Towards Fully Sparse Training: Information Restoration with Spatial Similarity

Weixiang Xu[1,2], **Xiangyu He**[1,2], **Ke Cheng**[1,2], **Peisong Wang**[1], **Jian Cheng**[1*]

[1]NLPR, Institute of Automation, Chinese Academy of Sciences
[2]School of Artificial Intelligence, University of Chinese Academy of Sciences
{xuweixiang2018,chengke2017}@ia.ac.cn, {xiangyu.he, peisong.wang, jcheng}@nlpr.ia.ac.cn

Abstract

The 2:4 structured sparsity pattern released by NVIDIA Ampere architecture, requiring four consecutive values containing at least two zeros, enables doubling math throughput for matrix multiplications. Recent works mainly focus on inference speedup via 2:4 sparsity while training acceleration has been largely overwhelmed where backpropagation consumes around 70% of the training time. However, unlike inference, training speedup with structured pruning is nontrivial due to the need to maintain the fidelity of gradients and reduce the additional overhead of performing 2:4 sparsity online. For the first time, this article proposes fully sparse training (FST) where 'fully' indicates that ALL matrix multiplications in forward/backward propagation are structurally pruned while maintaining accuracy. To this end, we begin with saliency analysis, investigating the sensitivity of different sparse objects to structured pruning. Based on the observation of spatial similarity among activations, we propose pruning activations with fixed 2:4 masks. Moreover, an Information Restoration block is proposed to retrieve the lost information, which can be implemented by efficient gradient-shift operation. Evaluation of accuracy and efficiency shows that we can achieve 2× training acceleration with negligible accuracy degradation on challenging large-scale classification and detection tasks.

Introduction

Network pruning (Han et al. 2015; Wen et al. 2016; He et al. 2019b) has drawn significant attention of researchers to alleviate the high computational complexity and intensive memory footprint in convolutional neural networks. Recent Ampere architecture, released by NVIDIA and equipped with Sparse Tensor Cores (NVIDIA 2020b), is the first commercial GPU architecture that supports fine-grained structured sparsity. Concretely, matrix-multiply operations can achieve 2× acceleration when one of the two arguments in GEMM meets the 2:4 structured sparsity requirement, providing a possibility for training/inference speedup. Here, the 2:4 sparsity means that each chunk of four adjacent elements in matrix rows has at least two zeros.

Attracted by the new characteristic, recent works focus on utilizing it to accelerate *inference*. For example, ASP (Mishra et al. 2021) obtains 2:4 pruned models by initializing from dense pre-trained models and training with a fixed mask. Independent of computationally expensive pre-training, SR-STE (Zhou et al. 2021) further proposes training from scratch with dynamic masks. However, the power of 2:4 sparsity in accelerating *training* has not been explored.

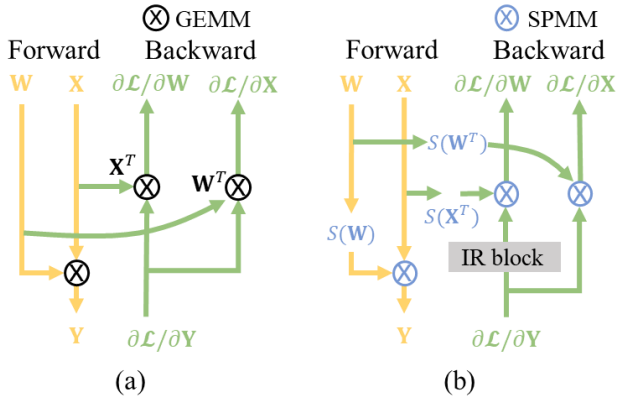

Figure 1: Computation Graph for (a) conventional training and (b) fully sparse training (FST). $S(\cdot)$ denotes the 2:4 structured pruning. GEMM/SPMM represents general/structured sparse matrix multiplications. IR indicates our information restoration block.

Unlike inference time acceleration, training with structured pruning is more challenging. First, the backpropagation dominates training costs. As shown in Figure 2, the backward procedure occupies up to 80% of the total calculation burden in SR-STE, to which previous inference speedup methods can not directly apply. Second, due to the accumulation in gradient descent optimization, the backpropagation is more sensitive to perturbation than inference (Zhou et al. 2016). As a result, the compression in backpropagation jeopardizes the optimization stability, causing severe accuracy degradation.

Moreover, unstructured pruning methods on gradients proposed by previous works (Sun et al. 2017; Goli and Aamodt 2020; Ye et al. 2020; Chmiel et al. 2021; M Abdelmoniem et al. 2021) are no longer applicable to structured cases for two reasons: 1) 2:4 structured sparsity en-

*Corresponding Author.

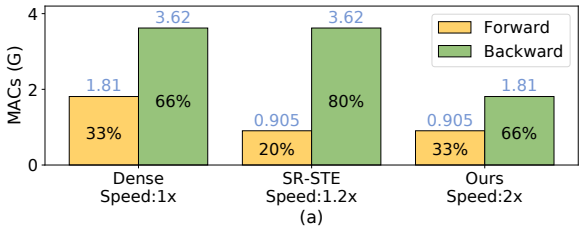

Figure 2: MACs for convolution layers with ResNet-18.

forces a stricter constraint than unstructured sparsity, which inevitably leads to misleading gradients during the backward pass, resulting in poor performance or even non-convergence (details in Table 1). 2) The extra time overhead introduced by pruning may be disastrous because the gradients involved in backpropagation are computed online (details in Table 2).

To investigate the above problem, we start with pruning saliency analysis and find activations $\mathbf{X}$ are relatively more robust to structured pruning than gradients $\frac{\partial \mathcal{L}}{\partial \mathbf{Y}}$. In addition to the sensitivity, the extra overhead introduced should also be noticed. Compared with network parameters, the dimension of intermediate activations is extremely vast. Thus to avoid the time-consuming sorting operation on $\mathbf{X}$, we propose 2:4 sparsity with a fixed mask based on the observation of spatial similarity in activations.

Moreover, we further propose an Information Restoration (IR) block to retrieve the lost information caused by fixed-position pruning, which can be implemented by the computationally efficient gradient-shift operation in practice.

Together, these contributions enable our FST framework with 2:4 structured pruning, as shown in Figure 1. To the best of our knowledge, it is the first systematic framework that structurally pruning all three multiplications (Eq.1-3) in both forward and backward pass. Evaluations show that about 2× training acceleration can be obtained on various tasks with almost no performance degradation.

Related Work

Various methods have been proposed to compress or accelerate deep networks, such as low-bit quantization (Rastegari et al. 2016; Wang et al. 2020; He et al. 2020; Xu et al. 2020; Chen et al. 2021), knowledge distillation (Gou et al. 2021), efficient convolution (Lavin and Gray 2016; Zhao et al. 2021) and specific hardware design (Chen et al. 2016; Li et al. 2020, 2021).

