International Journal of Networking and Computing – www.ijnc.org, ISSN 2185-2847
Volume 12, Number 2, pages 270–294, July 2022

CNN Models Acceleration Using Filter Pruning and Sparse Tensor Core

Hong-Xuan Wei
Department of Computer Science and Information Engineering, National Taiwan University, Taipei, 106032, Taiwan

Pangfeng Liu
Department of Computer Science and Information Engineering, National Taiwan University, Taipei, 106032, Taiwan

Ding-Yong Hong
Institute of Information Science, Academia Sinica, Taipei, 115201, Taiwan

Jan-Jan Wu
Institute of Information Science, Academia Sinica, Taipei, 115201, Taiwan

An-Tai Chen
Department of Computer Science and Information Engineering, National Taiwan University, Taipei, 106032, Taiwan

Received: February 15, 2022
Revised: May 5, 2022
Accepted: June 2, 2022
Communicated by Hiroyuki Sato

Abstract

Convolutional neural network (CNN) is a state-of-the-art technique in machine learning and has achieved high accuracy in many computer vision applications. The number of the parameters of the CNN models is fast increasing for improving accuracy; therefore, it requires more computation time and memory space for both training and inference. As a result, reducing the model size and improving the inference speed have become critical issues for CNN. This paper focuses on filter pruning and special optimization for NVIDIA sparse tensor core. Filter pruning is a model compression technique that evaluates the importance of filters in the CNN model and removes the less critical filters. NVIDIA sparse tensor core is special hardware for CNN computation from NVIDIA Ampere GPU architecture, which can speed up a matrix multiplication if the matrix has a structure that manifests as a 2:4 pattern.

This paper proposed hybrid pruning to prune the CNN models. The hybrid pruning combines filter pruning and 2:4 pruning. We apply filter pruning to remove the redundant filters to reduce the model size. Next, we use 2:4 pruning to prune the model according to a 2:4 pattern to utilize the sparse tensor core hardware for speedup. In this hybrid pruning scenario, we also proposed two *hybrid metrics* to decide the filter's importance during filter pruning. The hybrid ranking metrics preserve the essential filters for both pruning steps and achieve higher accuracy than

traditional filter prunings by considering both metrics. We test our hybrid pruning algorithm on MNIST, SVHN, CIFAR-10 datasets using AlexNet. Our experiments concluded that our hybrid metrics achieve better accuracy than the classic L1-norm metric and the output L1-norm metric. When we prune away 40 percent of filters in the model, our methods have 2.8% to 3.3%, 2.9% to 3.5%, 2.5% to 2.7% higher accuracy than the classic L1-norm metric and the output L1-norm metric on these three datasets. We also evaluate the inference speed of the model from our hybrid pruning. We compare the hybrid pruning model with the models that result from either filter pruning or 2:4 pruning. We find that a hybrid pruning model runs up to 1.3x faster than the traditional filter pruning model with similar accuracy.

Keywords: Model Compression, Filter Pruning, CNN, Machine Learning, Sparse Tensor Core

1 Introduction

Convolutional neural network (CNN) is a critical technique in deep learning research. CNN is an effective solution to many computer vision problems, including image classification [13], [26], [9], [8], image segmentation [25], [15], and object detection [24], [3].

CNN models require a large number of parameters to achieve high accuracy. This large amount of parameters requires extensive time to train and inference and a large amount of memory to store. For example, EfficientNet-L2 [23], a state-of-the-art CNN model with 90.2% top-1 accuracy and 98.8% top-5 accuracy for the ImageNet dataset, contains 480M parameters.

Model compression is a common and effective technique for reducing the number of parameters in deep learning computation. For example, *parameter quantization* [4], *knowledge distillation* [1, 11], and network pruning [14], [7], [6] are the common approaches for model compression. These techniques reduce the size of the network with acceptable overall accuracy loss. Some of these techniques significantly improve inference and training speed since they can leverage hardware support or reduce computation time by considerably reducing the number of parameters.

NVIDIA Ampere architecture provides sparse tensor core [21] as hardware support to speed up matrix multiplication. To be able to benefit from sparse tensor cores, one of the input matrices must be in a specific *2:4 structure*, which will be described later in the paper. We use *2:4 pruning* to refer to the pruning methods that the pruning result is in a 2:4 structure. The machine learning model can achieve 1.3x - 1.5x end-to-end model inference speedup by using sparse tensor core after pruning the model according to the 2:4 structure, without loss of accuracy.

Filter pruning is a standard network pruning technique. It evaluates the filter importance in a CNN model and removes the redundant ones. Therefore, choosing the right evaluation metric is essential to remove the redundant weights. There are many metrics for evaluating the importance of filters from the literature. For example, Han et al. [14] selected the weights based on their magnitude and removed unimportant ones. Chen et al. [2] proposed a metric based on weight value and the entropy of the feature maps. Hu et al. [12] determined the importance of neurons by the *Average Percentage of Zeros* in feature maps, then removed those that have a higher percentage of zeros and seem to be redundant.

In this paper, we propose a *hybrid pruning* that combines *filter pruning* and *2:4 pruning*. The idea is to generate a smaller model that can speed up the sparse tensor core by filter pruning first; then, we apply a 2:4 pruning to ensure that the weight matrix is a 2:4 structure to take advantage of the performance provided by sparse tensor cores.

Our hybrid pruning also proposes a *hybrid ranking algorithm* to determine the importance of ranking among filters during filter pruning. The traditional algorithms from the literature only consider the filter pruning metric since they are not aware of the performance advantage of sparse tensor cores. As a result, the filters preserved by traditional pruning methods may not be that important, as far as 2:4 pruning is concerned. Our hybrid pruning considers the importance of the filters both after the filter pruning and after the 2:4 pruning and preserves crucial filters.

We use the AlexNet model as our primary model and test our approach on different datasets to validate our hybrid algorithm. Compared to traditional filter pruning methods, our algorithms select filters better, in the sense that they get higher accuracy while removing the same ratio of filters as the traditional filter pruning methods do. On the MNIST dataset, our approach achieves

5% more accuracy than the L1 metric does when both methods prune away 70% of the filters. Our approaches have about 3% higher accuracy on the SVHN dataset than the baseline when we prune away more than 40% of the filters. On the CIFAR-10 dataset, our approaches have about 2% higher accuracy when we prune away 40% of the filters.

We also test the pruned model on a GeForce RTX 3090 GPU and observe the inference time speedup. The inference speed of the pruned model is 1.86x faster than that of the original model when we use the sparse tensor core, with less than 1% of accuracy loss.

Every filter pruning method that reduces the model size will also reduce the inference speed because it has fewer parameters to compute. Therefore we also compare the model from our hybrid pruning method with the model from filter pruning only. The experiment results indicate that the hybrid pruning model running on the sparse tensor core is 1.3x faster than the traditional filter pruning model, while both have the same accuracy.

We organize the remainder of this paper as follows. Section 2 describes the related works. Section 3 describes the background of filter pruning and 2:4 pruning. Section 4 describes our methods in detail. Section 5 describes the setting and the results of our experiments. Section 6 summarizes our results and concludes.

2 Related Works

This section describes the characteristic of sparse tensor cores and the previous works that address model compression.

2.1 Sparse Tensor Core

Recently NVIDIA introduced a new Ampere GPU architecture that provides the sparse tensor core to speed up matrix multiplication. The sparse tensor core requires that the input matrix must be in a *2:4 format* to take advantage of the sparse tensor core hardware. A matrix is in a *2:4 format* if we partition every row into groups of four elements each, then every group should have at least two elements whose values are zeros.

Figure 1 illustrate the fast matrix multiplication of sparse matrix A and dense matrix B on sparse tensor cores. The matrix A must be in a 2:4 format. Let the white grids denote elements with zero values, and the green grids denote non-zero values in A. Then the system compresses the matrix A by removing zeros and compacting non-zero values by the rows of A and generates metadata that records the non-zero value address for A. Then, the sparse tensor core performs fast matrix multiplication of the compressed matrix of A (with the metadata) and the dense matrix B.

NVIDIA has evaluated the effectiveness of sparse tensor cores on various networks [20], including networks spanning vision, object detection, segmentation, natural language modeling, and translation. The evaluation indicates that one can prune a model into 2:4 structure, reduce the number of weights by half, run it on the sparse tensor cores with 1.3-1.5x speedup in inference speed, and have virtually no loss in model accuracy.

2.2 Unstructured Pruning

Model compression has become a crucial technique to deal with the increasing number of parameters in deep learning models in recent years. *Unstructured pruning* is a model compression technique that reduces the number of parameters by removing the *unimportant* weights in a model. The removed parameters do not need to follow a structure. Han et al. [7] proposed a scheme that measures the importance of weights as their absolute values and prunes away those unimportant ones. They found that they can prune away 90% of the weights from VGG-16 and AlexNet models without accuracy drop on the ImageNet dataset.

