Attentive Fine-Grained Structured Sparsity for Image Restoration

Junghun Oh¹ Heewon Kim¹ Seungjun Nah¹,³ Cheeun Hong¹ Jonghyun Choi⁴ Kyoung Mu Lee¹,²
¹Dept. of ECE, ASRI, ²IP AI, Seoul National University ³NVIDIA ⁴Yonsei University
¹{dh6dh, ghimhw, cheeun914, kyoungmu}@snu.ac.kr, ³seungjun.nah@gmail.com, ⁴jc@yonsei.ac.kr

Abstract

Image restoration tasks have witnessed great performance improvement in recent years by developing large deep models. Despite the outstanding performance, the heavy computation demanded by the deep models has restricted the application of image restoration. To lift the restriction, it is required to reduce the size of the networks while maintaining accuracy. Recently, N:M structured pruning has appeared as one of the effective and practical pruning approaches for making the model efficient with the accuracy constraint. However, it fails to account for different computational complexities and performance requirements for different layers of an image restoration network. To further optimize the trade-off between the efficiency and the restoration accuracy, we propose a novel pruning method that determines the pruning ratio for N:M structured sparsity at each layer. Extensive experimental results on super-resolution and deblurring tasks demonstrate the efficacy of our method which outperforms previous pruning methods significantly. PyTorch implementation for the proposed methods will be publicly available at https://github.com/JungHunOh/SLS_CVPR2022

1. Introduction

Advances in deep learning has brought success in image restoration tasks such as super-resolution [29, 48] and deblurring [39, 60, 61]. Due to the heavy computational burden required by such methods, however, computing high-resolution images in practical applications has been challenging. Network pruning is one of the most popular tools to alleviate the computational burden of neural networks by eliminating weights that are less critical to the accuracy. It has shown remarkable efficacy in finding submodels for a better trade-off between accuracy and efficiency for image classification [9, 15, 17, 30, 34] and segmentation [14, 57].

Unstructured pruning [11, 13, 26] aims to find and remove individual weights that have relatively less impact on model accuracy. Still, accelerating the resulting models is difficult due to irregular sparsity patterns of weight tensors, considering the complex nature of parallelization on GPUs. On the other hand, structured pruning removes predetermined structures (e.g., a filter in convolution layers [28] or a channel of feature maps [17]) to enable the acceleration of pruned networks on GPUs. However, we empirically find image restoration models to be often susceptible to substantial performance degradation from structured pruning.

Recently, N:M fine-grained structured sparsity [19, 37, 64] has emerged as a better alternative, combining the strengths of both pruning methods: namely, fine-grained sparsity from unstructured pruning and hardware acceleration-ability from structured pruning. N:M structured sparsity enforces N number of weights in each group of M number of consecutive weights to have non-zero val-
ues. Such sparsity constraint has the potential of hardware acceleration, where 2:4 sparsity pattern has recently been supported in NVIDIA Ampere generation GPUs [37]. However, training a network with $N:M$ sparsity has been proven to be difficult, which the existing works on $N:M$ sparsity have focused on improving [37, 64]. Such difficulty has prevented the direct application of the already developed pruning techniques, such as pruning with layer-wise varying pruning ratios [27, 34], which is known to be crucial to the performance of the pruned networks. Particularly, we observed that several layers (e.g. the last upsampling layer) in image restoration networks are very sensitive to pruning with respect to the performance.

Here, we propose a layer-wise $N:M$ sparsity search framework for efficient image restoration networks, named SLS. In the prior arts [9, 17, 30, 31, 33], a filter or a channel is used as a unit of pruning but it is challenging to define the pruning unit in the case of $N:M$ sparsity. To this end, we propose to consider the original weight tensor as the sum of sparse tensors whose configurations are determined by the magnitude of weights. We use each of sparse tensor as the unit of pruning in our $N:M$ sparsity search problem. To learn how many units to preserve, we propose a trainable score for each pruning unit, which is designed to ensure units with the lowest magnitude-based importance are removed first for better performance.