Network pruning, removing unimportant parameters, also attracts much attention. According to the procedure of obtaining a sparse model, pruning methods can be categorized into '**scratch-dense-sparse**' and '**scratch-sparse**'. The former depends on a well-trained dense model and needs further retraining (Han et al. 2015; He et al. 2019b; Mishra et al. 2021), which is not the scope of this work. The latter, training sparse models from scratch, saves training time but at the cost of performance. Based on the sparse object, 'scratch-sparse' can be further divided as follows.

Pruning Weight from Scratch Various influential works have emerged to explore dynamic sparsity training from scratch, which enable sparse weight changes during training. DeepR (Bellec et al. 2018) applies random walk in parameter space, and weights are determined to be pruned or reactivated during training. SET(Mocanu et al. 2018) simplifies the pruning-regrowing process based on weight magnitude. The prune-redistribute-regrowth cycle is adopted in (Dettmers and Zettlemoyer 2019; Mostafa and Wang 2019; Evci et al. 2020) to change weights according to different criteria dynamically. Instead of hand-crafted redistribute-regrowth cycles, recent DMPF (Lin et al. 2020) uses error compensation mechanisms. DST (Liu et al. 2020) directly trains sparse models with learnable masks, where non-differentiable step function is replaced with derivable approximation. Despite the high compression rate, all these methods perform sparsity in an unstructured way, which is hardware-unfriendly and can not be accelerated by commercial GPUs.

Pruning Activation Gradient Some other works focus on pruning activation gradients to speed up the communication stage of distributed training. meProp (Sun et al. 2017) and ReSprop (Goli and Aamodt 2020) apply unstructured pruning on activation gradients by keeping the top-k elements and updating only the corresponding small portion of parameters. Recent works (Ye et al. 2020; Chmiel et al. 2021; M Abdelmoniem et al. 2021) calculate the analytical result of pruning threshold for unstructured-sparsity gradients by assuming the gradients obey normal or log-normal distribution. However, all these methods apply unstructured pruning, which reduces communication bandwidth pressure but can not accelerate matrix multiplication in practice. Furthermore, we will show that activation gradients are fragile in sparsity, and applying structured pruning on gradients will lead to severe optimization collapse.

Problem Formulation

We define a convolutional layer as a tuple $\langle \mathbf{W}, \mathbf{X} \rangle$. $\mathbf{W} \in \mathbb{R}^{C_{out} \times [C_{in} \cdot k \cdot k]}$, where (C_{out}, C_{in}, k) refer to output channels, i.e., filter number, input channels and kernel size, respectively. $\mathbf{X} \in \mathbb{R}^{[C_{in} \cdot k \cdot k] \times [H \cdot W]}$, where (H, W) are height and width of output feature maps. There are three major matrix multiplications for each layer during training: forward propagation, activation gradient backpropagation and weight gradient computation.

Forward propagation

$$\mathbf{Y} = \mathbf{W}\mathbf{X} \tag{1}$$

Backward propagation

$$\frac{\partial \mathcal{L}}{\partial \mathbf{X}} = \mathbf{W}^T \frac{\partial \mathcal{L}}{\partial \mathbf{Y}} \tag{2}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}} = \frac{\partial \mathcal{L}}{\partial \mathbf{Y}} \mathbf{X}^T \tag{3}$$

where $\mathbf{Y} \in \mathbb{R}^{C_{out} \times [H \cdot W]}$ represents the output feature map. $\frac{\partial \mathcal{L}}{\partial \mathbf{Y}} \in \mathbb{R}^{C_{out} \times [H \cdot W]}$ denotes the activation gradient, which involves in the calculation of gradients of input $\frac{\partial \mathcal{L}}{\partial \mathbf{X}}$ and gradients of weight $\frac{\partial \mathcal{L}}{\partial \mathbf{W}}$. Then $\frac{\partial \mathcal{L}}{\partial \mathbf{X}}$ will be backpropagated layer by layer, while the weight gradients will participate in

weight updating as $\mathbf{W} = \mathbf{W} - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{W}}$, where η represents the learning rate.

The multiplications between large matrixes with enormous computational complexity account for up to 90% of training time (Johnson 2016; Sun et al. 2017). Among them, backpropagation consumes around 70% of the training time (Goli and Aamodt 2020). The computational burden motivates us to design a fully sparse training framework with the help of A100's 2:4 sparse property, which speeds up not only the forward propagation, but also the two multiplications in the backward pass. Therefore, our target is to speed up training 2:4 sparsity neural networks by sparsifying Eq.1-3 while maintaining accuracy.

Analysis of Pruning Saliency

Sparse Tensor Core introduced in NVIDIA Ampere GPU architecture enables a $2\times$ acceleration of regular matrix multiplications when one of the two matrixes is 2:4 sparsified. Therefore, we first need to decide which matrices in Eq.1-3 to be 2:4 structurally pruned.

Since $\mathbf{X}$ commonly has a much larger dimension than parameter matrix $\mathbf{W}$, it is intuitive to prune $\mathbf{W}$ in Eq. 1 due to the extra cost. However, when it comes to Eq. 2 and Eq. 3, deciding which matrix to prune is nontrivial because they may show different sensitivity against pruning, inevitably affecting the final performance. Exhaustively, according to different sparse objects, there are four pruning strategies as follows:

$$\begin{aligned}
&\text{WG:} \quad \frac{\widetilde{\partial \mathcal{L}}}{\partial \mathbf{X}} = \mathcal{S}(\mathbf{W}^T)\frac{\partial \mathcal{L}}{\partial \mathbf{Y}}, \frac{\widetilde{\partial \mathcal{L}}}{\partial \mathbf{W}} = \mathcal{S}(\frac{\partial \mathcal{L}}{\partial \mathbf{Y}})\mathbf{X}^T \\
&\text{WX:} \quad \frac{\widetilde{\partial \mathcal{L}}}{\partial \mathbf{X}} = \mathcal{S}(\mathbf{W}^T)\frac{\partial \mathcal{L}}{\partial \mathbf{Y}}, \frac{\widetilde{\partial \mathcal{L}}}{\partial \mathbf{W}} = \frac{\partial \mathcal{L}}{\partial \mathbf{Y}}\mathcal{S}(\mathbf{X}^T) \\
&\text{GG:} \quad \frac{\widetilde{\partial \mathcal{L}}}{\partial \mathbf{X}} = \mathbf{W}^T\mathcal{S}(\frac{\partial \mathcal{L}}{\partial \mathbf{Y}}), \frac{\widetilde{\partial \mathcal{L}}}{\partial \mathbf{W}} = \mathcal{S}(\frac{\partial \mathcal{L}}{\partial \mathbf{Y}})\mathbf{X}^T \\
&\text{GX:} \quad \frac{\widetilde{\partial \mathcal{L}}}{\partial \mathbf{X}} = \mathbf{W}^T\mathcal{S}(\frac{\partial \mathcal{L}}{\partial \mathbf{Y}}), \frac{\widetilde{\partial \mathcal{L}}}{\partial \mathbf{W}} = \frac{\partial \mathcal{L}}{\partial \mathbf{Y}}\mathcal{S}(\mathbf{X}^T)
\end{aligned} \tag{4}$$

where $\frac{\widetilde{\partial \mathcal{L}}}{\partial \mathbf{X}}$ and $\frac{\widetilde{\partial \mathcal{L}}}{\partial \mathbf{W}}$ represent approximated input activations and weight activations respectively, and $\mathcal{S}(\cdot)$ represents 2:4 sparsity operation.