Unstructured pruning makes the matrix sparse, so it needs to encode the matrix into special formats to reduce memory usage by skipping the zeros in the matrix. The commonly used formats include CSC (Compressed Sparse Column) or CSR (Compressed Sparse Row). However, matrix operations, e.g., matrix multiplication, on sparse matrices requires special hardware and library support

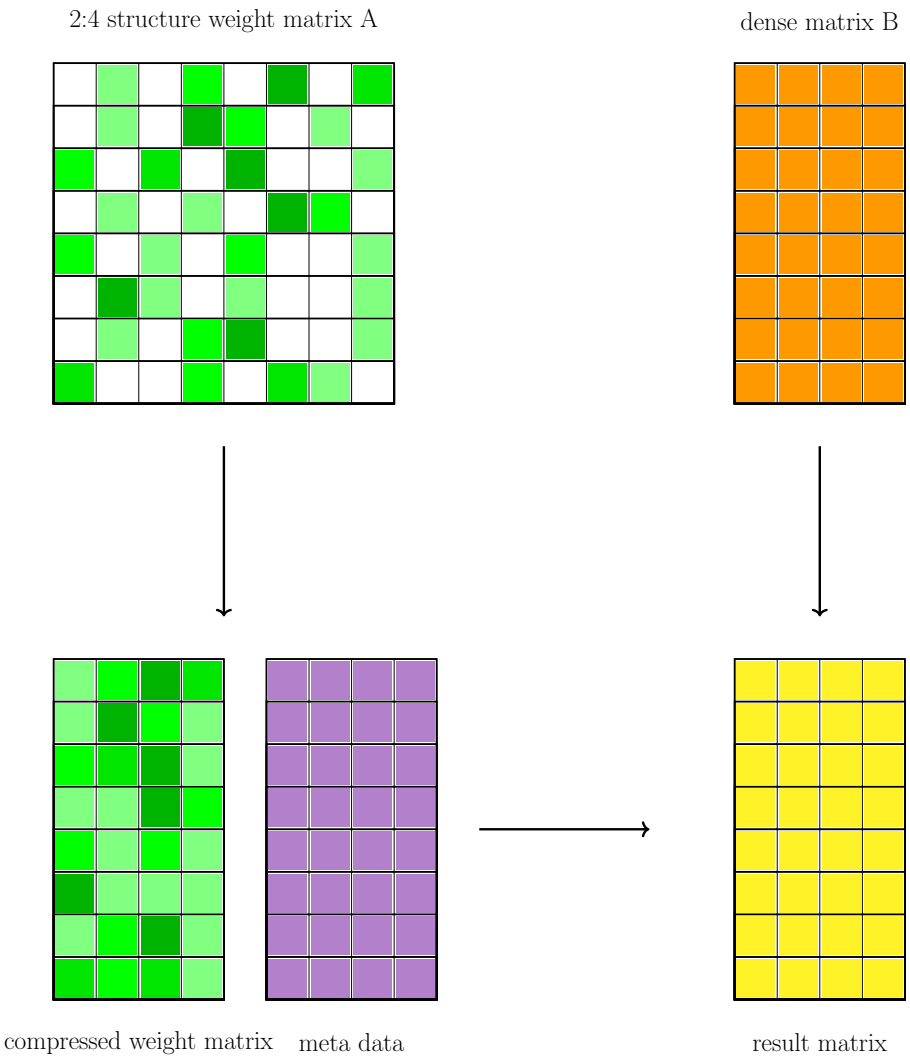


Figure 1: Sparse matrix multiplication on the sparse tensor core.

due to the sparsity of matrices. The hardware and library designed for dense matrix operations will not provide high training and inference speed for sparse matrix after pruning.

2.3 Structured Pruning

Structured pruning avoids the issues of unstructured pruning by reducing the number of parameters of the models in a *structured* manner. That is, the removed parameters must follow a structured rule. For example, structured pruning may remove the model's channels, filters, or stripes and keep the model structured. As a result, it can reduce the model size and improve the inference speed simultaneously without specialized hardware and libraries.

Filter pruning is one of the most common techniques of structured pruning. Filter pruning determines the importance of filters in CNN models and removes unimportant filters. Li et al. [14] proposed a method that removed the filters with the smallest sum of the absolute values of the filter weights. Chen et al. [2] also proposed a method based on weight magnitude that improves upon Li's method. They selected the filters by computing the L1-norm of output filter weights instead of input filter weights. Other works for filter pruning adopt a data-driven approach. Hu et al. [12] proposed a method based on the feature map values. They prune the filters with a higher percentage of zero

of its output feature map. Luo et al. [16] proposed a metric that determines the filter importance according to the entropy of the output feature map. They remove the filters with lower entropy, which have less information passing through the filter, and seem unimportant.

2.4 Relation to our Work

This paper proposes a new hybrid pruning method that combines the filter and 2:4 prunings to take advantage of the NVIDIA sparse tensor cores. We prune the filters of the CNN models first, then apply a 2:4 pruning to ensure that the model is in 2:4 format. We also propose two new hybrid metrics to select filters since hybrid pruning combines two pruning methods. The selection algorithm must consider the importance of two different pruning methods simultaneously. The traditional methods of filter pruning, which are not aware of the new tensor core hardware, do not consider the weights that might be pruned during the 2:4 pruning, leading to more loss of accuracy. With our method, we can preserve the filters that are both important for filter pruning and 2:4 pruning. From our experiment, we observe that our hybrid pruning achieves higher accuracy than the results reported from Li et al. [14], and Chen et al. [2].

3 Background

This section describes the notations and the background of parameter pruning used in this paper.

3.1 Filter Pruning

Before introducing filter pruning, we describe the notations of the convolution layers. Let $\mathrm{x}_{l,i} \in \mathbb{R}^{h_l \times w_l}$ be the i-th channel of the *input feature map* in the l-th convolution layer, where c_l is the number of the channels, h_l and w_l are the height and width of the input feature map respectively. Let x_{l+1} be the *output feature map* in the l-th convolution layer. Note that x_{l+1} is also the input feature map of the $l+1$-th layer.

In a convolution layer, we compute the output feature map $\mathrm{x}_{l+1,i}$ by applying a filter $F_{l,i} \in \mathbb{R}^{c_l \times s_l \times s_l}$ on the input feature map x_l, as indicated in Equation 1, where $\otimes$ is the convolution. Note that s_l is the kernel size of the convolution filter.

$$\mathrm{x}_{l+1,i} = F_{l,i} \otimes \mathrm{x}_l \tag{1}$$

Filter pruning is a critical technique to remove unnecessary/unimportant filters from a convolution neural network. For example, if we remove a filter $F_{l,1}$, then the corresponding channel in the output feature map, i.e., $\mathrm{x}_{l+1,1}$, will also be removed. In addition, the filters of the next layer, i.e., $F_{l+1,j}$, will also be affected by the removal of this filter. Its dimension now becomes $\mathbb{R}^{(c_{l+1}-1)\times s_{l+1} \times s_{l+1}}$.

Please refer to Figure 2 for an illustration of the effects of removing a filter. The filter pruning removes objects in dash lines. Since $F_{l,1}$ is pruned, $\mathrm{x}_{l+1,1}$ will no longer exist. Filter pruning will also remove the first channel of all filters $F_{l+1,j}$ in the next layer.

3.2 Filter Importance

There are many metrics to determine the importance of filters in filter pruning. Most traditional methods compute the importance using the value of weights.

3.2.1 Input L1 norm method

Li et al. [14] proposed a $L1$-norm based criteria, which is a classical metric to decide the filter importance. The metric determines the importance of filter $F_{l,i}$ by computing the sum of the absolute values of weight within a filter.

We define the importance of a filter as the *average* L1 norm of its weights. We use *Input L1* to denote this metric. The importance $V_{l,i}$ of the filter $F_{l,i}$ is computed by Equation 2, where $\|\cdot\|$ is

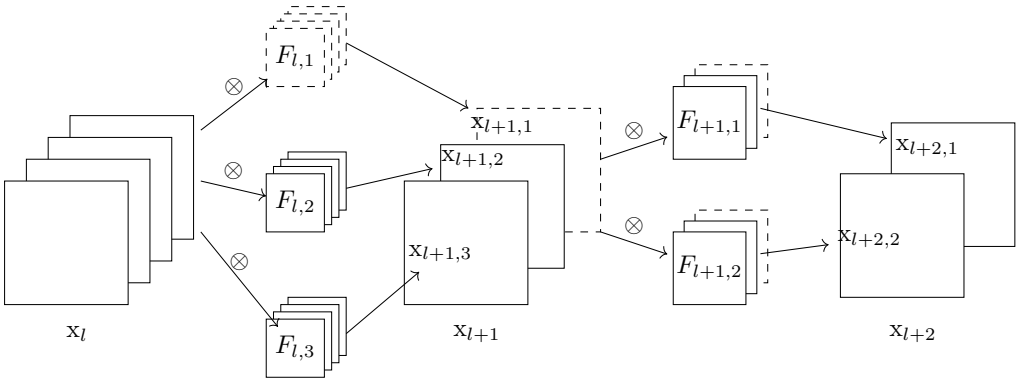


Figure 2: An illustration of filter removal.

denoted as L1-norm of a multi-dimensional matrix. The reason for taking the average is that the size of filters may be very different among different layers.

$$V_{l,i} = \frac{\|F_{l,i}\|}{c_l s_l^2} \tag{2}$$

3.2.2 Output L1 norm method

Chen et al. [2] proposed an *output L1-norm* metric to improve upon the classical L1-norm metric. Instead of computing the L1 metric from the input channels, the output L1 metric computes the L1 metric from the output channels. With the output $L1$-norm metric we compute the importance ($v_{l,i}$) of a filter($F_{l,i}$) in layer l by Equation 3. That is, we compute the average absolute values of weights on channel i for all filters(k denotes the k-th filter) in the following next layer ($F_{l+1,k,i}$).

$$v_{l,i} = \frac{\sum_{k=1}^{c_{l+2}} \|F_{l+1,k,i}\|}{c_{l+2} s_{l+1}^2} \tag{3}$$

3.3 2:4 Pruning

NVIDIA has developed a universal and straightforward recipe for pruning deep neural networks using the 2:4 structured sparsity pattern to accelerate inference speed on deep learning models with the sparse tensor core. After training a model with dense weights distribution, we apply a 2:4 pruning on the weight matrix to make it a 2:4 structure. A 2:4 pruning sets two weights with smaller absolute values to zero in a group of 4 weights in a row, as described earlier. Figure 3 shows an example of 2:4 pruning.

2.1	-1.2	1.3	0.5	1.1	1.5	1.7	0.8
0.2	-1.3	0.6	-0.1	-0.9	-0.3	0.2	1.3
1.5	0.1	-0.4	0.8	0.2	-1.3	0.1	-1.1
0.5	-0.9	0.0	1.0	1.5	-0.7	0.9	0.0

Origin matrix

→

2.1	0.0	1.3	0.0	0.0	1.5	1.7	0.0
0.0	-1.3	0.6	0.0	-0.9	0.0	0.0	1.3
1.5	0.0	0.0	0.8	0.0	-1.3	0.0	-1.1
0.0	-0.9	0.0	1.0	1.5	0.0	0.9	0.0

Pruned matrix

Figure 3: In 2:4 Pruning, we will set two of the elements in each four of filter values that have smaller absolute value to zero.

Note that after the weights matrix is in a 2:4 pattern, we still need to fine-tune the non-zero weights by retraining the model to reduce accuracy loss.

4 Scheme

We introduce the *hybrid pruning* that combines the general filter pruning and the special 2:4 pruning for the sparse tensor core to accelerate the inference on CNN models. It is vital to use proper criteria to preserve the essential filters in both filter pruning and 2:4 pruning. Our scheme considers the importance of both prunings and removes the filters that are not important to a combined metric from both prunings.

