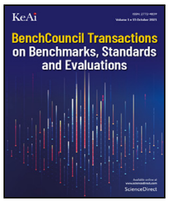
Contents lists available at ScienceDirect

BenchCouncil Transactions on Benchmarks, Standards and Evaluations

journal homepage: www.keaipublishing.com/en/journals/benchcouncil-transactions-on-benchmarks-standards-and-evaluations/



Research Article

Performance characterization and optimization of pruning patterns for sparse DNN inference

Yunjie Liu, Jingwei Sun [*], Jiaqiang Liu, Guangzhong Sun

University of Science and Technology of China, Hefei, China

ARTICLE INFO

Keywords:
Deep neural network
Pruning
Matrix multiplication

ABSTRACT

Deep neural networks are suffering from over parameterized high storage and high consumption problems. Pruning can effectively reduce storage and computation costs of deep neural networks by eliminating their redundant parameters. In existing pruning methods, filter pruning achieves more efficient inference, while element-wise pruning maintains better accuracy. To make a trade-off between the two endpoints, a variety of pruning patterns has been proposed. This study analyzes the performance characteristics of sparse DNNs pruned by different patterns, including element-wise, vector-wise, block-wise, and group-wise. Based on the analysis, we propose an efficient implementation of group-wise sparse DNN inference, which can make better use of GPUs. Experimental results on VGG, ResNet, BERT and ViT show that our optimized group-wise pruning pattern achieves much lower inference latency on GPU than other sparse patterns and the existing group-wise pattern implementation.

1. Introduction

Deep neural networks (DNNs) have achieved remarkable performance in the field of artificial intelligence and have attracted the interest of many researchers. In recent years, deep neural networks have been widely applied in numerous applications, including computer vision [1], natural language processing [2], recommendation systems [3], etc.

In order to achieve high accuracy, DNNs usually have the property of over-parameterized. In other words, they contain redundant parameters that cost large storage and are difficult to be deployed to resource-constrained devices. The inference latency of DNNs is also affected due to the large amount of computational operations. To address this issue, researchers have proposed various methods to compress DNN models. Pruning is a representative and effective model compression method. It identifies and removes redundant parameters in a DNN according to specific criteria. Ideally, after conducting pruning method, the amount of both model parameters and computational operations is reduced, and the inference time cost should also be reduced.

In practice, pruning does not ensure efficient inference. According to the granularity of pruned parameters, existing pruning methods falls along a spectrum between unstructured pruning and structured pruning. When the unstructured element-wise pattern is used for pruning, more parameters can be pruned. However, the pruned model is unfriendly on commodity GPU architectures due to unaligned and non-coalescing data access [4–6]. In this case, although the number of model parameters and the number of computational operations are reduced, the inference time can be even higher than that of the dense model due to the imperfect hardware support for the sparse computation [7]. When pruning uses the structured filter pruning, the parallel computing ability of GPUs can be better utilized due to its regular computation. However with this method, the number of pruned parameters is limited, while the model accuracy may drops significantly. To make a trade-off between the two endpoints, a variety of pruning patterns has been proposed, such as vector-wise [8,9] and block-wise pruning [10,11]. Vector-wise pruning divides the parameters of each row into vectors with equal size, and prune equal proportions of the parameters in each vector. Block-wise pruning divides the weight matrix into matrix blocks of specific shapes and removes the redundant blocks according to importance criteria of each block. Compared with filter pruning, these structured pruning patterns reserve more regular, balanced, and partially dense non-zero elements. However, the resulting sparse models from these pruning patterns still require specific runtime support.

In this study, we analyze the performance characteristics of different sparse DNNs, including element-wise, vector-wise, block-wise, and group-wise patterns. We find that these pruning patterns with off-the-shelf sparse computing libraries (e.g., cuSPARSE) are difficult to make full use of GPU ability. We then propose an efficient implementation of structured sparse DNN inference based on group-wise pattern. More specifically, for convolutional neural networks (CNNs), group-wise pattern removes the parameters with the same indexes in

* Corresponding author.
E-mail addresses: lyj06@mail.ustc.edu.cn (Y. Liu), sunjw@ustc.edu.cn (J. Sun), jqliu42@mail.ustc.edu.cn (J. Liu), gzsun@ustc.edu.cn (G. Sun).

https://doi.org/10.1016/j.tbench.2023.100090
Received 7 December 2022; Received in revised form 5 March 2023; Accepted 5 March 2023
Available online 7 March 2023

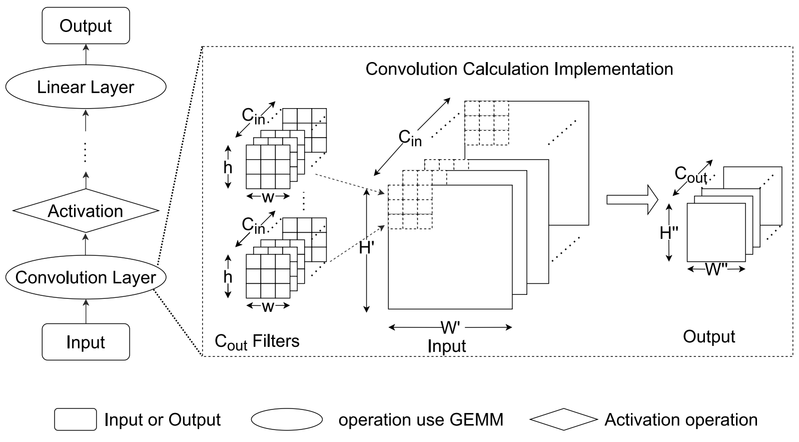


Fig. 1. Convolutional neural network.

all channels. For recurrent neural networks (RNNs) and transformer-based models, group-wise pattern removes rows of each weight matrix. Based on this pruning pattern, we convert the dominant computation kernels of pruned CNNs, RNNs, and Transformers to general matrix multiplication (GEMM) operations. Current deep learning programming frameworks (e.g., PyTorch, Tensorflow) and hardware platforms support well-developed GEMM operations. Therefore, our implementation can make better use of GPUs. Besides, group-wise pruning pattern only constrains the layout of non-zero elements. It is easy to combine the pattern with existing sophisticated pruning schedules and importance criteria, like Dynamic Sparse Training [12], Lottery Ticket Hypothesis [4,13], Magnitude [14,15], Taylor [16], Hessian [17], etc.

The main contributions of this paper are summarized as follows:

- We conduct an empirical study on existing mainstream fine-grained and structured pruning patterns. We compare their inference performance under varying conditions and indicate their inefficiency on GPU with off-the-shelf sparse computing library.
- We propose an efficient implementation of group-wise pruning pattern. The implementation converts group-wise sparse matrix-matrix multiplication into GEMM operations and optimizes the memory accesses according to GPU hardware characteristics. It makes full use of existing runtime libraries and GPU hardware support.

2. Background and related work

2.1. DNN model pruning

Generally, neural networks have over-parameterized property and contain redundant parameters. By analyzing and removing these redundant parameters during or after training, a neural network can be optimized to obtain a lower execution time and consume less memory resources when it is deployed to a target device. This process is the pruning of neural networks.

LeCun et al. in [18] pioneered the optimal brain damage (OBD) method that treats the individual weights as a unit. Hassibi et al. [19, 20] proposed an optimal brain surgeon (OBS) method based on the optimal brain damage method with the addition of an update step based on the surgical recovery weights, based on the diagonal assumption, the extreme value assumption and the quadratic assumption. Later, Han et al. [21] proposed that learning only the important connections in the network can reduce the number of model parameters and computation without affecting the final accuracy of the network, and proposed the classical pruning-retraining framework. Li et al. [15] proposed a compression technique based on convolutional kernel pruning. Hu et al. [22] proposed to use both the base model output and the pruned classification loss function to supervise the channel selection at each layer, especially introducing additional losses to encode the difference between the features in the base model and the pruned model feature maps. By considering reconstruction error, additional loss and classification loss simultaneously, the accuracy of the pruned model is greatly improved.