Furthermore, since image restoration tasks often have different computational constraints, we present an adaptive inference method that uses several models trained by SLS with different efficiency. The proposed adaptive inference technique determines which pruned model should be used at inference time depending on the restoration difficulty of an input image patch. The adaptive inference method further improves the efficiency-accuracy trade-off and enables a flexible adoption of the computational budgets.

We summarize our contributions as follows:

- Observing the pruning sensitivity of each layer to be different, we propose a novel method, SLS, to determine the layer-wise $N:M$ sparsity levels.
- From the mixture of the pruned models with different computational costs and accuracy, we propose to find a better trade-off with additional controllability at inference time.
- By extensive experiments with super-resolution and deblurring, we empirically validate our pruned models generally achieve state-of-the-art performance.

2. Related Work

Image Restoration. Motion blur or low resolution are common artifacts in images and restoring high-quality from such low-quality inputs has been widely studied in computer vision. In deep learning literature, many neural network architectures have been proposed to mitigate the artifacts. In image deblurring, multi-scale architectures [7, 39, 52], stacked networks [50, 58, 60], recurrent models [61] were proposed primarily to achieve better restoration quality. Due to the difficulty of ill-posed problem, such methods have employed complex architectures with high model capacity, leading to slow execution speed. Similarly, the advances in deep super-resolution from pioneering SR-CNN [5] were made by studying various network architectures such as deep networks [23, 29], attention mechanisms [4, 42, 62], and dense connections [12, 53, 63].

In order to make deblurring and super-resolution fast, many efforts have been made to design light-weight architectures. For deblurring, [25] adopted Inception-ResNet-v2 [51] and MobileNetV2 [46] to build feature pyramid networks. Also, [44] used a shallow recurrent model and progressively deblurred an image. Similarly, for super-resolution, [2] and [32] proposed to use an effective convolutional module for an efficient yet accurate network. Furthermore, instead of designing models manually, neural architecture search (NAS) was adopted [3, 22, 49] to find efficient model structures. Different from the previous methods designing light-weight architectures, our method makes the existing models efficient by using $N:M$ sparsity.

Network Pruning. From the early studies in neural networks, the redundant weights that have negligible impact on the final output have been witnessed and pruning aims to remove such unnecessary components [13, 26]. In unstructured pruning [6, 10, 11, 45], unimportant individual weight connects were eliminated, with the primary focus on accuracy preservation. However, due to the irregular sparsity patterns, accelerating the pruned networks requires specially designed hardware, limiting the application in practice [54]. On the other hand, structured pruning removes groups of the weights (e.g., layers, filters, and channels of a feature map) in the network architecture, leading to easier acceleration on off-the-shelf devices. Especially, filter or channel pruning methods [9, 15–17, 27, 28, 30, 33, 35, 38, 56] have risen as a popular pruning strategy. They either train networks to find optimally pruned networks [9, 15, 27, 33, 56] or propose metrics to measure the relative importance of filters or channels in pre-trained models [16, 17, 28, 30, 35, 38, 43]. However, we observed that such coarse-grained structured sparsity can lead to significant damage to the performance in image restoration networks.

Recently, several approaches have been proposed in order to make the structured pruning at a more fine-grained level. Block-level sparsity with matrix math pipelines [8] is successfully accelerated with the known pruning structure. However, it requires to increase the feature size to maintain the original accuracy. A more fine-grained balanced sparsity is proposed in [55] which can be accelerated by grouping weights and pruning each group with a uni-
3. Proposed Method

3.1. \(N:M\) sparsity

A weight tensor with \(N:M\) sparsity indicates a type of a weight tensor that satisfies the following conditions (See Figure 2a): (1) The number of input channels is divisible by \(M\). (2) Each group of \(M\) consecutive weights should have at least \(N\) non-zero weights. With a weight satisfying the above constraints, the weight and input tensor are compressed by ignoring the zero weights and the corresponding input feature values. Then, the computations of the tensor multiplication between the compressed weight and input tensor are reduced to \(\frac{N}{M}\) of the original computations.