4.1 Hybrid Pruning

Hybrid pruning uses the following three steps to convert an input model to a pruned 2:4 structured model so that it can utilize tensor core hardware to reduce inference time.

1. A filter pruning removes unimportant filters in three stages.
   (a) Evaluate the importance of every filter.
   (b) Remove those unimportant filters.
   (c) Fine-tune the pruned model.

   The filter pruning repeats until it has removed a significant amount of filters to reduce the size of the model for much more efficient inference.
2. Apply a standard 2:4 pruning to convert the model into a 2:4 structure. The goal is to further reduce the inference time by taking advantage of the tensor core hardware, which mandates that the input matrix must be in a 2:4 structure.
3. Retrain the model with a fixed number of epochs to improve the accuracy.

Note that the second step takes the result from the first step and runs it automatically on the hardware. This process is standard and is not the focus of this paper. The third step is also a standard process after applying pruning to a model. Instead this paper focuses on the first step, which selects important filters for the second step. We emphasize that the pruning criteria should consider the 2:4 pruning afterward. For ease of explanation, we will use *filter pruning* to refer to the first step in the rest of the paper.

4.2 Pruning Criteria

We now describe the metrics in the filter pruning, the first step of hybrid pruning (Section 4.1). All our metrics consider the effect of the 2:4 pattern.

4.2.1 Remanent L1-norm Metric

We propose a *remanent L1-norm metric* to measure the importance of the filters. Traditional input L1-norm considers *all* the weights in a filter. However, the 2:4 pruning will remove weights so that the weight matrix is in the 2:4 pattern; we need to consider that when defining our pruning criteria. As a result, we propose a remanent L1-norm metric that considers only those weights that will *remain* after 2:4 pruning. We partition every row of the weight matrix into groups of 4 elements, as shown in Figure 3. Let $w'_{l,i,k}$ be the sum of the two largest absolute values from the k-th group of the filter $F_{l,i}$, and n_l be the number of groups in $F_{l,i}$. Now by the remanent L1-norm metric we define the *remanent importance* $r_{l,i}$ of the i-th filter in the l-th layer ($F_{l,i}$) as in Equation 4. The remanent L1-norm metric only considers the two most significant elements out of four elements in each group.

$$r_{l,i} = \sum_{k=1}^{n_l} w'_{l,i,k} \tag{4}$$

4.2.2 Dominant Ratio Metric

We proposed the *dominant ratio metric* as another metric for pruning. The remanent L1-norm metric preserves the filters with the most significant remanent weights but does not consider the magnitude of the weights it prunes. For example, the metric may remove a vital weight because it is in the same group with two even larger weights. This removal may cause accuracy loss, and we should address this issue.

We then define the *dominant importance* of a filter as the ratio between the sum of all weights and the sum of all weights pruned, as in Equation 5, where $w_{l,i,k}$ be the sum of the absolute values from all four elements from the k-th group of the filter $F_{l,i}$.

$$d_{l,i} = \frac{\sum_{k=1}^{n_l} w_{l,i,k}}{\sum_{k=1}^{n_l}(w_{l,i,k} - w'_{l,i,k})} \tag{5}$$

We propose to use this *dominant ratio metric* to select filters. Those filters with higher dominant importance will remain in the model since they better align with the 2:4 structure. It is easy to see that when the sum of the two larger weights is much more significant than the two smaller weights, the larger the dominant importance becomes. Intuitively the higher the dominant importance is, the more critical the filter becomes because the 2:4 pruning later will not affect the accuracy.

4.2.3 Preservation Importance Metric

We define the *preservation ratio* for a group as the fraction of its weights that the 2:4 pruning will preserve. Formally we use $\alpha_{l,i,k}$) to denote the preservation ratio of the k-th group of the filter $u_{l,i}$ by the ratio between $w'_{l,i,k}$ and $w_{l,i,k}$, as in Equation 6.

$$\alpha_{l,i,k} = \frac{w'_{l,i,k}}{w_{l,i,k}} \tag{6}$$

We then define the preservation ratio for a filter. We define the *preservation importance* $p_{l,i}$ of filter $u_{l,i}$ as the harmonic mean of the preservation ratio of all its groups, as indicated by Equation 7. The harmonic mean is the inverse of the average of the inverse of its elements.

$$p_{l,i} = (\frac{\sum_{k=1}^{n_l} \alpha_{l,i,k}^{-1}}{n_l})^{-1} \tag{7}$$

4.3 Combination of Two Metrics

We define a *hybrid metric* that considers importance from both filter pruning and 2:4 structure pruning. We use u to denote the importance from filter pruning v for the importance from 2:4 structure pruning. There are many possible u's and v's. For example, we can choose output L1-norm as u and preservation importance metric as v. Figure 4 plots every filter as a dot with u (output L1-norm) as the x-coordinates and v (preservation importance L1-norm) as the y-coordinate. These are the 384 filters in the third convolutional layer of AlexNet. We need to define a metric to combine u and v into a hybrid importance function h. A simple hybrid metric h is the summation of u and v, i.e., $h = u + v$. If we use h to select filters, then it is like we use a line $u + v = c$ to sweep through filters by decreasing c. The line (with slope -1) will go from the upper right corner towards the origin and select filters it meets first.

In practice, we need to find a good h function that combines the importance of u and v to achieve high accuracy and efficient computation performance. We will consider both linear and non-linear h function combine u and v, and enrich the possible importance orders of our hybrid metric.

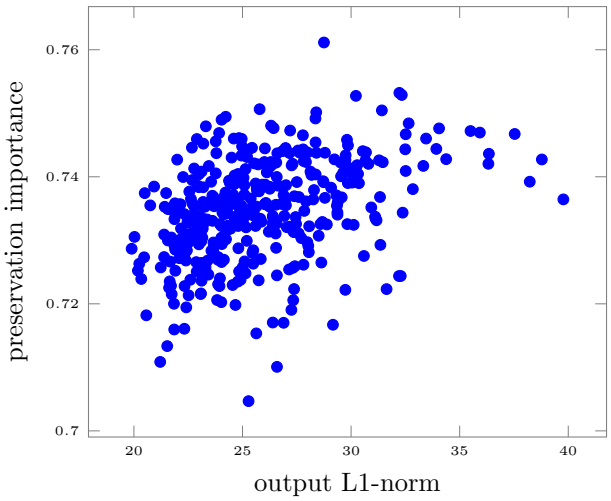


Figure 4: The distribution of output L1-norm and preservation importance metric.

We use a linear function h to combine u and v. Let h denote the new hybrid metric, which is a linear combination of u and v, as in Equation 8, where k is a number between 0 and 1.

$$h = (1 - k)u + kv \tag{8}$$

The linear function in Equation 8 provides a trivial criterion to determine whether one filter is more important than the other. For example, Figure 5 shows two filters separated by a linear function $h_{l,i} = \frac{2}{3}u_{l,i} + \frac{1}{3}v_{l,i}$. The slopes of lines to separate the filters by their importance are -2. Filter n_2 is more important than filter n_1 according to this metric, because n_2 is at the top-right side and n_1 is at the bottom-left side of the metric function.

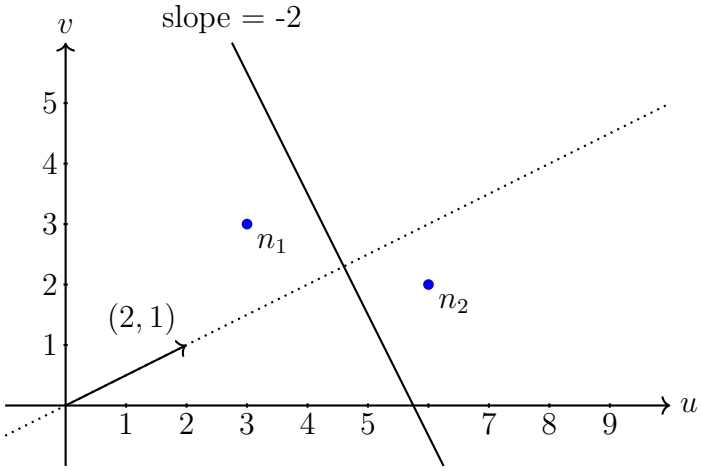


Figure 5: Two filters separated by a linear function h of slope -2.

It is difficult to determine whether a linear function (Equation 8) is sufficient to determine the relative importance of two filters. The only variable a linear function can control is its slope. We do not know if it is a fact that a line can separate all the important filters from unimportant ones. Even we do know this is true; it is still hard to determine the slope of this line.

We want to explore more possible ways to transform one or even two metrics with non-linear functions. We will denote these functions as *transformation functions*. That is, by applying a non-linear transformation function on metrics, then compute the linear function Equation 8. The goal

of the non-linear transformation functions is to change the relative order of the values among the two metrics and make further optimization possible.

We propose two non-linear transformation functions – *ranking* and *square of distance-to-minimum*. Both functions consider the metric values of their filters and require the metric values of other filters. As a result, these two functions map a vector of metric values from all filters to a vector of metric values.

Ranking considers only the ordinal number of the metric values (rank), not the metric values. As a result, the co-domain of the metric function is the set of permutations from 1 to the number of filters. We first calculate the metric values for all filters, then we sort the metric values in ascending order. By definition, the rank of a filter is the index of the filter in the sorted order. A filter with a larger metric value will have a larger rank.