2.2. Hardware-aware acceleration for pruned DNN models

Dense model. A convolutional neural networks mainly contains two layer types: convolutional layer and linear layer. The computation of linear layer can be simply regarded as matrix multiplication. The convolutional layer is shown in Fig. 1, which can be computed by converting the convolutional algorithm to matrix multiplication using im2col algorithm. For the BERT model and ViT model, its main computational part, encoders structure, can also be regarded as some column matrix multiplication, as shown in Fig. 2. Therefore, the key point of reducing the latency of a dense neural network model is to reduce the latency of matrix multiplication. Generalized matrix multiplication (GEMM) is used in deep learning to perform the above matrix operations. Since GEMM has been well-developed for a long time, most of the existing programming frameworks and commercial hardware can efficiently support dense model acceleration.

A model pruned by structured pruning, such as channel pruning or filter pruning, is also dense model, so it can be calculated by GEMM. Due to the strong constraint of the structured pruning pattern, the accuracy is usually worse than fine-grained pruning. Related studies focus on maintain higher accuracy. FlexPruner [23], a filter pruning method with flexible rate. It is based on a greedy strategy to select the filters to be pruned. Li et al. [24] extend the optimization space for pruning, so their method is able to compress the model more effectively. The MaskACC pruning method [25] dynamically reorganizes tensors and mask information used in convolutions to avoid unnecessary computations, so that the computational efficiency of the pruning process is improved.

Sparse model. GPUs are originally designed for dense linear algebra computation and are not ideal for sparse computations. Therefore, the design of pruning pattern is essential to the inference latency of pruned models. Zhu et al. [26] used a vector-wise pruning pattern to ensure a balanced workload for the pruned network. By adding a sparse mode with extended instruction set and hardware support, it can run on Tensor Core. Lin et al. [27] tiled the weights and divided them according to a similar vector-wise pattern to remove redundant whole vectors, which has a better trade-off between latency and performance compared to weight pruning and filter pruning. Anwar et al. [28] first explored kernel-level pruning, and proposed an intra-kernel strided pruning method, which prunes a sub-vector in a fixed stride. Guo et al. [29] proposed a Tile-wise mode, which first divides the matrix

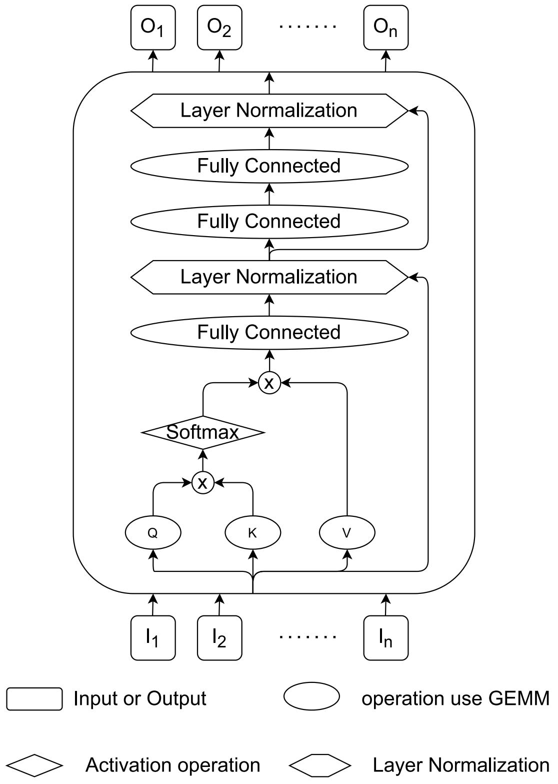


Fig. 2. Encoder structure in BERT model. Many of these operations use GEMM for completion.

into several larger Tile blocks according to the parallelism property during hardware computation, and prunes the ranks and columns within the blocks. Lebedev et al. proposed a group-wise pruning pattern in the convolutional layer in [30]. However, the pattern was combined with the Brain Damage criterion to propose a whole set of pruning method. We instead propose an efficient implementation of the group-wise pattern that focuses more on the inference time. Moreover, this work extends group-wise in the linear layer to achieve optimization of the whole neural network in terms of inference time.

In addition to innovations and research on pruning patterns, researchers also focus on efficient implementation and execution of pruned models. For instance, SparTA [31] is an end-to-end model sparsity framework that uses Tensor-with-Sparsity Attribute (TeSA) to build sparse models. Providing speedup for unstructured pruning and block-wise granularity pruning, it is compatible with a variety of sparse models and optimization techniques, facilitating sparse algorithms to explore better sparse models.

Compared with the existing works, the work proposed in this paper make better use of off-the-shelf dense computing libraries provided by vendors, e.g., cuBLAS. It has simpler implementation and higher portability. It avoids to use low-level APIs and hyper-parameters (e.g., tile width, block size) that are related to hardware architectures, so it can run on all NVIDIA GPUs, and achieve acceleration without specific tuning.

3. Performance characterization

3.1. Preliminary

The most intensive computation in a deep neural network mainly occurs in two layers: convolutional layer and linear layer. The computation of a convolutional layer transforms an input map U with C_{in} channels of size $W' \times H'$ into an output map V with C_{out} channels of size $W'' \times H''$. The relationship between the specific W', H', W'' and H'' is related to the padding and stride settings in the convolutional layer. The above transformation can be represented by the following formula:

$$V(c_o, x, y) = \sum_{c_i=1}^{C_{in}} \sum_{\substack{i=1\dots h\\ j=1\dots w}} K(c_o, c_i, i, j) \cdot U(c_i, x + i - \frac{h+1}{2}, y + j - \frac{w+1}{2}) \tag{1}$$

where K is a four-dimensional kernel tensor of size $C_{out} \times C_{in} \times h \times w$. The C_{out} corresponds to the output maps, the C_{in} corresponds to the input maps, and the h and w correspond to the convolutional kernel size.

To calculate (1), the kernel tensor K is reshaped into a two-dimensional weight matrix F with height $I' = C_{in} \times h \times w$ and width C_{out}. The input data U is reshaped into a two-dimensional input expansion matrix X with height $W'' \times H''$ and width $I' = C_{in} \times h \times w$. Each row consists of a square expansion that is computed with the corresponding convolutional kernel. Now we just need the following calculation [32]:

$$\tilde{V}(x', y') = \sum_{i=1}^{I'} X(x', i) * F(i, y') \tag{2}$$

The size of the matrix $\tilde{V}$ is $W'' \times H''$ in height and C_{out} in width and it contains all the output data of the convolutional layer. The correct output V is obtained by reshaping $\tilde{V}$.

A linear layer transforms a tensor with F_{in} dimensions to a tensor with F_{out} dimensions. The transformation can be expressed using the following equation:

$$Y(f_o) = \sum_{f_i=1}^{F_{in}} X(f_i) * W(f_i, f_o) \tag{3}$$

Where W is a tensor with height F_{in} and width F_{out}.

Since the computation of both convolutional layer and linear layer can be converted to matrix multiplication, the operation of pruning a layer is just reducing the parameters in the two-dimensional weight expansion matrix F of a convolutional layer or the W matrix of a linear layer.

3.2. Pruning patterns

A pruning method mainly consists of three components: pruning pattern, pruning schedule, and pruning criterion. Pruning pattern defines the layout of reserved non-zero elements. Pruning schedule determines the occurrence time of pruning, such as pruning after training [33,34], during training [12,35], and before training [36,37]. Pruning criterion measures the importance of a set of parameters, determining whether these parameters are pruned [4,12–17]. The AI research community usually concerns more about the schedule and the criterion, which have critical impact on the model compression ratio and inference accuracy. In this study, we focus on pruning pattern, which is essential to the execution latency and hardware utilization of pruned DNNs on GPUs.

Fig. 3 shows examples of element-wise, vector-wise, block-wise, and group-wise patterns, respectively.

The element-wise pattern [4–6] is also known as unstructured weight pruning. After the kernel tensor has been reshaped into a two-dimensional weight expansion matrix, unimportant individual parameters are removed according to specific pruning criterion. It can remove a large portion of parameters, resulting in a significant reduction of the model size. However, the layout of parameters obtained by such a pruning pattern is irregular and does not substantially help improve the execution performance of sparse DNN inference.