Several studies have focused on pruning activation gradients in an unstructured way (Sun et al. 2017; Goli and Aamodt 2020; Ye et al. 2020; Chmiel et al. 2021; Wiedemann et al. 2020; M Abdelmoniem et al. 2021) and find that $\frac{\partial \mathcal{L}}{\partial \mathbf{Y}}$ are almost full of small values close to zero, indicating that the third 'GG' strategy can be the best choice. However, our further experiments and analysis find that is not the case.

The straightforward strategy for deciding which to prune is selecting the matrix with small saliency, i.e., the pruned matrix has the smallest impact on the final accuracy. In order to explore the pruning saliency with 2:4 structured sparsity pattern, we begin with comparative experiments with the four different strategies above. Table 1 shows the performance.

Surprisingly, the third 'GG' and fourth 'GX' strategy with sparse activation gradients performs the worst and even collapse. In contrast, the second 'WX' strategy with sparse weights and activations performs the best, indicating that $\frac{\partial \mathcal{L}}{\partial \mathbf{Y}}$ is more sensitive to structured sparsity than $\mathbf{W}$ and $\mathbf{X}$ in the process of backpropagation. A similar phenomenon exists in fixed bit quantization, where DoReFa-Net (Zhou et al. 2016) argues that gradients need more bit-width than weights and activations to compensate for the accuracy degradation. We conjecture that this is because gradients are propagated layer by layer during backpropagation, and sparse reconstruction error will accumulate across multiple layers.

Method	Top1(%)
Dense	71.2
WG	67.5
WX	69.9
GG&GX	0.1

Table 1: Top-1 accuracy with ResNet-18.

Sparsity	Time(s)
Dense	0.179
$\mathcal{S}(\mathbf{W})$	0.0017
$\mathcal{S}(\partial\mathcal{L}/\partial\mathbf{Y})$	**0.29**
$\mathcal{S}(\mathbf{X}^T)$	**0.48**

Table 2: Execution time on sorting-based sparsity.

Besides performance, Table 2 further shows the averaged execution time on pruning different objects in an iteration, where the time on $\mathcal{S}(\partial\mathcal{L}/\partial\mathbf{Y})$ and $\mathcal{S}(\mathbf{X}^T)$ is even larger than the original training because of the vast dimensions. Taking the second layer of ResNet-18 with batch size 256 as an example, the number of elements in $\mathbf{W}$ is $64\times64\times3\times3$, while this number for $\mathbf{X}$ is $256 \times (64 \times 3 \times 3) \times (56 \times 56)$, about $12000\times$ than the former.

Methodology

Based on the saliency analysis above, we use sparse weights for Eq. 2 and sparse activations for Eq. 3 as our benchmark method. In this section, we will present pruning $\mathbf{W}$ of forward propagation, pruning $\mathbf{W}^T$ of activation gradient backpropagation, and pruning $\mathbf{X}^T$ of weight gradient computation respectively.

Forward Propagation with Sparse W

We first introduce our approach on 2:4 sparsity for Eq. 1. Considering the vast dimension of $\mathbf{X}$, we prune the parameter matrix $\mathbf{W}$ to avoid a large extra burden. During the forward propagation, output features are calculated as

$$\mathbf{Y} = \mathcal{S}(\mathbf{W})\mathbf{X} \tag{5}$$

where $\mathcal{S}(\cdot)$ is implemented by conducting Top-2 in every group of four consecutive absolute values. As shown in Table 2, the extra time is negligible because of the relatively small dimensions (compared with $\mathbf{X}$) and parallel implementation in deep learning libraries. During backward pass, the network parameters are updated as

$$\mathbf{W} = \mathbf{W} - \eta\frac{\partial \mathcal{L}}{\partial \mathcal{S}(\mathbf{W})} - \lambda[\mathbf{W} - \mathcal{S}(\mathbf{W})] \tag{6}$$

Here, $\frac{\partial \mathcal{L}}{\partial \mathcal{S}(\mathbf{W})}$ is an approximation to $\frac{\partial \mathcal{L}}{\partial \mathbf{W}}$ according to STE (Bengio, Léonard, and Courville 2013). The second refined term $\mathbf{W} - \mathcal{S}(\mathbf{W})$ is a gradual decay on pruned weights to reduce their changes during training according to SR-STE (Zhou et al. 2021). Following SR-STE, we set the decay coefficient λ as 0.0002 in all our experiments.

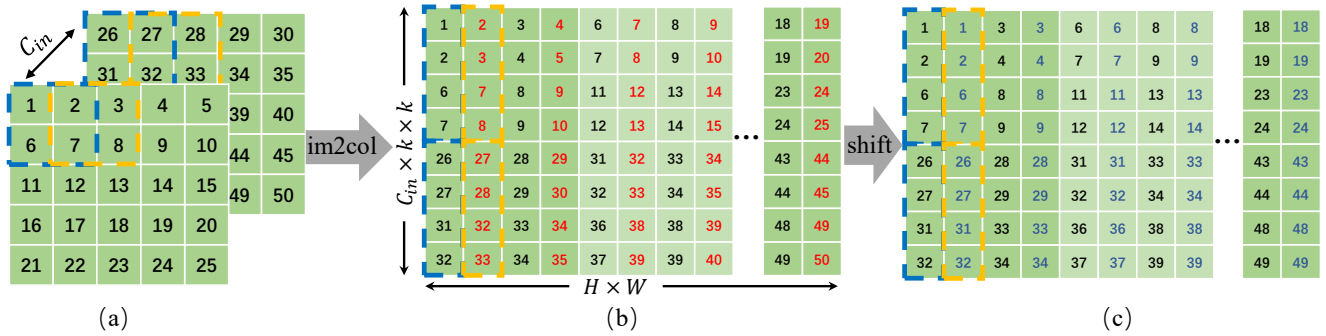

Figure 3: Pruning with fixed mask and retrieving lost information. The numbers indicates the location indexes. (a) A 2×2 convolutional kernel slides over a feature map of size [2,5,5]. (b) The operation *im2col* unfolds the original feature and stacks it as a 2D matrix $\mathbf{X}$ in certain layout order. Elements on two adjacent columns correspond to neighboring pixels on the original input channel. The second (red indexes) of the two adjacent columns is pruned fixedly. (c) Complementing the pruned columns with the corresponding adjacent columns (blue indexes), which is implemented by gradient-shift as Eq. 10 in practice.