Figure 2. (a) Visual illustration for \(N:M\) sparsity [37], where \(N = 3\) and \(M = 4\). We illustrate the sparse multiplication process only for the first row of the weight, where the non-zero weights and the input features at the corresponding positions are highlighted in green boxes. \(D\) refers to the spatial dimension of the input feature. (e.g., input height \(\times\) input width in a convolutional layer.) (b) Overview of the proposed method, SLS, which decomposes the weights into \(M\) (\(M = 4\) in this case) groups based on the weight magnitude (Equation (1)). According to Equation (2), the final pruned weights are constructed based on the binary mask \(b\), each value of which controls whether the corresponding group will be used. The binary mask will be generated based on the priority of each importance group, following the operations outlined by Equation (3) and (4). We illustrate the case when the searched sparsity is 3:4.

3.2. Overview

The conventional structured pruning methods [9, 17, 30, 31, 33] remove the predetermined structural units (e.g., filters or channels of a feature map). The common approach for finding layer-wise pruning ratios in structured pruning methods is to train a score value defined in each pruning unit and remove units with a small score value [9, 21, 33]. In the case of \(N:M\) sparsity pattern, however, it is challenging to determine such structural units because there are many possible configurations for preserving \(N\) weights out of \(M\) weights. To overcome the challenge, in Section 3.3, we consider the original weight tensor as the sum of \(M\) tensors with \(1:M\) sparsity whose configurations of remaining weights are determined by the magnitude of weight. We propose to use each sparse tensor as the unit of pruning. Then, we propose a differentiable sparsity search framework that learns score values for each pruning unit by using a straight-through estimator [20]. We empirically found that the magnitude-based importance and the score value can lead to a conflict with respect to the importance rank of a pruning unit, which brings a substantial performance loss. In Section 3.4, we eliminate the conflict by ensuring the score value is aligned to the magnitude-based importance. In Section 3.5, we present our loss function and propose a loss annealing strategy to control the speed of pruning during training. In Section 3.6, we propose an adaptive inference method to improve the efficiency-accuracy trade-off.
3.3. Differentiable N:M Sparsity Search

Let \( \mathbf{W}^l \in \mathbb{R}^{c_{out}^l \times c_{in}^l \times k_h^l \times k_w^l} \) denote the weight tensor in \( l \)-th convolutional layer. For notation simplicity, \( l \) is omitted unless otherwise noted. Our goal is to find the effective sparsity level of each layer given the target computational budgets. To this end, we represent the weight tensor as the sum of sparse tensors with 1:

\[
\mathbf{W} = \sum_{i=1}^{M} \mathbf{W}_i, \tag{1}
\]

where \( \mathbf{W}_i \) is a weight tensor with 1:M sparsity in which only the \( i \)-th important weight is remained every \( M \) consecutive weights. We define the importance of each weight as its magnitude, which has been widely used in the pruning literature [19, 64]. By doing so, we can view the sparse tensor as a pruning unit in our problem setting. In other words, \( \mathbf{W}_i \) is analogous to a filter or a channel in the existing structured pruning methods. Then, we formulate the pruning ratio search problem as follows:

\[
\hat{\mathbf{W}} = \sum_{i=1}^{M} b_i \cdot \mathbf{W}_i, \tag{2}
\]

where \( b_i \) is a binary value indicating whether \( \mathbf{W}_i \) should be removed or not. To optimize \( b_i \) using gradient descent, we adopt straight-through estimator (STE) [20]:

\[
b_i = \begin{cases} 
S(p_i, \tau) & \text{in a forward path,} \\
p_i & \text{in a backward path,} 
\end{cases} \tag{3}
\]

where \( S(\cdot, \tau) \) is a function that returns 1 for the greater value than a threshold \( \tau \) and 0 otherwise and \( p_i \) is a priority score of \( \mathbf{W}_i \) that is learned during training.

