we hope to explore these aspects in future work.

Error bounds lead to global insights. At the core of our contribution is our error analysis that enables us to link the global and local aspects of layer-wise compression techniques. We leverage our error bounds in practice to compress networks more effectively via an automated rank selection procedure without additional tedious hyperparameter tuning. However, we also have to rely on a proxy definition (maximum relative error) of the compression error to enable a tractable solution that we can implement efficiently. We hope these observations invigorate future research into compression techniques that come with tight error bounds – potentially even considering retraining – which can then naturally be wrapped into a global compression framework.

References

Jose M Alvarez and Mathieu Salzmann. Compression-aware training of deep networks. In *Advances in Neural Information Processing Systems*, pages 856–867, 2017.

Sanjeev Arora, Rong Ge, Behnam Neyshabur, and Yi Zhang. Stronger generalization bounds for deep nets via a compression approach. In *International Conference on Machine Learning*, pages 254–263, 2018.

Cenk Baykal, Lucas Liebenwein, Igor Gilitschenski, Dan Feldman, and Daniela Rus. Data-dependent coresets for compressing neural networks with applications to generalization bounds. In *International Conference on Learning Representations*, 2019a. URL https://openreview.net/forum?id=HJfwJ2A5KX.

Cenk Baykal, Lucas Liebenwein, Igor Gilitschenski, Dan Feldman, and Daniela Rus. Sipping neural networks: Sensitivity-informed provable pruning of neural networks. *arXiv preprint arXiv:1910.05422*, 2019b.

Jianda Chen, Shangyu Chen, and Sinno Jialin Pan. Storage efficient and dynamic flexible runtime channel pruning via deep reinforcement learning. *Advances in Neural Information Processing Systems*, 33, 2020.

Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017.

Patrick H. Chen, Si Si, Yang Li, Ciprian Chelba, and Cho-jui Hsieh. GroupReduce: Block-Wise Low-Rank Approximation for Neural Language Model Shrinking. *Advances in Neural Information Processing Systems*, 2018-December:10988–10998, jun 2018. URL http://arxiv.org/abs/1806.06950.

Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. In *International conference on machine learning*, pages 2285–2294, 2015a.

Wenlin Chen, James T. Wilson, Stephen Tyree, Kilian Q. Weinberger, and Yixin Chen. Compressing convolutional neural networks. *CoRR*, abs/1506.04449, 2015b. URL http://arxiv.org/abs/1506.04449.

Ting-Wu Chin, Ruizhou Ding, Cha Zhang, and Diana Marculescu. Towards efficient model compression via learned global ranking. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1518–1528, 2020.

Misha Denil, Babak Shakibi, Laurent Dinh, Marc Aurelio Ranzato, and Nando de Freitas. Predicting parameters in deep learning. In *Advances in Neural Information Processing Systems 26*, pages 2148–2156, 2013.

Emily Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting Linear Structure Within Convolutional Networks for Efficient Evaluation. *Advances in Neural Information Processing Systems*, 2(January):1269–1277, apr 2014. URL http://arxiv.org/abs/1404.0736.

Xiaohan Ding, Guiguang Ding, Yuchen Guo, Jungong Han, and Chenggang Yan. Approximated oracle filter pruning for destructive cnn width optimization. In *International Conference on Machine Learning*, pages 1607–1616. PMLR, 2019.

Xuanyi Dong, Junshi Huang, Yi Yang, and Shuicheng Yan. More is less: A more complicated network with less inference complexity. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5840–5848, 2017.

Mark Everingham, SM Ali Eslami, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes challenge: A retrospective. *International journal of computer vision*, 111(1):98–136, 2015.

Xitong Gao, Yiren Zhao, Łukasz Dudziak, Robert Mullins, and Cheng-zhong Xu. Dynamic channel pruning: Feature boosting and suppression. *arXiv preprint arXiv:1810.05331*, 2018.

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

Gene H Golub and Charles F Van Loan. *Matrix computations*, volume 3. JHU press, 2013.

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

Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *CoRR*, abs/1510.00149, 2015. URL http://arxiv.org/abs/1510.00149.

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.

Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. Soft filter pruning for accelerating deep convolutional neural networks. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pages 2234–2240. AAAI Press, 2018.