Input Gradient Calculation with Sparse $\mathbf{W}^T$

Though $\mathbf{W}$ has been 2:4 sparsified for $\mathbf{WX}$, it can not be utilized to speed up $\mathbf{W}^T \frac{\partial \mathcal{L}}{\partial \mathbf{Y}}$ because of the transposition. The recent work (Hubara et al. 2021) presents Transposable Sparse Masks (TSM), requiring $\mathbf{W}$ and $\mathbf{W}^T$ to satisfy 2:4 sparsity pattern *simultaneously*. We show that TSM is suboptimal in two aspects: 1) The simultaneous sparse requirement is a strong constraint, destructing the sparsity diversity. For weights with n 4×4 patches, the number of all candidate configurations with 2:4 sparsity is 1296^n, but the number for TSM degenerates to 90^n, which weakens model capacity. 2) It is not easy to find the optimal transposable weights, and TSM formulates it as a min-cost flow problem. Due to the huge time complexity $O(n^3)$, TSM updates it every 40 iterations in a greedy manner. Both of the above reasons lead to suboptimal performance.

In this work, we consider the time overhead and the effect of sparsity diversity on model capacity. Instead of *simultaneously* forcing both $\mathbf{W}$ and $\mathbf{W}^T$ to satisfy 2:4 sparsity pattern like TSM, we *separately* sparsify $\mathbf{W}^T$ during backward. Experiments show that the simple separate sparsity performs better than simultaneous sparsity in terms of both time overhead and accuracy.

Weight Gradient Calculation with Sparse $\mathbf{X}^T$

Although activation is less sensitive to structured sparsity than gradient, the tradeoff between time cost and accuracy is still challenging. Since our target is training speedup, the time overhead of performing 2:4 sparsity should be negligible. Therefore, conducting 2:4 structured pruning by sorting every four continuous elements is no longer suitable for activations. As shown in Table 2, performing sorting-based sparsity on $\mathbf{X}$ takes even longer time than the training procedure, which is disastrous and unacceptable.

Observation Our method is based on the observation that neighboring activations in the convolution tend to be close in value. Specifically, it is obvious that adjacent pixels share close values for natural images, and we find that this property will be preserved throughout neural network layers. This is because that CNNs are established by convolutional layers that conduct in sliding window form, and the same small filter slides over the feature to produce the output channel. We will further explore this phenomenon from qualitative and quantitative aspects.

Before performing the general matrix multiplications (GEMM), *im2col* is first applied to the 3D input, turning it into the 2D domain (Chellapilla, Puri, and Simard 2006). Here pixels in the im2col domain are arranged according to a certain regularity. Figure 3 illustrates a toy example with a two-channel input, and the kernel size is 2×2. As can be seen in Figure 3(b), the elements of two adjacent columns (e.g. indexes $\{1, 2, 6, 7, 26, 27, 31, 32\}$ in the first column versus $\{2, 3, 7, 8, 27, 28, 32, 33\}$ in the second column) correspond to neighboring pixels on the original input channel in Figure 3(a), which share close values. The spatial similarity phenomenon and the layout regularity in the im2col domain motivate us to prune $\mathbf{X}^T$ at fixed positions.

Information Restoration Block To utilize Sparse Tensor Core on Ampere GPU, $\mathbf{X}^T$ should be 2:4 sparsified along the $H \cdot W$ direction. Instead of pruning by the time-consuming sorting, we prune $\mathbf{X}$ with a fixed mask by utilizing the correlation between adjacent columns. Concretely, we keep only one of the two adjacent columns in $\mathbf{X}$ while leaving the other column zeros. As shown in Figure 3(b), the indexes of pruned columns are shown in red. $\mathbf{X}^T$ is sparsified alternately from column to column, which can be expressed as

$$\mathcal{S}(\mathbf{X}^T)_{ij} = \begin{cases} \mathbf{X}^T_{ij} & i = 1, 3, 5 \cdots \\ 0 & else \end{cases} \tag{7}$$

Obviously, the pruning approach with the fixed mask pattern naturally satisfies the requirement of 2:4 sparsity in the $H{\cdot}W$ direction. We denote the pruned activation as:

$$\overline{\mathcal{S}}(\mathbf{X}^T) = \mathbf{X}^T - \mathcal{S}(\mathbf{X}^T) \tag{8}$$

which results in the weight gradient error, i.e., $\frac{\partial \mathcal{L}}{\partial \mathbf{W}} - \frac{\partial \widetilde{\mathcal{L}}}{\partial \mathbf{W}} = \frac{\partial \mathcal{L}}{\partial \mathbf{Y}}\mathbf{X}^T - \frac{\partial \mathcal{L}}{\partial \mathbf{Y}}\mathcal{S}(\mathbf{X}^T) = \frac{\partial \mathcal{L}}{\partial \mathbf{Y}}\overline{\mathcal{S}}(\mathbf{X}^T)$. Based on the observa-

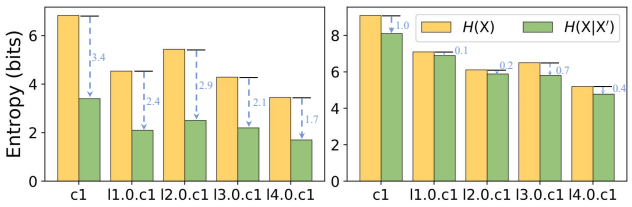

Figure 4: Feature entropy and conditional entropy from different layers. (a) Natural images from ImageNet as inputs. (b) White noise as inputs.

tion that neighboring elements share close values, we complement the pruned columns in $\mathcal{S}(\mathbf{X}^T)$ with the corresponding adjacent columns. As shown in Figure 3(c), the indexes of complements are shown in blue.