3.4. Priority-Ordered Pruning

In section 3.3, we introduce the two importance measures for each sparse tensor: the magnitude of weight is for the definition of the pruning unit and the priority score is for learning the pruning ratio. However, the importance rank indicated by the two measures can be different, by which a pruning unit with weights of larger magnitude can be removed first before one with weights of smaller magnitude. We found such misalignment leads to substantial performance degradation. To solve this problem, we propose Priority-Ordered Pruning (POP) method that aligns the two importance measures by design. Specifically, we define the priority score as follows:

\[
p_i = \prod_{n=1}^{i-1} k_n \tag{4}
\]

where \( p_1 \equiv 1 \) and \( k_n \) is a trainable parameter that is initialized to 1 and clamped to ensure \( 0 \leq k_n \leq 1 \). By Equation (4), \( p_{i+1} \leq p_i \) is guaranteed, so a pruning unit with weights of smaller magnitude is removed first. Figure 2b illustrates the overall process of the proposed learning framework.

3.5. Loss Function

For practical usage of the proposed method, we design our pruning framework as a budgeted pruning [9, 41], in which a network is pruned to meet the desired target computational budget. To this end, we first define the computational costs of a convolutional layer with \( N:M \) sparsity with respect to multiply-accumulate operations (MACs). Theoretically, the MACs of a convolutional layer with \( N:M \) sparsity are \( \frac{N}{M} \) of the original ones. Thus, the computational costs of a pruned convolutional layer are defined as follows:

\[
C_{pruned} = C_{original} \times \frac{\sum_{i=1}^{M} b_i}{M}, \tag{5}
\]

where \( C_{original} = (c_{out} \cdot c_{in} \cdot k_h \cdot k_w) \times (H \cdot W) \) denotes the original computational costs of the layer and \( H \) and \( W \) are the spatial sizes of the output of the layer. Then, we formulate our loss function as follows:

\[
L = L_{task} + \lambda_{reg} \sum_{l=1}^{L} C_{pruned}^{l}, \tag{6}
\]

where \( L \) is the total number of layers to be pruned, \( L_{task} \) is the task-specific loss function (e.g., \( L1 \) loss for image restoration tasks) and \( \sum_{l=1}^{L} C_{pruned}^{l} \) is the computational regularization loss. The two loss terms are balanced by the hyper-parameter \( \lambda_{reg} \). Note that the gradient from the computational regularization loss can flow to \( k_l \) by using STE in Equation (3). Starting from a pretrained network, the network is pruned until satisfying \( \sum_{l=1}^{L} C_{pruned}^{l} \leq C_{target} \), where \( C_{target} \) denotes a target computational budget. After reaching the target budget, all \( k_l \) are frozen and the pruned network is fine-tuned by optimizing the task-specific loss.

We empirically found that when the target computational budget is extremely low (e.g., \( C_{target} = 0.1 \times \sum_{l=1}^{L} C_{original}^{l} \)), \( \lambda_{reg} \) should be large enough to reach the target budget. However, a large \( \lambda_{reg} \) can result in an aggressive performance degradation because the network is pruned too fast, which is hard to be recovered even after a fine-tuning process. To solve this problem, we set \( \lambda_{reg} \) as a small value at the beginning of training and gradually increase it according to the pruned rate change. At every predetermined \( K \) epoch, we measure the pruned rate change during the last \( K \) epochs and update \( \lambda_{reg} \) by following rules:

\[
\lambda_{reg} = \begin{cases} 
\alpha \times \lambda_{reg} & \text{if } \Delta \frac{C_{pruned}}{C_{original}} \leq T, \\
\lambda_{reg} & \text{else}, \end{cases} \tag{7}
\]

17676
3.6. Adaptive Inference

Each region in a low-quality image often has different restoration difficulties. For example, in the case of image super-resolution tasks, flat areas such as the sky can be easily restored by using only a few computational resources. Also, in order to restore large images (e.g., 2K or 4K), it can be inevitable to process the whole image by dividing it into small patches due to the resource constraints [24]. From this motivation, we propose an adaptive inference method that determines which pruned models, trained by SLS, to use according to the restoration difficulty of an input image patch at inference time.

