Ph.D. Dissertation of Engineering

Towards an Effective Low-rank Compression of Neural Networks

심층신경망의 효과적인 저 차원 압축

August 2023

Program in
Digital Contents and Information
Graduate School of
Convergence Science and Technology
Seoul National University

Moonjung Eo

Towards an Effective Low-rank Compression of Neural Networks

Advisor Wonjong Rhee

Submitting a Ph.D. Dissertation of Engineering

August 2023

Program in Intelligence and Information
Graduate School of
Convergence Science and Technology
Seoul National University

Moonjung Eo

Confirming the Ph.D. Dissertation written by
Moonjung Eo

August 2023

Chair	Nojun Kwak	(Seal)
Vice Chair	Wonjong Rhee	(Seal)
Examiner	Bongwon Suh	(Seal)
Examiner	Dongsuk Jeon	(Seal)
Examiner	Daeyoung Choi	(Seal)

Abstract

Compression of neural networks has emerged as one of the essential research topics, especially for edge devices that have limited computation power and storage capacity. The most popular compression methods include quantization, pruning of redundant parameters, knowledge distillation from a large network to a small one, and low-rank compression. The low-rank compression methodology has the potential to be a high-performance compression method, but it does not achieve high performance since it does not solve the challenge of determining the optimal rank of all the layers. This thesis explores two methods to solve the challenge and improve compression performance. First, we propose **BSR** (Beam-search and Stable Rank), a low-rank compression algorithm that embodies an efficient rank-selection method and a unique compression-friendly training method. For the rank selection, **BSR** employs a modified beam search that can perform a *joint* optimization of the rank allocations over all the layers in contrast to the previously used heuristic methods. For the compression-friendly training, **BSR** adopts a regularization loss derived from a modified stable rank, which can control the rank while incurring almost no harm in performance. Experiment results confirm that **BSR** is effective and superior compared to the existing low-rank compression methods. Second, we propose a fully joint learning framework called **LeSS** to simultaneously determine filters for filter pruning and ranks for low-rank decomposition. We provided a method for rank selection with a training method and confirmed a significant improvement in performance by integrating it with the existing pruning method, which has outstanding performance. **LeSS** does not depend on iterative or heuristic processes, and it satisfies the desired resource budget constraint. **LeSS** comprises two learning modules: mask learning for filter pruning and threshold learning for low-rank decomposition. The first module learns masks identifying the importance of the filters, and the second module learns the threshold of the singular values to

be removed such that only significant singular values remain. Because both modules are designed to be differentiable, they are easily combined and jointly optimized. **LeSS** outperforms state-of-the-art methods on a number of benchmarks demonstrating its effectiveness. Finally, to obtain high performance in transfer learning for fine-grained datasets, we propose mask learning for both rank and filter selection. The mask learning approach could be employed in transfer learning since it is more crucial to determine which singular values are useful rather than rank selection. Our approach to compression for transfer learning yielded either improved or comparable performance with uncompressed results. We anticipate these techniques will be broadly applicable to industrial domains.

Keywords: Structured Compression, Low-rank Compression, Filter Pruning, Beam-search, Mask Learning

Student number: 2017-35557

Table of Contents

List of Tables

List of Figures

Chapter 1. Introduction

The explosion of both data and computational power has contributed to AI's current renaissance. In particular, deep learning has been effectively applied to many difficult tasks. However, in order to obtain good performance in these various tasks, a huge network is required. In addition, high-speed hardware is required in order to quickly learn and infer from the huge network, which makes it nearly impossible to deploy on low-resource devices or edge devices. For this reason, many studies have recently been conducted to compress neural networks. This dissertation focused on low-rank compression among various compression techniques for compressing neural networks and developed it with the aim of further improving the compression performance.

As deep learning becomes widely adopted by the industry, the demand for compression techniques that are highly effective and easy to use is sharply increasing. The most popular compression methods include quantization (Rastegari et al. 2016; Wu et al. 2016), pruning of redundant parameters (Han, Mao, and Dally 2015; Lebedev and Lempitsky 2016; Srinivas and Babu 2015), knowledge distillation from a large network to a small one (Hinton, Vinyals, and Dean 2015; Kim, Park, and Kwak 2018; Romero et al. 2014; Zagoruyko and Komodakis 2016), and network factorization (Alvarez and Salzmann 2017; Denton et al. 2014; Masana et al. 2017; Xue, Li, and Gong 2013). In this dissertation, we focus on low-rank compression that is based on a matrix factorization and low-rank approximation of weight matrices. This compression approach has numerous benefits. First, low-rank approaches have a long history of use in the domains of linear algebra, signal processing, and statistics with techniques such as singular value decomposition (SVD) and established software packages such as BLAS and LAPACK. Second, when the weights of a neural network are compressed using matrices with low ranks, both the size of the network and the computational requirements for the forward pass are explicitly reduced. Importantly, the computational reductions

can be achieved without explicit hardware assistance. This hardware friendliness is in striking contrast to other compression strategies, such as quantized or element-wise pruned models, which need the construction of a specialized processor in order to be deployed efficiently. As mentioned above, typical low-rank compression is based on a direct application of SVD (Singular Value Decomposition). For a trained neural network, the layer l's weight matrix $\mathbf{W}_l$ of size $m_l \times n_l$ is decomposed, and only the top r_l dimensions are kept. This reduces the computation and memory requirements from $m_l n_l$ to $(m_l + n_l)r_l$, and the reduction can be large when a small r_l is chosen. The traditional approaches can be divided into two main streams. One considers only a post-application of SVD on a fully trained neural network, and the other additionally performs compression-friendly training before SVD.

For the other stream, several works introduced an additional step of compression-friendly training (Alvarez and Salzmann 2017; Idelbayev and Carreira-Perpinán 2020; Li and Shi 2018). This can be a promising strategy because it is common sense to train a large network to secure a desirable learning dynamics (Luo, Wu, and Lin 2017; Carreira-Perpinán and Idelbayev 2018), but at the same time the network can be regularized in many different ways without harming the performance (Choi et al. 2021). For rank selection, however, most of them still used heuristics and failed to achieve competitive performance. Recently, (Idelbayev and Carreira-Perpinán 2020) achieved state-of-the-art performance for low-rank compression by calculating target ranks r with a variant of Eckhart-Young theorem and by performing a regularized training with respect to the weight matrices truncated according to the target rank vector $\mathbf{r}$. The existing compression-friendly works, however, require re-calculations of $\mathbf{r}$ during the training, train weight matrices that are not really low-rank, and demand extensive tuning, especially for obtaining a compressed network of a desired compression ratio.

Hence, the works presented in this thesis aims to enhance the existing low-rank compression methods in five different aspects. First, we adopt a modified beam-search with a performance validation for selecting the target rank vector $\mathbf{r}$. Compared to

the previous works, our method allows a joint search of rank assignments over all the layers. The previous works relied on a simple heuristic (e.g., all layers should be truncated to keep the same portion of energy (Alvarez and Salzmann 2017)) or an implicit connection of all layers (e.g. through a common penalty parameter (Idelbayev and Carreira-Perpinán 2020)). Secondly, we adopt a modified stable rank for the rank-regularized training. Because our modified stable rank does not rely on any instance of weight matrix truncation, it can be continuously enforced without any update as long as the target rank vector $\mathbf{r}$ remains constant. The previous works used a forced truncation in the middle of training (Alvarez and Salzmann 2017) or a norm distance regularization with the truncated weight matrices (Idelbayev and Carreira-Perpinán 2020). In both methods, iteration between the weight truncation step and the training step was necessary. In our method, we calculate and set the target rank vector $\mathbf{r}$ only once. Our modified stable rank turns out to be very effective at controlling the singular values. As a result of the first and the second aspects, our low-rank compression method can achieve performance on par with or even better than the latest pruning methods. Because low-rank compression can be easily combined with quantization, just like pruning, a very competitive overall compression performance can be achieved. Thirdly, we are the first to propose a successful SGD-learning solution for rank-selection (*threshold learning*). The results can achieve SOTA over the existing pruning and low-rank methods. Fourthly, we propose a unified method where joint learning over filter selection and rank selection is performed. Obviously, we are the first to propose a *unified learning* because learning of rank-selection was not available before. Finally, we suggest a fully mask learning method of rank selection and filter selection for the transfer learning regime. Our method of compression for transfer learning yielded either superior or comparable performance to uncompressed results.

1.1. Thesis Outline

In each chapter, we propose techniques for improving low-rank compression that we anticipate will be more broadly applicable to industrial domains. Concretely, the chapters are organized as follows:

- Chapter 2 gives a brief overview of structured compression methods such as low-rank compression and filter pruning.
- Chapter 3 provides a method to increase the performance of the low-rank compression using a novel rank selection method and a rank regularizer. We propose the rank selection algorithm based on beam-search and the rank regularizer modified from a stable rank.
- Chapter 4 considers the learning-based rank selection method for improving the performance of low-rank compression. Furthermore, We present a unified framework by joint learning over filter selection and rank selection. The unified framework shows enhanced compression performance.
- Chapter 5 provides mask learning for structured compression to obtain higher performance in transfer learning. The compressed model via our method shows a better or comparable performance than the non-compressed model.
- Finally, chapter 6 concludes and discusses the future outlook.

1.2. Related Publications

Portions of this thesis appeared in the following publications:

- Chapter 3: Moonjung Eo*, Suhyun Kang*, and Wonjong Rhee. "An effective low-rank compression with a joint rank selection followed by a compression-friendly training." Neural Networks (2023).

- Chapter 4: Moonjung Eo, Suhyun Kang, and Wonjong Rhee. "LeSS: Learning to Select a Structured Architecture over Filter Pruning and Low-rank Decomposition." Under review

Chapter 2. Background

This chapter provides a comprehensive summary of the different compression methods that have been researched previously. We begin by providing a brief overview of the different types of compression methods being studied in deep neural networks in Section 2.1. We then shift our attention to the structured compression methodology and present a comprehensive summary in Section 2.2. Specifically, we delve into the two main approaches of structured compression, filter pruning, and low-rank compression, and categorize each methodology into regularized training and selection problems. Our analysis focuses on the technical differences between the methodologies, and we provide an in-depth exploration of each.

2.1. Compression of Deep Neural Networks

It is generally known that learning on deep and wide networks, then reducing the network size is better than learning from scratch on shallow and narrow networks. In other words, it disapproves that the trained network is over-parameterized. To alleviate this problem, many researchers have attempted to make deep neural networks compact by pruning redundant individual parameters (Srinivas and Babu 2015; Lebedev and Lempitsky 2016; Han, Mao, and Dally 2015), quantizing weights by reducing the number of bits (Rastegari et al. 2016; Wu et al. 2016), distilling knowledge from a large network to a smaller one (Romero et al. 2014; Zagoruyko and Komodakis 2016; Hinton, Vinyals, and Dean 2015; Kim, Park, and Kwak 2018), and factorizing a network (Alvarez and Salzmann 2017; Masana et al. 2017; Xue, Li, and Gong 2013; Denton et al. 2014), among others. A simple way to reduce the size of a deep and wide neural network is network pruning, which directly removes some of the unimportant parameters. This process is conducted during training or by analyzing each parameter's influence on the training accuracy (Han, Mao, and Dally 2015; Lebedev and Lempitsky 2016;

Srinivas and Babu 2015; Zhou, Alvarez, and Porikli 2016) or objective function after the network has been trained (Liu et al. 2015; Hu et al. 2016). The network pruning methods require a pruning threshold, which plays an important role and is manually set according to the targeted loss of accuracy.

Parameter quantization seeks to find efficient representations of parameters with fewer bits (Buciluǎ, Caruana, and Niculescu-Mizil 2006; Courbariaux et al. 2016; Gupta et al. 2015; Han et al. 2015; Rastegari et al. 2016; Wu et al. 2016; Yu et al. 2017). The authors of (Courbariaux et al. 2016; Rastegari et al. 2016) proposed to quantize parameters in the network into +1 or -1, and the author of (Gupta et al. 2015) used a 16 bit fixed-point number representation when using stochastic rounding to accelerate computing speed and consume fewer hardware resources. The authors of (Han et al. 2015) proposed a 'deep compression' method to combine pruning, trained quantization, and Huffman coding. Their results show a significant reduction in the number of parameters without affecting network performance.

Knowledge distillation aims to transfer essential knowledge of large and complicated networks to smaller and simpler networks. This method was pioneered by the authors of (Hinton, Vinyals, and Dean 2015) and (Buciluǎ, Caruana, and Niculescu-Mizil 2006), and their approach used a softened version of the final output of a teacher network to transfer information to a smaller student network. Additionally, the author of (Romero et al. 2014) transferred information from hidden layers of the teacher network to the student network. Recently, the author of (Kim, Park, and Kwak 2018) proposed an efficient factor extractor and a translator using an unsupervised learning method and showed that it enhanced the performance of the student network.

An efficient way for matrix factorization is applying SVD to the parameters (Denton et al. 2014; Xue, Li, and Gong 2013; Yu et al. 2017; Zhou, Alvarez, and Porikli 2016). Recently, the authors of (Masana et al. 2017) have proposed an advanced matrix factorization method called domain adaptive low rank (DALR) for transfer learning, which considers activation statistics in compression. Their results show better

compression and transfer learning performances than earlier methods. In (Alvarez and Salzmann 2017), a regularizer is introduced for reducing weight rank while training and has shown considerable improvement in compression performance. For using this method, however, the optimization should be divided into two steps: First is minimizing cross-entropy through back-propagation, and the second is reducing weight rank by zeroing small singular values after truncated SVD. Thus, optimization might be unstably converged to satisfy both objective functions. To build lightweight models, there is a movement to design the network architecture to be compact from scratch. For example, famous models such as GoogleNet (Szegedy et al. 2015) and ResNet (He et al. 2016) avoided using fully-connected layers, which occupy a large percentage rather than used global average pooling. Even though these design strategies help to build a deeper and more compact network, there is still a lot of necessity for the compression of these models as they are over-parameterized most of the time. Neural Architecture Search (NAS) is another recent approach for automatically searching for suitable neural network architectures based on certain constraints. Few NAS approaches use reinforcement learning (He et al. 2018b), evolutionary algorithms (Chen et al. 2019a), and Bayesian optimization method (Cho et al. 2020) for the dynamic exploration of architectures.

In this study, among various approaches to making a network compact, we focus on the structured compression method, which consists of low-rank compression and filter pruning. Because of the structured property, the resulting compression can be easily implemented without requiring any specialized software or hardware support (Lin et al. 2020), thus it is considered to be a practical and influential method.

2.2. Structured Compression of Deep Neural Networks

Structured compression is a technique that aims to reduce the computational complexity of deep neural networks by decreasing the number of parameters in the network without creating a sparse structure. Structured compression is particularly well-suited

for current hardware, which is optimized for dense matrix multiplications and can completely remove any zero entries. This technique allows for memory reduction and faster training/inference times without the need for additional specialized hardware (Wen et al. 2017).

Structured compression can be categorized into two techniques: low-rank compression and filter pruning. Low-rank compression divides the original layer of the network into two low-rank layers, which is based on the assumption of the parameter's low-rankness. This technique decomposes the weight matrix of a layer into two smaller matrices with lower rank, significantly reducing the number of parameters in the network, leading to faster computation and reduced memory requirements (Denton et al. 2014).

Filter pruning is another technique used in structured compression to remove less informative filters from the original network based on a structural assumption of sparsity in DNN parameters. The process involves setting the weights of the filters to zero and retraining the network to ensure that the remaining filters can compensate for the missing filters without significantly affecting the overall accuracy of the network. This technique can also significantly reduce the number of parameters in the network, leading to faster computation and reduced memory requirements (Li et al. 2016).

Both low-rank compression and filter pruning techniques can be used individually or in combination to achieve even greater compression and performance gains. However, it is crucial to carefully evaluate the performance and trade-offs of different structured compression techniques for each application since the effectiveness of these techniques can vary depending on the specific network architecture and task at hand (Han, Mao, and Dally 2015).

2.2.1. Low-Rank Compression

Low-rank compression of deep neural networks can be achieved through a variety of strategies that have been extensively explored in the literature. Three main approaches

stand out, each with its own advantages and limitations.

The first strategy is to regularize the network to be low-rank during training. This involves adding a penalty term to the loss function that encourages the weight matrices of the network to have a low-rank structure. This method has been shown to effectively reduce the number of parameters in the network while maintaining high accuracy. However, it requires additional computation during training and may not be suitable for networks with already low-rank structures.

The second strategy is to select low-rank architectures of pre-trained networks. This approach involves searching for a low-rank structure in the weight matrices of a pre-trained network and using this structure to compress the network. This method has been shown to achieve high compression rates with only a minor decrease in accuracy. However, searching for the optimal low-rank structure may require extensive computational resources.

Each strategy will be further explained in the following subsections.

Regularized training

Rank regularized training is a technique used to achieve low-rank compression of deep neural networks by adding a regularization penalty to the weight matrices of the network to encourage a low-rank structure. This approach has been extensively studied in the literature, and a number of methods have been proposed to implement this technique.

The paper (Xu et al. 2019) proposes a method called Trained Rank Pruning (TRP) that integrates low-rank approximation and regularization into the training process of DNNs. TRP alternates between low-rank approximation and training, maintaining the capacity of the original network while imposing low-rank constraints during training. A nuclear regularization optimized by stochastic sub-gradient descent is utilized to further promote low rank in TRP. Networks trained with TRP have a low-rank structure in nature and can be approximated with negligible performance loss, eliminating the need for fine-tuning after low-rank approximation. The proposed method outperforms previous

compression methods using low-rank approximation on CIFAR-10 and ImageNet datasets.

Another method (Wu, Guo, and Zhang 2019) called Sparse Low Rank (SLR) improves the compression rate of low-rank approximation by sparsifying the decomposed matrix, giving minimal rank for unimportant neurons while retaining the rank of important ones. The proposed approach relatively represents the decomposition matrix entries based on widely used pruning strategies for neuron ranking. Structured sparsity introduced to the truncated SVD by our approach can achieve higher speed-up using Basic Linear Algebra Subprograms (BLAS) libraries in addition to higher compression. The proposed approach outperforms vanilla truncated SVD and a pruning baseline, achieving better compression rates with minimal or no loss in accuracy.

Recently, the paper (Idelbayev and Carreira-Perpinán 2020) proposes a method that learns the rank of each layer in a deep neural network, which can be used to compress the network by performing low-rank approximation on each layer. The rank of each layer is learned using a Bayesian optimization approach that maximizes the compression rate while minimizing the loss in accuracy. The proposed method outperforms several baseline methods on the CIFAR-10 and ImageNet datasets.

Additionally, the paper (Yin et al. 2021) proposes a method that uses tensor decomposition to compress deep neural networks. The proposed method optimizes the tensor decomposition by minimizing the Frobenius norm of the difference between the original weights and the decomposed weights subject to a sparsity constraint. The sparsity constraint is enforced using a group-lasso regularization term. The proposed method outperforms several baseline methods on the CIFAR-10 and ImageNet datasets.

In this section, the related works of regularized training strategy are explained. Training a neural network with a penalty that encourages the weight matrices to be low-rank is suggested, and the rank selection was performed by humans through repeated experiments (Jaderberg, Vedaldi, and Zisserman 2014; Denton et al. 2014; Tai et al. 2015).

Rank selection

Early Methods In the early days of research on neural network compression (Xue, Li, and Gong 2013), applying SVD to a single layer was a popular approach to reducing computational complexity and the number of parameters. Studies in this area showed that varying the rank of a single layer can significantly impact both model accuracy and FLOPs. The main takeaway from these studies was that SVD could effectively compress neural networks into low-rank matrices on a single layer. This research was significant in demonstrating the feasibility of using SVD for neural network compression and providing insights into the relationship between layer rank, accuracy, and FLOPs.

Heuristic Criteria This section provides a comprehensive review of the heuristics used for rank selection in previous literature. In the previous studies, to determine the ranks in advance, heuristic strategies such as a greedy approach or a thresholding method are used (Kim et al. 2015; Kim, Khan, and Kyung 2019; Wen et al. 2017; Xu et al. 2020). A common approach is to determine the target rank of each layer through trial and error, which involves fine-tuning the approximated network and can be highly time-consuming. The most popular and widely used heuristics for selecting the rank of the ith convolutional layer is as follows (Zhang et al. 2015a; Alvarez and Salzmann 2017; Li and Shi 2018): Given a set of singular values $\sigma = \sigma_1, \sigma_2, ..., \sigma_{k_i}$ for the ith convolutional layer, they defined $S_{\sigma,i} = \{j \mid j \leq k_i\}$ such that the first k_i singular values and their corresponding singular vectors account for a certain percentage of the total variation. Specifically, they chose k_i to be the largest integer satisfying the following constraint:

$$\Sigma_{j=1}^{k_i} \sigma_{i,j}^2 \leq \beta \cdot \Sigma_{\forall j} \sigma_{i,j}^2 \tag{2.1}$$

where β is a pre-defined percentage of variation to be retained in the ith layer accounted by the low-rank approximation, and the $\sigma_{i,j}$ is the jth largest singular value of the ith convolutional layer. It is evident that the approximated network's computation and memory costs are monotonic functions of the compression ratio β.

Another heuristic for selecting the rank of each layer was proposed in (Zhang et al. 2015b). In this approach, the set of singular values for the ith convolutional layer is denoted by σ_i, and the target rank is determined by maximizing a product of singular values subject to the constraint on computation cost. Specifically, $S_{\sigma,i} = \{j \mid j \leq k_i\}$ is chosen to maximize the objective function:

$$\Pi_{\forall i}(\Sigma_{j=1}^{k_i}\sigma_{i,j}^2) \tag{2.2}$$

subject to the constraint that the computation cost is less than a pre-defined threshold. The intuition behind this approach is that the product of singular values reflects the energy of the feature maps. Selecting the top k_i singular values with the highest energy can lead to a good trade-off between accuracy and complexity. A greedy search algorithm is employed to approximately solve this problem. Due to the use of the greedy algorithm, only a single constraint is considered. An improved method for rank selection was proposed by (Li and Shi 2018), which addresses the limitations of the previous heuristics by formulating the problem as a mixed-integer optimization problem. The key idea is to use masking variables to select the singular values for each layer while enforcing multiple constraints on computation cost and memory cost.

To achieve this, Equation 2.2 is re-formulated as follows:

$$\begin{aligned} \text{maximize}_m \ & \Pi_{\forall i}(\Sigma_{\forall j} m_{i,j}\sigma_{i,j}^2) \\ \text{subject to } & N_c(m) \leq N_{C,max} \\ & N_M(m) \leq N_{M,max} \\ & m_{i,j} \in \{0,1\} \end{aligned} \tag{2.3}$$

Where $m_{i,j}$ is the masking variable for the jth singular value in the ith convolutional layer, and $N_c(m)$ and $N_M(m)$ denote the computation cost and memory cost, respectively, which are functions of the masking variables. The objective function is formulated as the geometric mean of the selected singular values, and the constraints on computation cost and memory cost are enforced using the masking variables. Note that

the masking variables and the singular values can only take non-negative values; thus, the objective in Equation 2.3 is equivalent to maximizing the geometric mean. Even with the 0-1 integer constraint on the masking variables, modern numerical optimization engines can efficiently solve this mixed-integer program with provable optimality. This method provides a more flexible and accurate approach for rank selection, as it can handle multiple constraints and provides a provably optimal solution. In the paper (Kim et al. 2016), the authors utilized variational Bayesian matrix factorization (VBMF) (Nakajima et al. 2015) to determine the desired rank. The corrupted observation V, which equals U plus additional noise ΣZ, is decomposed by VBMF using a Bayesian approach. The aim is to identify a matrix U that can be decomposed as B multiplied by A multiplied by T, where the rank of U is not greater than r. This approach is referred to as R-VBMF (Rank due to VBMF). It is important to note that the user cannot freely set $N_{C,max}$ or $N_{M,max}$ with R-VBMF. Instead, the heuristic determines the computational and memory costs of the approximated network. The following works formulate the rank selection as optimization problems in which the singular values appear in the formulations (Kim, Khan, and Kyung 2019; Idelbayev and Carreira-Perpinán 2020; Liebenwein et al. 2021; Yaguchi et al. 2019). However, their methods are limited in that either heuristics or approximations are used to solve the problem and are layer-wise rank selection methods that do not result in globally optimal configurations. Another line of research involves using a genetic algorithm to determine the proper rank configuration. For the genetic strategy (Li et al. 2022), as used in neural architecture search, it includes processes that several parameters are heuristically determined, and the overall iterative search process is expensive. Distinct from previous research, we make the rank selection problem differentiable making it the first learnable method (i.e., does not depend on heuristics and no need for iteration design for an approximation). However, none of the prior methods were able to formulate the rank selection problem differently; hence, numerous approximations and heuristics were employed to solve the problem.