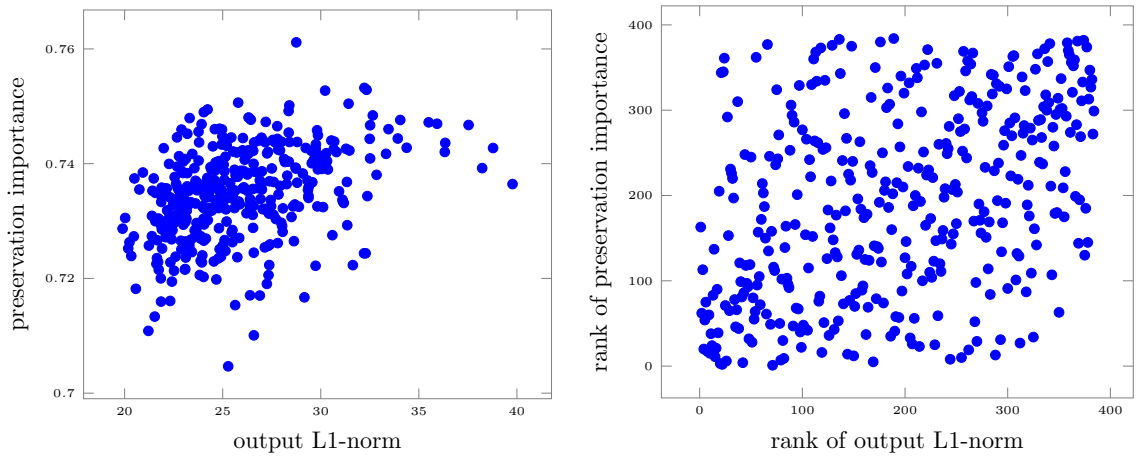
Square of distance-to-minimum is the second non-linear transformation function we use. The function value of the i-th metric values (s_i) is the square of the difference between the i-th metric value and the *minimum* metric value among all filters, as in Equation 9, where m^* is the minimum among all m_i, where i is from 1 to the number of filters.

$$s_i = (m_i - m^*)^2 \tag{9}$$

4.4 Two Hybrid Metrics

According to our scheme, we select the two metrics, u and v, for our hybrid metrics with the following two criteria. The first metric u should perform well in general filter pruning. The second metric v should reflect the unchanged ratio of a filter after 2:4 pruning. For the metric u, we select output L1-norm because output L1-norm performs better than input L1-norm in most cases. For the metric v, we choose preservation importance metric from the three metrics in Section 4.2. Preservation importance represents the overall ratio of preserved weight of groups and avoids the influence of the magnitude of all groups in a filter.

We propose two hybrid metrics that combine filter pruning and 2:4 structure pruning. The first hybrid metric uses ranking of output L1-norm as u, and ranking from preservation importance as v. This hybrid metric considers only the ranks, not the actual values, in output L1-norm and preservation importance. Figure 6 compare the distribution of filters in the u-v plane before and after applying the ranking transformation.

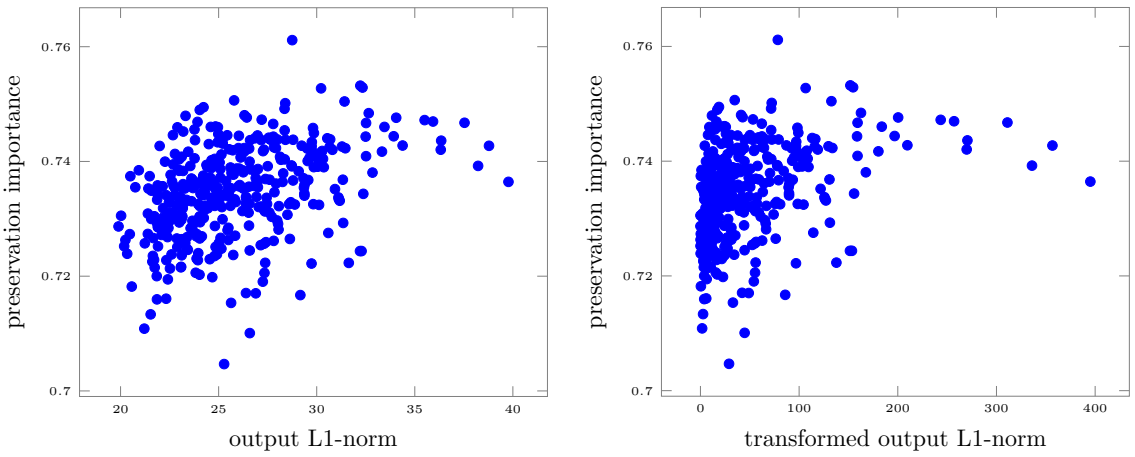


(a) Before (b) After

Figure 6: The distributions before and after performing the transformations in *hybrid metric 1*.

The second hybrid metric uses the square of distance-to-minimum from output L1-norm as u, and the preservation importance metric without transformation as v. Applying the square of distance-to-minimum transformation on output L1-norm emphasizes its contribution. Figure 7 compares the

distribution of filters on the u-v plane before and after applying the square of distance-to-minimum to output L1-norm. Since the magnitude of u is larger than that of v, it dictates the importance of most filters. Only those filters with the lowest output L1-norm metrics will consider preservation importance.



(a) Before (b) After

Figure 7: The distributions before and after performing the transformation in *hybrid metric 2*.

We observe Figure 6 and Figure 7 and conclude that by using the transformation functions, both hybrid metrics can spread out the distribution of filters more evenly in the u-v plane than without. As a result, it should be easier for both metrics to find a good separation line while using Equation 8 to select essential filters.

5 Evaluation

5.1 Experiment Settings

We mainly use AlexNet [13] in the experiments. AlexNet is a traditional CNN model with only five convolutional layers and is shallower than most other CNN models. As a result, if we do not keep the essential filters, the accuracy will drop significantly since the model is small and shallow. We also run accuracy testing experiments on VGG16 model [26] and the SimpleNet model [8]. Both models have more convolutional layers than the AlexNet so that we can compare our approach with other prunings on complex models.

Our experiments use four datasets – MNIST, Street View House Numbers (SVHN), CIFAR-10, and CIFAR-100. The MNIST dataset consists of 70,000 grey-scale images of 28×28 resolutions for 0-9 digits. These images consist of 60,000 training images and 10,000 testing images. The other three datasets all have 32×32 color images, which are different from those in the MNIST dataset. The SVHN dataset contains 73257 training images and 26032 testing images. The CIFAR-10 dataset consists of 50000 training images and 10000 testing images in 10 classes. The CIFAR-100 dataset has 50000 training images and 10000 testing images, and they are in 100 classes.

We use the PyTorch [22] framework for fine-tuning the model and the NVIDIA cutlass library[19] for testing the sparse tensor core performance. We conduct all the experiments on a server with one GeForce RTX 3090 GPU.

There are two sets of experiments. First, we compare our metrics with the input L1-norm and output L1-norm methods during hybrid pruning in accuracy tests. Then, we compare the speedup of hybrid pruning with that of only applying filter pruning or only 2:4 pruning.

5.2 Performance of Pruning Methods

We first describe the accuracy results from two sets of filter prunings. All filter prunings are *global*, i.e., they evaluate and compare the importance of *all* filters in *all* convolution layers and prune those that are not important.

- The first metric set does not consider the 2:4 pattern in the first step of hybrid pruning (Section 4.1), and include input L1-norm and output L1-norm.
- The second metric set does consider the 2:4 pattern in the first step of hybrid pruning (Section 4.1), and includes the remanent L1-norm metric, the dominant ratio metric, preservation importance metric, and the two hybrid metrics from Section 4.4.

Note that for the following experiments, we use 0.5 as the k value in Equation 8 for hybrid metrics if the value is not mentioned to compare with the results of using other metrics.

5.2.1 AlexNet

Figure 8, Figure 9 and Figure 10 depicts the accuracy of AlexNet model on the MNIST, SVHN, and CIFAR-10 datasets, under different percentages of remaining filters.

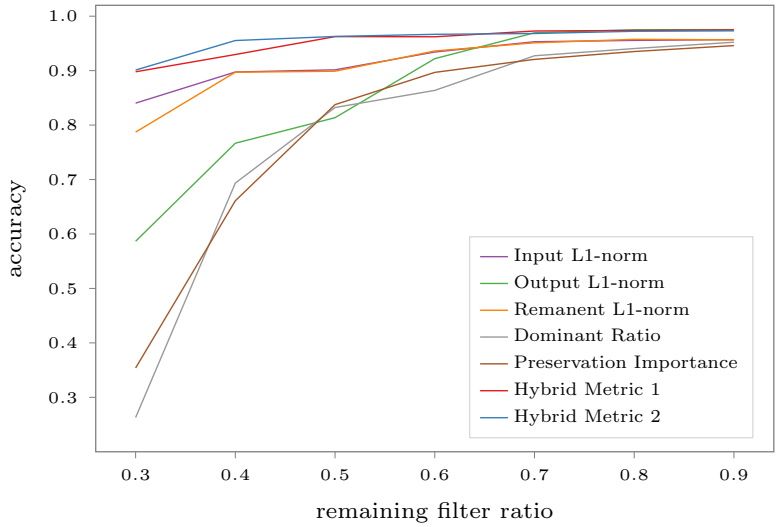


Figure 8: The accuracy of the AlextNet Models on MNIST dataset after applying hybrid pruning.

We observe that the two *hybrid metrics* outperform all other metrics, The hybrid metrics outperform traditional filter pruning methods, e.g., input L1-norm and output L1-norm, because they do not consider the filter weights removed by 2:4 pruning.

Not all metrics considering 2:4 pruning result in high accuracy of pruned models. For example, remanent L1-norm metric only considers the input L1-norm of the weights that will remain after 2:4 pruning. The experiments conclude that remanent L1-norm metric only has a similar performance as the input-L1 norm metric. The dominant ratio metric is another example.

The dominant ratio metric is the worst among all metrics in the experiment. The reason is that groups having a high dominant ratio may not necessarily have significant weights. They only need to be much larger than the other two in a group of four elements. This lack of global comparison could be why the dominant ratio is not a good metric.

5.2.2 VGG-16

In Figure 11, we observe similar results as in Figure 8, Figure 9 and Figure 10, the hybrid metrics perform better than the other metrics and the dominant ratio metric performs the worst. The

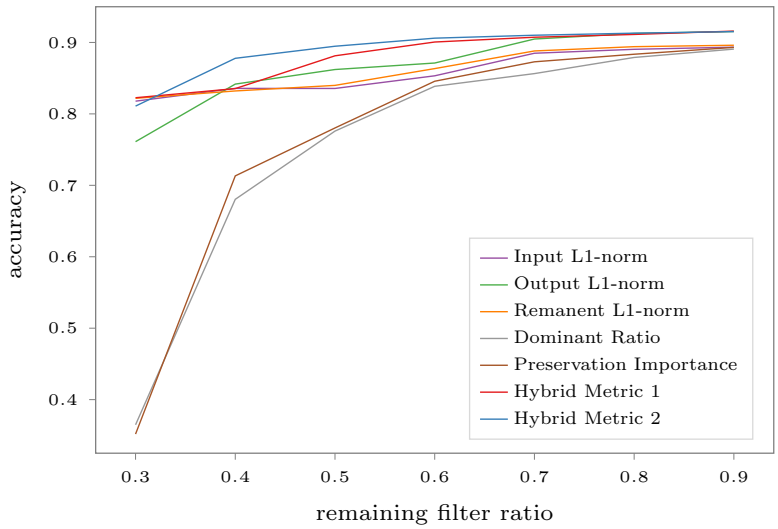


Figure 9: The accuracy of the AlexNet models on SVHN dataset after applying hybrid pruning.