The vector-wise pattern [8,9] retains more local structure compared to element-wise. Vector-wise pattern divides the weight expansion matrix into vectors of equal size. For example, if the weight expansion

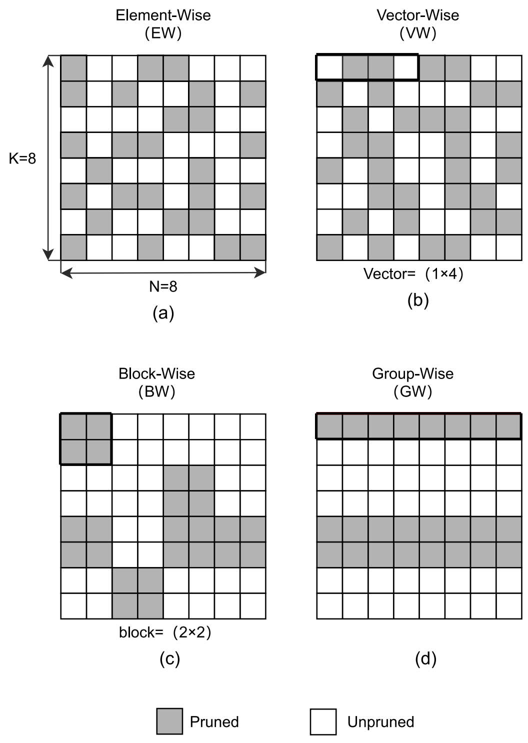


Fig. 3. Comparison of different pruning patterns. M and N represent the number of rows and columns of the expanded matrix F of the weight matrix after im2col algorithm. In the example, M is 8 and N is 8.

matrix is 12 × 16 and the vector size is artificially specified as 4, the weight expansion matrix will be divided into 12 × 4 vectors. Within each vector an equal proportion of the redundant weights are determined to be pruned, i.e., each vector contains the same number of zero elements. The redundant weights are not restricted in position within the vectors. This pruning pattern retains a more even distribution of weights because the number of pruned weights is the same within each vector.

The block-wise pattern [10,11] divides the weight expansion matrix into matrix blocks of size $m \times n$. For example, if the weight expansion matrix is 12 × 16 and the block size is artificially specified as 2 × 2, the weight expansion matrix will be divided into 6 × 8 blocks. The importance score of each block is calculated according to specific pruning criterion over the entire block.

The group-wise pattern divides the weights of different channels at the same position into a group. When the kernel tensor is expanded into a weight expansion matrix, each group forms exactly one row. At this point the unimportant rows are removed by calculating the importance score of each row according to the pruning criterion.

After removing the corresponding combination of weights from the dense model according to different pruning patterns and pruning criteria, the remaining weights form the pruned sparse neural network model.

3.3. Comparison of pruning patterns

We conduct an empirical comparison on element-wise [5], vector-wise [9], block-wise [11] and group-wise [30] pruning patterns, with different sparse ratios and tasks. The element-wise pattern uses the implementation method in [5]. The pruning criterion defines the uth index of the weight tensor W is defined as

$$score(u; W) := \frac{(W[u])^2}{\sum_{v \neq u}(W[v])^2} \tag{4}$$

As the index value increases, the score becomes smaller and smaller. After sorting each layer, the scores of all tensors are calculated and the global pruning is performed. According to the algorithm, it can be seen that this method pruning operation after the model training. The vector-wise pattern uses the implementation method in [9]. This method prunes the weights with smaller absolute values in each weight vector. The pruning schedule for this pruning method is during training. The block-wise pattern uses the method proposed in [11]. The method assigns to each block a trainable parameter m with an initial value of 1 and a range between 0 and 1. m is trained with the model and the corresponding block is pruned when this parameter is less than or equal to 0. The pruning schedule of this pruning method is also during training. The group-wise pattern uses the pattern proposed in [30]. There is no working code implementation, so the pruning criterion in DST [12] method is used to combine with the pattern. The method binds a trainable threshold at each layer and prunes groups with mean values less than the threshold. The pruning schedule for this pruning method is also during training.

Note that due to the inherent settings of pruning methods, the sparse ratios cannot be controlled to keep exactly the same. The evaluated models are VGG-16 [38], ResNet-18 [39] and Vision Transformer (ViT) [40] models. VGG-16 consists of 13 convolutional layers and 3 linear layers. ResNet-18 consists of one convolutional layer, eight residual blocks and one linear layer. Each residual block contains two convolutional layers. The ViT consists of a patch embedding layer and 12 Transformer encoders. The patch embedding layer contains a convolutional layer and each encoder contains two linear layers. We mainly perform experiments on the CIFAR-10 dataset and Tiny-Imagenet dataset, and conduct some additional experiments on the ImageNet dataset. CIFAR-10 dataset has 10 categories of images, and the corresponding task is an image classification task to predict image categories given a single image in the test set. The corresponding tasks in Tiny-ImageNet dataset and Imagenet dataset are similar to CIFAR-10. However, the number of categories in the Tiny-imagenet dataset is 200, and the number of categories in the Imagenet dataset is 1000. We also choose BERT-base model [41] as a representative in the NLP domain. It is based on the Transformer [42] implementation with 12 encoder modules. The dataset for evaluating BERT model is QQP dataset [43], which is a collection of question pairs from the community question and answer site Quora. It is a similarity and interpretation task to determine whether a pair of questions is semantically equivalent. The hardware platform we use is an Nvidia GeForce RTX 2080 Ti GPU. All pruned models are computed using cuSPARSE library.

Table 1 shows the inference time and accuracy variation (compared with non-pruned models) results of VGG-16 and ResNet-18 models with different reserved parameter ratios on the CIFAR-10 dataset. Table 2 shows the results on the Tiny-ImageNet dataset. Table 4 shows the inference time and accuracy variation of the BERT-base model on the QQP dataset.

According to the results, vector-wise and block-wise perform better than element-wise on convolutional and Transformer-based networks. However, all the pruned models have much worse performance than the corresponding dense models. Sparse computation can outperform dense computation only when the sparsity is extremely high, which will lead to significant accuracy drop and is impractical for applications. Therefore, we need a pruning pattern that can leverage an off-the-shelf dense computation library and does not need impractically high sparsity.

4. Efficient group-wise pruning

Sparse computation introduces irregular data access and is consequently time-consuming on GPUs. Through the introduce and analysis in Section 3, we can find that group-wise pruning pattern has more potential to make better use of GPU architecture. In this study, we proposes an efficient implementation of group-wise pruning pattern that enables dense computation for pruned models.

Table 1
Inference time of pruned models and dense models on CIFAR-10 dataset.

Models	Parameter (%)	Pruning pattern	Latency (ms)	Change of acc (%)
VGG-16	74.97	element-wise	6.82	+0.04
	75.00	vector-wise	6.97	+0.09
	78.74	block-wise	5.97	+0.12
	73.84	group-wise	8.63	−0.25
	49.97	element-wise	5.41	−0.23
	50.00	vector-wise	4.77	−0.32
	55.19	block-wise	5.21	−0.15
	49.90	group-wise	7.03	+0.05
	24.97	element-wise	3.27	+0.03
	25.00	vector-wise	2.55	−0.19
	25.26	block-wise	3.19	−0.38
	25.57	group-wise	2.30	−0.86
	100	dense	0.11	0.0
ResNet-18	75.99	element-wise	11.62	+0.59
	75.00	vector-wise	5.84	+0.04
	73.67	block-wise	11.93	+0.22
	77.27	group-wise	7.51	−0.35
	49.98	element-wise	10.17	+0.49
	50.00	vector-wise	3.99	−0.46
	47.41	block-wise	8.67	−0.37
	46.76	group-wise	5.86	−0.38
	24.97	element-wise	8.36	−0.06
	25.00	vector-wise	2.91	−0.53
	23.60	block-wise	5.53	−0.56
	39.35	group-wise	2.64	−1.38
	100	dense	0.70	0.0

4.1. *Group-wise pattern*

As introduced in Section 3.2, in the group-wise pattern the same position weights for different output channels will be clipped.

After group-wise pruning, the convolutional layer weights are expanded using the method described in Section 3.2. As shown in Fig. 4. The values of M and N are equal to $C_{in} \times h \times w$ and C_{out}, respectively. The masked parts of the figure indicate the redundant weights. It can be observed that the expanded weights to be pruned are complete rows in the matrix. By slicing and concatenating, the pruned matrix can be converted to a dense matrix. If the row vectors are removed from the weight matrix, then the corresponding column vectors in the input expansion matrix also need to be removed. Similarly, the input matrix can be converted into a dense matrix for calculation.