2.2.2. Filter Pruning

We focused on structured pruning, which removes filters while maintaining the model's regular structure, and the resulting compression can be easily implemented with off-the-shelf CPUs/GPUs (Sui et al. 2021). Pruning is a commonly employed technique in pre-trained deep neural networks (DNNs) to decrease the number of parameters. This can result in reduced storage and model runtime while maintaining performance by retraining the pruned network. Iterative weight pruning is a method that prunes the network while simultaneously retraining it until the desired network size and accuracy are achieved. Pruning is not typically carried out randomly but rather involves removing unimportant weights or neurons using well-informed pruning criteria. Random pruning can have a negative impact on the model's performance and requires more retraining steps to compensate for the removal of important weights or neurons.

Criterion-based methods

Criterion-based methods focus on using a specific metric to evaluate the importance of filters within a layer. One commonly used metric is the L_2-norm/L_1-norm of the filter weights (Li et al. 2016), which measures the magnitude of the filter values. Filters with smaller weights are considered less important and can be pruned without significantly affecting the accuracy of the CNN. Other metrics that have been proposed include the geometric median of the filters and the error contribution toward the output of the current layer.

The L_2-norm or L_1-norm is calculated by summing the squares or absolute values of the weights in a filter, respectively. Filters with a lower L_2-norm or L_1-norm have smaller magnitude weights and are considered less important. By pruning filters with low L_2-norm or L_1-norm, the size of the CNN can be reduced without significantly affecting its accuracy.

The geometric median of the filters is another metric that can be used to evaluate the importance of filters (He et al. 2019b). The geometric median is the point that

minimizes the sum of the distances to all the points in a set. In the context of CNNs, the geometric median of a set of filters is the filter that is closest to all the other filters in terms of their Euclidean distance. Filters that are far away from the geometric median are considered less important and can be pruned.

The error contribution towards the output of the current layer is another metric that can be used to evaluate the importance of filters (Luo, Wu, and Lin 2017; He, Zhang, and Sun 2017). Filters that contribute less to the output of the current layer are considered less important and can be pruned. This method was first proposed in the context of channel pruning, where entire channels of filters are pruned based on their error contribution.

The advantage of criterion-based pruning methods is that they are relatively simple to implement and do not require additional training or specialized modules. However, they often require numerous trial-and-error experiments to achieve the desired resource budget. Additionally, these methods can result in a sparse CNN, where some filters are pruned and others are kept. Sparse CNNs can be challenging to deploy efficiently on hardware because the irregular structure of the network can lead to inefficient memory access patterns.

Learning based methods

The learning-based methods focus on learning each channel's importance by associating scalar gate values. One such method is AutoPruner (Luo and Wu 2020a), which introduces mask generating modules that are appended at the end of each layer. These modules generate channel-wise masks, which are multiplied element-wise with the feature maps. This way, less important channels are removed from the feature maps, and the network's computational cost is reduced. Similarly, SSS (Huang and Wang 2018) proposes a channel pruning method that uses an optimization framework to learn the channel importance scores. SSS employs a sparse softmax function to generate the channel importance scores, which are then used to select the most important channels

for each layer.

Another learning-based method is proposed in PFS (Wang et al. 2020), which proposes a mask learning method for filter selection. PFS starts with an initialized network with frozen weight parameters and learns the binary mask for each filter using a sigmoid function with a constant steepness. The binary mask is then multiplied element-wise with the filter to obtain the pruned filter. The advantage of PFS is that it does not require additional training and can be easily applied to pre-trained models.

In general, learning-based methods tend to perform better than criteria-based methods, as they can adaptively learn the importance of each filter/channel during training, leading to better compression ratios and higher accuracy retention. However, they require additional training and specialized modules, which can increase computational overhead and training time. Additionally, the learned pruning patterns may not be transferable to different architectures, making the method less generalizable.

2.3. Low-rank decomposition in other fields

Natural language processing LoRA (Hu et al. 2021) addresses the challenge of adapting large language models to new tasks with limited labeled data. Traditional methods involve fine-tuning the entire model, which is computationally expensive and data-intensive. LoRA introduces a low-rank adaptation technique that decomposes the model into low-rank and high-rank components. The low-rank part captures general knowledge shared across tasks, while the high-rank part captures task-specific information. By updating only the high-rank component, LoRA reduces computational costs and allows effective adaptation with limited labeled data. This innovation leads to a substantial reduction in memory usage by approximately threefold and a remarkable decrease in the number of parameters, around 10000 times. Models like RoBERTA, DeBERTa, GPT-2, and GPT-3 demonstrate similar or even superior fine-tuning performance with this technique. Although a drawback exists with the pre-defined hyperparameter 'r', our method combines seamlessly, enabling automatic determination of 'r', resulting

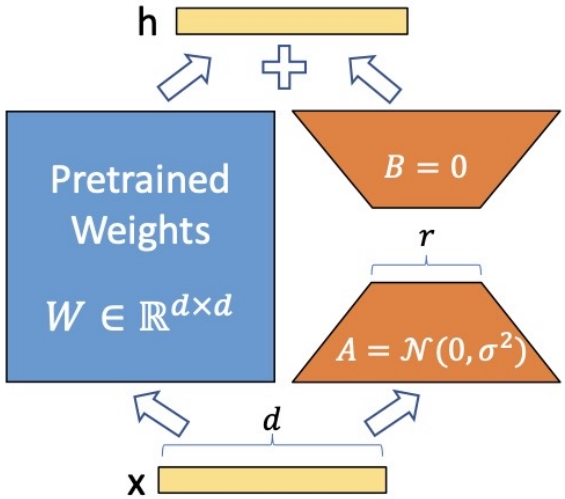


Figure 2.1. The basic LoRA module from the original LoRA study (Figures were adapted from figure 1 of the reference [Hu et al. 2021]). The structure contains two decomposed layers, enabling a small portion of the whole weights to be updated on novel datasets.

in prominent performance enhancements and significantly reducing the search time for finding the optimal 'r'. As a result, the integration of our methodology not only achieves greater efficiency in fine-tuning but also offers a more streamlined and effective approach to improving model performance across various tasks and applications.

Domain generalization The EDG method (Piratla, Netrapalli, and Sarawagi 2020) is proposed to improve domain generalization, which is training models to perform well on unseen domains. Traditional methods are computationally expensive and require abundant labeled data. This approach uses low-rank decomposition to reduce complexity. The method decomposes model parameters into common and specific components. The common component captures shared knowledge, while the specific component captures domain-specific information. By updating only the specific component, which is smaller, the approach reduces the computational burden. With this decomposition, the model can generalize to unseen domains with limited labeled data, leveraging common knowledge.

Image restoration The reweighted Low-Rank factorization with deep prior for the image restoration method (Chen et al. 2022) begins by decomposing the corrupted image into low-rank and sparse components. The low-rank component captures the underlying structure of the image, while the sparse component represents the noise and artifacts. This decomposition is performed using a reweighted optimization process, which enhances the ability to handle complex noise patterns. To incorporate deep learning priors, a deep neural network is trained to learn the characteristics of clean images. The network is then used to guide the restoration process by enforcing similarity to clean images during the optimization. The combination of reweighted low-rank factorization and deep learning priors leads to improved image restoration results.

2.4. Thesis Roadmap

In this thesis, we study how to improve the performance of compressed neural networks using low-rank compression. In Chapter 3, the method using a rank regularizer and beam-search algorithm is proposed. In Chapter 4, combining the low-rank and filter pruning methods is proposed.

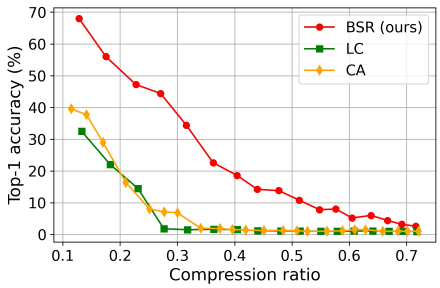
Chapter 3. An Effective Low-Rank Compression with a Joint Rank Selection Followed by a Compression-Friendly Training

3.1. Introduction

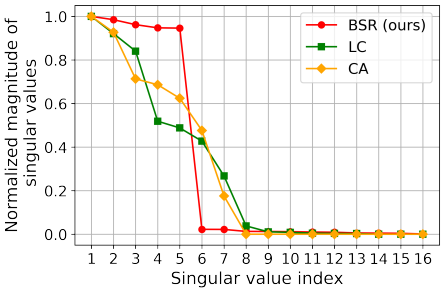
Compression of neural networks has emerged as one of the essential research topics, especially for edge devices that have limited computation power and storage capacity. The most popular compression methods include quantization (Rastegari et al. 2016; Wu et al. 2016), pruning of redundant parameters (Han, Mao, and Dally 2015; Lebedev and Lempitsky 2016; Srinivas and Babu 2015), knowledge distillation from a large network to a small one (Hinton, Vinyals, and Dean 2015; Kim, Park, and Kwak 2018; Romero et al. 2014; Zagoruyko and Komodakis 2016), and network factorization (Alvarez and Salzmann 2017; Denton et al. 2014; Masana et al. 2017; Xue, Li, and Gong 2013).

In this work, we develop a low-rank compression algorithm. To perform a low-rank compression of a neural network, the weight tensor of a layer needs to be decomposed and compressed. While there are many ways to approach this problem, such as a direct tensor decomposition (Kim et al. 2015; Lebedev et al. 2014; Garipov et al. 2016; Novikov et al. 2015), we limit our focus to the basic approach of reshaping the tensor into a two-dimensional matrix followed by an SVD (Singular Value Decomposition) as in (Denton et al. 2014; Tukan et al. 2020; Yu et al. 2017). For a trained neural network, the layer l's tensor is flattened into the weight matrix $\mathbf{W}_l$ of size $m_l \times n_l$, and it is decomposed to keep only the top r_l dimensions. This reduces the computation and memory requirements from $m_l n_l$ to $(m_l + n_l)r_l$, and the reduction can be large when a small r_l is chosen. We focus on two main challenges for developing a highly effective low-rank compression algorithm – the first is *rank-selection* method, and the second

is *compression-friendly training* method. For the first challenge of rank selection, the main problem is how to select r_l. To be precise, the problem is to select the rank vector $\mathbf{r} = [r_1, r_2, \cdots, r_L]^T$ for the L layers such that the desired compression ratio can be achieved without harming the performance too much (Jaderberg, Vedaldi, and Zisserman 2014; Denton et al. 2014; Tai et al. 2015; Zhang et al. 2015a; Wen et al. 2017). We focus on two main challenges for developing a highly effective low-rank compression algorithm – the first is *rank-selection* method, and the second is *compression-friendly training* method. For the first challenge of rank selection, the main problem is how to select r_l. To be precise, the problem is to select the rank vector $\mathbf{r} = [r_1, r_2, \cdots, r_L]^T$ for the L layers such that the desired compression ratio can be achieved without harming the performance too much (Jaderberg, Vedaldi, and Zisserman 2014; Denton et al. 2014; Tai et al. 2015; Zhang et al. 2015a; Wen et al. 2017). Because rank selection is a nonlinear optimization problem, some of the early studies manually selected the rank of each layer, and some of the later studies proposed energy metrics that are based on summary statistics of each layer's singular values. These studies, however, are limited because they decouple the rank selection into each layer and fail to reflect the aspects that are important for a joint rank selection over all L layers.



(a) Rank selection



(b) Singular value distribution

Figure 3.1. Comparison with the baseline algorithms of CA and LC. (a) Performance of rank selection methods: a base neural network (ResNet56 on CIFAR-100) was truncated by the selected ranks, and no further fine-tuning was applied. Modified beam-search (*mBS*) clearly outperforms the other two. (b) Effectiveness of rank regularized training: a base neural network (ResNet56 on CIFAR-100) was regularized by each compression-friendly training method. For the target rank of five, modified stable rank (*mSR*) is the only one that clearly minimizes the singular values other than the top five.

In our work, we propose *mBS* (modified Beam-Search) that can perform a joint selection of ranks over all the layers using a beam search. Also, *mBS* adopts validation accuracy as the joint selection metric. Because the validation performance is directly evaluated for each candidate rank vector, the actual performance after the completion of low-rank compression correlates well with the selection metric. Most of the previous rank-selection methods are based on heuristics or summary statistics. The performance of *mBS* is shown in Figure 3.1a where *mBS* is compared against two low-rank compression algorithms, CA (Alvarez and Salzmann 2017) and LC (Idelbayev and Carreira-Perpinán 2020). Both are low-rank compression algorithms that utilize compression-friendly training. To make the beam search fast, the size of the rank-selection validation dataset is kept small – around the size of a mini-batch. Additionally, the beam search algorithm is modified for speed up as will be explained in Section 3.5.2.

For the second challenge of compression-friendly training, the goal is to train the neural network to be ready for low-rank decomposition. The previous works used a forced weight matrix truncation in the middle of training (Alvarez and Salzmann 2017) or a norm distance regularization with the truncated weight matrices (Idelbayev and Carreira-Perpinán 2020). Both require updates on the regularization loss as the training proceeds. In our method of *mSR* (modified Stable Rank), the regularization loss depends only on the target rank of each layer. Because we first fix the target rank of each layer using *mBS* and never update it, *mSR* can have smoother learning dynamics and achieve better compression-friendly training, as can be seen in Figure 3.1b.

The *BSR* (Beam-search and Stable Rank) algorithm is defined as a sequential application of *mBS* for a practical rank-selection and *mSR* for effective compression-friendly training. As we will show in Section 3.6, it outperforms the latest benchmark low-rank compression algorithms and performs comparably against the state-of-the-art pruning algorithms.

3.2. Contributions

The **main contributions** of our study can be summarized as follows:

- We propose a new rank selection and regularized training method called *BSR* for effective low-rank compression.
- *BSR* consists of two novel algorithms: (1) The *mBS* (modified Beam Search) algorithm, which can perform a joint selection of ranks over all the layers. (2) The *mSR* (modified Stable Rank) regularizer, which can train a model into a compression-friendly form.
- Through experimental validation, we have confirmed that our rank regularizer is more effective in regularizing compared to conventional nuclear norm-based regularizers.
- Through experimental validation, we have confirmed that our rank regularizer is more effective in regularizing compared to conventional nuclear norm-based regularizers.
- In the field of compression, we have innovatively utilized a modified version of the beam search method to discover rank configurations. This is the first time such an approach has been applied in this context. Through experimental validation, we have empirically confirmed that our method is more effective in finding optimal rank configurations compared to existing techniques.
- For popular vision benchmarks, *BSR* outperforms the state-of-the-art low-rank compression methods with large margins.
- We compare *BSR* with pruning compression, which is also a structured compression method. *BSR* provides a competitive performance compared to the recent pruning algorithms.

- We demonstrate that *BSR* can be easily combined with the quantization method for additional compression.

3.3. Related works

We propose a novel low-rank compression algorithm, *BSR*, to overcome two main challenges in the field of low-rank compression. *BSR* consists of rank selection and rank-regularized training. For rank selection, we propose a modified beam search algorithm reformulated to find rank configurations efficiently. For rank-regularized training, we propose a modified stable rank regularizer that does not require repetitive updates of the regularization loss. In this section, we summarize beam-search-related works in Section 3.3.1, and rank regularizer-related works in Section 3.3.2.

3.3.1. Beam search

Beam search is a technique for searching a tree or graph, especially when the solution space is vast (Xu and Fern 2007; Antoniol et al. 1995; Furcy and Koenig 2005). It is based on a heuristic of developing K solutions in parallel while repeatedly inspecting adjacent nodes of the K solutions in the graph. K is commonly referred to as the beam size, and $K = 1$ corresponds to the greedy search (Huang, Fayong, and Guo 2012), and $K = \infty$ corresponds to the breadth-first search (Meister, Vieira, and Cotterell 2020). Obviously, beam search is a compromise between the two, where K is the control parameter. Beam search has been widely adopted, especially for natural languages processing tasks such as speech recognition (Lowerre 1976), neural machine translation (Lowerre 1976; Boulanger-Lewandowski, Bengio, and Vincent 2013), scheduling (Habenicht and Monch 2002), and generation of a natural language adversarial attack (Tengfei.Z et al. 2021). In our work, we form a graph by defining nodes as possible rank vectors. For the graph, we modify the basic beam search algorithm for a faster search where deeper (e.g., level s) children nodes can be searched.

3.3.2. Stable rank and rank regularization

Formally speaking, the stable rank of a matrix is defined as the ratio between the squared Frobenius norm and the squared spectral norm (Rudelson and Vershynin 2007). Simply speaking, the definition boils down to the ratio between the 'sum of squared singular values' and the 'squared value of the largest singular value'. Therefore, a smaller stable rank implies a relatively larger portion of energy in the largest singular value. In previous works, a stable rank normalization was studied for improving generalization (Sanyal, Torr, and Dokania 2019), and a stable rank approximation was utilized for performance tuning (Choi et al. 2021). In our work, we modify the stable rank's denominator to the sum of top r singular values such that we can concentrate the activation signals in the top r dimensions. While our approach requires only the target rank r to be specified, the previous compression-friendly training calculated the truncated matrices and directly used their element values for the regularization. Therefore, the previous methods required a frequent update of the truncated matrices during the training. In our *BSR* method, we calculate the target rank vector only once in the beginning and keep it fixed. Other rank regularizers are summarized in Table 3.1. All the methods are based on nuclear norms, and the nuclear norm and variants of nuclear norms have been proven the most effective regularizer for rank control. However, as demonstrated in the experimental results section of the main text, we have empirically verified that our method, derived by modifying the stable rank, effectively controls the rank in comparison to other approaches based on nuclear norms.

Name	Regularizer
Nuclear norm	$\Sigma_{i=1}^{k}\lambda\sigma_i$
Elastic-Net (Kim, Lee, and Oh 2015)	$\Sigma_{i=1}^{k}\lambda(\sigma_i+\gamma\sigma_i^2)$
Sp-norm (Mohan and Fazel 2012; Nie, Huang, and Ding 2012)	$\Sigma_{i=1}^{k}\lambda\sigma_i^p$
TNN (Hu et al. 2012)	$\Sigma_{i=r+1}^{k}\lambda\sigma_i$
WNN (Gu et al. 2014; 2017)	$\Sigma_{i=r+1}^{k}\lambda w_i\sigma_i$
CNN (Sun, Xiang, and Ye 2013)	$\Sigma_{i=r+1}^{k}\lambda min(\sigma_i,\theta)$
Logarithm (Friedman 2012)	$\Sigma_{i=1}^{k}\frac{\lambda log(\gamma\sigma_i+1)}{log(\gamma+1)}$
LNN (Peng et al. 2015)	$\Sigma_{i=1}^{k}\lambda log(\sigma_i+1)$
Geman (Geman and Yang 1995)	$\Sigma_{i=1}^{k}\frac{\lambda\sigma_i}{\sigma_i+\gamma}$
Laplace (Trzasko and Manduca 2008)	$\Sigma_{i=1}^{k}\lambda(1-exp(-\frac{\sigma_i}{\gamma}))$

Table 3.1. Commonly used rank regularizer in many fields.

3.4. The basics of low-rank compression

3.4.1. The basic process

A typical process of low-rank compression consists of four steps: 1) train a deep neural network, 2) select rank assignments over L layers, 3) factorize weight matrices using truncated SVD (Denton et al. 2014; Masana et al. 2017; Xue, Li, and Gong 2013) according to the selected ranks, and 4) fine-tune the truncated model to recover the performance as much as possible. In our work, we mainly focus on the rank selection step and an additional step of compression-friendly training. The additional step is placed between step 2 and step 3.

3.4.2. Compression ratio

Consider an L-layer neural network with $\mathbf{W} = (\mathbf{W}_1, \mathbf{W}_2, \cdots, \mathbf{W}_L)$, where $\mathbf{W}_l \in \mathbb{R}^{m_l \times n_l}$, as its weight matrices. Without a low-rank compression, the rank vector $\mathbf{r}$ corresponds to the full rank vector of $\mathbf{r}_{full} = [R_1, \cdots, R_L]^T$ where $R_l = \min(m_l, n_l)$. The rank selection is performed over the set of $\mathcal{R} = \{\mathbf{r} \in \mathbb{N}^L \mid \mathbf{r} = [r_1, r_2, \cdots, r_L]^T, 0 < r_l \leq \min(m_l, n_l)\}$, and the selected rank vector is denoted as $\mathbf{r}_{select}$. With $\mathbf{r}_{select}$, we can perform a truncated SVD. For lth layer's weight matrix $\mathbf{W}_l = \mathbf{U}_l \mathbf{S}_l \mathbf{V}_l^T$, we keep only the largest r_l singular values to obtain $\hat{\mathbf{W}}_l = \hat{\mathbf{U}}_l \hat{\mathbf{S}}_l \hat{\mathbf{V}}_l^T$ where $\hat{\mathbf{U}}_l \in \mathbb{R}^{m \times r_l}$, $\hat{\mathbf{S}}_l \in \mathbb{R}^{r_l \times r_l}$, and $\hat{\mathbf{V}}_l \in \mathbb{R}^{n \times r_l}$. Then, the compression is achieved by replacing $\mathbf{W}_l$ with a cascade of two matrices: $\hat{\mathbf{S}}_l \hat{\mathbf{V}}_l^T$ and $\hat{\mathbf{U}}_l$. Obviously, the computational benefit stems from the reduction in the matrix multiplication loads: mn for the original $\mathbf{W}_l$ and $(m+n)r_l$ for $\hat{\mathbf{S}}_l \hat{\mathbf{V}}_l^T$ and $\hat{\mathbf{U}}_l$. Similar results hold for convolutional layers. Finally, the compression ratio $C(\mathbf{r})$ for the selected rank vector $\mathbf{r}$ can be calculated as

$$C(\mathbf{r}) = 1 - \frac{\sum_{l=1}^{L} \{m_l n_l \mathbb{1}_l + r_l(m_l + n_l)(1 - \mathbb{1}_l)\}}{\sum_{l=1}^{L} m_l n_l} \tag{3.1}$$

where $\mathbb{1}_l$ is a simplified notation of $\mathbb{1}(m_l, n_l, r_l)$ that is defined as 1 if $m_l n_l \leq r_l(m_l + n_l)$ and 0 otherwise.

3.5. Methodology

3.5.1. Overall process

The overall process of *BSR* low-rank compression is shown in Figure 4.3. Starting from a fully trained network, phase one of *BSR* performs rank selection using *mBS* algorithm where it requires only the desired compression ratio (C_d) as the input and a small validation dataset (D_{val}). Once phase one is completed, $\mathbf{r}_{select}$ is fixed, and rank-regularized training is performed in phase two. The strength of *mSR* is controlled by λ, where its strength is gradually increased in a scheduled manner. Upon the completion of phase two, the trained network is truncated using singular value decomposition according to $\mathbf{r}_{select}$. Then, a final fine-tuning is performed to complete the compression.

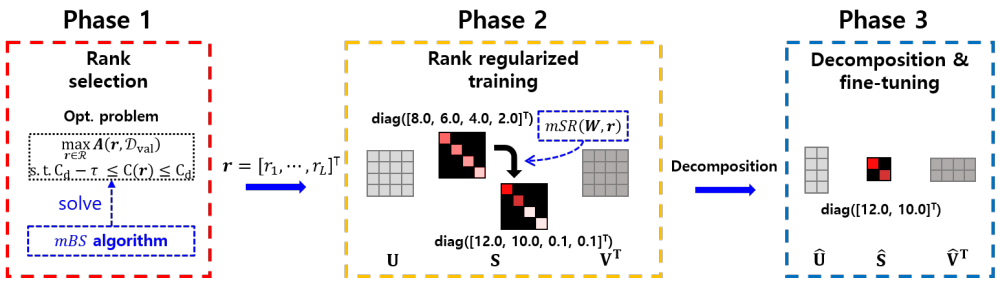


Figure 3.2. Overall process of *BSR* algorithm.