Due to the interleaved property between $\mathcal{S}(\mathbf{X}^T)$ and $\overline{\mathcal{S}}(\mathbf{X}^T)$, $\overline{\mathcal{S}}(\mathbf{X}^T)$ can be approximated by shifting the whole $\mathcal{S}(\mathbf{X}^T)$ *down* by one pixel (note here it is $\mathbf{X}^T$, while the one shown in Figure 3b is $\mathbf{X}$):

$$\overline{\mathcal{S}}(\mathbf{X}^T) \approx \boldsymbol{T}\mathcal{S}(\mathbf{X}^T) \tag{9}$$

where $\boldsymbol{T}$ is a transformation matrix obtained from the product by a series of elementary matrixes of size $HW \times HW$. And the physical meaning of pre-multiplying by $\boldsymbol{T}$ is equivalent to performing down-shift by one pixel. Following weight gradient calculation of Eq. 3, by filling up the pruned activations with Eq. 8, we have the following modified gradient calculation formulation,

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \mathbf{W}} &= \frac{\partial \mathcal{L}}{\partial \mathbf{Y}}(\mathcal{S}(\mathbf{X}^T) + \overline{\mathcal{S}}(\mathbf{X}^T)) \\ &\approx \frac{\partial \mathcal{L}}{\partial \mathbf{Y}}(\mathcal{S}(\mathbf{X}^T) + \boldsymbol{T}\mathcal{S}(\mathbf{X}^T)) \\ &= (\frac{\partial \mathcal{L}}{\partial \mathbf{Y}} + \frac{\partial \mathcal{L}}{\partial \mathbf{Y}}\boldsymbol{T})\mathcal{S}(\mathbf{X}^T) \end{aligned} \tag{10}$$

Here the activation gradient $\frac{\partial \mathcal{L}}{\partial \mathbf{Y}}$ is post-multiplied by the corresponding transformation matrix $\boldsymbol{T}$, which is equivalent to shifting $\frac{\partial \mathcal{L}}{\partial \mathbf{Y}}$ *left* by one element. Details can be found in Appendix. Therefore, in Eq. 10, the activation gradient $\frac{\partial \mathcal{L}}{\partial \mathbf{Y}}$ is first shifted and then added to the original one, and the new gradient is then multiplied by the 2:4 sparse activation, which can be accelerated on Ampere architecture. We call the process as Information Restoration, as shown in Figure 1(b). We will show that the shift-based implementation is hardware-friendly and time-efficient in experiment section.

Exploring Spatial Similarity

In the above section, we alternately prune $\mathbf{X}$ by columns and complement the pruned columns with neighboring pixels, based on the observation that adjacent pixels share close values even for deeper layers. In this section, we further demonstrate it in terms of qualitative and quantitative studies.

Spatial Entropy of Input Features

In order to quantitatively evaluate the property that adjacent elements of input features share close value, we resort to the well-known information entropy, which is usually a measure of how much total information the message contains. The metric used to measure the amount of information contained in an image is the so-called image entropy, which is originally introduced by (Frieden 1972) in the context of image reconstruction. An image $\mathbf{X}$ is regarded as a set of positive numbers $x_1, x_2, ..., x_N$ to be determined, and on which the image entropy is defined as $H(\mathbf{X}) = -\sum_{i=1}^{N} p(x_i) \log p(x_i)$.

Based on the adjacent similarity hypothesis, an intuition is that if the information in the adjacent columns $\mathbf{X}'$ has been known, the new information amount that the original feature $\mathbf{X}$ contains will be less. Namely, the uncertainty will decrease. A counterexample is the white noise image, for which the uncertainty of it does not decrease even if its adjacent columns are known. Based on this intuition, we introduce conditional entropy as

$$H(\mathbf{X}|\mathbf{X}') = \sum_{i=1}^{N} p(x_i') H(\mathbf{X}|\mathbf{X}' = x_i') \tag{11}$$

which quantifies the information amount of $\mathbf{X}$ given that its neighboring $\mathbf{X}'$ is known. We use $H(\mathbf{X}) - H(\mathbf{X}|\mathbf{X}')$ to denote the 'similarity' between adjacent columns in features. Figure 4(a) details the results on five layers in ResNet-18, averaged over 1000 images. Concretely, we uniformly quantize the continuous intermediate features with 8-bit (256 discrete intervals), such that we can obtain a histogram of feature distribution. It can be seen that the similarity property is kept throughout networks, even for deep layers such as 'layer4.0.conv1'. For comparison with unnatural images, we use randomly generated white noise as network inputs. As shown in Figure 4(b), since the random noise image does not have the spatial similarity as natural images, the decline of uncertainty in $\mathbf{X}$ is slight.

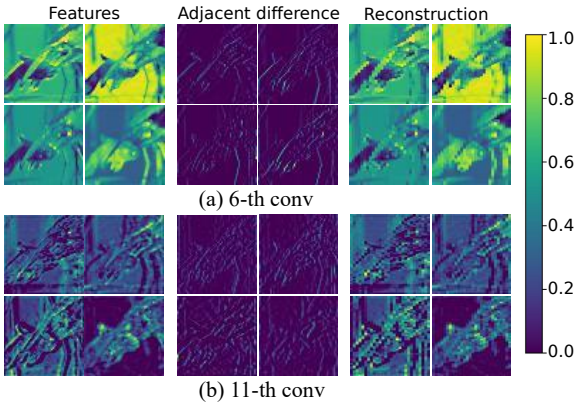

Figure 5: Intermediate features $\mathbf{X}$, adjacent difference maps $\mathbf{X}^{ad}$ and reconstruction features $\mathbf{X}^r$ from different layers of ResNet-18. Most of the values in $\mathbf{X}^{ad}$ are near 0, indicating that neighboring elements in $\mathbf{X}$ share close values.

Neighboring Difference Visualization

To further demonstrate the spatial similarity, we visualize the intermediate input features $\mathbf{X}$ and their corresponding

adjacent difference maps $\mathbf{X}^{ad}$ of ResNet-18 in Figure 5. The adjacent difference map indicates the difference between two adjacent columns, and its i-th column is obtained by $\mathbf{X}^{ad}_{:,i} = \mathbf{X}_{:,i} - \mathbf{X}_{:,i+1}$. The reconstruction features $\mathbf{X}^{r}_{:,i+1} = \mathbf{X}^{r}_{:,i}$ $(i = 0, 2, 4...)$ is also visualized to demonstrate the approximation degree of Eq. 9. All values are normalized into $[0, 1]$ for comparison. As can be seen, except for edges in the original features, values at most positions are close to zero, which is also the reason that our approximation in Eq. 10 can achieve better performance.

Experiments

In this section, we evaluate the proposed FST in terms of accuracy and efficiency. Our experiments are conducted on image classification and object detection.