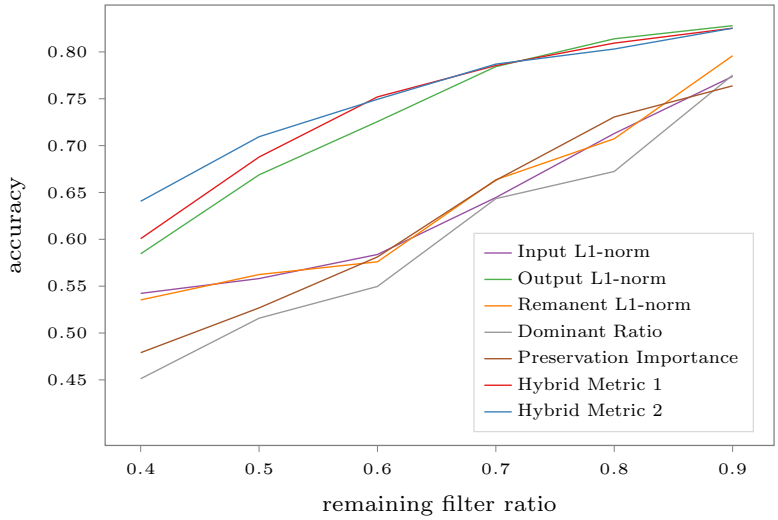


Figure 10: The accuracy of the AlextNet models on CIFAR-10 dataset after applying hybrid pruning.

remanent L1-norm metric performs similarly to the input L1-norm metric.

However, in Figure 12, we observed that the hybrid metrics do *not* perform the best. Hybrid metric 1 is even worse than the input L1-norm metric and remanent L1-norm metric. Hybrid metric 2 performs better than hybrid metric 1 but still worse than remanent L1-norm metric. From Figure 12, we also observe that the accuracy of the output L1-norm metric drops significantly after the percentage of the number of remaining filters drops below 50%, and it drops much faster than input L1-norm metric does. The accuracy drop of the output L1-norm metric affects the hybrid metrics because both of the hybrid metrics use the transformation of output L1-norm metric. Therefore, the hybrid metrics do not perform the best among all metrics for the CIFAR-10 dataset.

5.2.3 SimpleNet

Figure 13 and Figure 14 present the result of the SimpleNet model, which has deeper layers but fewer parameters than the AlexNet model. We observe that the hybrid metrics still outperform the other metrics. *Remanent L1-norm metric* presents a little bit worse than *input L1-norm metric* in

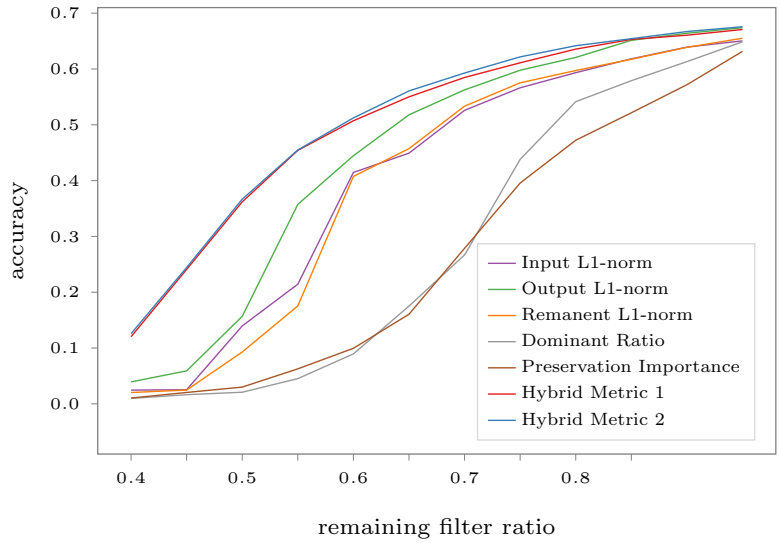


Figure 11: The accuracy of the VGG16 models on CIFAR-100 dataset after applying hybrid pruning.

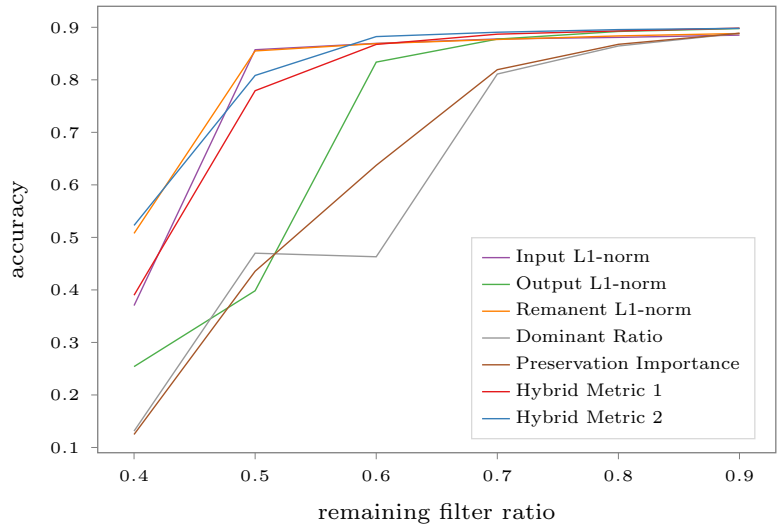


Figure 12: The accuracy of the VGG16 models on CIFAR-10 dataset after applying hybrid pruning.

the SimpleNet model. The accuracy drops sharply by using *dominant ratio metric* which means that this metric can not evaluate filter importance correctly in hybrid pruning.

Figure 11 and Figure 12 present the accuracy of the VGG16 model, which has a large number of parameters and convolutional layers, after hybrid pruning on CIFAR-100 and CIFAR-10, respectively.

From these experiments, we conclude that hybrid metrics outperform other metrics in most cases. The other two metrics that consider both filter pruning and 2:4 pruning perform only satisfactorily. Remanent L1-norm metric performs similarly to the input L1-norm metric does. The dominant ratio metric performs poorly in these experiments.

5.3 The Reversed Order

One might wonder if switching the order of the first and the third steps of the hybrid pruning (Section 4.1) will improve performance. The intuition is that if we first apply the standard 2:4

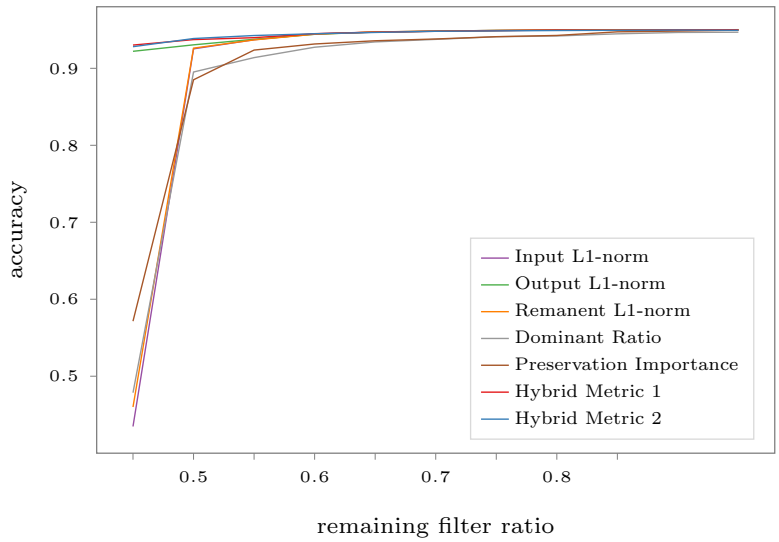


Figure 13: The accuracy of the SimpleNet models on SVHN dataset after applying hybrid pruning.

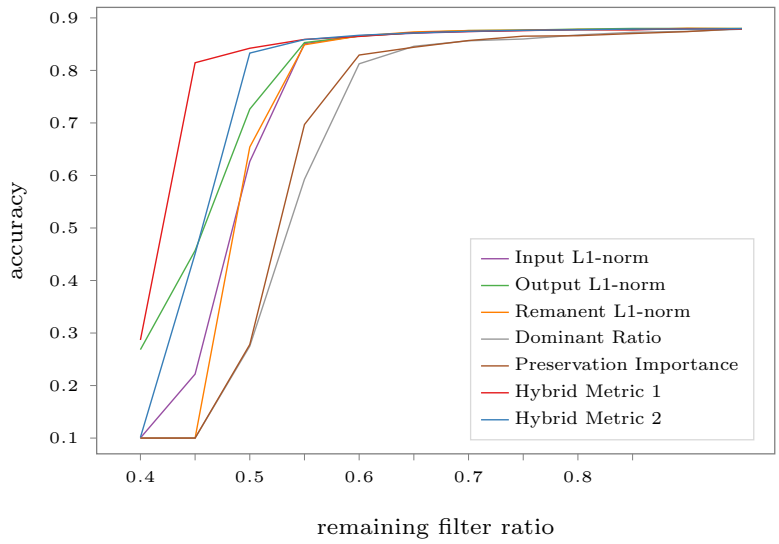


Figure 14: The accuracy of the SimpleNet models on CIFAR-10 dataset after applying hybrid pruning.

pruning to ensure that every group of four elements has at least two zeros, then we can use whatever filter pruning without worrying about the 2:4 structure. We will prune an entire filter, and the 2:4 structure will not change. We will refer to this change of order as *reversed hybrid pruning*.

Figure 15 and Figure 16 compare the accuracy of the original and the reversed hybrid pruning. We tried the reversed hybrid pruning on input L1-norm and output L1-norm after a standard 2:4 pruning. We observe that the reversed hybrid pruning performs similarly to the original hybrid pruning. In some cases, the reversed hybrid pruning achieves higher accuracy than the original one, but only by a small margin.