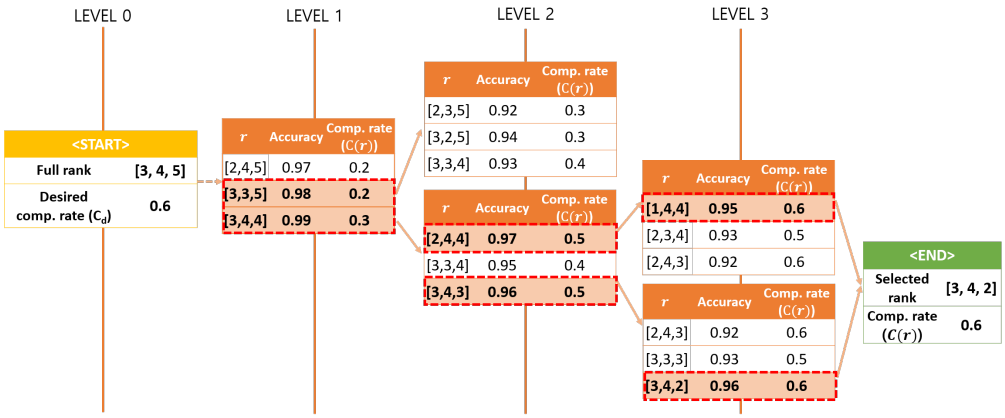


Figure 3.3. Illustration of *mBS* search process for $L = 3$, $K = 2$, $s = 1$, and $C_d = 0.6$.

3.5.2. Modified beam-search (*mBS*) for rank selection

When a neural network with $\mathbf{W}$ is truncated according to the rank vector $\mathbf{r}$, the accuracy can be evaluated with a small validation dataset $\mathcal{D}_{val}$ and the accuracy is denoted as $A(\mathbf{r}, \mathcal{D}_{val})$. The corresponding compression ratio can be calculated as $C(\mathbf{r})$. Our goal of rank selection is to find the $\mathbf{r}$ with the highest accuracy while the compression ratio is sufficiently close to the desired compression ratio C_d. This problem can be formulated as below.

$$\begin{aligned} &\max_{\mathbf{r}\in\mathcal{R}} A(\mathbf{r}, \mathcal{D}_{val}) \\ &\text{s.t.} \quad C_d - \tau \leq C(\mathbf{r}) \leq C_d \end{aligned} \tag{3.2}$$

Note that since C_d is a real number, we have introduced a small constant τ for relaxing the desired compression ratio. This relaxation forces the returned solution to have a compression ratio close to C_d, and it is also utilized as a part of the exit criteria. The problem in Equation 3.2 is a combinatorial optimization problem (Reeves 1993), and a simple greedy algorithm can be applied as in (Zhang et al. 2015a). Because the cardinality of the search space $\mathcal{R}$ is extremely large, however, greedy algorithms hardly produce good results in a reasonable computation time. On the other hand, a full search is also unacceptable because of its long search time. As a compromise, we adopt a beam-search framework and make adequate adjustments. Before presenting the details of *mBS*, a simple illustration of how Equation 3.2 can be solved with our modified beam search is presented in Figure 3.3. The details of *mBS* can be explained as the following.

- **Stage 1:** Initialize the level as $lv_{[1]} = 1$. Initialize the top-K set as $\mathcal{B}_{[1]} = \{\mathbf{r}^1_{[1]}\}$, where $\mathbf{r}^1_{[1]} = \mathbf{r}_{full}$ (full rank assignments).

- **Stage 2:** Move to the next level by adding the pre-chosen step size s, $lv_{[t]} = lv_{[t-1]} + s$. For each element in $\mathcal{B}_{[t-1]}$, find all of its descendants in $lv_{[t]}$ and add them to the candidate set $\mathcal{T}_{[t]}$. Exclude the descendants that perform excessive compression by checking the condition $C(\mathbf{r}_{[t]}) \leq C_d$.

- **Stage 3:** Calculate the new top-K set at level $lv_{[t]}$ by evaluating the small validation dataset and by finding the ordered top K elements:

$$\mathcal{B}_{[t]} = \underset{\mathbf{r}^1_{[t]},\cdots\mathbf{r}^K_{[t]}\in\mathcal{T}_{[t]}}{\arg\max}\ A(\mathbf{r}_{[t]}, \mathcal{D}_{val})$$

- **Stage 4:** Repeat **Stage 2** and **Stage 3** until $C_d - \tau \leq C(\mathbf{r}^1_{[t]}) \leq C_d$ is satisfied for the best element $\mathbf{r}^1_{[t]}$. If the condition is satisfied, return $\mathbf{r}^1_{[t]}$ as $\mathbf{r}_{select}$.

An implementation of this process is provided in Algorithm 1. Compared to the standard beam search, the main modification we make is the introduction of the *level step size* s for the speed-up of the search. For the rank selection, a weight matrix's rank needs to be reduced by a sufficient amount and satisfy the minimum condition of $r_l(m_l + n_l) \leq m_l n_l$ to achieve a positive compression effect. Because most of the weight matrices can allow a significant amount of rank reduction without harming the performance at the beginning of the search, we can choose s as a number between three and ten to make the search faster. As the search progresses, we reduce s and improve the search resolution whenever no candidate can be found at the next level. Besides s, the beam size K is another important parameter that determines the trade-off between speed and search resolution. Instead of performing a fine and slow search with a small s and a large K, we perform a fast search a few times with different configurations and choose the best $\mathbf{r}_{select}$. The configuration details can be found in Section 3.6.1, and ablation studies for K and s can be found in Section 3.6.3.

Algorithm 1 modified Beam Search (*mBS*) for rank selection

Input: desired comp. ratio C_d; validation data $\mathcal{D}_{val}$; beam size K; level step size s

Output: selected rank $\mathbf{r}_{select}$

Required: ratio function $C(\mathbf{r})$; base network $\mathbf{M}(\mathbf{r})$ with rank $\mathbf{r}$; evaluation function $A(\mathbf{r}, \mathcal{D})$

Initialize: $\mathbf{r}_{select} \leftarrow \mathbf{r}_{full}$;

top-K rank set $\mathcal{B} \leftarrow \{(\mathbf{r}_{full},\ C(\mathbf{r}_{full}),\ A(\mathbf{r}_{full}, \mathcal{D}_{val})\}$

while ($\mathcal{B}$ is changed) $\vee$ ($C_d - \tau \leq \mathbf{r}_{select} \leq C_d$) **do**
  $\mathcal{T} = \{\phi\}$
  for $(\mathbf{r}_p,\ C_p,\ A_p)$ in $\mathcal{B}$ **do**
    for $i = 1$ to L **do**
      $\mathbf{r}_c \leftarrow \mathbf{r}_p$
      $\mathbf{r}_c[i] \leftarrow \mathbf{r}_c[i] - s$
      $C_c \leftarrow C(\mathbf{r}_c)$
      if $C_c \leq C_d$ **then**
        $A_c \leftarrow A(\mathbf{r}_c, \mathcal{D}_{val})$
        $\mathcal{T} \cup (\mathbf{r}_c, C_c, A_c)$
      end if
    end for
  end for
  $\mathcal{B} \leftarrow \mathbf{TopK}(\mathcal{T}; keys = [\mathbf{A}, \mathbf{r}, \mathbf{rand}])$
  $\mathbf{r}_{select} \leftarrow \mathcal{B}[0]$
end while
return $\mathbf{r}_{select}$

3.5.3. Modified stable rank (*mSR*) for regularized training

For a weight matrix $\mathbf{W}_l \in \mathbb{R}^{m_l \times n_l}$, stable rank is defined as

$$SR(\mathbf{W}_l) = \frac{\|\mathbf{W}_l\|_F^2}{\|\mathbf{W}_l\|_2^2} = \frac{\sum_{i=1}^{R_l}(\sigma_i^l)^2}{(\sigma_1^l)^2}, \tag{3.3}$$

where σ_i^l is the ith singular value of $\mathbf{W}_l$. Because our goal of compression-friendly training is to have almost no energy in the dimensions other than the top r_l dimensions, we modify the stable rank as below.

$$mSR(\mathbf{W}_l, r_l) = \frac{tr(\mathbf{\Sigma}_l^{r_l:R_l})}{tr(\mathbf{\Sigma}_l^{1:r_l})} = \frac{\sum_{i=r_l+1}^{R_l}\sigma_i^l}{\sum_{i=1}^{r_l}\sigma_i^l} \tag{3.4}$$

The modified stable rank *mSR* is different from the stable rank in four ways. First, it is dependent on the input parameter r_l. Second, the summation in the denominator is performed over the largest r_l singular values. Third, the largest r_l singular values are excluded in the numerator's summation. Fourth, the singular values are not squared. The third and fourth differences make *mSR* regularization allocate less energy in the undesired dimensions as shown in Figure 3.1a. The compression-friendly training is performed by minimizing the loss of $\mathcal{L}(\mathbf{W}) + \lambda \sum_{l=1}^{L} mSR(\mathbf{W}_l, r_l)$, where the first term is the original loss of the learning task and the second term is the *mSR* as a penalty loss.

Through our empirical evaluations, we have confirmed that *mSR* can stably affect the weight matrices. In fact, the gradient of *mSR* can be easily derived. To do so, we decompose $\mathbf{W}_l = \mathbf{U}_l\mathbf{\Sigma}_l\mathbf{V}_l^T$ into two parts by allocating the first r_l dimensions into $\mathbf{W}_l^{1:r_l}$ and the remaining dimensions into $\mathbf{W}_l^{r_l:R_l}$ as below.

$$\begin{aligned}\mathbf{W}_l &= \mathbf{W}_l^{1:r_l} + \mathbf{W}_l^{r_l:R_l} \\ &= \mathbf{U}_l^{1:r_l}\mathbf{\Sigma}_l^{1:r_l}(\mathbf{V}_l^{1:r_l})^T + \mathbf{U}_l^{r_l:R_l}\mathbf{\Sigma}_l^{r_l:R_l}(\mathbf{V}_l^{r_l:R_l})^T\end{aligned} \tag{3.5}$$

Then, the derivative can be derived as the following.

$$
\begin{aligned}
\frac{\partial mSR(\mathbf{W}_l, r_l)}{\partial \mathbf{W}_l} &= \frac{\partial \left(\frac{tr(\mathbf{\Sigma}_l^{r_l:R_l})}{tr(\mathbf{\Sigma}_l^{1:r_l})} \right)}{\partial \mathbf{W}_l} \\
&= \frac{1}{(tr(\mathbf{\Sigma}_l^{1:r_l}))^2} \left(tr(\mathbf{\Sigma}_l^{1:r_l}) \frac{\partial tr(\mathbf{\Sigma}_l^{r_l:R_l})}{\partial \mathbf{W}_l} - tr(\mathbf{\Sigma}_l^{r_l:R_l}) \frac{\partial tr(\mathbf{\Sigma}_l^{1:r_l})}{\partial \mathbf{W}_l} \right) \\
&= \frac{tr(\mathbf{\Sigma}_l^{r_l:R_l})}{tr(\mathbf{\Sigma}_l^{1:r_l})} \left[\frac{1}{tr(\mathbf{\Sigma}_l^{r_l:R_l})} \left(\mathbf{V}_\mathbf{l}^{\mathbf{r_l}:\mathbf{R_l}} (\mathbf{U}_l^{r_l:R_l})^T - \frac{\partial tr((\mathbf{U}_l^{r_l:R_l})^T \mathbf{W}_l^{r_l:R_l} \mathbf{V}_l^{r_l:R_l})}{\partial \mathbf{W}_l} \right) \right. \\
&\quad \left. - \frac{1}{tr(\mathbf{\Sigma}_l^{1:r_l})} \left(\mathbf{V}_l^{1:r_l} (\mathbf{U}_l^{1:r_l})^T - \frac{\partial tr((\mathbf{U}_l^{1:r_l})^T \mathbf{W}_l^{r_l:R_l} \mathbf{V}_l^{1:r_l})}{\partial \mathbf{W}_l} \right) \right] \\
&= \frac{tr(\mathbf{\Sigma}_l^{r_l:R_l})}{tr(\mathbf{\Sigma}_l^{1:r_l})} \left(\frac{\mathbf{U}_l^{r_l:R_l} (\mathbf{V}_l^{r_l:R_l})^T}{tr(\mathbf{\Sigma}_l^{r_l:R_l})} - \frac{\mathbf{U}_l^{1:r_l} (\mathbf{V}_l^{1:r_l})^T}{tr(\mathbf{\Sigma}_l^{1:r_l})} \right).
\end{aligned}
\tag{3.6}
$$

A crucial issue with the above *mSR* regularization is its effect on the computational overhead. Use of *mSR* requires a repeated calculation of singular value decomposition (SVD) that is computationally intensive. To deal with this issue, we adopted the randomized matrix decomposition method in (Erichson et al. 2016). Because the target rank r_l is typically chosen to be small, the practical choice of implementation has almost no effect on the regularization while significantly reducing the computational burden. Furthermore, we obtain an additional reduction by calculating SVD only once every 64 iterations. Consequently, the training time of *mSR* regularization remains almost the same as the un-regularized training.

3.6. Experiments

3.6.1. Experimental setting

Baseline models and datasets

To investigate the effectiveness and generalizability of *BSR*, we evaluate its compression performance for a variety of models and datasets. For the comparison against the low-rank compression methods, we mainly followed the experimental settings of

LC (Idelbayev and Carreira-Perpinán 2020): ResNet32 and ResNet56 on CIFAR-10, ResNet56 on CIFAR-100, and AlexNet on ImageNet (ILSVRC 2012). For the comparison against the structured pruning methods, we have chosen the two benchmarks that are most commonly evaluated in the pruning community: ResNet56 on CIFAR-10 and ResNet50 on ImageNet. We used the standard preprocessing techniques for CIFAR-10, CIFAR-100, and ImageNet (i.e., random horizontal flip, random crop, and normalization were applied). Most of the structured pruning works provide their performance results for at least one of the two benchmarks.

Rank selection configuration

BSR performs rank selection only once. Therefore, it is important to find a rank vector that can result in a high performance after compression-friendly training. To improve the search speed and to make the search algorithm *mBS* robust, we use three settings of (s, K) as $\{(3, 5), (5, 5), (10, 5)\}$. Search for $\mathbf{r}_{select}$ is performed three times with the three settings, and the best-performing one is selected as the final choice. Then, compression-friendly training is performed only once after the final selection. Note that other low-rank algorithms require multiple attempts of compression-friendly training. Because we choose s that is larger than one, *mBS* might not be able to find a solution. When this happens, the level step size s is multiplied by $\gamma = 0.5$ and rounded, and the search is continued from the last K candidates.

Rank regularized training configuration

We used the initial learning rate of $\eta_0 = 0.01$ and a cosine annealing method was used as the learning rate scheduler. We used Nesterov's accelerated gradient method with a 0.9 momentum on mini-batches of size 256 for the ImageNet dataset and 128 for the others. A regularization strength scheduling was introduced for stable regularized training, where λ was gradually increased. To be specific, we used a regularization strength scheduling of $\lambda_j = \lambda_0 \cdot b$ where λ_j is updated every 15 epochs with the multiplication

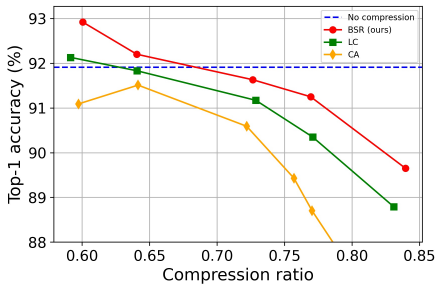
of constant b. The values of (λ_0, b) were chosen as one of (1.0, 1.2) and (2.0, 1.2) for ResNet50 on ImageNet and (0.02, 1.5) and (0.05,1.5) for all the others. For ResNet50 on ImageNet, a larger λ_0 value is beneficial for the early epochs.

3.6.2. Experimental results

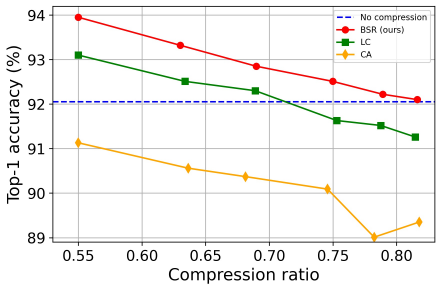
We provide three sets of experimental results. The first set focuses on the comparison against the low-rank compression algorithms, the second set focuses on applying *BSR* on lightweight networks, and the third set focuses on the comparison against the structured pruning algorithms.

Comparison against low-rank compression algorithms

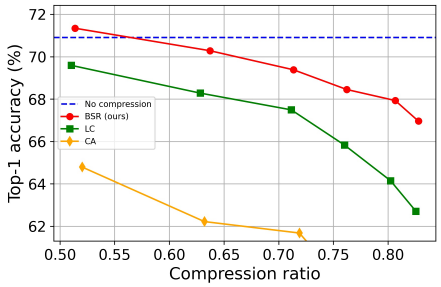
The results are provided in terms of compression ratio and FLOPs. For ResNet, convolution layers contain most of the parameters, and we applied *BSR* only over the convolutional layers. For AlexNet, the fully connected layers cannot be ignored, and we applied *BSR* over the fully connected layers and the convolutional layers. The results are shown in Figure **??**, where the upper four figures are the results of compression ratio, and the lower four figures are the results for FLOPs. In all figures, *BSR* outperforms LC and CA. For CA and LC, the compression ratio cannot be fully controlled, and the final compression ratio is dependent on the hyperparameter setting. We have tried numerous hyperparameter settings to generate the CA and LC results. For *BSR*, the compression ratio can be fully controlled without any need for tuning. Therefore, we have first generated LC results and have chosen the compression ratios of *BSR* to be the same as what LC ended up with.



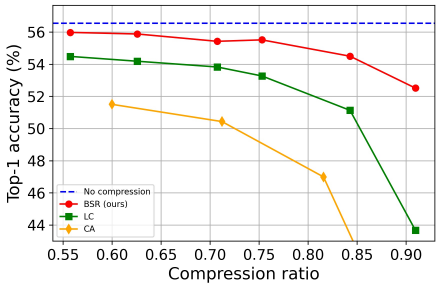
(a) ResNet32 on CIFAR10, Comp. ratio



(b) ResNet56 on CIFAR10, Comp. ratio

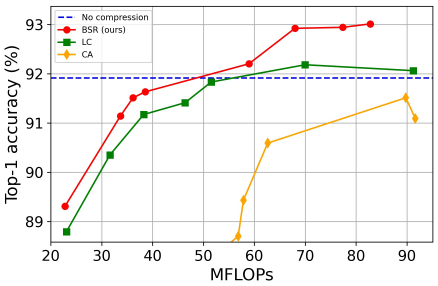


(c) ResNet56 on CIFAR100, Comp. ratio

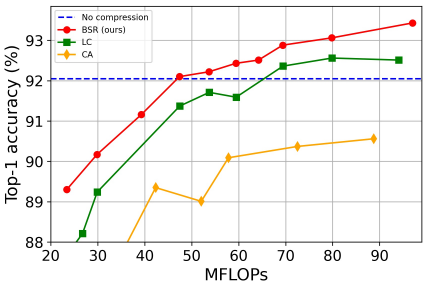


(d) Alexnet on ImageNet, Comp. ratio

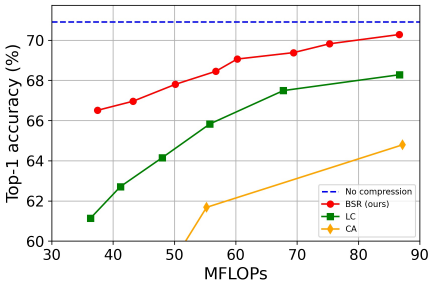
Figure 3.4. Comparison of *BSR* with CA and LC for (a) ResNet32 on CIFAR-10, (b) ResNet56 on CIFAR-10, (c) ResNet56 on CIFAR-100, and (d) AlexNet on ImageNet. For AlexNet, we used the pre-trained Pytorch model as the base network (56.55% for top-1 accuracy).



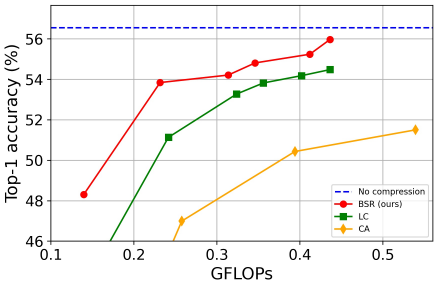
(a) ResNet32 on CIFAR10, Flops



(b) ResNet56 on CIFAR10, Flops

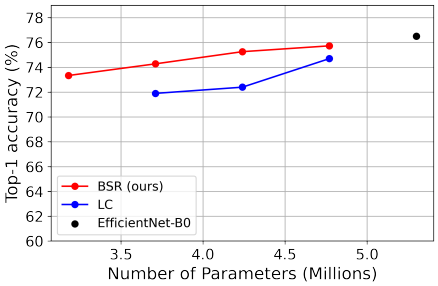


(c) ResNet56 on CIFAR100, Flops

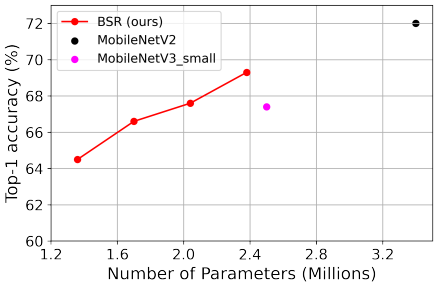


(d) Alexnet on ImageNet, Flops

Figure 3.5. Comparison of *BSR* with CA and LC for (a) ResNet32 on CIFAR-10, (b) ResNet56 on CIFAR-10, (c) ResNet56 on CIFAR-100, and (d) AlexNet on ImageNet. For AlexNet, we used the pre-trained Pytorch model as the base network (56.55% for top-1 accuracy).



(a) EfficientNet-B0 on ImageNet



(b) MobileNetV2 on ImageNet

Figure 3.6. Application of *BSR* on lightweight networks.

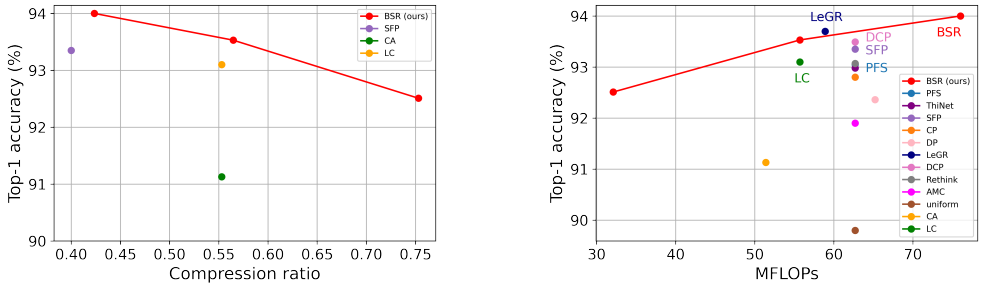
Compressing lightweight networks with *BSR*

EfficientNet and MobileNetV2 are well-known models that have been designed specifically as lightweight models with high performance. We applied *BSR* to the two networks to see if *BSR* can be useful for such lightweight networks. The results are shown in Figure 3.6. Although EfficientNet and MobileNetV2 are already compact, our method can further compress the models at the cost of a slight performance sacrifice. *BSR* outperforms LC method as shown in Figure 3.6a. In addition, MobileNetV2 compressed with *BSR* performs better than MobileNetV3 as shown in Fig 3.6b. Considering that MobileNetV3 is a compacter version in the MobileNet family, it is interesting to note that MobileNetV2 with *BSR* outperforms MobileNetV3.