To quantify the restoration difficulty of a patch, we assume that the more an image patch is hard to be restored, the larger error between the ground truth and the restored result. Since the ground truth is not available at inference time, we use a light-weight convolutional neural network that can estimate the mean squared error (MSE) between the ground truth and the restored result from a target model. Given several models trained by SLS with different target computational budgets, we train the MSE estimators for each candidate model using training datasets.

Formally, given an image patch \( x \), our adaptive inference method selects a model among the candidates that is obtained by the following operation:

\[
\argmax_i \frac{C_1 - C_i}{C_1} \times \beta + \frac{f_n(x) - f_i(x)}{f_n(x)},
\]

(8)

where \( C_i \) and \( f_i(\cdot) \) indicate the computational costs and the estimated MSE with respect to the \( i \)-th candidate model and \( n \) is the total number of candidates. We sort the index of each model with respect to the computational costs (\( C_{i+1} \leq C_i \)). By maximizing the two terms in Equation (8), our method tries to find the most efficient yet accurate model given \( x \). The hyper-parameter \( \beta \) enables flexible control of the computational costs of the selected model by making focus on either the computational costs or the performance. Figure 3 visualizes the proposed adaptive inference method.

4. Experiments

4.1. Dataset and Models

To validate the effectiveness of the proposed methods, we conduct experiments on image deblurring and super-resolution tasks. For image deblurring, we perform pruning on 3 model architectures that vary by the computational cost and the restoration accuracy: residual UNet [40], SRN [52] and DMPHN [60]. GOPRO dataset [39] is employed to train and evaluate the deblurring models. For image super-resolution, we use 3 popular and efficient architectures: EDSR [29], CARN [2] and RFDN [32]. We use DIV2K dataset [1] for training, Set14 [59], B100 [36] and Urban100 [18] benchmark datasets for evaluation.

4.2. Implementation Details

To make a fair comparison between different pruning methods, we train the networks with the same amount of iterations. The total training epochs are 4000 and 600, respectively for deblurring and super-resolution. Since the methods except for SR-STE [64] require a pretraining phase before pruning, we allocate half of the training epochs for pretraining and the rest for the pruning process for those methods. We set the hyper-parameters as \( \tau = 0.5, \alpha = 1.1, T = 0.1 \). \( \lambda_{reg} \) is set to \( 10^{-12} \) and \( 10^{-10} \) for image deblurring and super-resolution tasks, respectively. Also, we set \( M = 32 \) to allow extreme pruning ratios. For more details, please refer to the supplementary material.

4.3. Quantitative Comparison

In Table 1 and 2, we present the computational costs (GMACs) and the image restoration performance of the pruned models on image deblurring and super-resolution tasks, respectively. For each model, we train them using different computational budgets, 1/4, 1/8, and 1/16 of the original costs. Our method, SLS, is compared with the existing
Table 1. Image deblurring performance comparisons on GOPRO dataset [39].