We believe that this lack of performance gain from reversed hybrid pruning is similar to that of remanent L1-norm metric. Both of them preserve the filters that have a more significant absolute sum of remaining weights. Therefore, reversing the order of hybrid pruning will receive similar results as that of remanent L1-norm metric, which is similar to input L1-norm and output L1-norm

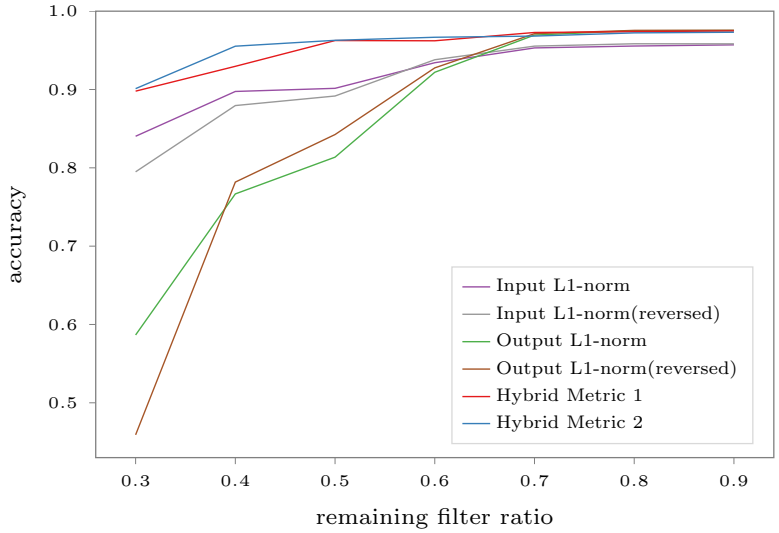


Figure 15: The accuracy of the AlexNet models on MNIST dataset after applying hybrid pruning.

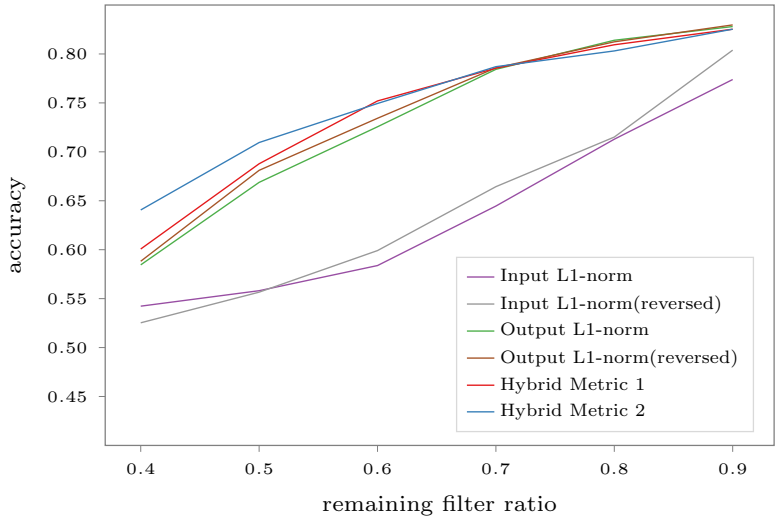


Figure 16: The accuracy of the AlexNet models on CIFAR-10 dataset after applying hybrid pruning.

metrics.

5.4 Linear Transformation of Hybrid Metrics

We study the effect of k on the accuracy in Equation 8 for hybrid pruning. Recall that the constant k controls the linear combination of u and v in Equation 8. For example, let u and v be output L1-norm and preservation importance metric. Then when $k = 0$, the hybrid metric becomes output L1-norm, and when $k = 1$, the hybrid metric becomes preservation importance.

Figure 17 shows the accuracy of the AlexNet model on SVHN and CIFAR-10 datasets when 40% of filters remain. We note that the best k values for different models are different. The best k is 0.8 with 88.2% accuracy in SVHN, and the best k is 0.9 with 62.6% accuracy in CIFAR-10.

We observe that the best k for different percentages of filters remain can be different. For example, Figure 18 shows the accuracy of VGG16 model on CIFAR-10 dataset when 40% and 50% filters remain. The accuracy drops when k is between 0.5 and 0.7. We conclude that the hybrid

metrics prune critical filters when k is between 0.5 and 0.7. As a result, we have two local maximums of accuracy as a function of k.

We also study the effect of k with the two proposed hybrid metrics. Figure 19 shows the accuracy of AlexNet and VGG16 models on CIFAR-10 dataset when pruning off 60% filters with the two hybrid metrics. There are multiple local maximums in each accuracy as a function of k.

We observe from Figure 17, Figure 18 and Figure 19 and conclude that the maximum accuracy does *not* appear at either endpoints when k is either 0 or 1. That means the hybrid metric can always find a k that produces the maximum accuracy than considering either u or v, i.e., filter pruning or 2:4 structure pruning. Therefore, combining two pruning metrics with a linear function is an effective approach to evaluate filter importance. However, searching for an optimal k might be difficult, considering that the accuracy function (as a function of k) may not have any structural information to leverage.

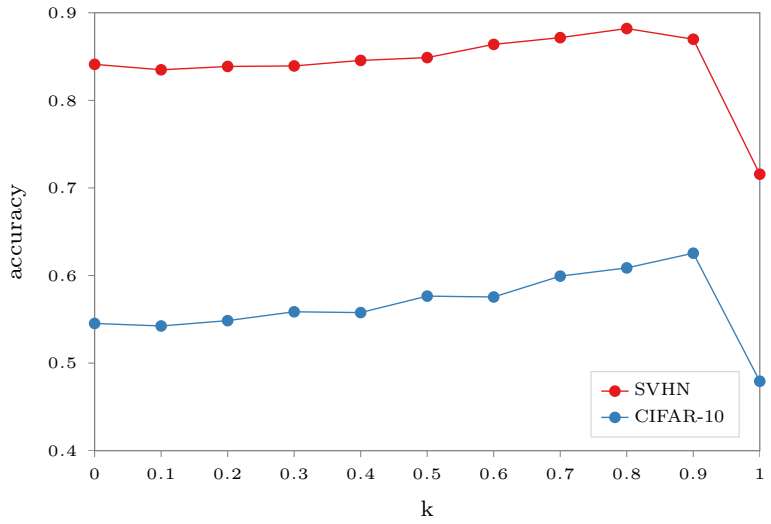


Figure 17: The accuracy of the the AlexNet models on CIFAR-10 and SVHN datasets with remaining filter ratio 0.4.

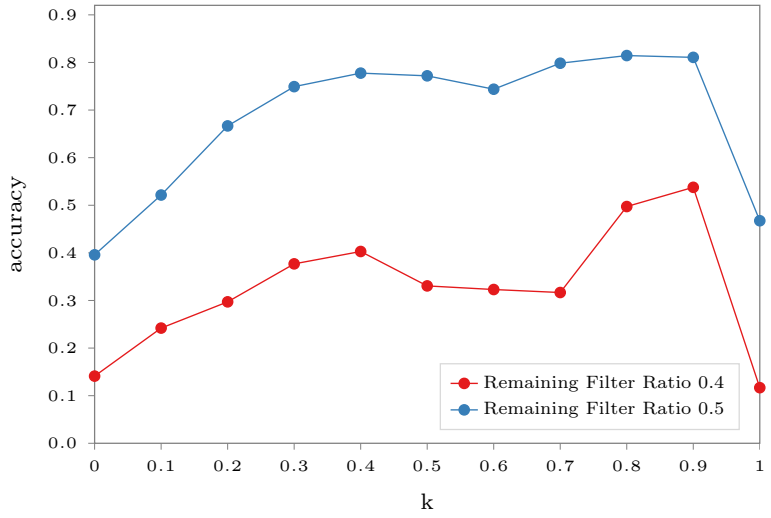


Figure 18: The accuracy of VGG16 on CIFAR-10 dataset with remaining filter ratio 0.4 and 0.5.

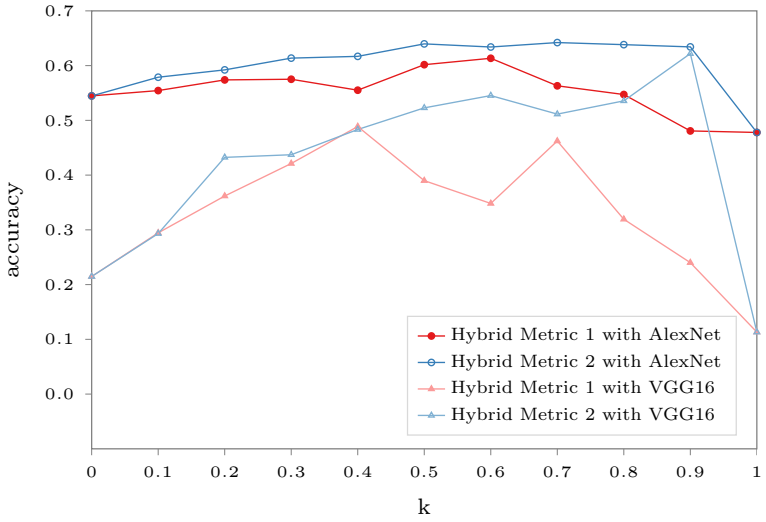


Figure 19: The accuracy of AlexNet and VGG16 models on CIFAR-10 dataset with remaining filter ratio 0.4.

5.5 Speedup of Hybrid Pruning

Hybrid pruning combines filter pruning and 2:4 pruning, reduces the size of a CNN model, and makes it into the 2:4 pattern to run on the sparse tensor core. This experiment compares the speedup of hybrid pruning against using only 2:4 pruning or filter pruning.

Figure 20 compares the inference time on an AlexNet by using/ not using the sparse tensor core, with decreasing percentage of remaining filters. Note that if we do not use the sparse tensor core, we do not need to apply 2:4 pruning. Therefore, the blue bars (not using the tensor core) present the time that only applies filter pruning, and the orange bars (using the tensor core) present the time that involves hybrid pruning. We observe that inference using sparse tensor core is 1.4x-1.5x faster than not using it. This result is similar to the results from NVIDIA [20].