Comparison against structured pruning algorithms

The previous low-rank methods typically perform worse than SOTA pruning methods. Because both low-rank and pruning are structured methods, we wanted to show that our new low-rank method is comparable to the SOTA pruning methods and, therefore, it is a competitive structured compression solution. We compare *BSR* against the latest structured pruning algorithms.

The benchmark pruning algorithms include naive uniform channel number shrinkage (uniform), ThiNet (Luo, Wu, and Lin 2017), Channel Pruning (CP) (He, Zhang, and Sun 2017), Discrimination-aware Channel Pruning (DCP) (Zhuang et al. 2018), Soft Filter Pruning (SFP) (He et al. 2018a), rethinking the value of network pruning (Rethink) (Liu et al. 2018), and include most recent methods such as Pruning From Scratch (PFS) (Wang et al. 2020), end-to-end trainable method (AutoPruner) (Luo and Wu 2020a), Compression Using Residual-connections and Limited-data (CURL) (Luo and Wu 2020b), and Learned Global Ranking(LeGR) (Chin et al. 2020).



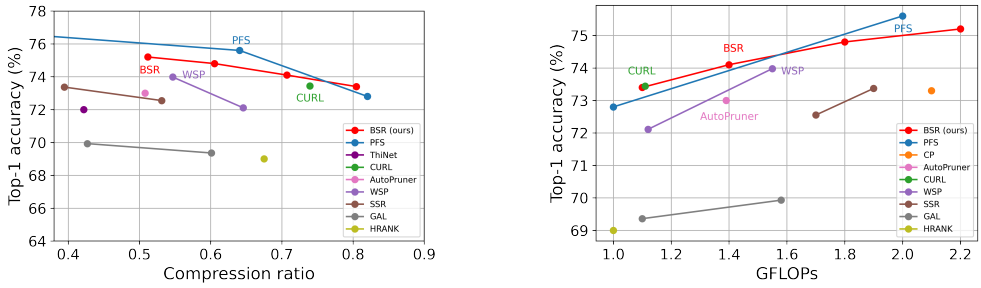
(a) Compression ratio

(b) FLOPs

Figure 3.7. Comparison for ResNet56 on CIFAR-10 (a) Compression ratio (b) FLOPs

Range of FLOPs reduction ratio	Method	Algorithm	Test acc. in %	Δ Test acc. in %	MFLOPs (Reduction ratio)	Number of parameters (Compression ratio)
40% ∼ 50%	Low-rank	**BSR (ours)**	92.73 → **94.00**	+ 1.27	76.0 (40 %)	0.49M (0.42)
	Pruning	DP (Kim et al. 2019)	92.66 → 92.36	- 0.30	65.2 (48 %)	N/A
50% ∼ 60%	Pruning	**LeGR** (Chin et al. 2020)	93.90 → **93.70**	- 0.20	58.9 (53 %)	N/A
	Low-rank	**BSR (ours)**	92.73 → **93.53**	+ 0.80	55.7 (56 %)	0.37M (0.56)
	Pruning	DCP (Zhuang et al. 2018)	93.80 → 93.49	- 0.31	62.7 (50 %)	N/A
	Pruning	SFP (He et al. 2018a)	93.59 → 93.35	- 0.24	62.7 (50 %)	0.51M (0.40)
	Low-rank	LC (Idelbayev and Carreira-Perpinán 2020)	92.73 → 93.10	+ 0.37	55.7 (56 %)	0.38M (0.55)
	Pruning	Rethink (Liu et al. 2018)	93.80 → 93.07	- 0.73	62.7 (50 %)	N/A
	Pruning	PFS (Wang et al. 2020)	93.23 → 93.05	- 0.18	62.7 (50 %)	N/A
	Pruning	ThiNet (Luo, Wu, and Lin 2017)	93.80 → 92.98	- 0.82	62.7 (50 %)	N/A
	Pruning	CP (He, Zhang, and Sun 2017)	93.80 → 92.80	- 1.00	62.7 (50 %)	N/A
	Low-rank	CA (Alvarez and Salzmann 2017)	92.73 → 91.13	- 1.60	51.4 (59 %)	0.38M (0.55)
	Pruning	AMC (He et al. 2018b)	92.80 → 91.90	- 0.90	62.7 (50 %)	N/A
	Pruning	Uniform	92.80 → 89.80	- 3.00	62.7 (50 %)	N/A
Above 60%	Low-rank	**BSR (ours)**	92.73 → **92.51**	- 0.22	32.1 (74 %)	0.21M (0.75)

Table 3.2. The performance for ResNet56 on CIFAR-10 is summarized for *BSR*, LC, CA, and structured pruning. For the structured pruning, we have aggregated the literature records that are available. Because most of the literature records are provided in FLOPs only, we have grouped the records according to the range of FLOPs reduction ratio for comparison.



(a) Compression ratio

(b) FLOPs

Figure 3.8. Comparison for ResNet50 on ImageNet (a) Compression ratio (b) FLOPs

Range of FLOPs reduction ratio	Method	Algorithm	Test acc. in %	Δ Test acc. in %	GFLOPs (Reduction ratio)	Number of parameters (Compression ratio)
0% ∼ 40%	Pruning	**PFS** (Wang et al. 2020)	76.1 → **76.7**	+ 0.6	3.0 (25 %)	17.9M (0.30)
	Pruning	ThiNet (Luo, Wu, and Lin 2017)	73.0 → 72.0	- 1.0	2.6 (37 %)	14.8M (0.42)
40% ∼ 50%	Pruning	**PFS** (Wang et al. 2020)	76.1 → **75.6**	- 0.5	2.0 (49 %)	9.2M (0.64)
	Low-rank	BSR (ours)	76.2 → 75.0	- 1.1	2.2 (47 %)	12.5M (0.51)
	Pruning	SFP (He et al. 2018a)	76.2 → 74.6	- 1.6	2.4 (42 %)	N/A
	Pruning	CP (He, Zhang, and Sun 2017)	76.1 → 73.3	- 2.8	2.1 (49 %)	N/A
50% ∼ 60%	Low-rank	**BSR (ours)**	76.2 → **74.8**	- 1.4	1.8 (55 %)	10.1M (0.60)
	Pruning	SSR (Lin et al. 2019a)	76.2 → 73.4	- 2.8	1.9 (55 %)	15.5M (0.39)
	Pruning	CP (He, Zhang, and Sun 2017)	76.1 → 73.3	- 2.8	2.1 (51 %)	N/A
60% ∼ 70%	Low-rank	**BSR (ours)**	76.2 → **74.1**	- 2.1	1.4 (67 %)	7.5M (0.70)
	Pruning	WSP (Guo et al. 2021)	76.2 → 74.0	- 2.2	1.5 (63 %)	11.6M (0.55)
	Pruning	AutoPruner (Luo and Wu 2020a)	76.1 → 73.0	- 3.1	1.4 (65 %)	12.6M (0.50)
	Pruning	SSR (Lin et al. 2019a)	76.2 → 72.6	- 3.6	1.7 (60 %)	12.0M (0.53)
	Pruning	GAL (Lin et al. 2019b)	76.2 → 69.9	- 6.3	1.6 (62 %)	14.7M (0.42)
Above 70%	Low-rank	**BSR (ours)**	76.2 → **73.4**	- 2.8	1.1 (73 %)	**5.0M (0.80)**
	Pruning	**CURL** (Luo and Wu 2020b)	76.2 → **73.4**	- 2.8	1.1 (73 %)	**6.7M (0.74)**
	Pruning	PFS (Wang et al. 2020)	76.1 → 72.8	- 3.3	1.0 (76 %)	4.6M (0.82)
	Pruning	WSP (Guo et al. 2021)	76.2 → 72.1	- 4.1	1.1 (73 %)	9.1M (0.64)
	Pruning	HRANK (Lin et al. 2020)	76.1 → 69.1	- 7.0	1.0 (76 %)	8.3M (0.68)

Table 3.3. The performance for ResNet50 on ImageNet is summarized for *BSR* and structured pruning. For the structured-pruning, we have aggregated the literature records that are available. To be consistent with Table 3.2, we have grouped the records according to the range of FLOPs reduction ratio. It is possible to group the records according to the parameter compression ratio, and the corresponding comparison plot is shown in Figure 3.8a.

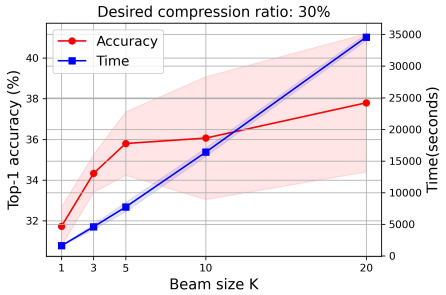
Table 3.2 summarizes the results for ResNet56 on CIFAR-10 and the graphical comparisons can be found in Figure 3.7. As can be seen in Table 3.2, the FLOPs information is provided in all the literature records but the number of parameter information is often missing. Therefore, we grouped the results according to the range of FLOPs reduction. For the cases with known compression ratio, the results are compared in Figure 3.7a where *BSR* outperforms all of the structured pruning algorithms. The FLOPs results are compared in Figure 3.7b, and *BSR* outperforms all of the structured pruning algorithms except for LeGR (Chin et al. 2020). The performance gap, however, is marginal. We compared the distributions of $\mathbf{r}_{select}$ as rank selection methods. Table 3.3 summarizes the results for ResNet50 on ImageNet, and the graphical comparisons can be found in Figure 3.8. For ResNet50 on ImageNet, both FLOPs information and number of parameter information is provided in most of the literature records. Nonetheless, we have organized the Table 3.3 with respect to the range of FLOPs reduction ratio to be consistent with Table 3.2. The parameter comparison results are shown in Figure 3.8a and it can be observed that *BSR* outperforms all but PFS (Wang et al. 2020). FLOPs comparison results are shown in Figure 3.8b where *BSR* can outperform PFS for small FLOPs values, but not for large FLOPs values. For 1.1 GFLOPs, CURL (Luo and Wu 2020b) achieves the same test accuracy as *BSR*. But *BSR* requires a smaller number of parameters (6.7M vs. 5.0M) as can be confirmed in Table 3.3.

3.6.3. Analysis of *BSR*

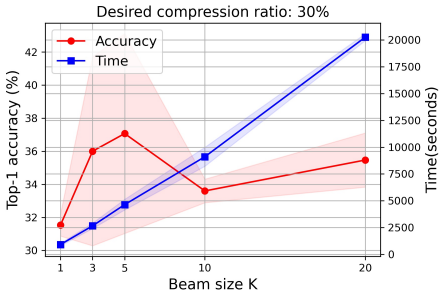
In this section, we analyze four aspects of *BSR*. Firstly, the sensitivity of K and s of the *mBS* algorithm is analyzed. Secondly, we look into the rank allocation of *mBS* and its characteristics. Thirdly, we analyze the training strategies of *mSR* including rank allocation updates during training and scheduled strengthening of the regularization. Finally, an ablation study of *BSR* is conducted to confirm the effectiveness of *mBS* and *mSR*.

***mBS*: sensitivity study**

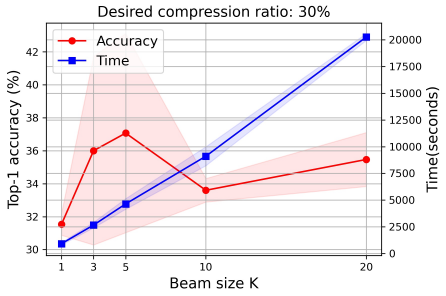
The effect of beam size K: We explore the effect of the level step size K for a fixed beam size s. Here, we fixed C_d to 0.3 and γ to 0.5. The results are shown in Figure 3.9, and we can observe that a larger K provides a better accuracy performance in general. The accuracy curve, however, exhibits a large variance and the average performance is even deteriorated when K is increased from one to three. This can be attributed to the nature of the rank selection problem. Because it is a non-convex problem, it is difficult to say what to can be expected with a certainty. Compared to the accuracy curve, the search time curve shows a monotonic behavior where search time increases as K is increased. Based on the results in Figure 3.9, we have chosen K to be five.



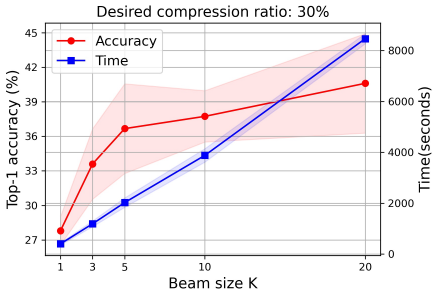
(a) $s = 3$



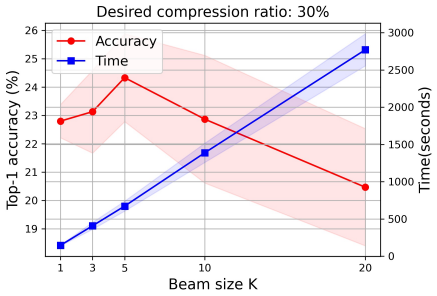
(b) $s = 5$



(c) $s = 5$



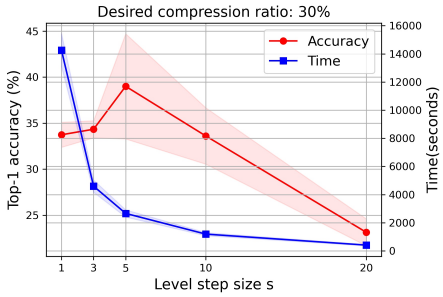
(d) $s = 10$



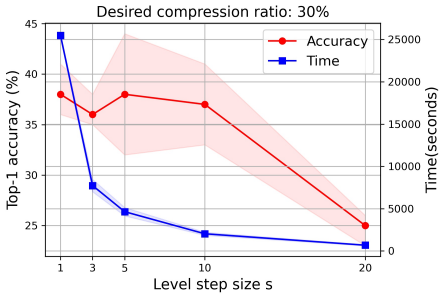
(e) $s = 20$

Figure 3.9. The effect of K parameters on *mBS* performance for a fixed s: a base neural network (ResNet56 on CIFAR-100) was truncated by the selected ranks and no further fine-tuning was applied for this analysis. Performance and search speed change as a function of K: (a) s is fixed at 3, (b) s is fixed at 5, (c) s is fixed at 10, (d) s is fixed at 20.

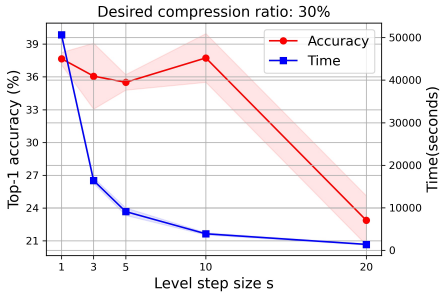
The effect of level step size s: We explore the effect of the level step size s for a fixed beam size K. Here, we fixed C_d to 0.3 and γ to 0.5. The analysis was repeated for a range of K, and the results are shown in Figure 3.10. We can observe that a smaller s provides a better accuracy performance. On the contrary, a smaller s makes the search time exponentially larger, especially for a very small s. Because of the trade-off, we used s between three and ten with an adaptive reduction of s when no solution was found.



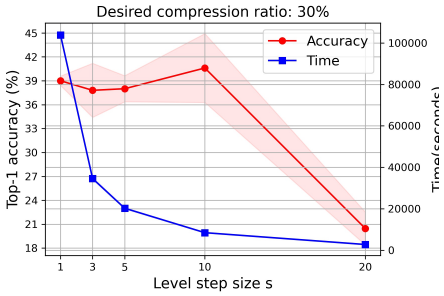
(a) $k = 3$



(b) $k = 5$

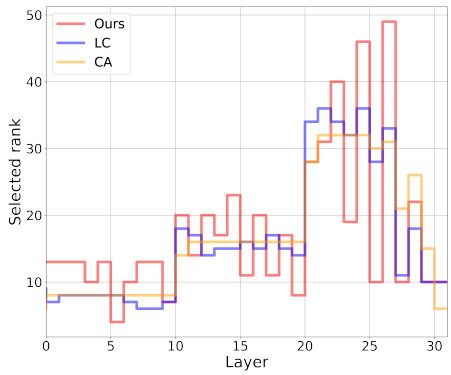


(c) $k = 10$

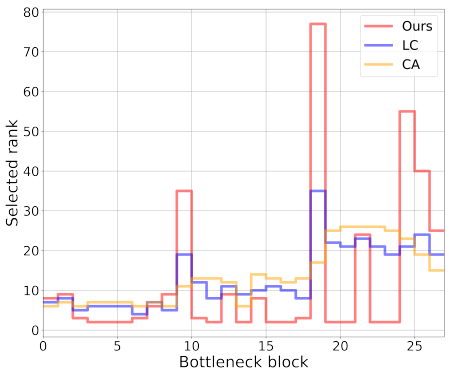


(d) $k = 20$

Figure 3.10. The effect of s parameters on *mBS* performance for a fixed K: a base neural network (ResNet56 on CIFAR-100) was truncated by the selected ranks and no further fine-tuning was applied for this analysis. Performance and search speed change as a function of s: (a) K is fixed at 3, (b) K is fixed at 5, (c) K is fixed at 10, (d) K is fixed at 20.



(a) ResNet32 on CIFAR-10

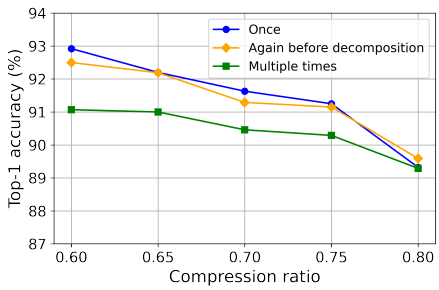


(b) ResNet56 on CIFAR-10

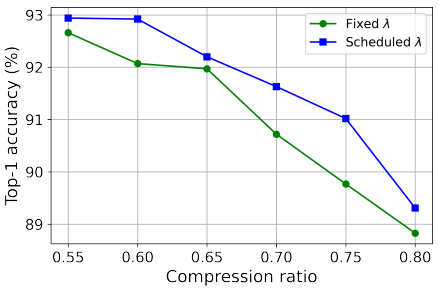
Figure 3.11. Selected rank distribution comparison with the baseline algorithms of CA and LC when compression ratio is 0.8.

mBS: characteristics of rank allocations

The results of rank allocation by LC, CA, and *BSR* are shown in Figure 3.11. It can be noted that *mBS* tends to allocate ranks in a less uniform way, as shown in Figure 3.11a. For a larger network with an extra redundancy, the rank allocation of *mBS* is even less uniform, as shown in Figure 3.11b. In fact, *mBS* chooses crucial blocks and allocates much larger resources to the chosen blocks. Compared to LC and CA, *mBS*' rank allocation tends to be closer to a layer selection algorithm. While it is difficult to judge if such a behavior is desired, the compression performance results seem to indicate that the behavior is indeed helpful.



(a) Rank selection updates



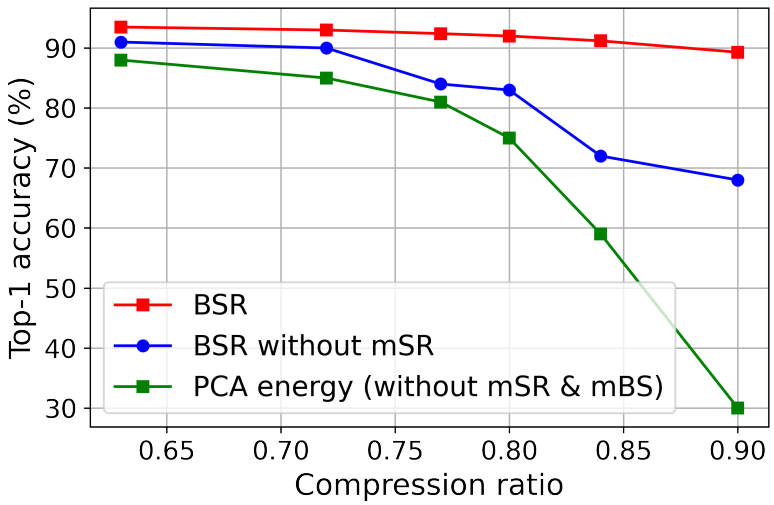
(b) Scheduled increase of λ

Figure 3.12. Performance of *BSR* for ResNet32 on CIFAR-10: (a) For rank selection, the best performance is achieved when the rank vector is set once and not updated. For multi-time, we have updated the target rank vector every 30 epochs. (b) For the scheduling of the strength of λ, the performance is improved with a scheduled increase in strength.

***mSR*: effects of training strategies**

Update number of $\mathbf{r}_{select}$ during training: In the previous works of CA and LC, the rank vector $\mathbf{r}$ is a moving target in the sense that CA performs truncated SVD multiple times during the compression-friendly training and LC updates the target weight matrices multiple times during the compression-friendly training. To investigate if *BSR* can benefit by updating $\mathbf{r}_{select}$, we have compared three different scenarios - $\mathbf{r}_{select}$ is calculated only once in the beginning ("once"), $\mathbf{r}_{select}$ is additionally updated once just before the decomposition and final fine-tuning (i.e., just before the phase three in Figure 4.3; "again before decomposition"), and $\mathbf{r}_{select}$ is updated at every 30 epochs ("multiple times"). The results are shown in Figure 3.12a. Interestingly, *BSR* performs best when $\mathbf{r}_{select}$ is determined only once in the beginning and never changed until the completion of the compression. This is closely related to the characteristics of *mSR*. As already mentioned, regularization through the modified stable rank does not rely on any particular instance of weight matrices. In fact, it only needs to know the target rank vector to be effective. Therefore, there is no need for any update during the training, and *BSR* can smoothly fine-tune the neural network to have the desired $\mathbf{r}_{select}$. Clearly, the regularization method of *mSR* has a positive effect on simplifying how $\mathbf{r}_{select}$ should be used.

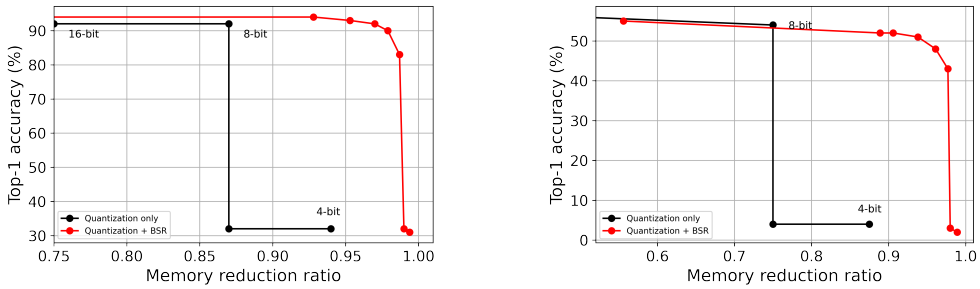
Scheduled strengthening of λ: The loss term during compression-friendly training is given by $\mathcal{L}(\mathbf{W}) + \lambda \sum_{l=1}^{L} \mathit{mSR}(\mathbf{W}_l, r_l)$. As the training continues, we can expect the weight matrices to be increasingly compliant with $\mathbf{r}_{select}$, thanks to the accumulated effect of *mSR* regularization. Then, a weak regularization might not be sufficient as the training continues. Comparison between fixed λ and scheduled λ (according to the explanation in Section 3.6.1) is shown in Figure 3.12b. As expected, the scheduled strengthening of λ is helpful for improving the compression performance.



(a) *BSR* ablation study for ResNet56 on CIFAR-10.

Ablation study of *BSR*

We performed ablation experiments of *BSR*. When the compression ratio is low, as shown in Figure 3.13a, there was no significant difference in performance among *BSR*, *BSR* without *mSR* and PCA energy (without *mSR* & *mBS*). However, the performance gap grows significantly as the compression ratio increases. In the case of PCA energy, performance degrades rapidly when the compression ratio is 70% or more, and it seems that performance recovery in the fine-tuning stage is nearly impossible. In the case of *BSR* without *mSR*, the performance drop was less severe than in the case of PCA energy; nonetheless, at a high compression ratio, it still demonstrated about 22% performance drop compared to *BSR*. This is because the allocated ranks become very small (When compressing more than 80%, rank 1 is selected over several layers); therefore, *BSR* is required in order to obtain a high-performance model with a high compression rate.