<table>
<thead>
<tr>
<th>Model</th>
<th>Method</th>
<th>GMACs</th>
<th>Num. Param.</th>
<th>PSNR_\text{\text{↑}} / SSIM_\text{\text{↑}} / LPIPS_\text{\text{↑}}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unpruned</td>
<td>Filler pruning</td>
<td>458.04</td>
<td>6.79M</td>
<td>29.46 / 0.8837 / 0.1686</td>
</tr>
<tr>
<td>One-shot (2:4)</td>
<td>Filler pruning</td>
<td>230.84</td>
<td>3.40M</td>
<td>29.55 / 0.8849 / 0.1662</td>
</tr>
<tr>
<td>One-shot (8:32)</td>
<td>Filler pruning</td>
<td>115.42</td>
<td>1.70M</td>
<td>28.79 / 0.8692 / 0.1893</td>
</tr>
<tr>
<td>SR-STE (8:32)</td>
<td>Filler pruning</td>
<td>117.24</td>
<td>1.70M</td>
<td>29.19 / 0.8771 / 0.1795</td>
</tr>
<tr>
<td>SLS (Ours)</td>
<td>Unpruned</td>
<td>116.64</td>
<td>1.55M</td>
<td>29.37 / 0.8811 / 0.1740</td>
</tr>
<tr>
<td>Filler pruning</td>
<td>One-shot (4:32)</td>
<td>65.27</td>
<td>0.86M</td>
<td>28.31 / 0.8570 / 0.2070</td>
</tr>
<tr>
<td>One-shot (4:32)</td>
<td>Unpruned</td>
<td>64.44</td>
<td>0.85M</td>
<td>28.64 / 0.8664 / 0.1985</td>
</tr>
<tr>
<td>SR-STE (4:32)</td>
<td>Unpruned</td>
<td>64.44</td>
<td>0.85M</td>
<td>28.33 / 0.8571 / 0.2070</td>
</tr>
<tr>
<td>SLS (Ours)</td>
<td>One-shot (4:32)</td>
<td>60.31</td>
<td>0.79K</td>
<td>28.98 / 0.8726 / 0.1870</td>
</tr>
<tr>
<td>Filler pruning</td>
<td>One-shot (4:32)</td>
<td>29.31</td>
<td>0.42K</td>
<td>27.65 / 0.8391 / 0.2325</td>
</tr>
<tr>
<td>One-shot (2:32)</td>
<td>SR-STE (4:32)</td>
<td>32.04</td>
<td>0.42K</td>
<td>27.77 / 0.8431 / 0.2325</td>
</tr>
<tr>
<td>SLS (Ours)</td>
<td>Unpruned</td>
<td>30.91</td>
<td>397.55K</td>
<td>28.36 / 0.8573 / 0.2117</td>
</tr>
</tbody>
</table>

Table 2. Image super-resolution performance (PSNR_\text{\text{↑}}) comparisons on benchmark datasets with the scaling factor of 4.

<table>
<thead>
<tr>
<th>Model</th>
<th>Method</th>
<th>GMACs</th>
<th>Num. Param.</th>
<th>PSNR_\text{\text{↑}} / SSIM_\text{\text{↑}} / LPIPS_\text{\text{↑}}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unpruned</td>
<td>Filler pruning</td>
<td>458.04</td>
<td>6.79M</td>
<td>29.46 / 0.8837 / 0.1686</td>
</tr>
<tr>
<td>One-shot (2:4)</td>
<td>Filler pruning</td>
<td>230.84</td>
<td>3.40M</td>
<td>29.55 / 0.8849 / 0.1662</td>
</tr>
<tr>
<td>One-shot (8:32)</td>
<td>Filler pruning</td>
<td>115.42</td>
<td>1.70M</td>
<td>28.79 / 0.8692 / 0.1893</td>
</tr>
<tr>
<td>SR-STE (8:32)</td>
<td>Filler pruning</td>
<td>117.24</td>
<td>1.70M</td>
<td>29.19 / 0.8771 / 0.1795</td>
</tr>
<tr>
<td>SLS (Ours)</td>
<td>Unpruned</td>
<td>116.64</td>
<td>1.55M</td>
<td>29.37 / 0.8811 / 0.1740</td>
</tr>
<tr>
<td>Filler pruning</td>
<td>One-shot (4:32)</td>
<td>65.27</td>
<td>0.86M</td>
<td>28.31 / 0.8570 / 0.2070</td>
</tr>
<tr>
<td>One-shot (4:32)</td>
<td>Unpruned</td>
<td>64.44</td>
<td>0.85M</td>
<td>28.64 / 0.8664 / 0.1985</td>
</tr>
<tr>
<td>SR-STE (4:32)</td>
<td>Unpruned</td>
<td>64.44</td>
<td>0.85M</td>
<td>28.33 / 0.8571 / 0.2070</td>
</tr>
<tr>
<td>SLS (Ours)</td>
<td>One-shot (4:32)</td>
<td>60.31</td>
<td>0.79K</td>
<td>28.98 / 0.8726 / 0.1870</td>
</tr>
<tr>
<td>Filler pruning</td>
<td>One-shot (4:32)</td>
<td>29.31</td>
<td>0.42K</td>
<td>27.65 / 0.8391 / 0.2325</td>
</tr>
<tr>
<td>One-shot (2:32)</td>
<td>SR-STE (4:32)</td>
<td>32.04</td>
<td>0.42K</td>
<td>27.77 / 0.8431 / 0.2325</td>
</tr>
<tr>
<td>SLS (Ours)</td>
<td>Unpruned</td>
<td>30.91</td>
<td>397.55K</td>
<td>28.36 / 0.8573 / 0.2117</td>
</tr>
</tbody>
</table>