Implementation Details

Different from SR-STE (Zhou et al. 2021), which implements 2:4 structured sparsity pattern by directly reshaping the 4-D weight (C_{out}, C_{in}, k, k) into a 2-D matrix $(\lfloor (C_{out} \cdot C_{in} \cdot k \cdot k)/4 \rceil, 4)$ and keeping Top-2 elements along the second dimension, we reshape weight as $(C_{out}, (C_{in} \cdot k \cdot k)/4, 4)$ and conduct 2:4 pruning on the third dimension, such that the layout of sparsity pattern matches the arrangement of im2col when performing convolution as Eq. 1. When $C_{in} \cdot k \cdot k$ is not divisible by 4 (only happens for the first convolution in ResNet), we make it a multiple of 4 by zero padding.

Image Classification

To verify the effectiveness of our method, we first evaluate it on the large-scale ImageNet.We follow hyperparameter settings as (Zhou et al. 2021): all models are trained with batch size 256 for 120 epochs, and learning rate is annealed from 0.1 to 0 with a cosine scheduler. In order to reproduce their reported accuracy, we set weight decay as 7e-5 and use label smooth. The ColorAugmentation released in their implementation is not adopted in our experiments because we find that it will actually degrade accuracy slightly.

ASFP takes channel pruning on weights and allows the pruned filters to be updated during the training procedure. ASP (Mishra et al. 2021) utilizes the 'scratch-dense-sparse' procedure and relies on the pre-trained dense models. Note that TSM (Hubara et al. 2021) utilizes 4:8 rather than 2:4 sparsity pattern, which also has a 50% compression rate. 4:8 structured sparsity relaxes the constraints on sparse patterns, which improves accuracy but at the cost of increasing the hardware overhead. Moreover, Ampere's Sparse Tensor Core does not support 4:8 sparsity pattern because it only provides 4×4 matrix processing array (NVIDIA 2020a). Therefore, we only consider 2:4 pruning in this work.

Table 3 shows the results compared with other state-of-the-art methods. With negligible accuracy degradation (0.2% on ResNet-18 compared with conventional dense training), our FST can achieve nearly 2× actual acceleration on the newest NVIDIA GPUs. Although a higher compression rate can be achieved, the unstructured sparsity pattern (Sun et al. 2017; Goli and Aamodt 2020; Lin et al. 2020; Evci et al. 2020) cannot be directly employed to commercial GPUs due to lacking CUDA instructions support (Han et al. 2016, 2017; Mishra et al. 2021). In contrast, structured pruning (He et al. 2019a), also known as channel pruning, can be directly accelerated without dedicated hardware, but at the expense of a $\sim 3\%$ accuracy drop when structurally pruning 30% weights.

Dataset	Model	Sparsity	mAP(%)	Speedup
VOC	VGG16-SSD	Dense	75.8	1×
		FST	76.2	2×
	R50 Faster-RCNN	Dense	79.5	1×
		FST	79.4	2×
COCO	VGG16-SSD	Dense	25.5	1×
		FST	25.5	2×
	R50 Faster-RCNN	Dense	37.4	1×
		FST	37.2	2×

Table 4: Object detection results (bounding box AP) on PASCAL VOC and COCO2017.

Object Detection

To further evaluate the effectiveness, we conduct experiments on object detection task. The PASCAL VOC dataset contains around 16k training images with 20 different classes, while the COCO dataset consists of about 80k training images from 80 different categories. The single-stage object detector SSD (Liu et al. 2016), and the two-stage Faster RCNN (Ren et al. 2015) are evaluated with different backbones. All experiments on object detection tasks are conducted with MMDetection (Chen et al. 2019), and the mean average precision (mAP) is used as the evaluation criterion. The results are shown in Table 4. We can observe that, on both one/two-stage detection algorithms, our approach can fully 2:4 sparsify the forward and backward pass with less than 0.2% accuracy drop, which demonstrates the generalization ability of our method.

Ablation Study

In this section, we perform ablation studies to clearly show the influence of pruning different parts $\mathcal{S}(\mathrm{W})$, $\mathcal{S}(\mathrm{W}^T)$ and $\mathcal{S}(\mathrm{X}^T)$, and the effectiveness of our Information Restoration block on retrieving performance. Among them, the sparsity on weights, namely $\mathcal{S}(\mathrm{W})$ and $\mathcal{S}(\mathrm{W}^T)$ is conducted by setting the two minimum elements in every four consecutive elements as zeros, while the sparsity on activations $\mathcal{S}(\mathrm{X}^T)$ is performed by pruning columns at fixed positions, avoiding the sizeable time overhead.

We take vanilla ResNet-18 on ImageNet as our baseline and design five ablation experiments. As shown in Table 5, it is worth noticing that: 1) In the third line, pruning weights in both forward and backward passes does not degenerate the accuracy and even makes a small improvement because of regularization introduced by sparsity. 2) In the fourth line, the activations are further pruned at fixed positions as Eq. 7. Compared with magnitude-based pruning, fixed-position-based pruning inevitably cast away more critical elements, significantly impacting the training performance because of the misleading weight updating directions

Model	Method	Pattern	Ratio	Eq. 1	Eq. 2	Eq. 3	Top-1(%)	Scratch	AS	Speedup
ResNet-18	Baseline	Dense	0	–	–	–	71.2	✓	✗	1×
	meProp (Sun et al. 2017)	Unstructured	50%×2	–	$\partial\mathcal{L}/\partial \mathrm{Y}$	$\partial\mathcal{L}/\partial \mathrm{Y}$	32.5	✓	✗	–
	ReS (Goli and Aamodt 2020)	Unstructured	50%×2	–	$\partial\mathcal{L}/\partial \mathrm{Y}$	$\partial\mathcal{L}/\partial \mathrm{Y}$	69.8	✓	✗	–
	ASFP (He et al. 2019a)	Channel	30%×1	W	–	–	68.0	✗	✓	1.1×
	ASP (Mishra et al. 2021)	2:4 structured	50%×1	W	–	–	70.7	✗	✗	–
	SR-STE (Zhou et al. 2021)	2:4 structured	50%×1	W	–	–	71.2	✓	✓	1.2×
	TSM (Hubara et al. 2021)	2:4 structured	50%×2	W	W^T	–	70.7	✓	✓	1.5×
	FST	2:4 structured	50%×2	W	W^T	–	**71.3**	✓	✓	1.5×
	FST	2:4 structured	50%×3	W	W^T	X^T	71.0	✓	✓	**2**×
ResNet-50	Baseline	Dense	0	–	–	–	77.3	✓	✗	1×
	DPF (Lin et al. 2020)*	Unstructured	50%×1	W	–	–	76.5	✓	✗	–
	RigL (Evci et al. 2020)*	Unstructured	50%×1	W	–	–	76.2	✓	✗	–
	ASFP (He et al. 2019a)	Channel	30%×1	W	–	–	75.5	✗	✓	1.1×
	PruneT (Lym et al. 2019)	Channel	-	W	W^T	–	75.0	✓	✓	1.5×
	ClickT (Zhang et al. 2021)	Predefined	-	W	W^T	–	76.2	✓	✓	1.6×
	SparseT (Gong et al. 2020)	Unstructured	-	X	$\partial\mathcal{L}/\partial \mathrm{Y}$	X^T	NA	✓	✓	1.3×
	ASP (Mishra et al. 2021)	2:4 structured	50%×1	W	–	–	76.8	✗	✗	–
	SR-STE (Zhou et al. 2021)	2:4 structured	50%×1	W	–	–	77.0	✓	✓	1.2×
	TSM (Hubara et al. 2021)	2:4 structured	50%×2	W	W^T	–	76.8	✓	✓	1.5×
	FST	2:4 structured	50%×2	W	W^T	–	**77.1**	✓	✓	1.5×
	FST	2:4 structured	50%×3	W	W^T	X^T	77.0	✓	✓	**2**×