The above is the definition of group-wise pattern in convolutional layers of convolutional neural networks. It can also be applied to NLP models. The most heavy computation in NLP models, like RNNs and Transformer-based models, is direct multiplication of two weight matrices. We can simply follow the same idea of group-wise pruning pattern for CNNs. Each row of the weight matrix is removed or reserved simultaneously. Then the reserved rows are concatenated into a dense matrix. By this way group-wise is extended to linear layers.

4.2. *Inference with group-wise pattern*

Mask. When inferring with a group-wise sparse model, the model needs to know which weights have been pruned and then skips corresponding computation. Note that the pruned weights are determined by the pruning criteria. Our implementation of inference with group-wise pattern is not bound with any specific pruning criteria, so it does not suppose the exact locations of the redundant weights when we get

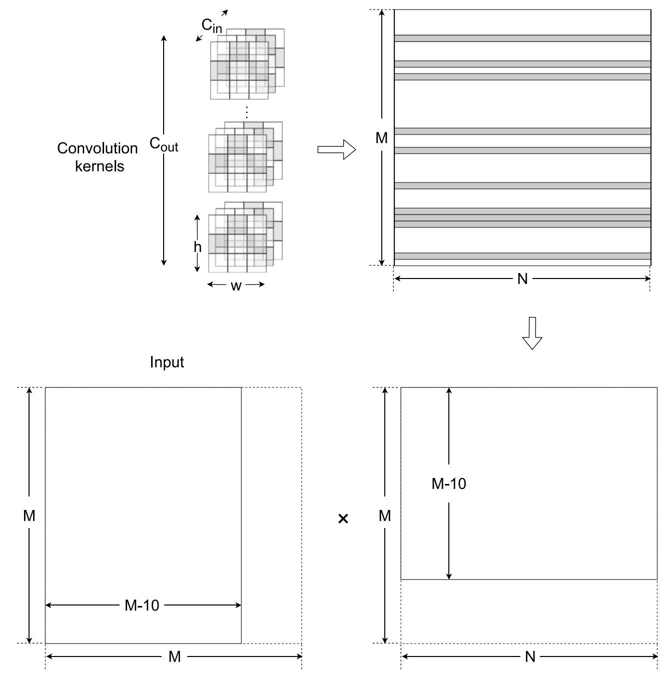


Fig. 4. Group-wise pattern. M and N represent the number of rows and columns of the expanded matrix of the weight matrix after im2col algorithm. M represent the number of rows of the expanded matrix of the input data matrix after im2col algorithm.

Algorithm 1 Inference of group-wise 2D convolutional layer

```
Input: input data $X$, layer parameters $W$
Output: output of this layer $output$
1: $W_{tile}$ = im2col($W$)
2: $W_{groups}$ = $W_{tile}$ splitted in groups
3: $X_{tile}$ = im2col($X$)
4: if use mask then
5:     $Index_{pruned}$ = [$i$ if mask[$i$] is 0]
6:     $w = W_{groups}$
7: else
8:     $Index_{pruned}$ = [$i$ if sum($W_{groups}[i]$) is 0]
9:     $w$ = index_select($W_{groups}[i]$) if $i$ not in $Index_{pruned}$
10: end if
11: //remove data that is not involved in the calculation
12: $x$ = index_select($X_{tile}[:j]$) if $j$ not in $Index_{pruned}$
13: $output = x \times w$
14: return $output$
```

Algorithm 2 Inference of group-wise linear layer

```
Input: Input data $X$, layer parameters $W$
Output: Output of this layer $output$
1: $W_{groups}$ = $W$ splitted in groups
2: if use mask then
3:     $Index_{pruned}$ = [$i$ if mask[$i$] is 0]
4:     $w = W_{groups}$
5: else
6:     $Index_{pruned}$ = [$i$ if sum($W_{tile}[i]$) is 0]
7:     $w$ = index_select($W_{groups}[i]$) if $i$ not in $Index_{pruned}$
8: end if
9: //remove data that is not involved in the calculation
10: $x$ = index_select($X[:j]$) if $j$ not in $Index_{pruned}$
11: $output = x \times w$
12: return $output$
```

Table 2
Inference time of pruned models and dense models on Tiny-ImageNet dataset.

Models	Parameters(%)	Pruning patterns	Latency(ms)	Change of acc(%)
VGG-16	72.93	element-wise	22.15	−11.52
	75.00	vector-wise	9.24	−12.34
	75.82	block-wise	17.75	−4.13
	71.74	group-wise	38.02	−4.19
	48.64	element-wise	18.54	−9.35
	50	vector-wise	8.55	−11.57
	49.2	block-wise	14.34	−4.76
	49.16	group-wise	27.86	−4.76
	24.8	element-wise	14.27	−11.71
	25	vector-wise	3.49	−11.3
	26.93	block-wise	9.89	−14.1
	30.81	group-wise	16.13	−11.68
	100	dense	0.5	0.0
ResNet-18	74.57	element-wise	47.01	−6.05
	75	vector-wise	9.59	−8.92
	75.66	block-wise	28.47	+1.2
	74.74	group-wise	30.60	+0.11
	49.27	element-wise	41.18	−4.77
	50	vector-wise	6.17	−9.87
	50.42	block-wise	18.81	−2.82
	49.23	group-wise	25.55	−0.56
	24.24	element-wise	33.87	−4.63
	25	vector-wise	3.86	−8.36
	27.66	block-wise	18.06	−4.15
	29.55	group-wise	12.98	−6.23
	100	dense	1.02	0.0

Table 3
Inference time of ViT models and dense models on CIFAR-10 and Tiny-Imagenet datasets.

Datasets	Parameters(%)	Pruning patterns	Latency(ms)	Change of acc(%)
CIFAR-10	75	element-wise	85.83	−0.56
		vector-wise	85.36	−1.08
		block-wise	28.42	+0.96
		group-wise	81.00	−1.22
	50	element-wise	57.62	−2.10
		vector-wise	58.71	−0.72
		block-wise	28.79	+0.20
		group-wise	55.55	−1.36
	25	element-wise	28.32	−0.58
		vector-wise	27.96	−0.48
		block-wise	18.19	+0.44
		group-wise	28.77	−1.95
	100	dense	3.10	0.0
Tiny-Imagenet	75	element-wise	85.98	−1.67
		vector-wise	85.54	−1.79
		block-wise	28.18	+0.37
		group-wise	81.34	0.0
	50	element-wise	57.97	−3.77
		vector-wise	58.77	−2.25
		block-wise	28.99	+0.12
		group-wise	55.99	−1.01
	25	element-wise	28.18	−3.95
		vector-wise	28.06	−2.73
		block-wise	18.05	−1.33
		group-wise	29.06	−1.12
	100	dense	3.12	0.0

a trained and pruned model. Consequently, the location information should be given when the inference starts.

Table 4
Inference time of pruned BERT-base model on QQP dataset.

Pruning pattern	Parameter (%)	Latency (ms)	Change of acc (%)
element-wise	73.46	19.66	−0.24
	43.90	20.46	−0.68
	31.66	25.32	−1.21
vector-wise	75.00	20.96	−0.19
	50.00	16.53	−0.8
	25.00	20.08	−2.42
block-wise	75.18	14.09	−0.65
	51.70	11.09	−1.64
	26.74	9.53	−2.56
group-wise	81.71	15.81	−0.28
	58.26	11.70	−1.85
	29.02	6.82	−6.32
dense	100	2.48	0.0

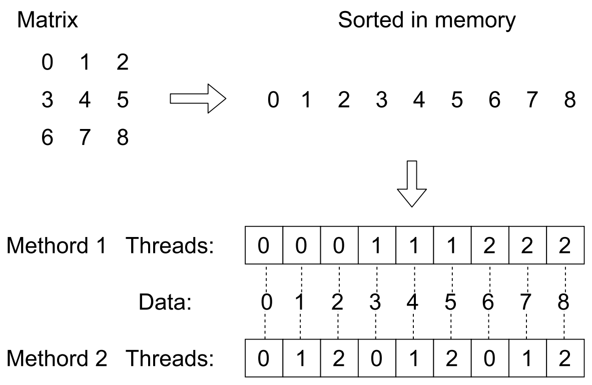


Fig. 5. Two methods of accessing data.