(a) ResNet56 on CIFAR10

(b) AlexNet on ImageNet

Figure 3.14. Comparison of memory usage for (a) ResNet56 on CIFAR10, and (b) AlexNet on ImageNet. A significant improvement is achieved by using both quantization and *BSR*.

3.7. Discussion

3.7.1. Combined use with quantization

As known well, low-rank compression can be used together with quantization. The performance for using both together is shown in Figure 3.14. Because the number of parameters is not affected by quantization, we are showing the actual memory usage as the metric of compression. Compared to using quantization only, the hybrid use of quantization and *BSR* provide a significant improvement.

We were also able to reach a range of memory usage that could not be attained by quantization alone by using *BSR* together. Even with more realistic datasets and larger networks, using *BSR* and quantization together is effective in reducing memory usage. This feature can make it much easier to use deep learning models on edge devices. Quantization and our low-rank compression are extremely easy to use and can be applied to practically any situation.

3.7.2. Limitations and future works

It might be possible to improve *BSR* in a few different ways. While our modified stable rank works well, it might be possible to identify a rank surrogate with better learning dynamics. While we choose only one $\mathbf{r}_{selected}$ and perform only a single compression-friendly training, it might be helpful to choose multiple rank vectors, train all, and choose the best. This can be an obvious way of improving performance at the cost of extra computation. Pruning and low-rank compression cannot be used at the same time, but it might be helpful to apply them sequentially. In general, combining multiple compression techniques to generate synergy remains as a future work.

3.8. Conclusion

We have introduced a new low-rank compression method called *BSR*. Its main improvements compared to the previous works are in the rank selection algorithm and the rank regularized training. The design of the modified beam-search is based on the idea that beam-search is a superior way of balancing search performance and search speed. Modifications such as the introduction of level step size s and the compression rate constraint play important roles. The design of a modified stable rank is based on a careful analysis of how the weight matrix's rank is regularized. To our best knowledge, our modified stable rank is the first regularization method that truly controls the rank. As a result, *BSR* performs very well.

Chapter 4. Learning to Select a Structured Architecture over Filter Pruning and Low-rank Decomposition

4.1. Introduction

Deep neural networks (DNNs) have achieved state-of-the-art performance in various fields, such as image classification, object detection, and video understanding. However, millions of parameters and high computational costs make their deployment on the edge and mobile devices challenging. To overcome this problem, DNN compression has been extensively studied in recent years. Among the various compression techniques, filter pruning and low-rank decomposition are two representative directions, both of which do not require hardware modification. They aim to reduce a heavy network to a lightweight form with two different structural assumptions.

Filter pruning removes less useful filters from an original network based on the structural assumption that some DNN filters are redundant. On the other hand, low-rank decomposition reduces the number of weight parameters of the original network by replacing each layer with two low-rank layers based on the assumption that some dimensions of the weights are less influential. Considering the distinct structural assumptions of the two compression approaches, investigating the efficient integration of filter pruning and low-rank decomposition towards a more effective model compression would be important and meaningful. Figure 4.1 presents a brief comparison of the existing works.

Two distinct structural assumptions result in two different architectural design spaces that are coarse-grained and fine-grained. Filter pruning compresses a model at the granularity of a full set of filters that correspond to an output channel (i.e., coarse granularity). This can efficiently remove uninformative filters at a low compression rate but can discard a large amount of useful information in the chosen set of filters as the

compression rate increases. On the other hand, low-rank decomposition compresses a model at the granularity of a weight matrix's singular-vector dimension (i.e., fine-granularity). Compared to the structural assumption of redundant filters, the structural assumption of less influential singular-vector dimensions can be a weaker assumption for a practical DNN. However, low-rank decomposition allows a fine-granularity removal and can avoid important information being eliminated all at once. It is obvious that properly utilizing the two compression approaches with different levels of granularity has the potential to provide a superior compression solution.

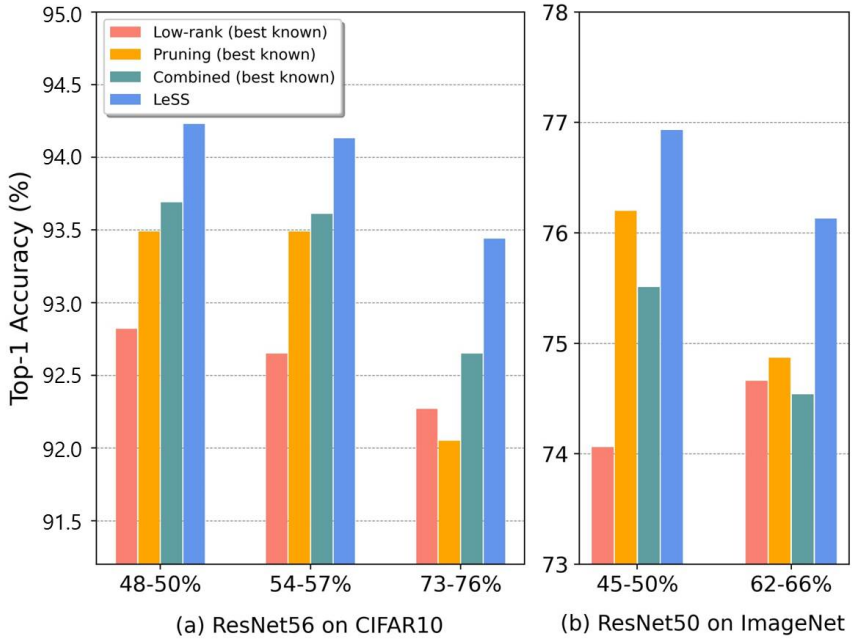


Figure 4.1. Performance comparison for each compression method according to the FLOP reduction rate category for (a) ResNet56 on CIFAR10 and (b) ResNet50 on ImageNet. The SOTA performance of each compression method is selected with respect to the FLOP reduction rate category. (a) (Idelbayev and Carreira-Perpinán 2020; Zhuang et al. 2018; Li et al. 2020) are selected for the 48-50% category, (Idelbayev and Carreira-Perpinán 2020; Yu, Mazaheri, and Jannesari 2022; Ruan et al. 2020) for the 54-57% category, and (Idelbayev and Carreira-Perpinán 2020; Sui et al. 2021; Li et al. 2020) for the 73-76% category. (b)(Xu et al. 2020; Sui et al. 2021; Ruan et al. 2020) are selected for the 45-50% category, (Phan et al. 2020; Shang et al. 2022; Li et al. 2020) for the 62-66% category.

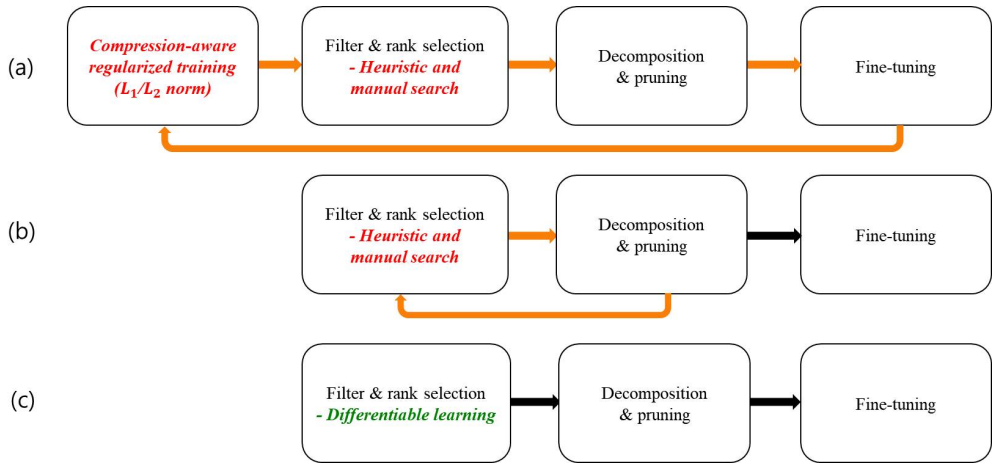


Figure 4.2. Comparison of three representative hybrid compression processes. Orange arrows indicate the iterative process that is computationally burdensome. (a) Based on compression-aware regularized training and heuristic filter/rank selection. To satisfy the target resource budget, the full process needs to be conducted iteratively. (b) Based on the iterative process for heuristic filter/rank selection. (c) **LeSS**: No iteration. Differentiable learning that is efficient and effective. Fully joint selection of filters and ranks. Satisfies target resource budget.

Recently, several works have been performed to integrate two compression techniques. First of all, compressing the weights by sequentially employing pruning and low-rank decomposition has been proposed (Dubey, Chatterjee, and Ahuja 2018; Chen et al. 2019b; 2020). However, sequentially employing algorithms did not exhibit a synergistic effect on the compression performance because the two compression techniques are not completely independent, as demonstrated in (Yu et al. 2017). To leverage the benefits of two different compression techniques, compression-aware regularized training methods (Ruan et al. 2020; Guo et al. 2019; Li et al. 2020) making a model compression-friendly were proposed to simultaneously handle the sparsity and low-rankness of the weight parameters (Figure 4.2a). However, controlling the compression rate with these methods is difficult because it requires numerous trials and errors to satisfy the target compression rate. In addition, the final rank and filter selection process for acquiring a compressed network is conducted by heuristic strategies such as a greedy approach or energy thresholding. Most recently, in (Li et al. 2022), a collaborative compression scheme is introduced to iteratively remove filters and determine ranks layer-by-layer. The method requires a multi-step heuristic removal process to satisfy the target compression rate (Figure 4.2b).

Although hybrid compression methods have been developed, as shown in Figure 4.1, performance improvements of the hybrid methods have not been significant compared to previous state-of-the-art (SOTA) methods with a single compression method (either filter pruning or low-rank decomposition). In addition, the performance of the compressed model with a single compression method is sometimes higher than that of hybrid compression methods (see Figure 4.1b). The existing hybrid compression methods are not guaranteed to converge to a desirable solution because the final decision (i.e., rank and filter selection) for acquiring a compressed network is dependent on heuristics.

In this work, we propose a simple learning algorithm for a joint filter and rank selection within desired resource budget constraint (Figure 4.2c), and it is called as **Le**arning to **S**elect a **S**tructured architecture (**LeSS**). Indeed, filter and rank selection

is a well-known combinatorial optimization problem (COP), which is NP-hard and requires an exhaustive search in a solution space. In order to provide polynomial-time solutions, various heuristic approaches (e.g., greedy approach and energy thresholding) have been investigated. To avoid heuristics and solve the COP very efficiently, **LeSS** is designed to be learned in a differentiable way by reformulating COP as a continuous nonlinear problem. **LeSS** consists of two modules: mask learning inspired by (Wang et al. 2020) for filter selection and novel threshold learning for rank selection. To be more specific, masks are learned so that the masks of less informative filters can be set to zero and thresholds are learned so that less important singular values of each layer can be set to zero. As shown in Figure 4.2c, **LeSS** does not require compression-aware regularized training, which is a burdening process, applies efficient differentiable learning for selecting filters and ranks, performs a fully joint selection over filters and ranks, and satisfies target resource budget without an iterative process.

As for the experiments, we demonstrate the effectiveness of **LeSS** on three datasets (CIFAR10, CIFAR100, ImageNet-1k) over five popular benchmark networks (ResNet18, ResNet50, ResNet56, VGG16, MobileNetV2). In all the experiments, **LeSS** consistently outperforms the prior SOTA of pruning, low-rank decomposition, and hybrid algorithms by a large margin. **LeSS** can even compress the lightweight network of MobileNetV2 without any loss in accuracy.

4.2. Contribution

The main contributions of our work can be summarized as follows:

One of the key highlights of our research is the introduction of a groundbreaking algorithm within the field of low-rank compression. This algorithm marks the first-ever approach that enables rank selection based on SGD (Stochastic Gradient Descent). Building upon this milestone, we seamlessly integrated this SGD-based rank selection algorithm with filter pruning, creating a powerful hybrid compression framework.

Through the implementation of our newly proposed algorithm, we achieved remark-

able performance that rivals state-of-the-art (SOTA) methods using low-rank compression alone. Additionally, when our algorithm was combined with pruning, it surpassed existing SOTA methods by a significant margin across various networks. Notably, our approach demonstrated exceptional superiority even on lightweight contemporary models like MobileNetV2. Furthermore, the algorithm's outstanding performance extended across diverse datasets, showcasing its versatility and applicability.

By introducing the first algorithm to enable rank selection based on SGD and successfully incorporating it into a hybrid compression framework, our work pushes the boundaries of compression techniques for deep neural networks. Our research paves the way for enhanced model efficiency, reduced resource demands, and improved performance in practical applications.

4.3. Related works

4.3.1. Hybrid compression methods

Previous research (Dubey, Chatterjee, and Ahuja 2018; Chen et al. 2020) has proposed separate compression stages to integrate multiple compression techniques. These stages sequentially adopt one compression technique in each step and ignore the interrelations of the different compression methods. For instance, in (Dubey, Chatterjee, and Ahuja 2018), filter pruning is conducted first to reduce the weights, and the weights are then decomposed using a corset-based decomposition technique. In addition, several compression-aware training approaches are proposed using regularization to make a network compression-friendly (Chen et al. 2019b; Ruan et al. 2020; Guo et al. 2019; Li et al. 2020). For example, in (Li et al. 2020), they first introduce a sparsity-inducing matrix at each weight and then impose group sparsity constraints during training. However, determining a good balance between compression rate and accuracy is challenging under the desired compression rate with these compression-aware methods. Recently, (Li et al. 2022) proposes a collaborative compression method to employ the global

compression rate optimization method to obtain the compression rate of each layer and adopt a multi-step heuristic removal strategy. Our hybrid compression method does not need heuristics on selecting filters and ranks, and does not require compression-aware regularized training.

4.4. Background

4.4.1. Selection problem for DNN compression

The selection problem for DNN compression can be formulated as constrained optimization whose objective is to determine the optimal ν satisfying the resource budget such as FLOPs, Macs, and the number of parameters while preserving good performance:

$$\min_{\nu} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(f(x_i; h(\mathbf{W}, \nu)), y_i) \quad \text{s.t. } \mathcal{B}(\nu) \leq \mathcal{B}_d. \tag{4.1}$$

Here, y_i is the corresponding label of input x_i, $\mathcal{L}$ is a cross-entropy loss function, f is a pre-trained model with parameters $\mathbf{W}$, and ν are selection variables determining the structure of $\mathbf{W}$. For filter pruning, ν is the collection of binary channel masks associated with the output filter of each layer (i.e., $\mathbf{c} = \{c_1, \cdots, c_L \mid c_l \in \{0,1\}^p$, where p is the output filter number of the l-th layer$\}$); for low-rank decomposition, ν is the collection of ranks of each layer (i.e., $\mathbf{r} = \{r_1, \cdots, r_L \mid r_l \in \mathbb{N}$, where L is the number of layers$\}$); For hybrid compression, ν is a collection of the rank and the binary mask of each layer (i.e., $\{(r_l, c_l)\}_{l=1}^{L}$). h is a function that returns parameters whose structure is determined by ν. Generally, h is non-differentiable because ν is a discrete set. $\mathcal{B}(\cdot)$ is a function that measures the resource budget. $\mathcal{B}_d$ is the desired resource budget.

4.4.2. Tensor Matricization

In our work, *matricization* is used to transform the tensor of convolutional kernels into a matrix to conduct singular value decomposition (SVD) operation. *Matricization* is the process of reshaping the elements of an D-dimensional tensor $\mathbf{X} \in \mathbb{R}^{I_1 \times \cdots \times I_D}$ into

a matrix (Kolda and Bader 2009; Kolda 2006). Let the ordered sets $\mathcal{R} = \{r_1, ..., r_L\}$ and $\mathcal{C} = \{c_1, ..., c_M\}$ be a partitioning of the modes $\mathcal{D} = \{1, ..., D\}$. The matricization function ψ of an D-dimensional tensor $\mathbf{X} \in \mathbb{R}^{I_1 \times \cdots \times I_D}$ is defined as:

$$\begin{aligned} \psi : \mathbf{X} &\longmapsto \mathbf{X}_{(\mathcal{R} \times \mathcal{C}: I_D)} \in \mathbb{R}^{J \times K}, \\ \text{where } J = \prod_{n \in \mathcal{R}} I_n \quad &\text{and} \quad K = \prod_{n \in \mathcal{C}} I_n. \end{aligned} \tag{4.2}$$

For example, the weight tensor of a convolutional layer is represented as a 4-D tensor ($\mathbf{W} \in \mathbb{R}^{C_{out} \times C_{in} \times k \times k}$) where it is composed of kernels, and it can be unfolded into a matrix as six different forms. The two most common forms used in low-rank decomposition are as follows: ① $\mathbf{W} \in \mathbb{R}^{C_{out} \times (C_{in} kk)}$, ② $\mathbf{W} \in \mathbb{R}^{(C_{out} k) \times (C_{in} k)}$.

4.4.3. CNN decomposition scheme

To decompose a convolutional layer with $C_{in},\ C_{out}$ (input/output channels) and k (kernel size), one of the following low-rank structures is used depending on the matricization form.

Scheme 1 When we use the first reshaping form ① introduced in Section 4.4.2, the convolutional weights can be considered as a linear layer with the shape of $C_{out} \times C_{in} k^2$. Then, the rank-$r$ approximation presents two linear mappings with weight shapes $C_{out} \times r$ and $r \times C_{in} k^2$. These linear mappings can be deployed as a sequence of two convolutional layers: $\mathbf{W_1} \in \mathbb{R}^{r \times C_{in} \times k \times k}$, and $\mathbf{W_2} \in \mathbb{R}^{C_{out} \times r \times 1 \times 1}$ (Wen et al. 2017; Xu et al. 2020; Li and Shi 2018).

Scheme 2 When we use the second reshaping form ② introduced in Section 4.4.2, the convolutional weights can be considered as a linear layer of $C_{out} k \times C_{in} k$. For this scheme, an approximation of rank r has two linear mappings with weight shapes $C_{out} k \times r$ and $r \times C_{in} k$. These can be implemented as a sequence of two convolutional layers as follows: $\mathbf{W_1} \in \mathbb{R}^{r \times C_{in} \times k \times 1}$, and $\mathbf{W_2} \in \mathbb{R}^{C_{out} \times r \times 1 \times k}$ (Tai et al. 2015; Jaderberg, Vedaldi, and Zisserman 2014).

We use *Scheme 1* throughout all experiments in our study based on the comparison

results in Discussion 4.7.1.

4.5. Learning framework for the selection problem in hybrid compression

In this section, we propose the new learning framework, called **LeSS**, which determines informative filters and the optimal ranks in hybrid compression. The general idea of **LeSS** is to transform the problem of solving (4.1) over discrete variables $\mathbf{c}$ and $\mathbf{r}$ into minimizing a differentiable surrogate function h_{LeSS} over continuous variables $\mathbf{z_c}$ and $\mathbf{z_r}$. The discrete solution can be approximated using differentiable functions $g_{\mathbf{c}}$ and $g_{\mathbf{r}}$, and this allows us to use gradients that are not available in discrete problems. More specifically, we re-define the problem (4.1) as follows:

$$\begin{aligned}\min_{\mathbf{z_c},\mathbf{z_r}} \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(f(x_i; h_{\text{LeSS}}(\mathbf{W}, \mathbf{z_c}, \mathbf{z_r}))), y_i) \\ +\lambda \left\| \mathcal{B}(g_{\mathbf{c}}(\mathbf{z_c}), g_{\mathbf{r}}(\mathbf{z_r})) - \mathcal{B}_d \right\|^2\end{aligned} \tag{4.3}$$

where $g_{\mathbf{c}}(\mathbf{z_c})$ is the number of filters for each channel, $g_{\mathbf{r}}(\mathbf{z_r})$ is the rank for each layer, and λ is a hyper-parameter used to regularize the budget constraint. For each layer, **LeSS** consists of the surrogate function h_{LeSS} which is composed of two modules, s_m and s_t (i.e., $h_{\text{LeSS}} = s_t \circ s_m$).

Module s_m (mask learning for filter selection): To construct a function h_{LeSS}, we describe a module s_m used for filter selection. Let $\mathbf{z_c} = \{M_1, \cdots, M_L \mid M_l \in \mathbb{R}^{C^l_{out} \times (C^l_{in} kk)}\}$ be a set of diagonal matrices (i.e., mask matrix). To establish s_m, we first define a scheduled sigmoid function to generate the approximate binary masks as follows:

$$\sigma_s(x) = \frac{1}{1 + \exp(-1 * \mu_i * (x - 0.5))} \tag{4.4}$$

where $\mu_i = \min(\alpha, \mu_{i-1}+\beta)$.

Note that μ_i is the scheduling factor affecting the steepness of sigmoid in iteration i, and it is updated every iteration and does not exceed α. During the beginning phases of training, μ_i is kept at a very low value; it is then increased as the optimization process progresses. When μ_i grows large enough, the values of approximate binary masks will become almost 0 or 1. That is, it is completely determined which filter should be removed. For each weight $\mathbf{W_l}$ of the l-th layer, we define a function s_m by Eq. (4.2) and Eq. (4.4) as follows:

$$s_m(\mathbf{W_l}, M_l) = \psi^{-1}(\sigma_s(M_l) \cdot \psi(\mathbf{W_l})) \tag{4.5}$$

To approximate the number of filters corresponding to the continuous variable $\mathbf{z_c}$, we define the set-valued function $g_\mathbf{c}$ as follows:

$$g_\mathbf{c}(\mathbf{z_c}) = \{\mathbf{1}^T \cdot \text{diag}(\sigma_s(M_l))\}_{l=1}^{L} \tag{4.6}$$

Since all functions constituting Eq. (4.5) and Eq. (4.6) are differentiable, we can easily confirm that s_m and $g_\mathbf{c}$ are differentiable. As in the previous works on filter pruning (Huang and Wang 2018; Wang et al. 2020), we follow the mask strategy whose mask variables are learnable parameters. In contrast to the previous works, however, we adopt a *scheduled* sigmoid function because the scheduling technique has been proven to be useful for gradient-descent-based optimization in the general deep learning field (Kwon et al. 2020; Loshchilov and Hutter 2016; Zhai et al. 2022; Zhou et al. 2021).

Although the introduction of the scheduled factor is a simple technique, we find that this can result in a significant improvement in performance, as shown in Table 4.1.

Flop reduction rate	0.5	0.6	0.7
Sigmoid with μ_i	**94.13**	**93.20**	**92.76**
Sigmoid without μ_i	92.84	92.77	92.25

Table 4.1. Performance comparison of implementing $\mathcal{T}_{\text{DML-S}}$ with and without scheduled factor μ_i. Results are shown for ResNet56 on CIFAR10.

Module s_t (threshold learning for rank selection): To proceed, we explain the s_t module used for rank selection. Let $\mathbf{z_r} = \{\gamma_1, \cdots, \gamma_L \mid \gamma_l \in \mathbb{R}\}$ be a threshold set. To construct s_t, we introduce a singular value thresholding (SVT) function. For a matrix $M \in \mathbb{R}^{m \times n}$ and threshold γ, SVT is defined as follows:

$$\text{SVT}(M, \gamma) = U \cdot \text{ReLU}(\Sigma - \gamma) \cdot V^T \tag{4.7}$$

where U is an $m \times m$ real unitary matrix, Σ is an $m \times n$ rectangular diagonal matrix with non-negative real numbers on the diagonal, V is an $n \times n$ real unitary matrix, and $\text{ReLU}(\cdot)$ is a rectified linear unit activation function. For each weight W_l of the l-th layer, we define a function s_t by (4.2) and (4.7) as follows:

$$s_t(\mathbf{W_l}, \gamma_l) = \psi^{-1}(\text{SVT}(\psi(\mathbf{W_l}), \gamma_l)) \tag{4.8}$$

To approximate the rank corresponding to $\mathbf{z_r}$, we also define the set-valued function $g_\mathbf{r}$ as:

$$g_\mathbf{r}(\mathbf{z_r}) = \{\mathbf{1}^T \cdot (\tanh(\text{ReLU}(\Sigma_l - \gamma_l) \cdot \tau))\}_{l=1}^{L} \tag{4.9}$$

where Σ_l is a diagonal matrix whose diagonal entries are singular values of the l-th layer weight matrix W_l and τ is a scaling hyper-parameter used to control the steepness of $\tanh$. Similar to s_m and $g_\mathbf{c}$, we can easily confirm that s_t and $g_\mathbf{r}$ are differentiable.

