DominoSearch: Find layer-wise fine-grained N:M sparse schemes from dense neural networks

Anonymous Author(s)
Affiliation
Address
email

Abstract

Neural pruning is a widely-used compression technique for Deep Neural Networks (DNNs). Recent innovations in Hardware Architectures (e.g. Nvidia Ampere Sparse Tensor Core) and N:M fine-grained Sparse Neural Network algorithms (i.e. every M-weights contains N non-zero values) reveal a promising research line of neural pruning. However, the existing N:M algorithms only address the challenge of how to train N:M sparse neural networks in a uniform fashion (i.e. every layer has the same N:M sparsity) and suffer from a significant accuracy drop for high sparsity (i.e. when sparsity > 80%). To tackle this problem, we present a novel technique – ***DominoSearch*** to find mixed N:M sparsity schemes from pre-trained dense deep neural networks to achieve higher accuracy than the uniform-sparsity scheme with equivalent complexity constraints (e.g. model size or FLOPs). For instance, for the same model size with 2.1M parameters (87.5% sparsity), our layer-wise N:M sparse ResNet18 outperforms its uniform counterpart by 2.1% top-1 accuracy, on the large-scale ImageNet dataset. For the same computational complexity of 227M FLOPs, our layer-wise sparse ResNet18 outperforms the uniform one by 1.3% top-1 accuracy. Furthermore, our layer-wise fine-grained N:M sparse ResNet50 achieves 76.7% top-1 accuracy with 5.0M parameters[1]. This is competitive to 76.2% top-1 accuracy achieved by the state-of-the-art layer-wise **unstructured-sparsity** model with same number of parameters, which is believed to be the upper-bound of Neural Network pruning with respect to accuracy-sparsity trade-off. We believe that our work can build a strong baseline for further sparse DNN research and encourage future hardware-algorithm co-design work. Our code and models are publicly available at URL[2].

1 Introduction

Modern deep neural networks (DNNs) achieve remarkable success in various tasks such as computer vision and language processing at a cost of memory footprint, computation, energy and storage. These have all become bottlenecks of deploying DNNs on commodity hardware for real-world applications. To address these challenges, many model compression techniques have been proposed to optimize DNN models. Neural network pruning is one promising technique orthogonal to quantization [1], network architecture search [2] and knowledge distillation [3].

Recent innovations in hardware architectures (e.g. Nvidia Ampere Sparse Tensor Core [4]) and N:M fine-grained sparse DNNs pruning algorithms [5, 6] reveal a promising research line of neural pruning that may achieve a high sparsity ratio while maintains a regular sparse structure. On the

[1]Parameters, model size and sparsity are alternately used in this paper. They are equivalent in this context.
[2]will be available on Github upon acceptance. Now it can be found in supplementary material.

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

other hand, traditional unstructured sparsity can achieve relatively higher sparse ratio but cannot maintain regular sparse structure which makes it difficult for acceleration [7]. Traditional structured sparsity can achieve regular sparse structure but may not achieve high compression ratio[5]. Figure 1 illustrates traditional unstructured/structured pruning and N:M fine-grained structured pruning.

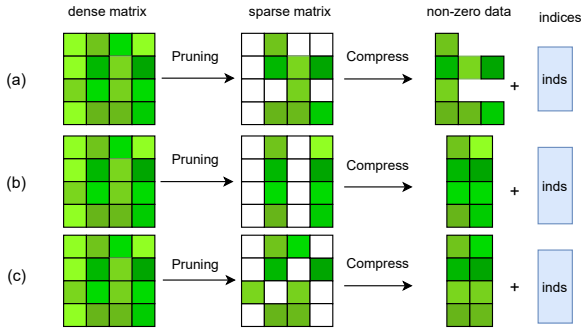

Figure 1: (a) Unstructured pruning treats every weight in the dense matrix individually which will produce a sparse matrix with irregular non-zero data. (b) Coarse-grained pruning treats a filter/channel as an atomic component and can produce a regular sparse matrix. (c) Fine-grained N:M pruning performs like unstructured pruning inside each group (group size M = 4 in this example) while it enforces an additional constraint that there are at most N non-zero values to be kept.

Applying uniform fine-grained N:M sparsity training, or fine-tuning to DNNs, will lead to a sub-optimal solution because layers are treated equally. Modern DNNs tend to have a number of layers with a highly non-uniform distribution of redundancy. This well-known phenomenon has been exploited by recent unstructured pruning methods [8, 9, 10, 11] that have achieved higher accuracy than the uniform counterparts especially for relatively high sparsity. However, the existing methods mainly focus on unstructured pruning that treats weights within a layer individually and does not have fine-grained position constraints, which cannot be used for fine-grained N:M sparsity. As a consequence, a novel method for layer-wise fine-grained N:M sparsity is needed.

Furthermore, sparsity acceleration relies on the underlying software-architecture system of the platform. We can make a reasonable assumption that there will be more but not arbitrary N:M schemes supported in future software-architecture designs. Although the flexibility of N and M is still an open system design research question, we assume a policy of fixed M and flexible N in this paper because it can provide fixed window size (i.e. size = M) for sparse weight matrix compression.