Existing pruning methods usually adopt two ways to keep the location of pruned weights: using masks [5], or directly setting the pruned weights to zeros [9]. If using masks, binary mask matrices are used to indicate whether the weights at corresponding locations are pruned. Our implementation covers both two ways. Algorithm 1 and Algorithm 2 show the implementations of inference with Group-wise sparse convolutional layers and linear layers, respectively. In Algorithm 1, we first split the weight W and the input data X of the current layer into W_{tile} and X_{tile} using the im2col algorithm, and divide the tiling weight into W_{groups} according to the group-wise pattern. If the model is marked with a mask, the indexes of the pruning part is extracted according to the mask. If the model is marked with changing the pruning weights to zero, the indexes of the pruning part is extracted according to the zeroing group, and the weight to be pruned is removed. According to the pruning indexes, the input data of this layer are extracted from the X_{tile} correspondingly, concatenated into a dense matrix, and then GEMM is calculated to output the calculation results of this layer. Algorithm 2 directly changes the weight of the linear layer into W_{groups} according to the group-wise pattern, and then extracts the input data with the same calculation steps as convolution to obtain the output of the linear layer. In these two algorithms, i refers to the group index of data not involved in the operation (the weight to be pruned and the input data not involved in the operation), and j refers to the group index of data to be retained.

Memory accesses coalesce. Memory accesses coalesce is the use of consecutive threads to access data at consecutive addresses. As shown in Fig. 5, a matrix of size 3×3 is accessed using 3 threads and the matrix is stored in memory in a linear fashion. There are two ways to access the matrix. The first one, thread 0 accesses the 0th, 1st, and 2nd data, and thread 1 accesses the 3rd, 4th, and 5th data, and the contiguous threads do not access contiguous memory; The second way, thread 0 accesses the 0th, 3rd, and 6th data, and thread 1 accesses the 1st, 4th,

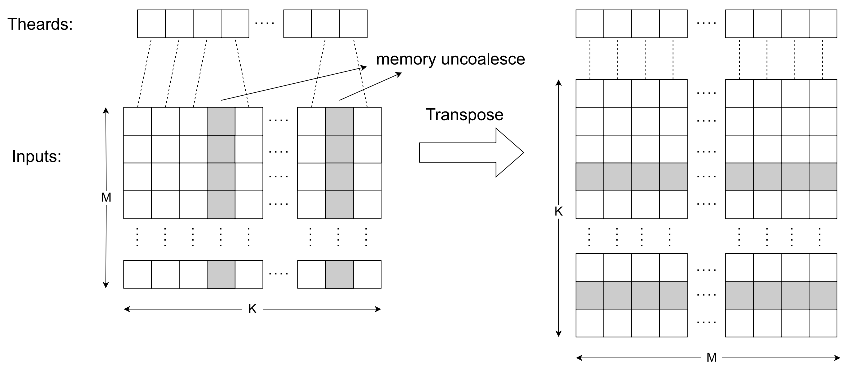


Fig. 6. Implementation of memory accesses coalesce in group-wise pattern.

and 7th data, and the contiguous threads access contiguous memory. Either way, each thread makes 3 accesses, but the second way is a coalesce memory access that requires fewer memory transactions and is therefore more efficient than the first.

In the group-wise pruning pattern, some rows of a weight matrix are pruned, so the columns of the matrix corresponding to the input data tiled at that layer then do not participate in the computation and need to be removed. When the columns of the input matrix are skipped, uncoalesced memory accesses are introduced frequently, which is inefficient on the GPU. Then the contiguous accesses to the initial input matrix become uncoalesced, which may lead to severe performance degradation. Uncoalesced memory accesses require multiple memory transactions. To alleviate this issue, the matrix can be transposed to improve its memory access efficiency, as shown in Fig. 6. In this case, column skipping becomes row skipping, eliminating uncoalesced accesses and improving access efficiency.

5. Evaluation

5.1. Setup

Benchmark. The evaluated models are VGG-16, ResNet-18, ViT and BERT-base, which cover the fields of computer vision and NLP. VGG-16 and ResNet-18 are classical CNN models. We perform inference latency evaluation on the CIFAR-10 dataset. For a convolutional layer, it is tiled after pruning. And for a linear layer, we prune it directly according to the same pattern after tiling by convolutional computation.

For the most popular family of Transformer models, we use the ViT and BERT-base models with 12-layer encoder. The ViT model is also applicable to CIFAR-10 and Tiny-Imagenet datasets for experiments. The BERT-base model downstream task being evaluated is a sentence classification task on the widely used QQP dataset.

In our experiments, the sparse CNN models and the ViT model are trained from scratch, and these models are pruned with element-wise, vector-wise, block-wise and group-wise sparse model with 100 epochs at different target sparsity levels, depending on the dataset size. The NLP models are evaluated with pre-trained models and fine-tuned by 10 epochs at each target sparsity level. They are also pruned by applying the patterns of element-wise, vector-wise, block-wise and group-wise, respectively.

Baseline. The models obtained by element-wise, vector-wise and block-wise pruning patterns are sparse models, so they are computed using the cuSPARSE library. Group-wise can be computed using the cuBLAS library through a series of processes. All experiments are performed on an NVIDIA GeForce RTX 2080 Ti GPU using FP32. The convolutional operations in the CNN models are converted to GEMM by the im2col method.

Table 5
Inference latency of group-wise pattern on CIFAR-10 dataset.

Models	Parameter (%)	Latency (ms)	Change of acc (%)
VGG-16	73.84	0.31	−0.25
	49.9	0.20	+0.05
	25.57	0.11	−0.86
ResNet-18	77.27	0.78	−0.35
	46.76	0.75	−0.38
	39.35	0.64	−1.38
ViT	75	2.80	−1.22
	50	2.37	−1.36
	25	1.96	−1.95

Table 6
Inference latency of group-wise pattern on Tiny-ImageNet dataset.

Models	Parameters(%)	Latency(ms)	Change of acc(%)
VGG-16	71.74	0.49	−4.19
	47.27	0.34	−4.76
	30.98	0.25	−11.68
ResNet-18	75.4	1.07	+0.11
	49.23	0.82	−0.56
	29.55	0.57	−6.23
ViT	75.0	2.71	−0.0
	50.0	2.37	−1.01
	25.0	1.92	−1.12

Table 7
Inference latency of group-wise pattern on QQP dataset.

Models	Parameter (%)	Latency (ms)	Change of acc (%)
BERT-base	81.71	2.43	−0.28
	58.26	2.01	−1.85
	31.21	1.47	−8.72

5.2. Result and analysis

We compare the latency of group-wise, element-wise, vector-wise and block-wise patterns on multiple models. The results of the group-wise pattern are listed in Tables 5 to 7. Figs. 7 to 10 show a comparison in inference time between the efficient group-wise introduced in this paper and the three pruning patterns element-wise, vector-wise, and block-wise in Section 3.2. The data of efficient group-wise come from Tables 5 to 7, and the data of other patterns come from Tables 1 to 4. The number of parameters on the horizontal axis is only an approximate range, rather than the exact value. For example, 25% means that with a model residual parameter of approximately 25%, it may be 27% or 23% of the true figure. The data represented in the figures are a visualization of the tables in Section 3.3 and Tables 5 to 7 in this section. It can be seen that the inference delay with group-wise pattern is significantly reduced. Supplementary experiments on Imagenet dataset are shown in Table 8.

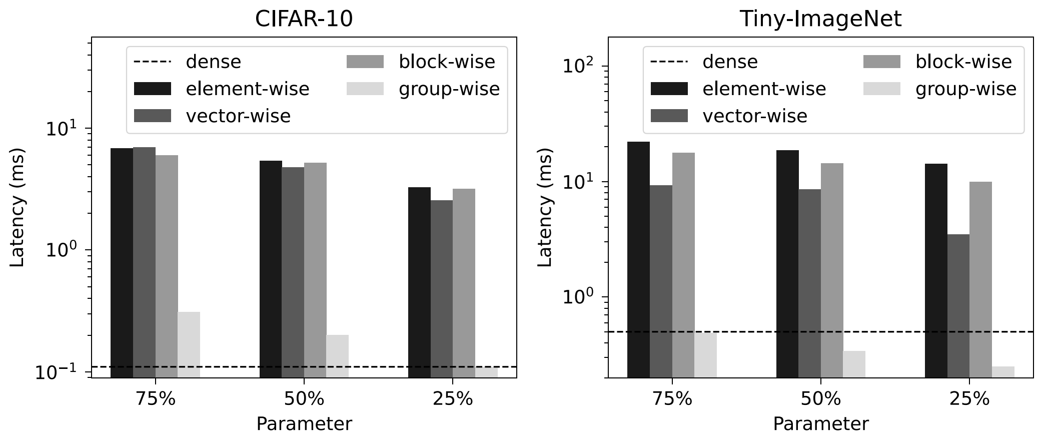


Fig. 7. This figure shows the inference latency of VGG-16 model using different pruning patterns on CIFAR-10 and Tiny-ImageNet datasets.