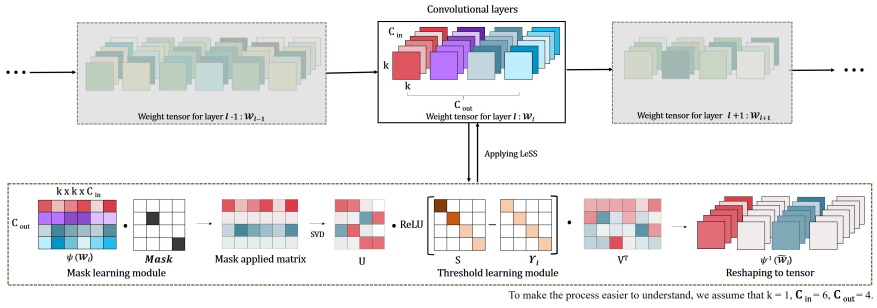


Figure 4.3. Illustration of **LeSS** algorithm's forward process.

Algorithm 2 Learning to Select Structured Architecture

Input: desired resource budget $\mathcal{B}_d$; model parameters $\mathbf{W} = \{\mathbf{W}_l\}_{l=1}^{L}$.

Output: thresholding variables $\mathbf{z_r} = \{\gamma_l\}_{l=1}^{L}$; mask variables $\mathbf{z_c} = \{M_l\}_{l=1}^{L}$

Require: function for measuring resource budget $\mathcal{B}(\cdot, \cdot)$; epoch E; iteration T

Initialize $\mathbf{z_r} = \mathbf{0}$, $\mathbf{z_c} = \mathbf{1}$

Calculate $\mathcal{B}_{\text{cal}} = \mathcal{B}(g_{\mathbf{c}}(\mathbf{z_c})g_{\mathbf{r}}(\mathbf{z_r}))$

while $\|\mathcal{B}_{\text{cal}} - \mathcal{B}_d\| \leq \epsilon$ **do**
  for ep $= 1:$ E **do**
    for iter $= 1:$ T **do**
      for $l = 1:$ L **do**
        $W_l^{\text{comp}} = h_{\text{LeSS}}(\mathbf{W_l}, M_l, \gamma_l)$ (see (4.11), (4.12))
      end for
      $\mathbf{W}^{\text{comp}} = \{W_l^{\text{comp}}\}_{l=1}^{L}$
      $\mathcal{B}_{\text{cal}} = \mathcal{B}(g_{\mathbf{c}}(\mathbf{z_c}), g_{\mathbf{r}}(\mathbf{z_r}))$
      Update $\mathbf{z_r}, \mathbf{z_c} \leftarrow$
        $\arg\min_{\mathbf{z_r}, \mathbf{z_c}} \mathcal{L}(\mathbf{W}^{\text{comp}}) + \lambda \|\mathcal{B}_{\text{cal}} - \mathcal{B}_d\|^2$
    end for
  end for
end while

Budget $\mathcal{B}(g_{\mathbf{c}}(\mathbf{z_c}), g_{\mathbf{r}}(\mathbf{z_r}))$: FLOP is used for the resource budget in this paper and the formula of budget calculation is as follows:

$$\sum_{l=1}^{L} \frac{A_l \cdot k_l \cdot k_l \cdot g_{\mathbf{r}}(l)(\mathbf{z_r}) \cdot (g_{\mathbf{c}}(\mathbf{z_c})(l-1) + g_{\mathbf{c}}(\mathbf{z_c})(l))}{A_l \cdot k_l \cdot k_l \cdot C_{in}^{l} \cdot C_{out}^{l}} \tag{4.10}$$

A_l denotes the area of the l-th layer's feature maps, and k_l is the kernel size of the l-th layer. C_{in}^{l} and C_{out}^{l} denote l-th layer's input- and output-channels of the original model, respectively. $g_{\mathbf{r}}(\mathbf{z_r})(l)$ and $g_{\mathbf{c}}(\mathbf{z_c})(l)$ are the l-th layer's selected rank and number of selected filters, respectively. Because all elements in Eq. (2) are differentiable, the budget function $\mathcal{B}(g_{\mathbf{c}}(\mathbf{z_c}), g_{\mathbf{r}}(\mathbf{z_r}))$ is differentiable. We use FLOP resource budget, but other resource budget (e.g., number of parameters) also can be used.

Here, we define the surrogate function, h_{LeSS} for two types of layers (convolutonal layers and linear layers). If the l-th layer is a convolutional layer, the surrogate function h_{LeSS} is defined as:

$$h_{\text{LeSS}}(\mathbf{W}_l, M_l, \gamma_l) = s_t(s_m(\mathbf{W}_l, M_l), \gamma_l) \tag{4.11}$$

When the l-th layer is a linear layer, h_{LeSS} is defined as below:

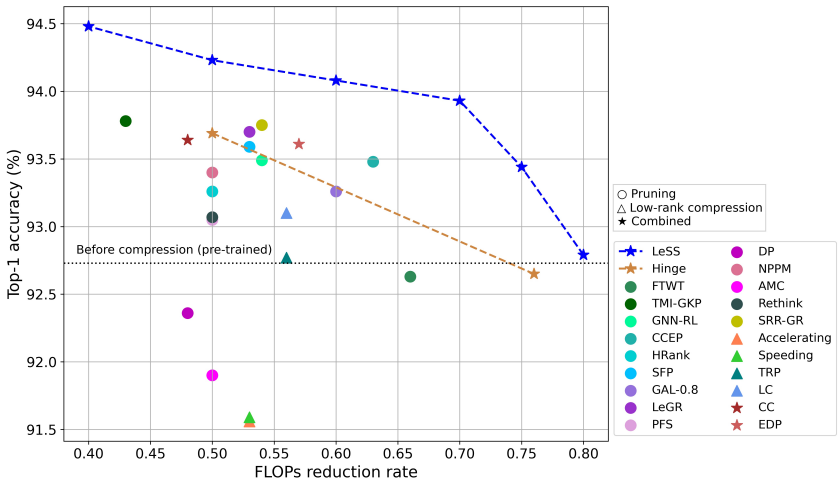
$$h_{\text{LeSS}}(\mathbf{W}_l, M_l, \gamma_l) = s_t(\mathbf{W}_l, \gamma_l) \tag{4.12}$$

Note that M_l is not used for linear layers, as structured pruning (filter pruning) cannot be applied to linear layers. Algorithm 2 provides the training procedure and Figure 4.3 shows the forward process of **LeSS**. The two **LeSS** modules are learned simultaneously in the training process. Therefore, **LeSS** can efficiently use two different compression techniques by considering the impact of the reduction in the number of filters and rank on the model's performance.

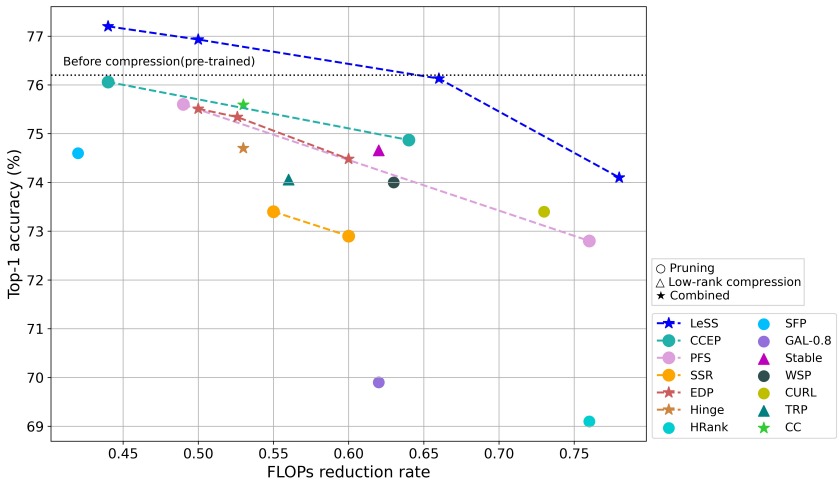
After training is completed, we require an exact binary mask and rank for each layer to directly compress the model. To acquire these, we round up the binary mask in $\mathbf{z_c}$ and the rank in $g_{\mathbf{r}}(\mathbf{z_r})$.

In the l-th layer weight, we prune the filter whose exact binary mask is zero. Subsequently, we perform low-rank decomposition with the exact rank on the pruned weight.

Finally, as in previous studies (Alvarez and Salzmann 2017; Alwani, Wang, and Madhavan 2022; Cai et al. 2021a; Chin et al. 2020; Idelbayev and Carreira-Perpinán 2020), we fine-tune the compressed model to further improve performance.



(a) ResNet56 on CIFAR10



(b) ResNet50 on ImageNet

Figure 4.4. Comparison of our method with SOTA pruning, low-rank decomposition, and hybrid compression methods for (a) ResNet56 on CIFAR10 and (b) ResNet50 on ImageNet.

4.6. Experiments

4.6.1. Experimental settings

To validate the effectiveness and generalizability of **LeSS**, we evaluate its compression performance on five benchmarks: VGG16 on CIFAR10, ResNet56 on CIFAR10 and CIFAR100, and ResNet50 and MobileNetV2 on ImageNet-1k. These are the most commonly examined benchmarks in the compression studies. In particular, MobileNetV2 has recently become one of the most important benchmarks in terms of practicality. We follow the original VGG and ResNet training settings to prepare a baseline network. For ResNet50 and MobileNetV2, we use the official PyTorch pre-trained models. The standard data augmentation techniques are used for all the datasets. Full training dataset is used to learn masks and thresholds. We fix the hyper-parameter $\lambda = 1$ in Eq. (4.3), $\alpha = 50$ for μ_i in Eq. (4.4), and use $2/\sigma_1$ for τ in Eq. (4.9). After the compression process (pruning and low-rank decomposition) is complete, the compressed network is fine-tuned for 100 epochs as the previous methods did. Fine-tuning is conducted similarly to the original training, but the initial learning rate is reduced by one-tenth. CIFAR-10 is a 10-class classification dataset that includes 50k images for training and 10k images for validation. CIFAR-100 is a classification dataset that has 100 different classes and contains 50k images for training and 10k images for validation. ImageNet is a huge image classification dataset that contains 1.2 million training images and 50k validation images with 1,000 different categories. For all the comparisons that we are able to find in the literature, the full comparison tables can be found in Supplementary A. We provide graphical summaries for the two cases (ResNet56 on CIFAR10 and ResNet50 on ImageNet) where sufficiently large number of comparisons exist, and provide table summaries where less comparison points are available.

ResNet56 on CIFAR10 Figure 4.4a shows the comparison results for ResNet56 on CIFAR10. **LeSS** outperforms the previous methods by a large margin across all

FLOP reduction rates. In particular, the 50% FLOP reduction rate is investigated by a bunch of previous methods, and **LeSS** achieves the best performance under this constraint. Note that the compressed model produced by **LeSS** consistently exhibits higher performance than that of the original (baseline) model across all FLOP reduction rates. **LeSS** reduces the FLOPs by 40% compared with the baseline model yet improves accuracy by 1.6%. This demonstrates that our compression method correctly eliminates redundant dimensions and filters, resulting in a generalizable compressed model. Full comparison results are summarized in Appendix D.

Dataset	Model	Compression method	Algorithm	Baseline (%)	Test acc.(%)	Δ Test acc.(%)	MFLOPs (Reduction ratio)	Params (Compression ratio)
CIFAR10	VGG16	Low-rank	LC (Idelbayev and Carreira-Perpinán 2020)	93.43	93.83	+ 0.40	132 (57.8 %)	3.16 M (78.7 %)
		Pruning	SSS (Huang and Wang 2018)	93.96	93.02	- 0.94	183 (41.6 %)	3.93 M (73.8 %)
			CP (He, Zhang, and Sun 2017)	93.99	93.67	- 0.32	156 (50.0 %)	N/A
			GAL-0.1 (Lin et al. 2019b)	93.96	93.42	- 0.54	141 (54.8 %)	12.25 M (17.8 %)
			DECORE (Alwani, Wang, and Madhavan 2022)	93.96	93.56	- 0.40	110 (64.8 %)	1.66 M (89.0 %)
		Hybrid	Hinge (Li et al. 2020)	94.02	93.59	- 0.43	122 (60.9 %)	11.92 M (20.0 %)
			LeSS	94.14	**93.87**	- 0.27	125 (60.0 %)	2.24 M (85.0 %)
			LeSS	94.14	**93.74**	- 0.40	109 (65.0 %)	1.79 M (88.0 %)
			LeSS	94.14	**93.64**	- 0.50	94 (70.0 %)	1.47 M (90.1 %)
CIFAR100	ResNet56	Low-rank	CA (Alvarez and Salzmann 2017)	72.39	64.79	- 7.60	75 (40.0 %)	N/A
			LC (Idelbayev and Carreira-Perpinán 2020)	72.39	69.82	- 2.57	59 (52.5 %)	N/A
		Pruning	ASFP (He et al. 2019a)	72.92	69.35	- 3.57	59 (52.6 %)	N/A
			ASRFP (Cai et al. 2021b)	72.92	69.16	- 3.32	59 (52.6 %)	N/A
			GHFP (Cai et al. 2021a)	72.92	69.62	- 3.30	59 (52.6 %)	N/A
			PGMPF (Cai et al. 2022)	72.92	70.21	- 2.71	59 (52.6 %)	N/A
		Hybrid	**LeSS**	72.39	**72.12**	- 0.27	63 (50.0 %)	0.41 M (56.4 %)
			LeSS	72.39	**71.05**	- 1.34	55 (55.0 %)	0.37 M (51.7 %)

Table 4.2. The performance comparison for VGG16 on CIFAR10 and for ResNet56 on CIFAR100.

VGG16 on CIFAR10 Table 4.2 presents the comparison results for VGG16 on CIFAR10 with the SOTA methods. The performance of our method is superior to that of all the most recent filter pruning, low-rank decomposition, and hybrid methods; in particular, our method shows no noticeable drop in performance even at higher FLOP reduction rates (maximum of 0.5 percentage point is dropped). This is because of the structural characteristics of the VGG16 model consisting of convolutional and fully-connected layers. Because filter pruning is unable to compress fully-connected layers, the amount of compression achievable is limited. On the other hand, our hybrid method is able to successfully compress fully-connected layers using low-rank decomposition. Clearly, the performance obtained when the model is compressed by 70% using our method is superior to the performance obtained when the model is compressed by 60% using Hinge (Li et al. 2020).

ResNet56 on CIFAR100 The result of comparing with the SOTA methods for ResNet56 on CIFAR100 can also be found in Table 4.2. **LeSS** is able to preserve the original model's performance at half the original model's FLOP rate. In addition, compared with other methods, **LeSS** exhibits the best performance even when the FLOPs is reduced by a large percentage (@ 55% reduction).

ResNet50 and ResNet18 on ImageNet The result of ResNet50 on ImageNet can be founded in Figure 4.4b. The graphical summary confirms that **LeSS** shows superior performance than that of the other SOTA methods in all FLOP reduction rates. For instance, when we compare the difference in the FLOP rate between our method and the CC algorithm (Li et al. 2021) at the same performance (75.59%), our method can accelerate the inference time by 14% more than the CC method (Li et al. 2021) (0.53 *vs.* 0.68). In addition, even when ResNet50 is compressed by 50% FLOP reduction, our method exhibits higher performance than the baseline performance. The result of ResNet18 on ImageNet is summarized in Table 4.3. Because no experimental results of hybrid algorithms for ResNet18 on ImageNet are available, the performances of

algorithms employing only a single compression method are compared. When compared with recent SOTA methods, **LeSS** outperforms them in all FLOP reduction rates, and even when a model is compressed up to 70%, the performance is higher than the baseline performance. That is, **LeSS** removes redundant dimensions and filters effectively.

MobileNetV2 on ImageNet Performance comparison result for ImageNet on light-weight MobileNetV2 is summarized in Table 4.3. MobileNetV2 is a well-known computationally efficient model, which makes it harder to compress. Nevertheless, our method surprisingly increases the model's top-1 accuracy up to 72% when the FLOP reduction rate is 35%. Furthermore, despite the fact that the inference time is accelerated more than twice that of the original model, the performance reduction is only 0.17 percentage point. From these results, it is convinced that our method can efficiently reduce the size of a network while keeping performance as high as possible, even if the model size is already small.

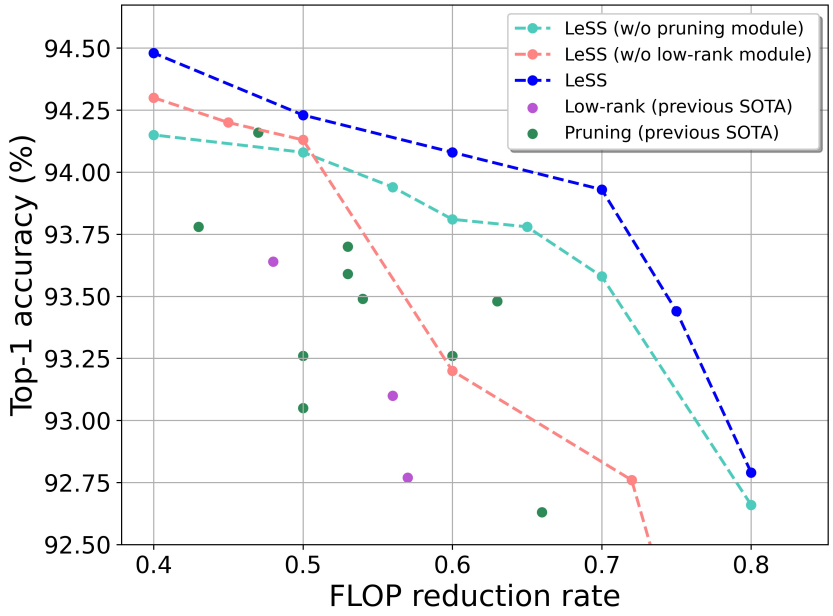
Dataset	Model	Compression method	Algorithm	Baseline (%)	Test acc.(%)	Δ Test acc.(%)	GFLOPs (Reduction ratio)	Params (Compression ratio)
ImageNet	ResNet18	Low-rank	Stable (Phan et al. 2020)	69.76	68.62	- 1.14	1.00 (45 %)	N/A
			TRP (Xu et al. 2020)	69.10	65.51	- 3.59	0.73 (60 %)	N/A
			ALDS (Liebenwein et al. 2021)	69.62	69.24	- 0.38	0.64 (65 %)	N/A
		Pruning	SFP (He et al. 2018a)	70.28	67.10	- 3.18	1.06 (42 %)	N/A
			FPGM (He et al. 2019b)	70.28	68.41	- 1.87	1.06 (42 %)	7.10 M (39 %)
			PFP (Liebenwein et al. 2019)	69.74	65.65	- 4.09	1.04 (43 %)	N/A
			DMCP (Guo et al. 2020)	N/A	69.00	N/A	1.04 (43 %)	N/A
			CHEX (Hou et al. 2022)	N/A	69.60	N/A	1.03 (43 %)	N/A
			SCOP (Tang et al. 2020)	69.76	68.62	- 1.14	1.00 (45 %)	N/A
			FBS (Gao et al. 2018)	69.76	68.17	- 1.59	0.91 (50 %)	N/A
			CGNET (Hua et al. 2019)	69.76	68.30	- 1.46	0.89 (51 %)	N/A
			GNN (Yu, Mazaheri, and Jannesari 2022)	69.76	68.66	-1.10	0.89 (51 %)	N/A
			ManiDP (Tang et al. 2021)	69.76	68.88	- 0.88	0.89 (51 %)	N/A
			PGMPF (Cai et al. 2022)	70.23	66.67	- 3.56	0.84 (54 %)	N/A
		Hybrid	**LeSS**	69.76	**71.24**	+ 1.48	0.91 (50 %)	4.68 M (60 %)
			LeSS	69.76	**70.82**	+ 1.06	0.70 (62 %)	3.51 M (70 %)
			LeSS	69.76	**70.15**	+ 0.39	0.55 (70 %)	2.81 M (76 %)
	MobileNetV2	Low-rank	LC (Idelbayev and Carreira-Perpinán 2020)	71.80	69.80	- 2.00	0.21 (30 %)	N/A
		Pruning	PFS (Wang et al. 2020)	71.80	70.90	- 0.90	0.21 (30 %)	2.60 M (26 %)
			AMC (He et al. 2018b)	71.80	70.80	- 1.00	0.22 (27 %)	2.30 M (34 %)
			MetaPruning (Liu et al. 2019)	72.00	71.20	- 0.80	0.22 (27 %)	N/A
			LeGR (Chin et al. 2020)	71.80	71.40	- 0.20	0.21 (30 %)	N/A
			NPPM (Gao et al. 2021)	72.02	72.04	+ 0.02	0.21 (30 %)	N/A
			GNN (Yu, Mazaheri, and Jannesari 2022)	71.87	70.04	- 1.83	0.17 (42 %)	N/A
		Hybrid	EDP (Ruan et al. 2020)	N/A	71.00	N/A	0.22 (27 %)	N/A
			LeSS	71.80	**72.16**	+ 0.20	0.19 (35 %)	2.24 M (36 %)
			LeSS	71.80	**71.63**	- 0.17	0.14 (55 %)	1.54 M (56 %)

Table 4.3. Performance comparison for ResNet18 and MobileNetV2 on ImageNet.

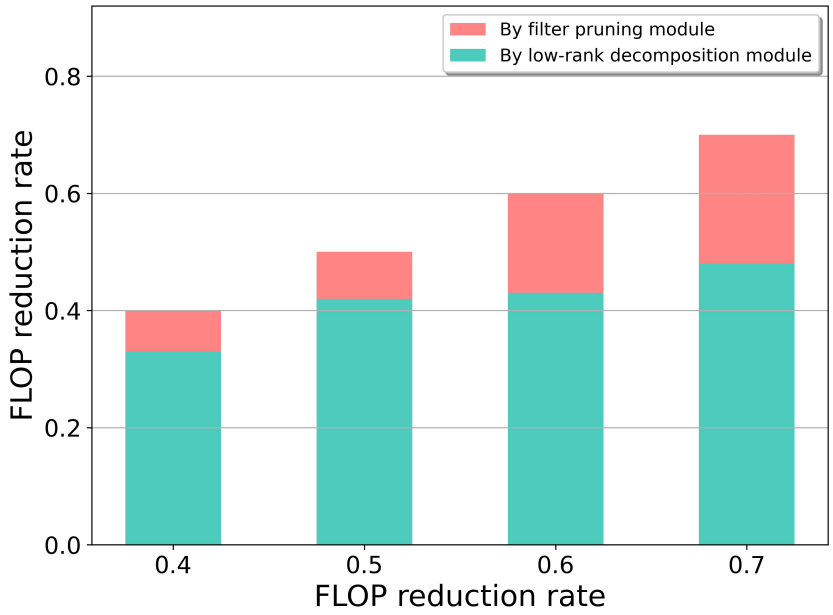
4.7. Analysis and discussion.