Yang He, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi Yang. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4340–4349, 2019.

Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1389–1397, 2017.

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

Weizhe Hua, Yuan Zhou, Christopher De Sa, Zhiru Zhang, and G Edward Suh. Channel gating neural networks. *arXiv preprint arXiv:1805.12549*, 2018.

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

Yerlan Idelbayev and Miguel A Carreira-Perpinán. Low-rank compression of neural nets: Learning the rank of each layer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8049–8059, 2020.

Y Ioannou, D Robertson, J Shotton, R Cipolla, and A Criminisi. Training cnns with low-rank filters for efficient image classification. In *4th International Conference on Learning Representations, ICLR 2016-Conference Track Proceedings*, 2016.

Yani Ioannou, Duncan Robertson, Jamie Shotton, Roberto Cipolla, and Antonio Criminisi. Training cnns with low-rank filters for efficient image classification. *arXiv preprint arXiv:1511.06744*, 2015.

Yani Ioannou, Duncan Robertson, Roberto Cipolla, and Antonio Criminisi. Deep roots: Improving cnn efficiency with hierarchical filter groups. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1231–1240, 2017.

Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. In *Proceedings of the British Machine Vision Conference. BMVA Press*, 2014.

Hyeji Kim, Muhammad Umar Karim Khan, and Chong-Min Kyung. Efficient neural network compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12569–12577, 2019.

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. *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, nov 2015a. URL http://arxiv.org/abs/1511.06530.

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

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

Valero Laparra, Jesús Malo, and Gustau Camps-Valls. Dimensionality reduction via regression in hyperspectral imagery. *IEEE Journal of Selected Topics in Signal Processing*, 9(6):1026–1036, 2015.

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

Chong Li and CJ Shi. Constrained optimization based low-rank approximation of deep neural networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 732–747, 2018.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

Jiashi Li, Qi Qi, Jingyu Wang, Ce Ge, Yujian Li, Zhangzhang Yue, and Haifeng Sun. Oicsr: Out-in-channel sparsity regularization for compact deep neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7046–7055, 2019a.

Yawei Li, Shuhang Gu, Luc Van Gool, and Radu Timofte. Learning filter basis for convolutional neural network compression. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5623–5632, 2019b.

Lucas Liebenwein, Cenk Baykal, Harry Lang, Dan Feldman, and Daniela Rus. Provable filter pruning for efficient neural networks. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=BJxkOlSYDH.

Lucas Liebenwein, Cenk Baykal, Brandon Carter, David Gifford, and Daniela Rus. Lost in pruning: The effects of pruning neural networks beyond test accuracy. *Proceedings of Machine Learning and Systems*, 3, 2021.

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*, pages 1529–1538, 2020a.

Tao Lin, Sebastian U. Stich, Luis Barba, Daniil Dmitriev, and Martin Jaggi. Dynamic model pruning with feedback. In *International Conference on Learning Representations*, 2020b. URL https://openreview.net/forum?id=SJem8lSFwB.

Zechun Liu, Haoyuan Mu, Xiangyu Zhang, Zichao Guo, Xin Yang, Kwang-Ting Cheng, and Jian Sun. Metapruning: Meta learning for automatic neural network channel pruning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3296–3305, 2019a.

Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2736–2744, 2017.

Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *International Conference on Learning Representations*, 2019b. URL https://openreview.net/forum?id=rJlnB3C5Ym.

Jian-Hao Luo and Jianxin Wu. Autopruner: An end-to-end trainable filter pruning method for efficient deep model inference. *Pattern Recognition*, 107:107461, 2020.

Alaa Maalouf, Harry Lang, Daniela Rus, and Dan Feldman. Deep learning meets projective clustering. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=EQfpYwF3-b.

Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016.

Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 11264–11272, 2019.

Alexander Novikov, Dmitry Podoprikhin, Anton Osokin, and Dmitry Vetrov. Tensorizing neural networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems-Volume 1*, pages 442–450, 2015.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

Xi Peng, Zhang Yi, and Huajin Tang. Robust subspace clustering via thresholding ridge regression. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

Alex Renda, Jonathan Frankle, and Michael Carbin. Comparing fine-tuning and rewinding in neural network pruning. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=S1gSj0NKvB.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115 (3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

Tara N Sainath, Brian Kingsbury, Vikas Sindhwani, Ebru Arisoy, and Bhuvana Ramabhadran. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6655–6659. IEEE, 2013.