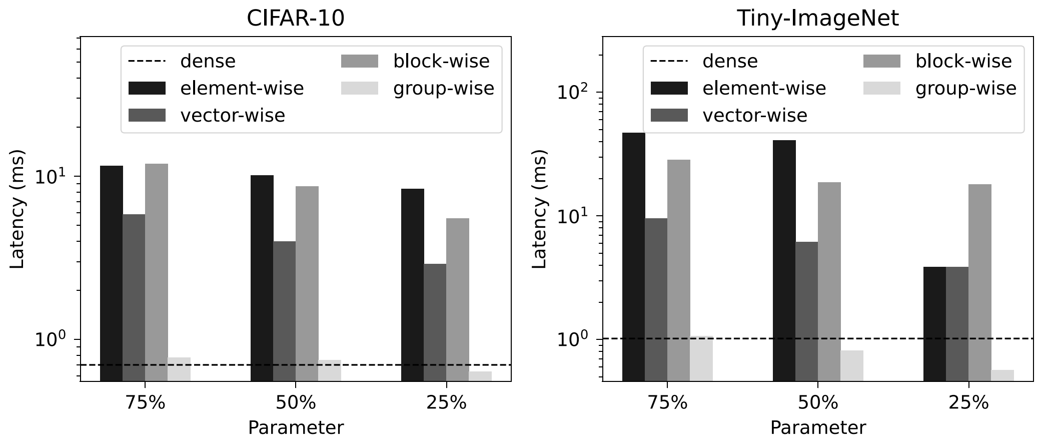


Fig. 8. This figure shows the inference latency of ResNet-18 model using different pruning patterns on CIFAR-10 and Tiny-ImageNet datasets.

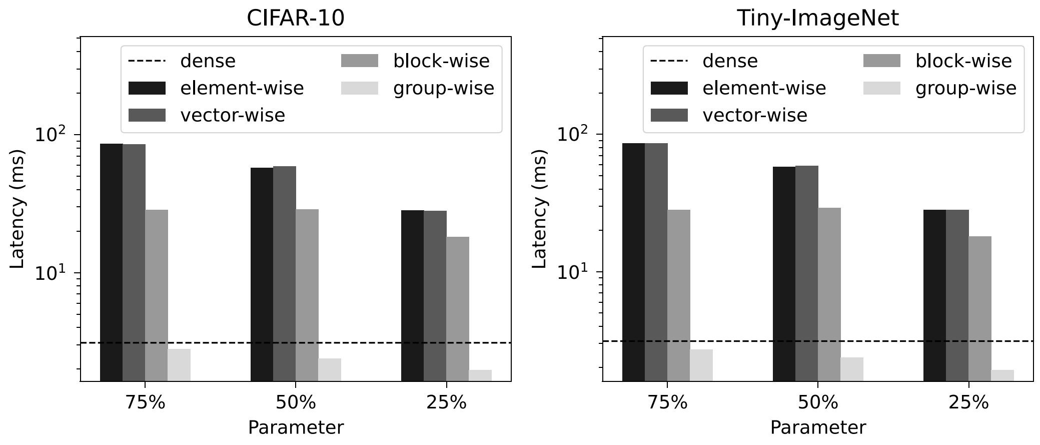


Fig. 9. This figure shows the inference latency of ViT model using different pruning patterns on CIFAR-10 and Tiny-ImageNet datasets.

The experimental results show that the group-wise pattern has shorter latency than all the other sparse patterns at the same sparsity. Group-wise effectively takes advantage of the dense GEMM acceleration, which makes fast inference possible even after pruning to obtain a sparse model. When compared with the dense model, effective latency reduction will be achieved on ResNet-18, BERT-base and ViT models. Poor performance is achieved on VGG-16 when using small datasets, but effective latency reduction is achieved on larger datasets. Because small datasets have less data to be calculated, the reduced calculation time after pruning is insufficient to counteract the overhead introduced by extra steps for pruning. Even so, when the remaining parameter ratio is 25%, the inference latency of the sparse model can be equivalent to or less than that of the dense model. And further compression can bring more acceleration. In most cases, pruned models achieve similar latency to the dense model at about 75% remaining parameter ratio, and the inference latency of the group-wise pattern will be lower than the dense model as the number of parameters continues to decrease.

Fig. 11 shows the comparison of the inference time between the group-wise implementation in this paper and Lebedev's implementation in [30]. Since [30] only focuses on convolutional layers, we take a convolutional layer from VGG-16 model as an example for comparison. The configuration of the convolutional layer set as 512 input channels,

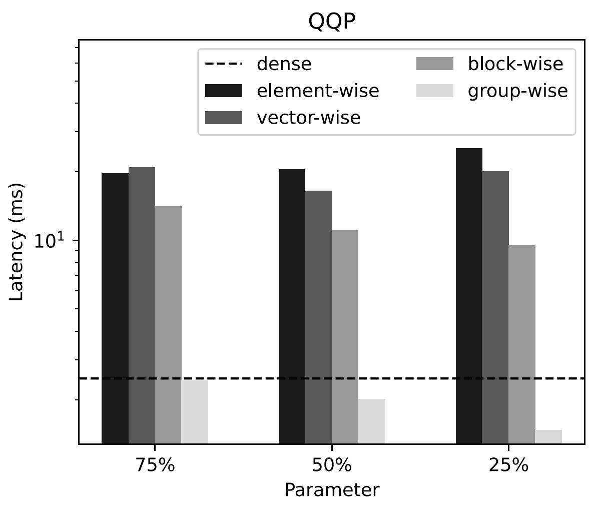


Fig. 10. This figure shows the inference latency of BERT-base model using different pruning patterns on QQP datasets.

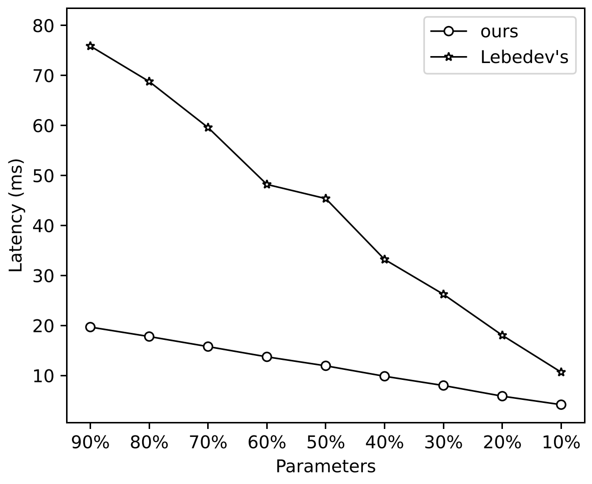


Fig. 11. Latency of different sparsity of convolutional layer using different implementations. 'Lebedev's' is the implementation in [30], and 'ours' is the implementation in this paper.

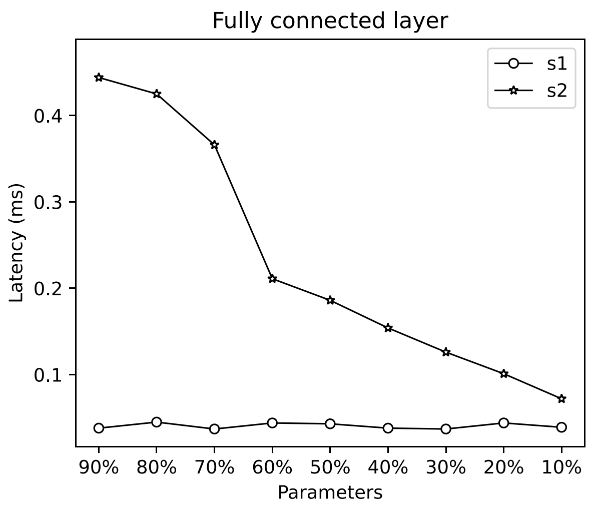


Fig. 12. Latency of group-wise pattern in linear layer. In the figure, s1, s2 correspond to the matrix concatenating and multiplication steps, respectively. The measured batch size is 64.