4.7.1. Learning strategy analysis

LeSS consists of two modules (mask learning, threshold learning) and they are jointly conducted for filter and rank selection. For analyzing each module's effectiveness in **LeSS**, we intentionally make each module disabled in turn. The experiment is conducted for ResNet56 on CIFAR10. As shown in Figure 4.5a, the individual modules work fairly well, better or comparable to most of the recent filter pruning and low-rank decomposition algorithms. In particular, filter pruning shows higher performances at relatively low FLOP reduction rates, whereas low-rank decomposition shows higher performance at relatively large FLOP reduction rates. This characteristic is also confirmed in Figure 4.1. The result indicates that the structural assumption that some of the filters are redundant is correct, but only up to a certain level. Once the redundant filters are removed, pruning starts to suffer with a sharp performance reduction because some of the required filters need to be removed as a whole. Note that pruning does not allow a partial reduction for each filter. On the other hand, low-rank decomposition does not suffer from this problem because it allows a finer grain control of dimension reduction. As expected, the hybrid algorithm (**LeSS**) always shows the best performance in all FLOP intervals. In addition, overall performance of our mask module for pruning is comparable to the previous results, while that of rank selection module is always higher than the previous results. This leads to the conclusion that the most important factors in achieving high performance of **LeSS** are that the differentiable rank selection module is properly designed for high reduction rates and that our joint selection method can effectively combine filter selection and rank selection. We investigate the ratio of each compression module in the hybrid method, and the results are shown in Figure 4.5b. The utilization of low-rank decomposition is substantial across all FLOP reduction rates, and the contribution of pruning to compression increases with the FLOP reduction rate. That is, while each strategy contributes to compression at differing degrees, higher

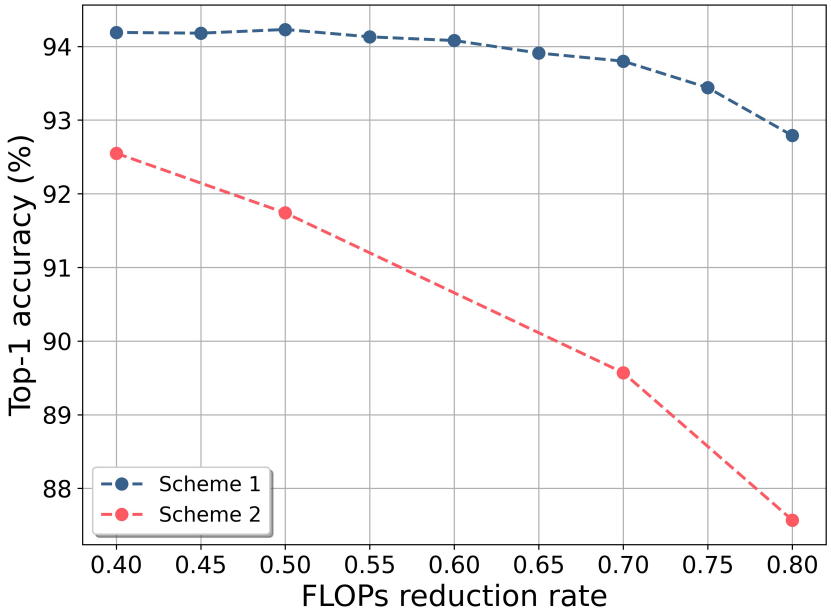


(a) Performance comparison of each method for ResNet56 on CIFAR10.

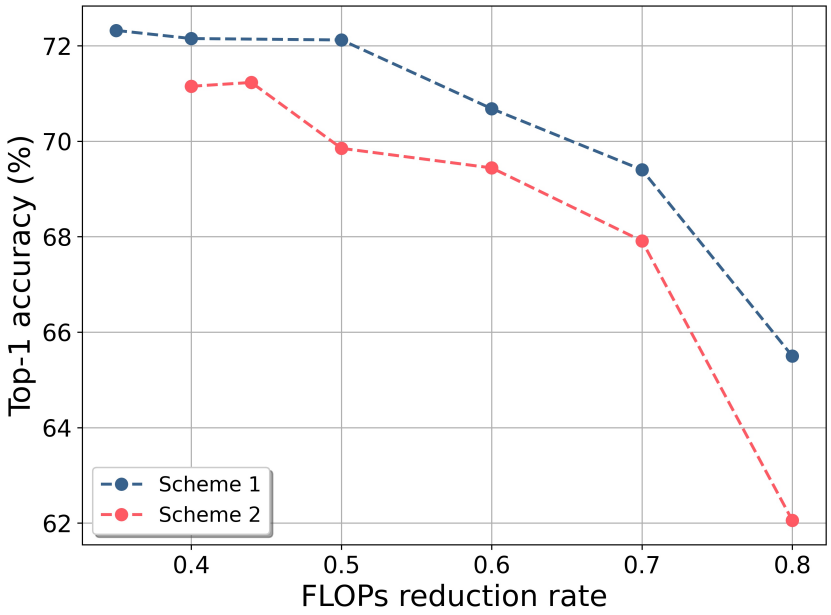


(b) The contribution of each method to FLOP reduction.

Figure 4.5. Analysis of learning techniques.



(a) ResNet56 on CIFAR10



(b) ResNet56 on CIFAR100

Figure 4.6. Comparison results depending on matricization scheme.

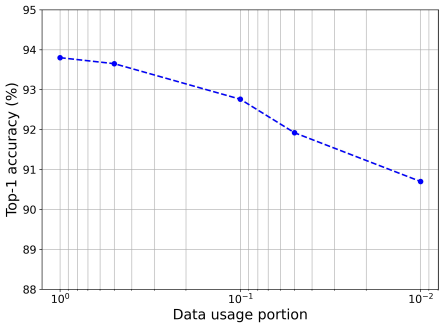
performances can be obtained by employing both modules rather than one.

4.7.2. Influence of matricization scheme

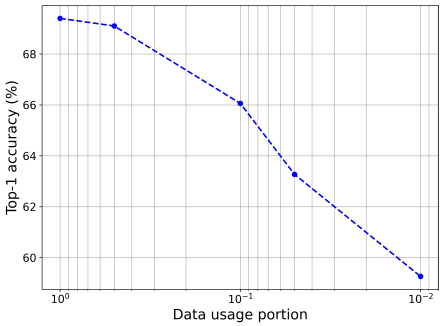
In many compression studies, particularly on low-rank decomposition requiring the reshaping of tensors into matrices, the two matricization schemes described in Section 4.4.3 are typically used. Most studies use *Scheme1* rather than *Scheme 2*. We conduct experiments to compare which scheme performs better for our method. The results are shown in Figure 4.6. *Scheme 1* demonstrates significantly better performances and the performance gap increases with the FLOP reduction rate. Note that typically *Scheme2* forms a matrix that is square or closer to a square matrix.

4.7.3. Data efficiency of LeSS

We evaluate the data efficiency of **LeSS** for ResNet56 on CIFAR10 and CIFAR100 datasets. The experiment is conducted based on the assumption that $\mathbf{z_c}$ and $\mathbf{z_r}$ in **LeSS** are trained to be biased toward the training data when numerous training data are used. However, contrary to our assumption, the performance of the model steadily drops as the used training data size reduces, and the performance drastically decreases below a training data usage of 0.5% (Figure 4.7). This confirms that learning an appropriate mask and threshold from the entire training data is beneficial rather than over-fitting to the dataset.



(a) ResNet56 on CIFAR10



(b) ResNet56 on CIFAR100

Figure 4.7. Comparison results on data efficiency of **LeSS**

4.7.4. Extension to higher-order SVD

In this work, we focus on the matrix decomposition technique (i.e. SVD) for the design of s_m. However, we can easily extend our method to the tensor decomposition technique (e.g., a higher-order SVD (HOSVD)) by properly designing s_m.

For applying our method to HOSVD, since the spatial dimension in the convolution layer is small (5 or 7 is used) and it does not need to be decomposed, only two unfolding matrices are typically considered for a 4-D convolutional weight tensor. That is just two γ_l sets corresponding to two rank sets need to be considered for each unfolding matrix. As such, LeSS can be easily extended to the tensor decomposition technique. Using our methodology to compare the performance difference between matrix- and tensor-based decomposition is considered a promising research direction; we left this as a future work.

4.7.5. Extension to transformer architecture

Our method can be extended to Transformer architectures. Transformers can be broadly categorized into three main components, and we believe our approach can be applied to each of them.

Firstly, we can apply a masked pruning module to the attention module. This would involve selectively pruning connections within the attention mechanism, thereby reducing the number of attention weights and improving computational efficiency.

Secondly, a gate-based masked pruning module can be utilized to control the number of heads in the multi-head attention mechanism. By selectively pruning heads, we can adjust the model's capacity and fine-tune the attention mechanism based on the specific task or dataset.

Lastly, by applying both low-rank thresholding and masked pruning modules to the fully connected layers, we can achieve a significant reduction in parameters and computational requirements. This combined approach offers a groundbreaking way to reduce both the number of parameters and the computational complexity associated

with the fully connected layers of the Transformer model.

By extending our method to Transformers and applying these variations, we anticipate substantial improvements in parameter efficiency and computational effectiveness, contributing to more efficient and scalable Transformer models.

4.7.6. Discussion on the reasons for the improved performance of compressed models compared to the uncompressed baseline model

The enhanced performance of compressed deep neural networks compared to the uncompressed baseline model can be attributed to several factors.

Firstly, compression techniques effectively eliminate redundant or unnecessary information from the network, such as redundant weights or filters. This reduction in redundancy allows the compressed model to focus on the most essential features and parameters, resulting in improved efficiency and performance.

Secondly, compression methods, such as weight pruning or quantization, act as a form of regularization. By reducing the complexity of the model, these techniques help prevent overfitting and promote generalization. The regularization effect contributes to improved performance in the compressed model. Moreover, compressed models often exhibit enhanced generalization capabilities. By removing excessive details and noise during compression, the models become better at capturing underlying patterns and can generalize well to unseen data. This improved generalization leads to superior performance compared to the uncompressed baseline model. Additionally, during the compression process, optimization opportunities arise. Techniques such as fine-tuning and knowledge distillation are commonly applied to enhance the compressed model's performance further. Fine-tuning allows the model to adapt and optimize its parameters specifically for the given task or dataset, while knowledge distillation transfers knowledge from a larger uncompressed model to the compressed model, benefiting its performance. In summary, the improved performance of compressed deep neural networks is a result of the removal of redundancy, regularization effects,

enhanced generalization, increased model capacity, and the utilization of additional optimization opportunities introduced by compression techniques.

4.8. Conclusion

In this study, we propose a unified compression algorithm called **LeSS** to integrate filter pruning and low-rank decomposition via a joint learning framework. Our framework consists of two learning strategies: mask learning for filter pruning and threshold learning for low-rank decomposition. Both strategies are differentiable and are jointly optimized to satisfy the desired resource constraint. Our unified framework does not require compression-aware regularized training that is burdensome and is not dependent on heuristics when selecting filters and ranks. Several experiments confirm that our compression method is effective and efficient. Although this study demonstrates the effectiveness of our technique for vision tasks, its effectiveness for natural language processing and audio tasks using large-scale networks (e.g., transformer, BERT) remains as a future work.

Chapter 5. Conclusion and limitations

In this dissertation, we put forth a number of approaches that aim to enhance the low-rank compression process. Our first contribution is proposing a new low-rank compression technique, called BSR, which surpasses previous methods due to its rank selection algorithm and rank regularized training. The modified beam-search algorithm is founded on the belief that it presents an optimal way to balance search speed and performance. Important alterations, such as the introduction of a level step size and compression rate constraint, are also implemented. Our modified stable rank method is the first regularization approach that directly regulates the rank, and this is a significant factor in BSR's outstanding performance.

Furthermore, we introduce a unified compression algorithm named LeSS, which combines filter pruning and low-rank decomposition with a joint learning framework. Our framework involves two learning strategies: mask learning for filter pruning and threshold learning for low-rank decomposition. Both methods are differentiable and are jointly optimized to fulfill the specified resource constraint. Unlike previous techniques, our unified approach does not necessitate compression-aware regularized training, which is difficult to perform, and it does not rely on heuristics when choosing filters and ranks. Numerous experiments prove that our compression method is both effective and efficient. While our research highlights the usefulness of our method for vision tasks, its usefulness for natural language processing and audio tasks that use large-scale networks (e.g., transformer, BERT) will be investigated in future work.

We expect that our proposed techniques will have a broad range of applications in various industrial fields.

In our study, we dedicated significant attention to the SVD method as the primary focus. However, it is crucial to acknowledge the existence of other tensor decomposition techniques, such as Tucker decomposition, that warrant further exploration in future

research endeavors. By incorporating these alternative methods into our analysis, albeit with the need for additional hyperparameter tuning, there lies a substantial potential for further enhancing the accuracy and performance of our approach.

Moreover, it is important to note that our research primarily concentrated on image classification tasks. While this served as a valuable starting point, expanding our investigations to encompass a broader range of tasks, such as object detection, would be invaluable for gaining a comprehensive understanding of the versatility and robustness of our method. Furthermore, exploring the applicability of our approach on different datasets across various application domains can provide valuable insights into its generalizability and effectiveness in diverse contexts.

Additionally, it is worth mentioning that our compression experiments were conducted on a limited set of architectures. While the findings from our study contribute valuable knowledge, their applicability extends beyond the specific architectures examined. Notably, the insights gained from our research can be extended to a broader range of network architectures, including the increasingly popular transformer-based networks that have garnered significant attention in recent research. By systematically exploring the application of our method on these diverse architectures, we can further validate its effectiveness, uncover potential optimizations, and expand its practical utility in various real-world scenarios.

Bibliography

Alvarez, J. M., and Salzmann, M. 2017. Compression-aware training of deep networks. *Advances in neural information processing systems* 30:856–867.

Alwani, M.; Wang, Y.; and Madhavan, V. 2022. Decore: Deep compression with reinforcement learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 12349–12359.

Antoniol, G.; Brugnara, F.; Cettolo, M.; and Federico, M. 1995. Language model representations for beam-search decoding. In *1995 International Conference on Acoustics, Speech, and Signal Processing*, volume 1, 588–591. IEEE.

Boulanger-Lewandowski, N.; Bengio, Y.; and Vincent, P. 2013. Audio chord recognition with recurrent neural networks. In *ISMIR*, 335–340. Citeseer.

Buciluǎ, C.; Caruana, R.; and Niculescu-Mizil, A. 2006. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, 535–541.

Cai, L.; An, Z.; Yang, C.; and Xu, Y. 2021a. Soft and hard filter pruning via dimension reduction. In *2021 International Joint Conference on Neural Networks (IJCNN)*, 1–8. IEEE.

Cai, L.; An, Z.; Yang, C.; and Xu, Y. 2021b. Softer pruning, incremental regularization. In *2020 25th International Conference on Pattern Recognition (ICPR)*, 224–230. IEEE.

Cai, L.; An, Z.; Yang, C.; Yan, Y.; and Xu, Y. 2022. Prior gradient mask guided pruning-aware fine-tuning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 1.

Carreira-Perpinán, M. A., and Idelbayev, Y. 2018. "learning-compression" algorithms for neural net pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 8532–8541.

Chen, Y.; Yang, T.; Zhang, X.; Meng, G.; Xiao, X.; and Sun, J. 2019a. Detnas: Backbone search for object detection. In *Advances in Neural Information Processing Systems*, 6642–6652.

Chen, Z.; Lin, J.; Liu, S.; Chen, Z.; Li, W.; Zhao, J.; and Yan, W. 2019b. Exploiting weight-level sparsity in channel pruning with low-rank approximation. In *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, 1–5. IEEE.

Chen, Z.; Chen, Z.; Lin, J.; Liu, S.; and Li, W. 2020. Deep neural network acceleration based on low-rank approximated channel pruning. *IEEE Transactions on Circuits and Systems I: Regular Papers* 67(4):1232–1244.

Chen, L.; Jiang, X.; Liu, X.; and Haardt, M. 2022. Reweighted low-rank factorization with deep prior for image restoration. *IEEE Transactions on Signal Processing* 70:3514–3529.

Chin, T.-W.; Ding, R.; Zhang, C.; and Marculescu, D. 2020. Towards efficient model compression via learned global ranking. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 1518–1528.

Cho, H.; Kim, Y.; Lee, E.; Choi, D.; Lee, Y.; and Rhee, W. 2020. Basic enhancement strategies when using bayesian optimization for hyperparameter tuning of deep neural networks. *IEEE Access* 8:52588–52608.

Choi, D.; Lee, K.; Hwang, D.; and Rhee, W. 2021. Statistical characteristics of deep representations: An empirical investigation. In *International Conference on Artificial Neural Networks*, 43–55. Springer.

Courbariaux, M.; Hubara, I.; Soudry, D.; El-Yaniv, R.; and Bengio, Y. 2016. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*.

Denton, E. L.; Zaremba, W.; Bruna, J.; LeCun, Y.; and Fergus, R. 2014. Exploiting

linear structure within convolutional networks for efficient evaluation. In *Advances in neural information processing systems*, 1269–1277.

Dubey, A.; Chatterjee, M.; and Ahuja, N. 2018. Coreset-based neural network compression. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 454–470.

Elkerdawy, S.; Elhoushi, M.; Zhang, H.; and Ray, N. 2022. Fire together wire together: A dynamic pruning approach with self-supervised mask prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 12454–12463.

Erichson, N. B.; Voronin, S.; Brunton, S. L.; and Kutz, J. N. 2016. Randomized matrix decompositions using r. *arXiv preprint arXiv:1608.02148*.

Friedman, J. H. 2012. Fast sparse regression and classification. *International Journal of Forecasting* 28(3):722–738.

Furcy, D., and Koenig, S. 2005. Limited discrepancy beam search. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, IJCAI'05, 125–131. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Gao, X.; Zhao, Y.; Dudziak, Ł.; Mullins, R.; and Xu, C.-z. 2018. Dynamic channel pruning: Feature boosting and suppression. *arXiv preprint arXiv:1810.05331*.

Gao, S.; Huang, F.; Cai, W.; and Huang, H. 2021. Network pruning via performance maximization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 9270–9280.

Garipov, T.; Podoprikhin, D.; Novikov, A.; and Vetrov, D. 2016. Ultimate tensorization: compressing convolutional and fc layers alike. *arXiv preprint arXiv:1611.03214*.

Geman, D., and Yang, C. 1995. Nonlinear image recovery with half-quadratic regularization. *IEEE transactions on Image Processing* 4(7):932–946.

Gu, S.; Zhang, L.; Zuo, W.; and Feng, X. 2014. Weighted nuclear norm minimization

with application to image denoising. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2862–2869.

Gu, S.; Xie, Q.; Meng, D.; Zuo, W.; Feng, X.; and Zhang, L. 2017. Weighted nuclear norm minimization and its applications to low level vision. *International journal of computer vision* 121:183–208.

Guo, K.; Xie, X.; Xu, X.; and Xing, X. 2019. Compressing by learning in a low-rank and sparse decomposition form. *IEEE Access* 7:150823–150832.

Guo, S.; Wang, Y.; Li, Q.; and Yan, J. 2020. Dmcp: Differentiable markov channel pruning for neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 1539–1547.

Guo, Q.; Wu, X.-J.; Kittler, J.; and Feng, Z. 2021. Weak sub-network pruning for strong and efficient neural networks. *Neural Networks* 144:614–626.

Gupta, S.; Agrawal, A.; Gopalakrishnan, K.; and Narayanan, P. 2015. Deep learning with limited numerical precision. In *International conference on machine learning*, 1737–1746. PMLR.

Habenicht, K., and Monch, L. 2002. A finite-capacity beam-search-algorithm for production scheduling in semiconductor manufacturing. In *Proceedings of the Winter Simulation Conference*, volume 2, 1406–1413. IEEE.

Han, S.; Pool, J.; Tran, J.; and Dally, W. J. 2015. Learning both weights and connections for efficient neural networks. *arXiv preprint arXiv:1506.02626*.

Han, S.; Mao, H.; and Dally, W. J. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

He, Y.; Kang, G.; Dong, X.; Fu, Y.; and Yang, Y. 2018a. Soft filter pruning for accelerating deep convolutional neural networks. *arXiv preprint arXiv:1808.06866*.

He, Y.; Lin, J.; Liu, Z.; Wang, H.; Li, L.-J.; and Han, S. 2018b. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 784–800.

He, Y.; Dong, X.; Kang, G.; Fu, Y.; Yan, C.; and Yang, Y. 2019a. Asymptotic soft filter pruning for deep convolutional neural networks. *IEEE transactions on cybernetics* 50(8):3594–3604.

He, Y.; Liu, P.; Wang, Z.; Hu, Z.; and Yang, Y. 2019b. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 4340–4349.

He, Y.; Zhang, X.; and Sun, J. 2017. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE international conference on computer vision*, 1389–1397.

Hinton, G.; Vinyals, O.; and Dean, J. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Hou, Z.; Qin, M.; Sun, F.; Ma, X.; Yuan, K.; Xu, Y.; Chen, Y.-K.; Jin, R.; Xie, Y.; and Kung, S.-Y. 2022. Chex: Channel exploration for cnn model compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 12287–12298.

Hu, Y.; Zhang, D.; Ye, J.; Li, X.; and He, X. 2012. Fast and accurate matrix completion via truncated nuclear norm regularization. *IEEE transactions on pattern analysis and machine intelligence* 35(9):2117–2130.

Hu, H.; Peng, R.; Tai, Y.-W.; and Tang, C.-K. 2016. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*.

Hu, E. J.; Shen, Y.; Wallis, P.; Allen-Zhu, Z.; Li, Y.; Wang, S.; Wang, L.; and Chen, W. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Hua, W.; Zhou, Y.; De Sa, C. M.; Zhang, Z.; and Suh, G. E. 2019. Channel gating neural networks. *Advances in Neural Information Processing Systems* 32.

Huang, Z., and Wang, N. 2018. Data-driven sparse structure selection for deep neural networks. In *Proceedings of the European conference on computer vision (ECCV)*, 304–320.

Huang, L.; Fayong, S.; and Guo, Y. 2012. Structured perceptron with inexact search. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 142–151.

Idelbayev, Y., and Carreira-Perpinán, M. A. 2020. 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*, 8049–8059.

Jaderberg, M.; Vedaldi, A.; and Zisserman, A. 2014. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*.

Kim, Y.-D.; Park, E.; Yoo, S.; Choi, T.; Yang, L.; and Shin, D. 2015. Compression of deep convolutional neural networks for fast and low power mobile applications. *arXiv preprint arXiv:1511.06530*.

Kim, Y.-D.; Park, E.; Yoo, S.; Choi, T.; Yang, L.; and Shin, D. 2016. Compression of deep convolutional neural networks for fast and low power mobile applications. In *In Proc. of the 4th Int. Conf. Learning Representations (ICLR 2016)*.

Kim, J.; Park, C.; Jung, H.; and Choe, Y. 2019. Differentiable pruning method for neural networks. *CoRR*.

Kim, H.; Khan, M. U. K.; and Kyung, C.-M. 2019. Efficient neural network compres-

sion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 12569–12577.

Kim, E.; Lee, M.; and Oh, S. 2015. Elastic-net regularization of singular values for robust subspace learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 915–923.

Kim, J.; Park, S.; and Kwak, N. 2018. Paraphrasing complex network: Network compression via factor transfer. *arXiv preprint arXiv:1802.04977*.

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

Kolda, T. G. 2006. Multilinear operators for higher-order decompositions. Technical report, Sandia National Laboratories (SNL), Albuquerque, NM, and Livermore, CA ....

Kwon, W.; Yu, G.-I.; Jeong, E.; and Chun, B.-G. 2020. Nimble: Lightweight and parallel gpu task scheduling for deep learning. *Advances in Neural Information Processing Systems* 33:8343–8354.

Lebedev, V., and Lempitsky, V. 2016. Fast convnets using group-wise brain damage. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2554–2564.

Lebedev, V.; Ganin, Y.; Rakhuba, M.; Oseledets, I.; and Lempitsky, V. 2014. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *arXiv preprint arXiv:1412.6553*.

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

Li, H.; Kadav, A.; Durdanovic, I.; Samet, H.; and Graf, H. P. 2016. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*.

Li, Y.; Gu, S.; Mayer, C.; Gool, L. V.; and Timofte, R. 2020. Group sparsity: The hinge

between filter pruning and decomposition for network compression. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 8018–8027.

Li, Y.; Lin, S.; Liu, J.; Ye, Q.; Wang, M.; Chao, F.; Yang, F.; Ma, J.; Tian, Q.; and Ji, R. 2021. Towards compact cnns via collaborative compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 6438–6447.