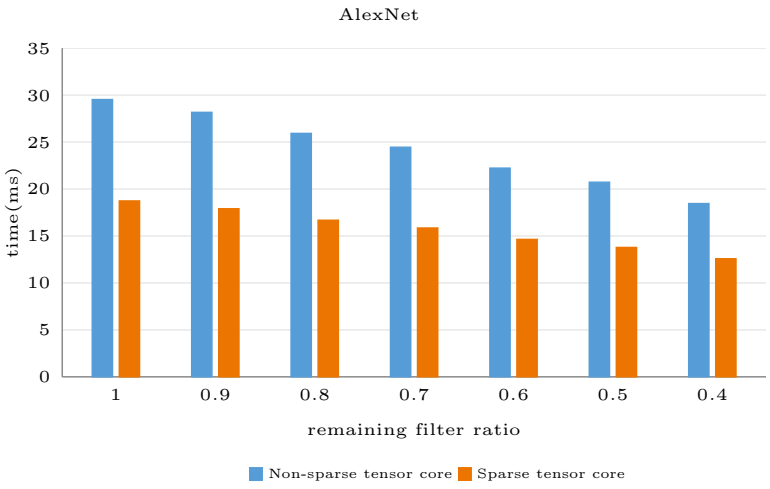


Figure 20: Comparison of the inference time of a batch that using sparse tensor core or not among different remaining ratio of filters by AlexNet model.

From Figure 20 we also observe that the speedup of using the tensor core decreases when the percentage of remaining filters decreases. The reason is that the sparse tensor core only increases the

matrix multiplication speed. The other operations, such as activation or pooling, do not benefit from using the tensor core. While we prune more filters, the percentage of work in matrix multiplication decreases, so the speedup decreases.

Next, we compare the accuracy of using tensor core with hybrid pruning and not using tensor core with only filter pruning. The hybrid pruning removes more parameters than the filter alone does. Even for the same percentage of the remaining filters, the hybrid pruning removes more weights since it will do an additional 2:4 pruning, and it can remove up to two elements out from each group of four.

Table 1 compares the accuracy of filter pruning only and the hybrid pruning, under the different percentage of the remaining number of filters, for the AlexNet model on MNIST dataset. From Table 1 we observe that the accuracy loss is insignificant. For most percentage of the remaining filters, the accuracy loss is about 1%. This loss is insignificant since from Figure 20 we can achieve significant speedup by hybrid pruning.

Table 1: The comparison of the accuracy of the AlexNet models on MNIST dataset with filter pruning, 2:4 pruning and hybrid pruning.

Remaining Filter Ratio	Original Model	2:4 Pruning
1.0	0.983	0.978
-	Filter Pruning	Hybrid Pruning
0.9	0.982	0.975
0.8	0.982	0.974
0.7	0.977	0.973
0.6	0.976	0.962
0.5	0.973	0.962
0.4	0.956	0.930

From Table 1 we also observe that to the same accuracy of 0.973, the filter pruning needs to keep 50% of filters, and the hybrid pruning needs to keep 70% of filters. Even so, from Figure 20 and table 1, we find that the hybrid pruning is still 1.3x faster than the filter pruning for the same accuracy of 0.973.

Next, from Table 1 we observe that the hybrid pruning can remove 50% of the filters with only 1.6% of accuracy drop from 0.978 by using only 2:4 pruning to 0.962. In addition, the hybrid pruning runs 1.36x faster than using the 2:4 pruning only.

5.5.1 Dimension Permutation in 2:4 Pruning

Table 1 shows that the accuracy of a pruned model after 2:4 pruning is lower than the accuracy of the pruned model before 2:4 pruning for at most 2.6%. However, Mishra et al. [18] have proposed a 2:4 pruning scheme to prune models into a 2:4 structure and claimed no loss of accuracy. As a result, we will conduct experiments to clarify the differences between these two 2:4 pruning approaches.

Mishra's workflow is different from how our hybrid pruning prunes the models into a 2:4 structure. First, we fix the number of retraining epochs after 2:4 pruning to five, which is less than the number of the training epochs for most CNN models. Mishra et al. retrain the model after applying the 2:4 structure by repeating the original training session of the model. Second, their 2:4 structure group weights of channels along the dimension of channels; that is, every four weights in a group belong to four different channels. Our 2:4 structure groups weights along the dimension of the image width, so the weights in a group belong to the same channel. Both 2:4 structures can run on the sparse tensor cores.

To examine the effects on the accuracy of the two pruned models from the two 2:4 structures, we modified the second step in our hybrid pruning by grouping filter weights along the dimension of channels as in Mishra's workflow. Other steps in hybrid pruning remain unchanged. We then compare the accuracy of this new 2:4 structure with the original one.

Table 2: Comparison of the accuracy of the pruned models in 2:4 structure and permuted 2:4 structure.

Remaining Filter Ratio	2:4 Structure	Permuted 2:4 Structure
1.0	0.978	0.979
0.9	0.975	0.978
0.8	0.974	0.978
0.7	0.973	0.974
0.6	0.962	0.971
0.5	0.962	0.958
0.4	0.930	0.940

Table 2 shows that the accuracy results of the 2:4 structure along the width and along the channels are similar. The accuracy difference is at most 1% for the same remaining filter ratios. Therefore, we believe that the accuracy drop after 2:4 pruning is due to fewer retraining epochs in hybrid pruning, not the 2:4 structure.

5.6 Filters Pruned by The Two Hybrid Metrics

We study the difference between the filters selected by two hybrid metrics. We apply hybrid pruning to the AlexNet model twice, once with the first hybrid metric and the second hybrid metric, using the same remaining filter ratio. We prune a set of filters for each hybrid metric and keep the remaining ones. Since the two pruned models have the same remaining filter ratio, the numbers of pruned filters are the same.

We calculate the percentage of the filters pruned by *both* metrics, among the filters they pruned individually. We also compare the remaining filters in the pruned models. We calculate *remaining filter intersection ratio* with the same way of calculating *pruned filter intersection ratio* but with the two sets of remaining filters.

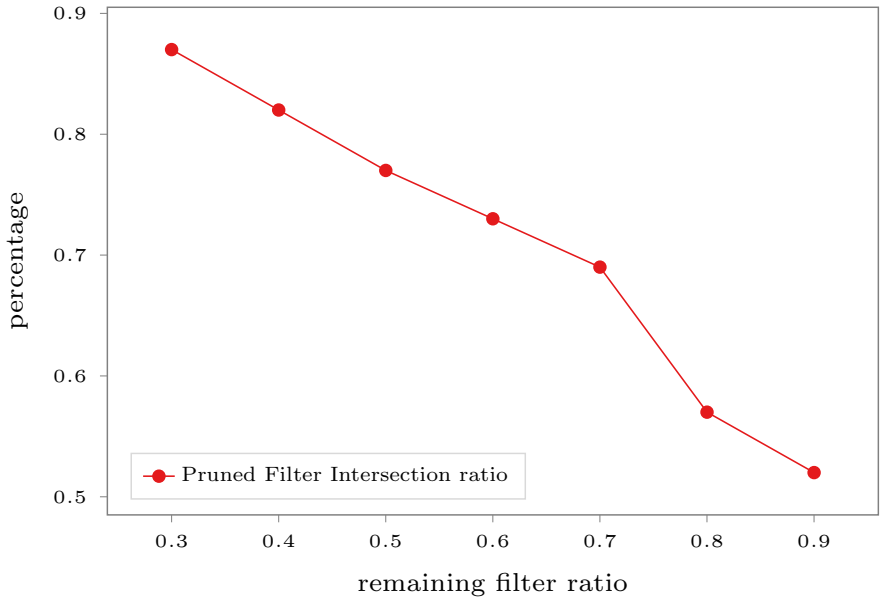
Figure 21a shows the percentages of the filters that are pruned by *both* metrics, among the filters they pruned individually. The percentages are from an AlexNet using MNIST as input and different remaining filter ratios. When the two metrics prune away 10% filters, about half of the pruned filters are the same. Also, from Figure 21b, we observe that accuracy from both pruned models is similar. Therefore, when we prune away only a few filters, the metrics may prune away a different set of filters without significant accuracy loss.

When we prune more filters, i.e., from 10% to 70%, the percentage of the filters pruned by both metrics increases from 52% to 87%. This convergence on pruned filters by both metrics indicates they must agree on what to keep since they can now only keep 30% of the filters. This consensus is crucial since the accuracy will drop if they prune away essential filters. From Figure 21b, we observe that accuracy from both pruned models is still higher than 90%, which suggests that not only do the two metrics have a consensus on what to keep, and this set of kept filters is indeed important. A metric must carefully select filters to prune when it needs to prune away a high percentage of filters since the remaining few filters are more crucial to the accuracy of the pruned model.

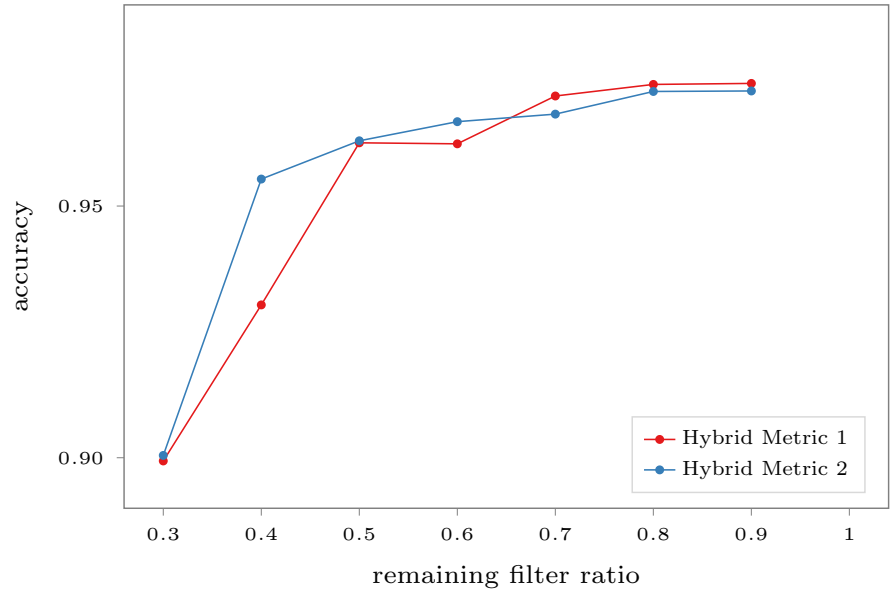
We observe from Figure 21a and 21b that despite that, two hybrid metrics agree upon more than 80% of the 60% filters they will prune, the accuracy drops sharply. We believe that the accuracy drop is because although the two metrics have consensus on 80% of the filter they want to prune, hybrid metric 1 failed to avoid pruning a few critical filters, and hybrid metric 2 did. As a result, hybrid metric 2 can still obtain high accuracy. On the other hand, when the remaining filters are only 30%, both metrics cannot keep enough critical filters, and both metrics' accuracy drops.