Table 8
When the VGG-16 model retains 50% of the parameters, use the experimental data of different patterns on full Imagenet dataset.

Parameters(%)	Pruning patterns	Latency(ms)	Change of acc(%)
50	element-wise	360.43	−5.29
	vector-wise	315.82	−8.97
	block-wise	44.25	−4.13
	group-wise	5.99	−5.26
100	dense	15.67	0.0

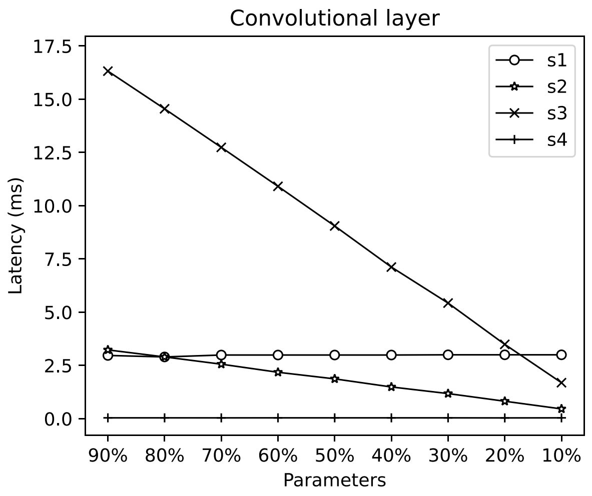


Fig. 13. Latency of group-wise pattern model in convolutional operation. In this figure, s1, s2, s3 and s4 denote im2col, matrix concatenating, multiplication, and reshaping results to feature map size steps, respectively. The measured batch size is 64.

512 output channels, 3 × 3 convolutional kernel size, 28 × 28 input data size, and 64 batch size. It can be seen that our implementation can further effectively reduce the inference latency of group-wise pruning pattern.

We also measured the latency of internal steps of convolutional layers and linear layers, as shown in Figs. 12 to 14, respectively. According to Figs. 12 to 14, the latency showed in Fig. 14 is shorter than others. The convolutional and linear layers are split into multiple kernel functions when measuring the internal steps, therefore some overhead is introduced. When measuring the time for the entire layer, it is only necessary to wait for the finish of the layer. So there is some difference in the overall time between the two sets of data.

The parameter settings for the convolutional layer are the same as the experimental settings in Fig. 11. The configuration of the linear layer: the input channel is set to 4096, the output channel is 4096, the input data size is 4096, and the batch size is 64. The above parameter settings are also from one of the layers in the VGG-16 model. Fig. 14 shows that, when the sparsity is around 20% and 10% for convolutional layer and linear layer, respectively, the inference latency is equivalent to that of dense matrix calculation. With the increase of sparsity, the latency advantage from pruning becomes more significant.

6. Conclusion

In this paper, we conduct an empirical comparison on existing mainstream pruning patterns, including element-wise, vector-wise, block-wise and group-wise pattern. After analyzing their inefficiency, we propose a more efficient implementation of the group-wise pattern on GPU using off-the-shelf GEMM library. Experimental results show that its inference latency on GPU is much lower than that of other sparse patterns. The proposed optimization implementation can further improve the inference speed of DNN models compared to existing group-wise approach. In addition, when the reserved parameters of the model are less than 75%, our group-wise inference performance can exceed that of dense models.

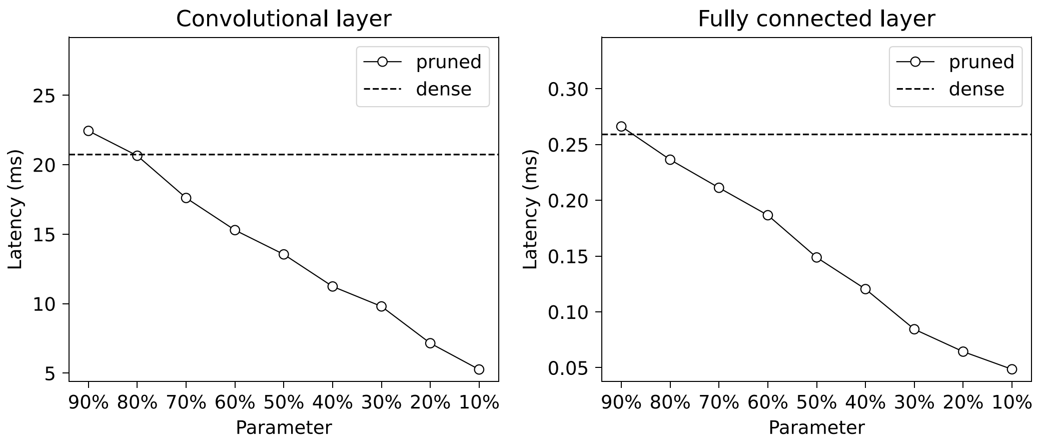


Fig. 14. Latency of different sparsity of convolutional layer and linear layer in group-wise pruned and dense model. The measured batch size is 64.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] S. Hijazi, R. Kumar, C. Rowen, et al., Using Convolutional Neural Networks for Image Recognition, Vol. 9, Cadence Design Systems Inc., San Jose, CA, USA, 2015.

[2] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al., Language models are unsupervised multitask learners, OpenAI Blog 1 (8) (2019) 9.

[3] G. Jain, T. Mahara, S.C. Sharma, S. Agarwal, H. Kim, TD-DNN: A time decay-based deep neural network for recommendation system, Appl. Sci. 12 (13) (2022) 6398.

[4] J. Diffenderfer, B. Kailkhura, Multi-prize lottery ticket hypothesis: Finding accurate binary neural networks by pruning a randomly weighted network, in: 9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021, OpenReview.net, 2021, pp. 1–23, URL: https://openreview.net/forum?id=U_mat0b9iv.

[5] J. Lee, S. Park, S. Mo, S. Ahn, J. Shin, Layer-adaptive sparsity for the magnitude-based pruning, in: 9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021, OpenReview.net, 2021, pp. 1–19, URL: https://openreview.net/forum?id=H6ATjJ0TKdf.

[6] V. Sehwag, S. Wang, P. Mittal, S. Jana, HYDRA: Pruning adversarially robust neural networks, in: H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, H. Lin (Eds.), Advances in Neural Information Processing Systems, Vol. 33, Curran Associates, Inc., 2020, pp. 19655–19666, URL: https://proceedings.neurips.cc/paper/2020/file/e3a72c791a69f87b05ea7742e04430ed-Paper.pdf.

[7] P. Hill, A. Jain, M. Hill, B. Zamirai, C. Hsu, M.A. Laurenzano, S.A. Mahlke, L. Tang, J. Mars, DeftNN: addressing bottlenecks for DNN execution on GPUs via synapse vector elimination and near-compute data fission, in: H.C. Hunter, J. Moreno, J.S. Emer, D. Sánchez (Eds.), Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2017, Cambridge, MA, USA, October 14-18, 2017, ACM, 2017, pp. 786–799.

[8] Z. Yao, S. Cao, W. Xiao, C. Zhang, L. Nie, Balanced sparsity for efficient dnn inference on gpu, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 33, 2019, pp. 5676–5683.

[9] A. Zhou, Y. Ma, J. Zhu, J. Liu, Z. Zhang, K. Yuan, W. Sun, H. Li, Learning N: M fine-grained structured sparse neural networks from scratch, in: 9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021, OpenReview.net, 2021, pp. 1–15, URL: https://openreview.net/forum?id=K9bw7vqp_s.

[10] S. Narang, E. Undersander, G.F. Diamos, Block-sparse recurrent neural networks, CoRR abs/1711.02782, 2017.

[11] D.T. Vooturi, G. Varma, K. Kothapalli, Dynamic block sparse reparameterization of convolutional neural networks, in: 2019 IEEE/CVF International Conference on Computer Vision Workshops, ICCV Workshops 2019, Seoul, Korea (South), October 27-28, 2019, IEEE, 2019, pp. 3046–3053, http://dx.doi.org/10.1109/ICCVW.2019.00367.