Li, N.; Pan, Y.; Chen, Y.; Ding, Z.; Zhao, D.; and Xu, Z. 2022. Heuristic rank selection with progressively searching tensor ring network. *Complex & Intelligent Systems* 8(2):771–785.

Liebenwein, L.; Baykal, C.; Lang, H.; Feldman, D.; and Rus, D. 2019. Provable filter pruning for efficient neural networks. *arXiv preprint arXiv:1911.07412*.

Liebenwein, L.; Maalouf, A.; Feldman, D.; and Rus, D. 2021. Compressing neural networks: Towards determining the optimal layer-wise decomposition. *Advances in Neural Information Processing Systems* 34:5328–5344.

Lin, S.; Ji, R.; Li, Y.; Deng, C.; and Li, X. 2019a. Toward compact convnets via structure-sparsity regularized filter pruning. *IEEE transactions on neural networks and learning systems* 31(2):574–588.

Lin, S.; Ji, R.; Yan, C.; Zhang, B.; Cao, L.; Ye, Q.; Huang, F.; and Doermann, D. 2019b. Towards optimal structured cnn pruning via generative adversarial learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2790–2799.

Lin, M.; Ji, R.; Wang, Y.; Zhang, Y.; Zhang, B.; Tian, Y.; and Shao, L. 2020. Hrank: Filter pruning using high-rank feature map. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 1529–1538.

Liu, B.; Wang, M.; Foroosh, H.; Tappen, M.; and Pensky, M. 2015. Sparse convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 806–814.

Liu, Z.; Sun, M.; Zhou, T.; Huang, G.; and Darrell, T. 2018. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*.

Liu, Z.; Mu, H.; Zhang, X.; Guo, Z.; Yang, X.; Cheng, K.-T.; and Sun, J. 2019. Metapruning: Meta learning for automatic neural network channel pruning. In *Proceedings of the IEEE/CVF international conference on computer vision*, 3296–3305.

Loshchilov, I., and Hutter, F. 2016. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*.

Lowerre, B. T. 1976. The harpy speech recognition system. In *Carnegie Mellon University*.

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

Luo, J.-H., and Wu, J. 2020b. Neural network pruning with residual-connections and limited-data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 1458–1467.

Luo, J.-H.; Wu, J.; and Lin, W. 2017. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE international conference on computer vision*, 5058–5066.

Masana, M.; van de Weijer, J.; Herranz, L.; Bagdanov, A. D.; and Alvarez, J. M. 2017. Domain-adaptive deep network compression. In *Proceedings of the IEEE International Conference on Computer Vision*, 4289–4297.

Meister, C.; Vieira, T.; and Cotterell, R. 2020. If beam search is the answer, what was the question? *arXiv preprint arXiv:2010.02650*.

Mohan, K., and Fazel, M. 2012. Iterative reweighted algorithms for matrix rank minimization. *The Journal of Machine Learning Research* 13(1):3441–3473.

Nakajima, S.; Tomioka, R.; Sugiyama, M.; and Babacan, S. D. 2015. Condition for

perfect dimensionality recovery by variational bayesian pca. *J. Mach. Learn. Res.* 16:3757–3811.

Nie, F.; Huang, H.; and Ding, C. 2012. Low-rank matrix recovery via efficient schatten p-norm minimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 26, 655–661.

Novikov, A.; Podoprikhin, D.; Osokin, A.; and Vetrov, D. P. 2015. Tensorizing neural networks. *Advances in neural information processing systems* 28.

Peng, C.; Kang, Z.; Li, H.; and Cheng, Q. 2015. Subspace clustering using log-determinant rank approximation. In *Proceedings of the 21th ACM SIGKDD international conference on Knowledge Discovery and Data Mining*, 925–934.

Phan, A.-H.; Sobolev, K.; Sozykin, K.; Ermilov, D.; Gusak, J.; Tichavskỳ, P.; Glukhov, V.; Oseledets, I.; and Cichocki, A. 2020. Stable low-rank tensor decomposition for compression of convolutional neural network. In *European Conference on Computer Vision*, 522–539. Springer.

Piratla, V.; Netrapalli, P.; and Sarawagi, S. 2020. Efficient domain generalization via common-specific low-rank decomposition. In *International Conference on Machine Learning*, 7728–7738. PMLR.

Rastegari, M.; Ordonez, V.; Redmon, J.; and Farhadi, A. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*, 525–542. Springer.

Reeves, C. R. 1993. Modern heuristic techniques for combinatorial problems. In *John Wiley & Sons, Inc*.

Romero, A.; Ballas, N.; Kahou, S. E.; Chassang, A.; Gatta, C.; and Bengio, Y. 2014. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*.

Ruan, X.; Liu, Y.; Yuan, C.; Li, B.; Hu, W.; Li, Y.; and Maybank, S. 2020. Edp: An effi-

cient decomposition and pruning scheme for convolutional neural network compression. *IEEE Transactions on Neural Networks and Learning Systems* 32(10):4499–4513.

Rudelson, M., and Vershynin, R. 2007. Sampling from large matrices: An approach through geometric functional analysis. *Journal of the ACM (JACM)* 54(4):21–es.

Sanyal, A.; Torr, P. H.; and Dokania, P. K. 2019. Stable rank normalization for improved generalization in neural networks and gans. *arXiv preprint arXiv:1906.04659*.

Shang, H.; Wu, J.-L.; Hong, W.; and Qian, C. 2022. Neural network pruning by cooperative coevolution. *arXiv preprint arXiv:2204.05639*.

Srinivas, S., and Babu, R. V. 2015. Data-free parameter pruning for deep neural networks. *arXiv preprint arXiv:1507.06149*.

Sui, Y.; Yin, M.; Xie, Y.; Phan, H.; Aliari Zonouz, S.; and Yuan, B. 2021. Chip: Channel independence-based pruning for compact neural networks. *Advances in Neural Information Processing Systems* 34:24604–24616.

Sun, Q.; Xiang, S.; and Ye, J. 2013. Robust principal component analysis via capped norms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, 311–319.

Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; and Rabinovich, A. 2015. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 1–9.

Tai, C.; Xiao, T.; Zhang, Y.; Wang, X.; et al. 2015. Convolutional neural networks with low-rank regularization. *arXiv preprint arXiv:1511.06067*.

Tang, Y.; Wang, Y.; Xu, Y.; Tao, D.; Xu, C.; Xu, C.; and Xu, C. 2020. Scop: Scientific control for reliable neural network pruning. *Advances in Neural Information Processing Systems* 33:10936–10947.

Tang, Y.; Wang, Y.; Xu, Y.; Deng, Y.; Xu, C.; Tao, D.; and Xu, C. 2021. Manifold

regularized dynamic network pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 5018–5028.

Tengfei.Z; Zhaocheng.G; Hanping.H; and Dingmeng.S. 2021. Generating natural language adversarial examples through an improved beam search algorithm. In *arXiv:2110.08036*.

Trzasko, J., and Manduca, A. 2008. Highly undersampled magnetic resonance image reconstruction via homotopic ℓ_{0}-minimization. *IEEE Transactions on Medical imaging* 28(1):106–121.

Tukan, M.; Maalouf, A.; Weksler, M.; and Feldman, D. 2020. Compressed deep networks: Goodbye svd, hello robust low-rank approximation. *arXiv preprint arXiv:2009.05647*.

Wang, Y.; Zhang, X.; Xie, L.; Zhou, J.; Su, H.; Zhang, B.; and Hu, X. 2020. Pruning from scratch. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 12273–12280.

Wang, Z.; Li, C.; and Wang, X. 2021. Convolutional neural network pruning with structural redundancy reduction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 14913–14922.

Wen, W.; Xu, C.; Wu, C.; Wang, Y.; Chen, Y.; and Li, H. 2017. Coordinating filters for faster deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, 658–666.

Wu, J.; Leng, C.; Wang, Y.; Hu, Q.; and Cheng, J. 2016. Quantized convolutional neural networks for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 4820–4828.

Wu, K.; Guo, Y.; and Zhang, C. 2019. Compressing deep neural networks with sparse matrix factorization. *IEEE transactions on neural networks and learning systems* 31(10):3828–3838.

Xu, Y., and Fern, A. 2007. On learning linear ranking functions for beam search. In *Proceedings of the 24th international conference on Machine learning*, 1047–1054.

Xu, Y.; Li, Y.; Zhang, S.; Wen, W.; Wang, B.; Dai, W.; Qi, Y.; Chen, Y.; Lin, W.; and Xiong, H. 2019. Trained rank pruning for efficient deep neural networks. In *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing-NeurIPS Edition (EMC2-NIPS)*, 14–17. IEEE.

Xu, Y.; Li, Y.; Zhang, S.; Wen, W.; Wang, B.; Qi, Y.; Chen, Y.; Lin, W.; and Xiong, H. 2020. Trp: Trained rank pruning for efficient deep neural networks. *arXiv preprint arXiv:2004.14566*.

Xue, J.; Li, J.; and Gong, Y. 2013. Restructuring of deep neural network acoustic models with singular value decomposition. In *Interspeech*, 2365–2369.

Yaguchi, A.; Suzuki, T.; Nitta, S.; Sakata, Y.; and Tanizawa, A. 2019. Decomposable-net: Scalable low-rank compression for neural networks. *arXiv preprint arXiv:1910.13141*.

Yin, M.; Sui, Y.; Liao, S.; and Yuan, B. 2021. Towards efficient tensor decomposition-based dnn model compression with optimization framework. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 10674–10683.

Yu, X.; Liu, T.; Wang, X.; and Tao, D. 2017. On compressing deep models by low rank and sparse decomposition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 7370–7379.

Yu, S.; Mazaheri, A.; and Jannesari, A. 2022. Topology-aware network pruning using multi-stage graph embedding and reinforcement learning. In *International Conference on Machine Learning*, 25656–25667. PMLR.

Zagoruyko, S., and Komodakis, N. 2016. Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer. *arXiv preprint arXiv:1612.03928*.

Zhai, X.; Kolesnikov, A.; Houlsby, N.; and Beyer, L. 2022. Scaling vision transform-

ers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 12104–12113.

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

Zhang, X.; Zou, J.; Ming, X.; He, K.; and Sun, J. 2015b. Efficient and accurate approximations of nonlinear convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and pattern Recognition*, 1984–1992.

Zhou, H.; Alvarez, J. M.; and Porikli, F. 2016. Less is more: Towards compact cnns. In *European Conference on Computer Vision*, 662–677. Springer.

Zhou, Y.; Ren, T.; Zhu, C.; Sun, X.; Liu, J.; Ding, X.; Xu, M.; and Ji, R. 2021. Trar: Routing the attention spans in transformer for visual question answering. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2074–2084.

Zhuang, Z.; Tan, M.; Zhuang, B.; Liu, J.; Guo, Y.; Wu, Q.; Huang, J.; and Zhu, J. 2018. Discrimination-aware channel pruning for deep neural networks. *arXiv preprint arXiv:1810.11809*.

Appendices

A. The SoTA compression methods

ResNet56 on CIFAR10 (Idelbayev and Carreira-Perpinán 2020; Zhuang et al. 2018; Li et al. 2020) are showing the SoTA performance for the 48-50% category, (Idelbayev and Carreira-Perpinán 2020; Yu, Mazaheri, and Jannesari 2022; Ruan et al. 2020) are showing the SoTA performance for the 54-57% category, and (Idelbayev and Carreira-Perpinán 2020; Sui et al. 2021; Li et al. 2020) are showing the SoTA performance for the 73-76% category.

ResNet50 on ImageNet (Xu et al. 2020; Sui et al. 2021; Ruan et al. 2020) are showing the SoTA performance for the 45-50% category, (Phan et al. 2020; Shang et al. 2022; Li et al. 2020) are showing the SoTA performance for the 62-66% category.

Note that the SoTA-achieving algorithm for each FLOP category can vary. This is because of two reasons. First, each algorithm's performance curve is different and achieving a SoTA performance in one FLOP category does not guarantee achieving SoTA in another FLOP category. Second, for some of the benchmark algorithms, the experimental results were partially available over the FLOP ranges.

B. Resource budget definition

Parameter budget By $g_{\mathbf{c}}$ and $g_{\mathbf{r}}$, we define the formula of the parameter resource budget as follows:

$$\frac{\sum_{l=1}^{L} g_{\mathbf{r}}(\mathbf{z_r})(l) \cdot k_l \cdot k_l \cdot g_{\mathbf{c}}(\mathbf{z_c})(l-1) + g_{\mathbf{r}}(\mathbf{z_r})(l) \cdot g_{\mathbf{c}}(\mathbf{z_c})(l)}{\sum_{l=1}^{L} k_l \cdot k_l \cdot C_{\text{in}}^{l} \cdot C_{\text{out}}^{l}} \tag{1}$$

k_l denotes the kernel size of the l-th layer. C_{in}^{l} and C_{out}^{l} denote l-th layer's input- and output-channels of the original model, respectively. $g_{\mathbf{r}}(\mathbf{z_r})(l)$ and $g_{\mathbf{c}}(\mathbf{z_c})(l)$ are the l-th

layer's selected rank and number of selected filters, respectively.

Volume budget This budget controls the size of activations, thus setting an upper bound on the amount of memory required for the inference process. We define volume budget as follows:

$$\frac{\sum_{l=1}^{L} A_l \cdot \big(g_{\mathbf{r}}(\mathbf{z_r})(l) + g_{\mathbf{c}}(\mathbf{z_c})(l)\big)}{\sum_{l=1}^{L} A_l \cdot C_{\text{out}}^{l}} \tag{2}$$

A_l denotes the area of the l-th layer's feature maps. C_{out}^{l} denote l-th layer's output channels of the original model. $g_{\mathbf{r}}(\mathbf{z_r})(l)$ and $g_{\mathbf{c}}(\mathbf{z_c})(l)$ are the l-th layer's selected rank and number of selected filters, respectively.

C. Implementation details

For reproducibility, we provide the details of implementation and hyper-parameters used for training DIOR. Our implementations are based on LC (Idelbayev and Carreira-Perpinán 2020) library. Our implementation codes will be made available online.

C.1. Hyper-parameter setting

For all the experiments, we use the hyper-parameters in Table A1.

Hyper-parameter	CIFAR10	CIFAR100	ImageNet
Batch size	128	128	64
Epochs for fine-tuning	100	100	100
Learning rate for baseline model	0.01	0.01	-
Learning rate for fine-tuning	0.001	0.001	0.001
λ	1	1	1
τ	$\frac{2}{\sigma_1}$	$\frac{2}{\sigma_1}$	$\frac{2}{\sigma_1}$
μ_0	5	5	5
α (in μ_i)	50	50	50
β (in μ_i)	4	4	8

Table A1. Hyper-parameters used for training DIOR on various experiments. σ_1 is the largest singular value for each layer.

C.2. Tuning details of hyper-parameters

We fix the hyper-parameter $\lambda = 1$ in Eq. (4.3). Both τ in Eq. (4.9) and μ_i in Eq. (4.4) are tuned via a light grid-search. τ controls which singular values should be treated as zero. We define τ as $\frac{C}{\sigma_1}$, where σ_1 is the largest singular value of each layer, and perform a light grid search for C. The normalization by σ_1 allows a single hyper-parameter C to be used over all the layers. Therefore, the normalization significantly simplifies the hyper-parameter search.

For μ_i, its α is simply set as a large constant of 50 because DIOR's performance is not sensitive to the choice, and its β is explored using a light grid search.

D. Full comparison results

We provide the full comparison results for ResNet56 on CIFAR10 and ResNet50 on ImageNet-1k in Table A2 and Table A3, respectively.

Dataset on Model	Compression method	Algorithm	Baseline (%)	Test acc.(%)	Δ Test acc.(%)	MFLOPs (Reduction ratio)	Params (Compression ratio)
CIFAR10 on ResNet56	Low-rank	Accelerating (Zhang et al. 2015a)	93.14	91.56	- 1.58	58.9 (53%)	N/A
		Speeding (Jaderberg, Vedaldi, and Zisserman 2014)	93.14	91.59	- 1.55	58.9 (53%)	N/A
		LC (Idelbayev and Carreira-Perpinán 2020)	92.73	93.10	+ 0.37	55.7 (56 %)	N/A
		TRP (Xu et al. 2020)	93.14	92.77	- 0.37	52.9 (57 %)	N/A
		CA (Alvarez and Salzmann 2017)	92.73	91.13	- 1.60	51.4 (59 %)	N/A
		BSR (**?**)	92.73	93.53	+ 0.80	55.7 (56 %)	0.37 M (56 %)
		BSR (**?**)	92.73	92.51	- 0.22	32.1 (74 %)	0.21 M (75 %)
	Pruning	CHIP (Sui et al. 2021)	93.26	94.16	+ 0.75	66.0 (47 %)	0.48 M (43 %)
		DP (Kim et al. 2019)	92.66	92.36	- 0.30	65.2 (48 %)	N/A
		AMC (He et al. 2018b)	92.80	91.90	- 0.90	62.7 (50 %)	N/A
		CP (He, Zhang, and Sun 2017)	93.80	92.80	- 1.00	62.7 (50 %)	N/A
		ThiNet (Luo, Wu, and Lin 2017)	93.80	92.98	- 0.82	62.7 (50 %)	0.41 M (50 %)
		PFS (Wang et al. 2020)	93.23	93.05	- 0.18	62.7 (50 %)	N/A
		Rethink (Liu et al. 2018)	93.80	93.07	- 0.73	62.7 (50 %)	N/A
		SFP (He et al. 2018a)	93.59	93.35	- 0.24	62.7 (50 %)	N/A
		NPPM (Gao et al. 2021)	93.04	93.40	+ 0.36	62.7 (50 %)	N/A
		DCP (Zhuang et al. 2018)	93.80	93.49	- 0.31	62.7 (50 %)	0.43 M (49 %)
		LeGR (Chin et al. 2020)	93.90	93.70	- 0.20	58.9 (53 %)	N/A
		SRR-GR (Wang, Li, and Wang 2021)	93.38	93.75	- 0.37	57.9 (54 %)	N/A
		GNN (Yu, Mazaheri, and Jannesari 2022)	93.39	93.49	+ 0.10	57.6 (54 %)	N/A
		FTWT (Elkerdawy et al. 2022)	93.66	92.63	- 1.03	47.4 (60 %)	N/A
		ASFP (He et al. 2019a)	94.85	89.72	- 5.13	35.2 (73 %)	N/A
		ASRFP (Cai et al. 2021b)	94.85	90.54	- 4.31	35.2 (73 %)	N/A
		GHFP (Cai et al. 2021a)	94.85	92.54	- 2.31	35.2 (73 %)	N/A
		CHIP (Sui et al. 2021)	93.26	92.05	- 1.21	34.8 (73 %)	0.24 M (72 %)
	Hybrid	CC (Li et al. 2021)	93.33	93.64	+ 0.31	65.2 (48 %)	0.44 M (48 %)
		Hinge (Li et al. 2020)	92.95	93.69	+ 0.74	62.7 (50 %)	0.41 M (51 %)
		DIOR	92.73	**94.23**	+ 1.50	62.7 (50 %)	0.43 M (49 %)
		DIOR	92.73	**94.13**	+ 1.40	55.1 (55 %)	0.39 M (54 %)
		EDP (Ruan et al. 2020)	93.61	93.61	0.00	53.0 (58 %)	0.39 M (54 %)
		DIOR	92.73	**94.08**	+ 1.35	47.4 (60 %)	0.37 M (57 %)
		DIOR	92.73	**93.44**	+ 0.71	32.2 (75 %)	0.22 M (74 %)
		Hinge (Li et al. 2020)	92.95	92.65	- 0.30	31.0 (76 %)	0.17 M (80 %)

Table A2. Performance comparison results for CIFAR-10 on ResNet56.

Dataset on Model	Compression method	Algorithm	Baseline (%)	Test acc.(%)	Δ Test acc.(%)	GFLOPs (Reduction ratio)	Params (Compression ratio)
ImageNet-1k on ResNet50	Low-rank	TRP (Xu et al. 2020)	75.90	74.06	- 1.84	2.3 (45 %)	N/A
		Stable (Phan et al. 2020)	76.15	74.66	- 1.47	1.6 (62 %)	N/A
		BSR (**?**)	76.20	75.00	- 1.2	2.2 (47 %)	12.5 M (51 %)
		BSR (**?**)	76.20	74.80	- 1.4	1.8 (55 %)	10.1 M (60 %)
		BSR (**?**)	76.20	74.10	- 2.1	1.4 (67 %)	7.5 M (70 %)
		BSR (**?**)	76.20	73.40	- 2.8	1.1 (73 %)	5.0 M (80 %)
	Pruning	PFS (Wang et al. 2020)	76.10	76.70	+ 0.60	3.0 (25 %)	N/A
		ThiNet (Luo, Wu, and Lin 2017)	73.00	72.04	- 0.96	2.4 (37 %)	16.9 M (34 %)
		SFP (He et al. 2018a)	76.20	74.60	- 1.60	2.4 (42 %)	N/A
		CCEP (Shang et al. 2022)	76.13	76.06	- 0.07	2.3 (44 %)	N/A
		CHIP (Sui et al. 2021)	76.20	76.30	+ 0.10	2.3 (45 %)	15.1 M (41 %)
		PFS (Wang et al. 2020)	76.10	75.60	- 0.50	2.0 (49 %)	N/A
		CP (He, Zhang, and Sun 2017)	76.10	73.30	- 2.80	2.1 (49 %)	N/A
		CP (He, Zhang, and Sun 2017)	76.10	73.30	- 2.80	2.1 (51 %)	N/A
		SSR (Lin et al. 2019a)	76.20	73.40	- 2.80	1.9 (55 %)	15.5 M (39 %)
		SSR (Lin et al. 2019a)	76.20	72.60	- 3.60	1.7 (60 %)	12.0 M (53 %)
		GAL (Lin et al. 2019b)	76.20	69.90	- 6.30	1.6 (62 %)	14.6 M (43 %)
		WSP (Guo et al. 2021)	76.13	73.91	- 2.22	1.5 (63 %)	11.6 M (54 %)
		CCEP (Shang et al. 2022)	76.13	74.87	- 1.26	1.5 (64 %)	N/A
		AutoPruner (Luo and Wu 2020a)	76.10	73.00	- 3.10	1.4 (65 %)	N/A
		WSP (Guo et al. 2021)	76.13	72.04	- 4.09	1.1 (73 %)	9.1 M (65 %)
		CURL (Luo and Wu 2020b)	76.20	73.40	- 2.80	1.1 (73 %)	6.7 M (74 %)
		PFS (Wang et al. 2020)	76.10	72.80	- 3.30	1.0 (76 %)	N/A
		CHIP (Sui et al. 2021)	76.20	73.30	- 2.90	1.0 (76 %)	7.8 M (69 %)
		HRANK (Lin et al. 2020)	76.10	69.10	- 7.00	1.0 (76 %)	8.3 M (67 %)
	Hybrid	EDP (Ruan et al. 2020)	75.90	75.51	- 0.39	2.1 (50 %)	N/A
		DIOR	76.20	**77.20**	+ 1.00	2.3 (44 %)	13.8 M (46 %)
		DIOR	76.20	**76.93**	+ 0.73	2.1 (50 %)	12.2 M (52 %)
		EDP (Ruan et al. 2020)	75.90	75.34	- 0.56	1.9 (53 %)	N/A
		CC (Li et al. 2021)	76.15	75.59	- 0.56	1.9 (53 %)	13.2 M (48 %)
		Hinge (Li et al. 2020)	N/A	74.7	N/A	1.9 (53 %)	N/A
		EDP (Ruan et al. 2020)	75.90	74.48	- 1.42	1.7 (60 %)	N/A
		CC (Li et al. 2021)	76.15	74.54	- 1.61	1.5 (63 %)	10.6 M (59 %)
		DIOR	76.20	**76.13**	- 0.07	1.4 (66 %)	8.4 M (67 %)
		DIOR	76.20	**74.10**	- 2.10	0.9 (78 %)	5.4 M (79 %)

Table A3. Performance comparison results for ImageNet-1k on ResNet50.