5.7 Retraining in Filter Pruning

The pruning mechanism (Section 4.1) benefits from both the correct selection of filters to prune and the retrain after the removal of the filters. We now demonstrate that the pruning benefits from the correction selection, even without the retrain.



(a) The percentages of the filters that are pruned by both metrics with the same remaining filter ratio.



(b) The accuracy of pruned models by the two hybrid metrics.

Figure 21: Comparison of the filters from the pruned models by hybrid metric 1 and hybrid metric 2 with the AlexNet model on MNIST.

Figure 22 compares the loss function of a simple output-L1 metric and our hybrid metrics after filter pruning under different pruning ratios and with or without retraining. First, we observe that retraining after pruning filters does reduce the loss of a pruned model (from light lines to dark lines), no matter what metric we used. This loss reduction becomes apparent when we prune away more than half of the filters in Figure 22. With a sufficient number of epochs of retraining and an adequate amount of remaining filters, the pruned model may recover its loss to that of the original model.

From Figure 22 we observe that the loss from hybrid metric 2 *without* retraining is already lower than the loss of the simple output L1-norm metric *with* retraining (the light blue line and the dark

red line). The loss from hybrid metric 1 without retraining is also lower than the loss of the simple output L1-norm metric without retraining (the light green line and the light red line). Therefore, it is clear that selecting the suitable filters to prune is the reason for the lower loss of the hybrid metrics.

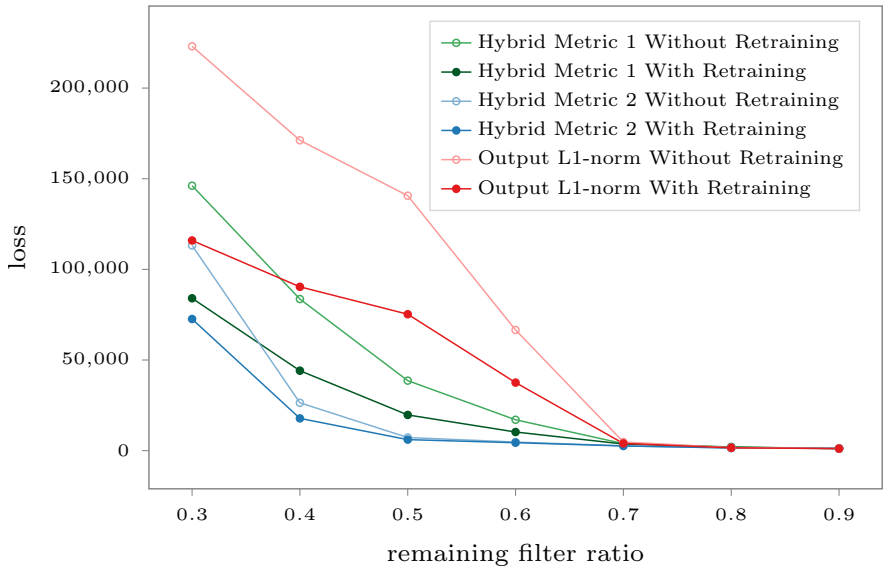


Figure 22: The loss of pruned models with/without retraining after filter pruning with output L1-norm and our hybrid metrics.

One may argue that our hybrid metrics' high accuracy of the pruned models results from retraining, not the hybrid metric that correctly selects filters to prune. Figure 23 shows the accuracy of the pruned models in Figure 22. We observe that the accuracy from hybrid metric 2 only benefits slightly from retraining, and both hybrid metric 1 and the output L1-norm benefit greatly from retraining when we remove half of the filters. We conclude that the high accuracy of hybrid metric 2, the highest among all three metrics, benefits most from the correct selection of filters to prune. For example, the accuracy of hybrid metric 2 without retraining (the light blue line) is already higher than the accuracy from output L1-norm without the retrain (the light red line). This phenomenon is particularly noticeable when we remove up to 60% of filters and note that the accuracy of hybrid metric 2 only improves slightly. We conclude that retraining improves accuracy, but our hybrid metrics benefit more from the correct selection of filters to prune.

6 Conclusion

This paper proposes a hybrid pruning scheme that combines filter pruning and the 2:4 pruning to utilize the sparse tensor core. Traditional pruning does not consider the effect of 2:4 pruning essential to the tensor core hardware. They cannot deal with the case when the standard 2:4 pruning removes the critical weights. Our hybrid pruning uses *hybrid metric* that considers both the L1-norm and the effects of 2:4 pruning. As a result, hybrid pruning preserves the filters essential to filter pruning and (or) the 2:4 pruning. Experiment results indicate that our hybrid pruning improves the inference time by 30% with similar accuracy loss than using only the filter pruning or the 2:4 pruning. We also note that not all methods that consider both pruning steps will improve the accuracy. The experiments show that the remanent L1-norm metric performs similarly to the traditional metric, and the dominant ratio metric performs poorly.

There are some possible future works for our hybrid pruning. For example, we can try to prune the network with a different granularity of structure instead of filters. There have been other granularity proposed in the literature, such as channel pruning [10], stripe pruning [17], and tiles pruning [5].

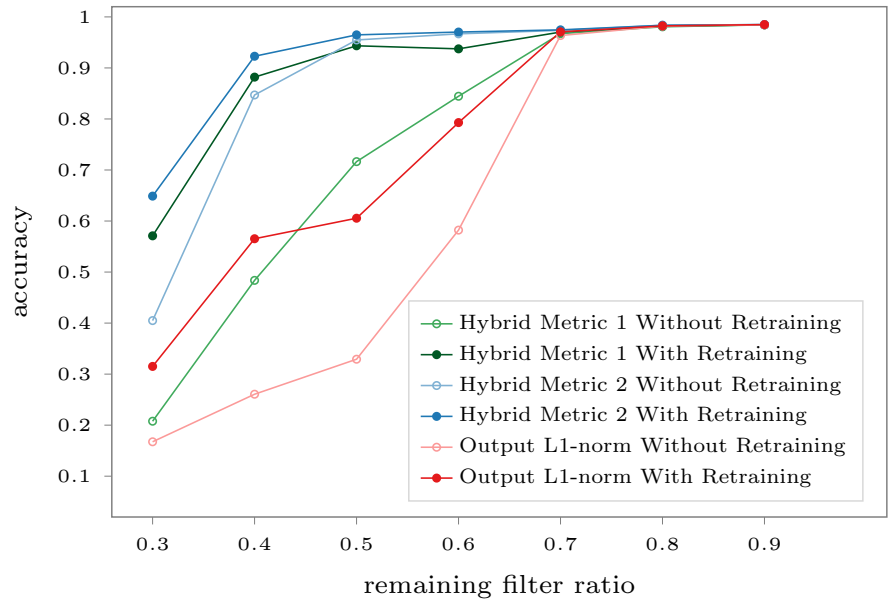


Figure 23: The accuracy of pruned models with/without retraining after filter pruning with output L1-norm and our hybrid metrics.

These pruning methods may have advantages over filter pruning, and some of them might be more suitable for specific structures of CNN and would be more beneficial for hybrid pruning. It may be easier for a pruning method by a specific granularity to make the model a 2:4 structure. We may reduce accuracy loss by pruning in this new granularity and get more speedup from the hybrid pruning and the sparse tensor core.

Our hybrid pruning can improve the inference speed of CNN models without a significant accuracy loss, and the hybrid metrics can help achieve better accuracy with the same amount of computation.

References

[1] Cristian Buciluǎ, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 535–541, 2006.

[2] Tse-Wen Chen, Pangfeng Liu, and Jan-Jan Wu. Exploiting data entropy for neural network compression. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 5007–5016. IEEE, 2020.

[3] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[4] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014.

[5] Cong Guo, Bo Yang Hsueh, Jingwen Leng, Yuxian Qiu, Yue Guan, Zehuan Wang, Xiaoying Jia, Xipeng Li, Minyi Guo, and Yuhao Zhu. Accelerating sparse dnn models without hardware-support via tile-wise sparsity. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15. IEEE, 2020.

[6] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[7] Song Han, Jeff Pool, John Tran, and William J Dally. Learning both weights and connections for efficient neural networks. *arXiv preprint arXiv:1506.02626*, 2015.

[8] Seyyed Hossein Hasanpour, Mohammad Rouhani, Mohsen Fayyaz, and Mohammad Sabokrou. Lets keep it simple, using simple architectures to outperform deeper and more complex architectures. *arXiv preprint arXiv:1608.06037*, 2016.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[10] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE international conference on computer vision*, pages 1389–1397, 2017.

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

[12] Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*, 2016.

[13] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

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

[15] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[16] Jian-Hao Luo and Jianxin Wu. An entropy-based pruning method for cnn compression. *arXiv preprint arXiv:1706.05791*, 2017.

[17] Fanxu Meng, Hao Cheng, Ke Li, Huixiang Luo, Xiaowei Guo, Guangming Lu, and Xing Sun. Pruning filter in filter. *arXiv preprint arXiv:2009.14410*, 2020.

[18] Asit Mishra, Jorge Albericio Latorre, Jeff Pool, Darko Stosic, Dusan Stosic, Ganesh Venkatesh, Chong Yu, and Paulius Micikevicius. Accelerating sparse deep neural networks. *arXiv preprint arXiv:2104.08378*, 2021.

[19] NVIDIA. Cutlass 2.5. *https://github.com/NVIDIA/cutlass*, 2019.

[20] NVIDIA. Nvidia a100 tensor core gpu architecture. *https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf*, 2020.

[21] NVIDIA. Nvidia ampere ga102 gpu architecture. *https://images.nvidia.com/aem-dam/en-zz/Solutions/geforce/ampere/pdf/NVIDIA-ampere-GA102-GPU-Architecture-Whitepaper-V1.pdf*, 2020.

[22] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019.

[23] Hieu Pham, Zihang Dai, Qizhe Xie, Minh-Thang Luong, and Quoc V Le. Meta pseudo labels. *arXiv preprint arXiv:2003.10580*, 2020.

[24] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

[25] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[26] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.