[12] J. Liu, Z. Xu, R. Shi, R.C.C. Cheung, H.K. So, Dynamic sparse training: Find efficient sparse network from scratch with trainable masked layers, in: 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020, OpenReview.net, 2020, pp. 1–14, URL: https://openreview.net/forum?id=SJlbGJrtDB.

[13] E. Malach, G. Yehudai, S. Shalev-Shwartz, O. Shamir, Proving the lottery ticket hypothesis: Pruning is all you need, in: Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event, in: Proceedings of Machine Learning Research, vol. 119, PMLR, 2020, pp. 6682–6691.

[14] S.J. Hanson, L.Y. Pratt, Comparing biases for minimal network construction with back-propagation, in: D.S. Touretzky (Ed.), Advances in Neural Information Processing Systems 1, NIPS Conference, Denver, Colorado, USA, 1988, Morgan Kaufmann, 1988, pp. 177–185.

[15] H. Li, A. Kadav, I. Durdanovic, H. Samet, H.P. Graf, Pruning filters for efficient ConvNets, in: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings, OpenReview.net, 2017, pp. 1–13, URL: https://openreview.net/forum?id=rJqFGTslg.

[16] P. Molchanov, S. Tyree, T. Karras, T. Aila, J. Kautz, Pruning convolutional neural networks for resource efficient inference, in: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings, OpenReview.net, 2017, pp. 1–17, URL: https://openreview.net/forum?id=SJGCiw5gl.

[17] H. Wang, C. Qin, Y. Zhang, Y. Fu, Neural pruning via growing regularization, in: International Conference on Learning Representations, 2021, pp. 1–16.

[18] Y. LeCun, J.S. Denker, S.A. Solla, Optimal brain damage, in: D.S. Touretzky (Ed.), Advances in Neural Information Processing Systems 2, NIPS Conference, Denver, Colorado, USA, November 27-30, 1989, Morgan Kaufmann, 1989, pp. 598–605.

[19] B. Hassibi, D. Stork, Second order derivatives for network pruning: Optimal brain surgeon, Adv. Neural Inf. Process. Syst. 5 (1992).

[20] B. Hassibi, D.G. Stork, G.J. Wolff, Optimal brain surgeon and general network pruning, in: IEEE International Conference on Neural Networks, IEEE, 1993, pp. 293–299.

[21] S. Han, J. Pool, J. Tran, W. Dally, Learning both weights and connections for efficient neural network, Adv. Neural Inf. Process. Syst. 28 (2015).

[22] Y. Hu, S. Sun, J. Li, J. Zhu, X. Wang, Q. Gu, Multi-loss-aware channel pruning of deep networks, in: 2019 IEEE International Conference on Image Processing, ICIP, IEEE, 2019, pp. 889–893.

[23] G. Li, X. Ma, X. Wang, H. Yue, J. Li, L. Liu, X. Feng, J. Xue, Optimizing deep neural networks on intelligent edge accelerators via flexible-rate filter pruning, J. Syst. Archit. 124 (2022) 102431, http://dx.doi.org/10.1016/j.sysarc.2022.102431, URL: https://www.sciencedirect.com/science/article/pii/S1383762122000303.

[24] G. Li, X. Ma, X. Wang, L. Liu, J. Xue, X. Feng, Fusion-catalyzed pruning for optimizing deep learning on intelligent edge devices, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 39 (11) (2020) 3614–3626, http://dx.doi.org/10.1109/TCAD.2020.3013050.

[25] X. Ma, G. Li, L. Liu, H. Liu, X. Wang, Accelerating deep neural network filter pruning with mask-aware convolutional computations on modern CPUs, Neurocomputing 505 (2022) 375–387, http://dx.doi.org/10.1016/j.neucom.2022.07.006, URL: https://www.sciencedirect.com/science/article/pii/S0925231222008669.

[26] M. Zhu, T. Zhang, Z. Gu, Y. Xie, Sparse tensor core: Algorithm and hardware co-design for vector-wise sparse neural networks on modern GPUs, in: Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2019, Columbus, OH, USA, October 12-16, 2019, ACM, 2019, pp. 359–371.

[27] M. Lin, Y. Zhang, Y. Li, B. Chen, F. Chao, M. Wang, S. Li, Y. Tian, R. Ji, 1 $\times$ n pattern for pruning convolutional neural networks, IEEE Trans. Pattern Anal. Mach. Intell. (2022) 1–11, http://dx.doi.org/10.1109/TPAMI.2022.3195774.

[28] S. Anwar, K. Hwang, W. Sung, Structured pruning of deep convolutional neural networks, ACM J. Emerg. Technol. Comput. Syst. (JETC) 13 (2017) 1–18.

[29] C. Guo, B.Y. Hsueh, J. Leng, Y. Qiu, Y. Guan, Z. Wang, X. Jia, X. Li, M. Guo, Y. Zhu, Accelerating sparse DNN models without hardware-support via tile-wise sparsity, in: C. Cuicchi, I. Qualters, W.T. Kramer (Eds.), Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2020, Virtual Event / Atlanta, Georgia, USA, November 9-19, 2020, IEEE/ACM, 2020, p. 16.

[30] V. Lebedev, V.S. Lempitsky, Fast ConvNets using group-wise brain damage, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016, IEEE Computer Society, 2016, pp. 2554–2564.

[31] N. Zheng, B. Lin, Q. Zhang, L. Ma, Y. Yang, F. Yang, Y. Wang, M. Yang, L. Zhou, SparTA: Deep-learning model sparsity via Tensor-with-Sparsity-Attribute, in: 16th USENIX Symposium on Operating Systems Design and Implementation, OSDI 22, USENIX Association, Carlsbad, CA, 2022, pp. 213–232, URL: https://www.usenix.org/conference/osdi22/presentation/zheng-ningxin.

[32] K. Chellapilla, S. Puri, P. Simard, High performance convolutional neural networks for document processing, in: Tenth International Workshop on Frontiers in Handwriting Recognition, Suvisoft, 2006, pp. 1–7.

[33] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, C. Zhang, Learning efficient convolutional networks through network slimming, in: IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017, IEEE Computer Society, 2017, pp. 2755–2763.

[34] W. Wen, C. Wu, Y. Wang, Y. Chen, H. Li, Learning structured sparsity in deep neural networks, in: D.D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon, R. Garnett (Eds.), Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain, 2016, pp. 2074–2082.

[35] Y. He, G. Kang, X. Dong, Y. Fu, Y. Yang, Soft filter pruning for accelerating deep convolutional neural networks, in: J. Lang (Ed.), Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden, ijcai.org, 2018, pp. 2234–2240.

[36] P. de Jorge, A. Sanyal, H.S. Behl, P.H.S. Torr, G. Rogez, P.K. Dokania, Progressive skeletonization: Trimming more fat from a network at initialization, in: 9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021, OpenReview.net, 2021, pp. 1–21, URL: https://openreview.net/forum?id=9GsFOUyUPi.

[37] N. Lee, T. Ajanthan, P.H.S. Torr, Snip: single-shot network pruning based on connection sensitivity, in: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, la, USA, May 6-9, 2019, OpenReview.net, 2019, pp. 1–15, URL: https://openreview.net/forum?id=B1VZqjAcYX.

[38] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, in: Y. Bengio, Y. LeCun (Eds.), 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings, 2015, pp. 1–14, URL: http://arxiv.org/abs/1409.1556.

[39] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016, IEEE Computer Society, 2016, pp. 770–778.

[40] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, N. Houlsby, An image is worth 16 × 16 words: Transformers for image recognition at scale, in: 9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021, OpenReview.net, 2021, pp. 1–21, URL: https://openreview.net/forum?id=YicbFdNTTy.

[41] J. Devlin, M. Chang, K. Lee, K. Toutanova, BERT: pre-training of deep bidirectional transformers for language understanding, in: J. Burstein, C. Doran, T. Solorio (Eds.), Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers), Association for Computational Linguistics, 2019, pp. 4171–4186.

[42] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, L. Kaiser, I. Polosukhin, Attention is all you need, in: Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA, 2017, pp. 5998–6008.

[43] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, S.R. Bowman, GLUE: A multi-task benchmark and analysis platform for natural language understanding, in: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, la, USA, May 6-9, 2019, OpenReview.net, 2019, pp. 1–20, URL: https://openreview.net/forum?id=rJ4km2R5t7.