Qinfeng Shi, James Petterson, Gideon Dror, John Langford, Alex Smola, and SVN Vishwanathan. Hash kernels for structured data. *Journal of Machine Learning Research*, 10(Nov):2615–2637, 2009.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

Sidak Pal Singh and Dan Alistarh. Woodfisher: Efficient second-order approximations for model compression. *arXiv preprint arXiv:2004.14340*, 2020.

Cheng Tai, Tong Xiao, Yi Zhang, Xiaogang Wang, et al. Convolutional neural networks with low-rank regularization. *arXiv preprint arXiv:1511.06067*, 2015.

Rishabh Tiwari, Udbhav Bamba, Arnav Chavan, and Deepak Gupta. Chipnet: Budget-aware pruning with heaviside continuous approximations. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=xCxXwTzx4L1.

Antonio Torralba, Rob Fergus, and William T Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE transactions on pattern analysis and machine intelligence*, 30(11):1958–1970, 2008.

Murad Tukan, Alaa Maalouf, Matan Weksler, and Dan Feldman. Compressed deep networks: Goodbye svd, hello robust low-rank approximation. *arXiv preprint arXiv:2009.05647*, 2020.

Karen Ullrich, Edward Meeds, and Max Welling. Soft weight-sharing for neural network compression. *arXiv preprint arXiv:1702.04008*, 2017.

Huan Wang, Can Qin, Yulun Zhang, and Yun Fu. Neural pruning via growing regularization. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=o966_Is_nPA.

Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 1113–1120, 2009.

Wei Wen, Cong Xu, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Coordinating Filters for Faster Deep Neural Networks. *Proceedings of the IEEE International Conference on Computer Vision*, 2017-Octob:658–666, mar 2017. URL http://arxiv.org/abs/1703.09746.

Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized Convolutional Neural Networks for Mobile Devices. In *Proceedings of the International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

Yuhui Xu, Yuxi Li, Shuai Zhang, Wei Wen, Botao Wang, Yingyong Qi, Yiran Chen, Weiyao Lin, and Hongkai Xiong. Trained rank pruning for efficient deep neural networks. *arXiv preprint arXiv:1812.02402*, 2018.

Yuhui Xu, Yuxi Li, Shuai Zhang, Wei Wen, Botao Wang, Yingyong Qi, Yiran Chen, Weiyao Lin, and Hongkai Xiong. TRP: Trained Rank Pruning for Efficient Deep Neural Networks. *IJCAI International Joint Conference on Artificial Intelligence*, 2021-Janua:977–983, apr 2020. URL http://arxiv.org/abs/2004.14566.

Jian Xue, Jinyu Li, and Yifan Gong. Restructuring of deep neural network acoustic models with singular value decomposition. In *Interspeech*, pages 2365–2369, 2013.

Jianbo Ye, Xin Lu, Zhe Lin, and James Z. Wang. Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=HJ94fqApW.

Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S Davis. Nisp: Pruning networks using neuron importance score propagation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9194–9203, 2018.

Xiyu Yu, Tongliang Liu, Xinchao Wang, and Dacheng Tao. On compressing deep models by low rank and sparse decomposition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7370–7379, 2017.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

Xiangyu Zhang, Jianhua Zou, Kaiming He, and Jian Sun. Accelerating very deep convolutional networks for classification and detection. *IEEE transactions on pattern analysis and machine intelligence*, 38(10):1943–1955, 2015a.

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

Zhuangwei Zhuang, Mingkui Tan, Bohan Zhuang, Jing Liu, Yong Guo, Qingyao Wu, Junzhou Huang, and Jinhui Zhu. Discrimination-aware channel pruning for deep neural networks. *arXiv preprint arXiv:1810.11809*, 2018.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 2.3 and 5
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See supplementary material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] see supplementary material.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See supplementary material.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes]
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] We include an implementation for our method and many other competing methods.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]