Table 3: Performance of ResNet-18/50 from scratch on ImageNet. The sparsity ratio (how much being pruned) and sparse objects in Eq.1-3 are reported. '×n' represents n objects are pruned. 'AS' indicates actual speedup on commodity GPU instead of on dedicated hardware. * indicates that results are reproduced with their released code by setting target sparse ratio as 50%.

$\widetilde{\frac{\partial\mathcal{L}}{\partial \mathbf{W}}} = \frac{\partial\mathcal{L}}{\partial \mathbf{Y}}\mathcal{S}(\mathbf{X}^T)$. 3) In the fifth line, the IR block remedies the accuracy degeneration, indicating that neighboring pixels help retrieve pruned information in $\mathbf{X}$.

	$\mathcal{S}(\mathrm{W})$	$\mathcal{S}(\mathrm{W}^T)$	$\mathcal{S}(\mathrm{X}^T)$	IR	Top-1(%)
1	✗	✗	✗	✗	71.2
2	✓	✗	✗	✗	71.2
3	✓	✓	✗	✗	71.3
4	✓	✓	✓	✗	69.4
5	✓	✓	✓	✓	71.0

Table 5: Ablation studies with ResNet-18 on ImageNet. 'IR' indicates our Information Restoration block as Eq. 10.

Extra Overhead Evaluation

Since our motivation is speeding up the training process, the extra computation overhead introduced in our FST framework should be considered. In other words, the extra time overhead introduced must be less than the time saved to achieve an overall acceleration. The extra latency in FST mainly comes from 1) sorting weights inside its 4-sized groups, which can be accelerated in parallel with popular deep learning frameworks. 2) shifting gradient by one element, which can be efficiently implemented by re-allocating the index with the CUDA kernel.

Therefore, to further justify the effectiveness in practice, we report the actual time overhead versus the training time. We take ResNet-18 with batch size 256 as an example, and the training time in an iteration is recorded. As for the extra latency, the time spent on performing 2:4 structured sparsity and conducting shift on gradients is reported in Table 6. The execution environment is as below: Tesla A100 GPU×1, PyTorch 1.7, CUDA 11.1. From the table, we observe that the total extra overhead consumes about 2% of the training time in an iteration, demonstrating our approach's value in practical acceleration. Recall that in Table 2, the time of sorting-based 2:4 sparsity on activations $\mathcal{S}(\mathbf{X}^T)$ is up to 0.48s. The efficiency experiments show that the Information Restoration block cooperates well with the fixed-position sparsity.

Training (s)	Extra Time Overhead (s)			
	$\mathcal{S}(\mathbf{W})$	$\mathcal{S}(\mathbf{W}^T)$	$shift(\frac{\partial\mathcal{L}}{\partial \mathbf{Y}})$	Total
0.184	0.0017	0.0017	0.0008	0.0042

Table 6: Actual execution time in an iteration. Averaged on 1000 iterations with batch size 256.

Conclusion

This work proposes FST (fully sparse training) framework under NVIDIA's newly released Ampere architecture. We select weights and activations, which are relatively more robust to structured pruning, as pruning objects. Based on the spatial similarity of activations, fixed-position 2:4 pruning is proposed to avoid large overhead. To alleviate the misleading weight updating directions, we further propose an IR block to retrieve the performance. Overall, FST achieves around 2× accelerating training without performance drop. In future, we will further study the effects of FST on natural language processing tasks with self-attention modules.

Acknowledgments

This work was supported in part by National Key Research and Development Program of China (No. 2020AAA0103402), the Strategic Priority Research Program of Chinese Academy of Sciences (No. XDA27040300), National Natural Science Foundation of China (No.61906193) and the Central Hardware Engineering Institute, 2012 Laboratories of Huawei.

References

Bellec, G.; Kappel, D.; Maass, W.; and Legenstein, R. A. 2018. Deep Rewiring: Training very sparse deep networks. In *International Conference on Learning Representations*.

Bengio, Y.; Léonard, N.; and Courville, A. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*.

Chellapilla, K.; Puri, S.; and Simard, P. 2006. High performance convolutional neural networks for document processing. In *Tenth international workshop on frontiers in handwriting recognition*. Suvisoft.

Chen, K.; Wang, J.; Pang, J.; Cao, Y.; Xiong, Y.; Li, X.; Sun, S.; Feng, W.; Liu, Z.; Xu, J.; et al. 2019. MMDetection: Open mmlab detection toolbox and benchmark. *arXiv preprint arXiv:1906.07155*.

Chen, W.; Wang, P.; Cheng, K.; and Cheng, J. 2021. Towards Mixed-Precision Quantization of Neural Networks via Constrained Optimization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 5350–5359.

Chen, Y.-H.; Krishna, T.; Emer, J. S.; and Sze, V. 2016. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE journal of solid-state circuits*, 52(1): 127–138.

Chmiel, B.; Ben-Uri, L.; Shkolnik, M.; Hoffer, E.; Banner, R.; and Soudry, D. 2021. Neural gradients are near-lognormal: improved quantized and sparse training. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*.

Dettmers, T.; and Zettlemoyer, L. 2019. Sparse networks from scratch: Faster training without losing performance. *arXiv preprint arXiv:1907.04840*.

Evci, U.; Gale, T.; Menick, J.; Castro, P. S.; and Elsen, E. 2020. Rigging the lottery: Making all tickets winners. In *International Conference on Machine Learning*, 2943–2952. PMLR.

Frieden, B. R. 1972. Restoring with maximum likelihood and maximum entropy. *JOSA*, 62(4): 511–518.

Goli, N.; and Aamodt, T. M. 2020. Resprop: Reuse sparsified backpropagation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 1548–1558.

Gong, Z.; Ji, H.; Fletcher, C. W.; Hughes, C. J.; and Torrellas, J. 2020. SparseTrain: Leveraging Dynamic Sparsity in Software for Training DNNs on General-Purpose SIMD Processors. In *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*, 279–292.