4.4. Qualitative Comparison

We present the qualitative results in Figure 5. In the case of image super-resolution tasks (the first two rows), we observe that the models trained by SLS can restore more sharp and clear textures, compared to the results from the other methods. Notably, the results from the filter pruning suffer checkerboard artifacts since pruning the last pixel shuffle upscaling layer [47] results in sparse pixel values. Similarly, the results on image deblurring tasks (the last two rows) show that the models trained by SLS can restore the detailed textures and cleaner car plates with better readability while other pruning methods fail to reconstruct such high-frequency details. The overall results show that under the same computational budgets, models pruned by SLS achieve perceptually satisfying performance.

4.5. Finding Optimal Pruning Ratio for Each Layer

Different from the previous methods that set a uniform N:M sparsity in all layers, our SLS finds the sparsity level for each layer via learning. In Figure 6, we visualize the searched level of sparsity of each layer. The results show that the searched N have a tendency to decrease when the computational costs of the corresponding layer become heavy. Interestingly, we found that this tendency is not shown in the last upscaling layer in RFDN (indicated as the 50-th layer in the figures). We empirically found that the last upscaling layer has a significant impact on the
Figure 5. Qualitative comparisons with the existing pruning methods. The first two rows show image super-resolution results from RFDN with the scaling factor of 4. The last two rows show image deblurring results from DMPHN. For each task, all pruned models have almost the same computational costs (1/16 of the original value) with respect to GMACs.

Table 3. Ablation studies on the proposed methods. In the 3rd column, the two numbers indicate the update periods (iterations) for updating \( \hat{W}_i \) in RFDN and UNet, respectively. For RFDN, we use Urban100 dataset. We set the target budget as \( \frac{1}{8} \) of the original computational costs.

<table>
<thead>
<tr>
<th>POP ( \lambda_{reg} ) annealing</th>
<th>Update period (Iterations)</th>
<th>PSNR_\text{r}</th>
<th>RFDN</th>
<th>UNet</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \chi ) ( \checkmark )</td>
<td>1000/131</td>
<td>25.46</td>
<td>28.57</td>
<td></td>
</tr>
<tr>
<td>( \checkmark ) ( \times )</td>
<td>1000/131</td>
<td>25.59</td>
<td>28.84</td>
<td></td>
</tr>
<tr>
<td>( \checkmark ) ( \checkmark )</td>
<td>No update</td>
<td>25.46</td>
<td>28.74</td>
<td></td>
</tr>
<tr>
<td>( \checkmark ) ( \checkmark )</td>
<td>1/1</td>
<td>25.61</td>
<td>28.88</td>
<td></td>
</tr>
<tr>
<td>( \checkmark ) ( \checkmark )</td>
<td>10000/1310</td>
<td>25.61</td>
<td>28.92</td>
<td></td>
</tr>
<tr>
<td>( \checkmark ) ( \checkmark )</td>
<td>1000/131</td>
<td>25.63</td>
<td>28.98</td>
<td></td>
</tr>
</tbody>
</table>