Gou, J.; Yu, B.; Maybank, S. J.; and Tao, D. 2021. Knowledge distillation: A survey. *International Journal of Computer Vision*, 129(6): 1789–1819.

Han, S.; Kang, J.; Mao, H.; Hu, Y.; Li, X.; Li, Y.; Xie, D.; Luo, H.; Yao, S.; Wang, Y.; et al. 2017. Ese: Efficient speech recognition engine with sparse lstm on fpga. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 75–84.

Han, S.; Liu, X.; Mao, H.; Pu, J.; Pedram, A.; Horowitz, M. A.; and Dally, W. J. 2016. EIE: Efficient inference engine on compressed deep neural network. *ACM SIGARCH Computer Architecture News*, 44(3): 243–254.

Han, S.; Pool, J.; Tran, J.; and Dally, W. J. 2015. Learning both Weights and Connections for Efficient Neural Network. In *Advances in neural information processing systems*, 1135–1143.

He, X.; Mo, Z.; Cheng, K.; Xu, W.; Hu, Q.; Wang, P.; Liu, Q.; and Cheng, J. 2020. Proxybnn: Learning binarized neural networks via proxy matrices. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part III 16*, 223–241. Springer.

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.

Hubara, I.; Chmiel, B.; Island, M.; Banner, R.; Naor, S.; and Soudry, D. 2021. Accelerated sparse neural training: A provable and efficient method to find n: m transposable masks. In *Advances in Neural Information Processing Systems*.

Johnson, J. 2016. cnn-benchmarks. https://github.com/jcjohnson/cnn-benchmarks. Accessed: 2022-03-16.

Lavin, A.; and Gray, S. 2016. Fast algorithms for convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 4013–4021.

Li, F.; Li, G.; Mo, Z.; He, X.; and Cheng, J. 2020. FSA: A Fine-Grained Systolic Accelerator for Sparse CNNs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(11): 3589–3600.

Li, G.; Liu, Z.; Li, F.; and Cheng, J. 2021. Block convolution: Towards memory-efficient inference of large-scale CNNs on FPGA. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*.

Lin, T.; Stich, S. U.; Barba, L.; Dmitriev, D.; and Jaggi, M. 2020. Dynamic Model Pruning with Feedback. In *International Conference on Learning Representations*.

Liu, J.; Xu, Z.; Shi, R.; Cheung, R. C. C.; and So, H. K. 2020. Dynamic Sparse Training: Find Efficient Sparse Network From Scratch With Trainable Masked Layers. In *International Conference on Learning Representations*.

Liu, W.; Anguelov, D.; Erhan, D.; Szegedy, C.; Reed, S.; Fu, C.-Y.; and Berg, A. C. 2016. Ssd: Single shot multibox detector. In *European conference on computer vision*, 21–37. Springer.

Lym, S.; Choukse, E.; Zangeneh, S.; Wen, W.; Sanghavi, S.; and Erez, M. 2019. PruneTrain: fast neural network training by dynamic sparse model reconfiguration. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 1–13.

M Abdelmoniem, A.; Elzanaty, A.; Alouini, M.-S.; and Canini, M. 2021. An Efficient Statistical-based Gradient Compression Technique for Distributed Training Systems. *Proceedings of Machine Learning and Systems*, 3.

Mishra, A.; Latorre, J. A.; Pool, J.; Stosic, D.; Stosic, D.; Venkatesh, G.; Yu, C.; and Micikevicius, P. 2021. Accelerating sparse deep neural networks. *arXiv preprint arXiv:2104.08378*.

Mocanu, D. C.; Mocanu, E.; Stone, P.; Nguyen, P. H.; Gibescu, M.; and Liotta, A. 2018. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature communications*, 9(1): 1–12.

Mostafa, H.; and Wang, X. 2019. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. In *International Conference on Machine Learning*, 4646–4655. PMLR.

NVIDIA. 2020a. Matrix multiply-accumulate operation using mma.sp instruction with sparse matrix. https://docs.nvidia.com/cuda/parallel-thread-execution/index.html. Accessed: 2022-03-16.

NVIDIA. 2020b. NVIDIA A100 Tensor Core GPU Architecture. https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf. Accessed: 2022-03-16.

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.

Ren, S.; He, K.; Girshick, R.; and Sun, J. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28: 91–99.

Sun, X.; Ren, X.; Ma, S.; and Wang, H. 2017. meprop: Sparsified back propagation for accelerated deep learning with reduced overfitting. In *International Conference on Machine Learning*, 3299–3308. PMLR.

Wang, P.; Chen, Q.; He, X.; and Cheng, J. 2020. Towards accurate post-training network quantization via bit-split and stitching. In *International Conference on Machine Learning*, 9847–9856. PMLR.

Wen, W.; Wu, C.; Wang, Y.; Chen, Y.; and Li, H. 2016. Learning structured sparsity in deep neural networks. *Advances in neural information processing systems*, 29: 2074–2082.

Wiedemann, S.; Mehari, T.; Kepp, K.; and Samek, W. 2020. Dithered backprop: A sparse and quantized backpropagation algorithm for more efficient deep neural network training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 720–721.

Xu, W.; He, X.; Zhao, T.; Hu, Q.; Wang, P.; and Cheng, J. 2020. Soft Threshold Ternary Networks. In *IJCAI*, 2298–2304.

Ye, X.; Dai, P.; Luo, J.; Guo, X.; Qi, Y.; Yang, J.; and Chen, Y. 2020. Accelerating CNN training by pruning activation gradients. In *European Conference on Computer Vision*, 322–338. Springer.

Zhang, C.; Yuan, G.; Niu, W.; Tian, J.; Jin, S.; Zhuang, D.; Jiang, Z.; Wang, Y.; Ren, B.; Song, S. L.; et al. 2021. ClickTrain: efficient and accurate end-to-end deep learning training via fine-grained architecture-preserving pruning. In *Proceedings of the ACM International Conference on Supercomputing*, 266–278.

Zhao, T.; Hu, Q.; He, X.; Xu, W.; Wang, J.; Leng, C.; and Cheng, J. 2021. ECBC: Efficient Convolution via Blocked Columnizing. *IEEE Transactions on Neural Networks and Learning Systems*.

Zhou, A.; Ma, Y.; Zhu, J.; Liu, J.; Zhang, Z.; Yuan, K.; Sun, W.; and Li, H. 2021. Learning N: M Fine-grained Structured Sparse Neural Networks From Scratch. In *International Conference on Learning Representations*.

Zhou, S.; Wu, Y.; Ni, Z.; Zhou, X.; Wen, H.; and Zou, Y. 2016. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*.