4.6. Ablation Studies

To validate the effectiveness of each component in SLS, we conduct ablation studies and the results are shown in Table 3. To investigate the effect of Priority-Ordered Pruning (POP), we defined the priority score values \( p \) not as the cumulative product of auxiliary trainable parameters as in Equation (4) but as independent trainable parameters. The performance since it is directly related to the final output. Thus, considering the better restoration performance in Table 2, these results demonstrate that SLS finds more effective pruning ratios for each layer by taking into account both the computational costs and the contribution to the performance of each layer.
analysis on the searched layer-wise $N:M$ sparsity. The results are obtained by training RFDN model with $C_{\text{target}} = \frac{1}{5}C_{\text{original}}$. We visualize the layer-wise pruning ratios in terms of (a) GMACs and (b) $N$. The layer is sorted with respect to MACs before pruning. The highlighted bar by the red arrow indicates the result of the last upsampling layer in RFDN. The brown line in (b) is for the comparison with the uniform pruning [37, 64].

Figure 6. Analysis on the searched layer-wise $N:M$ sparsity. The results are obtained by training RFDN model with $C_{\text{target}} = \frac{1}{5}C_{\text{original}}$. We visualize the layer-wise pruning ratios in terms of (a) GMACs and (b) $N$. The layer is sorted with respect to MACs before pruning. The highlighted bar by the red arrow indicates the result of the last upsampling layer in RFDN. The brown line in (b) is for the comparison with the uniform pruning [37, 64].

Figure 7. Results of the proposed adaptive inference method on Urban100 dataset with the scale factor of 4. We use the bicubic upsampler and three CARN models trained by SLS with different target budgets.

results show that there is a large performance loss when POP is not used, indicating that aligning the two importance measures (the magnitude of weights and the priority scores) is essential. For the ablation study on $\lambda_{\text{reg}}$ annealing, we set $\lambda_{\text{reg}}$ as a large value that is finally found when the annealing strategy is used and train the models with it. The results demonstrate that $\lambda_{\text{reg}}$ annealing brings an additional performance gain by controlling the speed of pruning process according to the current pruned rate. In Equation (1), we group the weight tensor into several sparse tensors by using the magnitude of weights. Since the weights change during training, we should update the sparse tensors, $\hat{W}_i$. To find an appropriate update period, we train models with different update periods. As expected, the pruned models suffer significant performance degradation when there is no update. Also, updating $\hat{W}_i$ at every iteration is not helpful, so we set the update period as 10000 and 1310 iterations (10 epochs) for super-resolution and deburring, respectively.

4.7. Adaptive Inference

Figure 7 shows the results of the proposed adaptive inference method on image super-resolution tasks. An input image is divided into several patches and each patch is restored by the selected model by Equation (8). As shown in the figure, our adaptive inference scheme not only enables the detailed control of computational budgets but also improves the trade-off between the computational costs and the restoration performance in terms of PSNR. For the experimental details and the results on deblurring tasks, please refer to the supplementary material.

5. Conclusion

In this paper, we propose a novel layer-wise pruning ratio search framework, SLS, tailored for $N:M$ sparsity. Our differentiable learning framework is trained end-to-end with the task-specific and the computational regularization loss to determine a more effective degree of sparsity for each layer. Compared with the previous methods with uniform $N:M$ sparsity at all layers, our results achieve state-of-the-art image restoration performance at similar computational budgets. Furthermore, our adaptive inference scheme facilitates the detailed control of the computational budgets with improved restoration performance.

Acknowledgment. This work was supported in part by IITP grant funded by the Korea government (MSIT) [No. 2021-0-01343, Artificial Intelligence Graduate School Program (Seoul National University), and No. 2021-0-02068, Artificial Intelligence Innovation Hub], and in part by AIRS Company in Hyundai Motor Company & Kia Motors Corporation through HMC/KIA-SNU AI Consortium.
References


[3] Xiangxiang Chu, Bo Zhang, Hailong Ma, Ruijun Xu, Jixiang Li, and Qingyuan Li. Fast, accurate and lightweight super-resolution with neural architecture search. In ICPR, 2020. 2


[31] Shaohui Lin, Rongrong Ji, Chengjian Yan, Baochang Zhang, Liujuan Cao, Qixiang Ye, Feiyue Huang, and David Doermann. Towards optimal structured cnn pruning via generative adversarial learning. In CVPR, 2019. 2, 3