Motivated by above-mentioned challenges, this work introduces an efficient framework to find a layer-wise fine-grained N:M scheme with pre-defined N:M candidates which achieves higher accuracy compared to the uniform counterpart with equivalent complexity constraint (e.g. model size or FLOPs).

We propose DominoSearch. DominoSearch is an iterative algorithm that finds layer-wise fine-grained sparse schemes from pre-trained dense weights. This is achieved using a magnitude-based criterion [8, 12] for the pruning selection threshold, in combination with a weight penalty to drive values towards the threshold. DominoSearch also uses a layer-wise penalty factor to balance the heuristic layer-wise redundancy of parameters[9] and layer-wise computation complexity (e.g. FLOPs). As indicated by the name *Domino*, the N value from the N:M ratio decrements during the search phase with the weight penalty acting as momentum.

Due to the layer-wise sparsity, our framework is able to achieve the state-of-the-art (SOTA) accuracy-model size trade-offs when comparing with unstructured sparsity and accuracy-FLOPs trade-offs when comparing with structured sparsity. Figure 2 demonstrates the advantages of layer-wise N:M sparsity under both scenarios on ResNet50 [13]. Details can be found in Table 2.

The major contributions of our paper can be summarized as follows: **(1)** We present DominoSearch - A novel and easy-to-use framework to search the layer-wise sparse schemes from pre-trained dense weights with a specified model complexity constraint. Our method does not rely on expensive design space exploration such as Network Architecture Search (NAS) or Reinforcement Learning (RL). To the best of our knowledge, this is the first work on layer-wise fine-grained N:M sparsity. **(2)** We

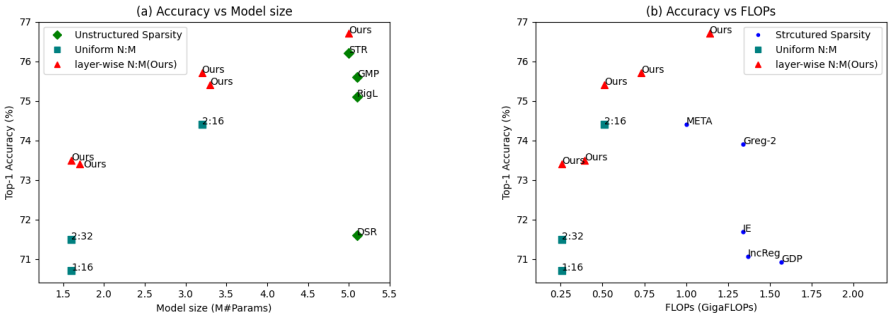

Figure 2: ResNet50 on ImageNet: Compare top-1 Accuracy of our layer-wise N:M sparse models with SOTA structured/unstructured sparsity under various FLOPs/model size.

propose a layer-wise penalty factor to balance the parameters redundancy that does not depend on input tensor/image size and complexity which depends on input tensor/image size (e.g. FLOPs). The computational complexity measurement FLOPs used in this paper can easily be replaced with latency, throughput or energy consumption when deployed on a specific platform. **(3)** We evaluate our solution on the large scale ImageNet dataset with both heavy ResNets [13] and compact RegNets [14], and smaller scale CIFAR100 dataset with ResNet56. Results of ResNet50 on ImageNet show that layer-wise N:M sparsity searched by our framework achieves SOTA accuracy-sparsity trade-off even compared with unstructured sparsity.

2 Related Work

Structure of sparsity: Traditionally, sparsity of DNNs is categorized into 1) unstructured sparsity and 2) structured sparsity. Unstructured sparsity can be achieved by weight pruning and reaches a relatively high sparse ratio without significant accuracy drop [8, 10, 12, 15]. However, unstructured sparsity has a large execution time overhead on modern hardware due to the large overhead of handling the irregular non-zero weights in the matrix; good speed-up can only be achieved when the sparse ratio is very high (i.e., > 95%) [7, 16]. On the other hand, coarse-grained structured sparsity such as filter-wise sparsity and channel-wise sparsity [16, 17] can maintain a hardware-friendly sparse structure but may not be able to achieve a high sparse ratio. In order to achieve a high compression ratio and a hardware-friendly sparse structure, various fine-grained structured sparsity patterns were proposed including Tile-wise [18] and N:M sparsity [4]. Our paper focuses on N:M fine-grained sparsity which has demonstrated success in getting accurate N:M sparse DNNs (i.e. trained from scratch [5] or fine-tuned [19] from dense pretrained weights) and usage in commodity hardware (e.g. Nvidia Ampere Tensor Core). Figure 1 illustrates two traditional types (structured/unstructured) and fine-grained N:M sparsity.

Layer-wise compression: Layer-wise compression is a well-known concept in the DNN optimization community. The non-uniform importance of different layers in DNNs is leveraged by mixed-precision quantization in which different bit-widths are allocated via NAS [20, 21] or RL [22]. Recent discoveries demonstrate that by carefully selecting [8, 9] or learning [10] the layer-wise sparsity, the sparse DNNs can achieve higher accuracy than their uniform counterpart. [9] proposes a training scheme for unstructured sparse neural networks and presents the Erdos-Renyi kernel (ERK) (extension of [11]) to heuristically select the layer-wise sparsity. [8] proposes a layer-adaptive magnitude-based pruning score to choose the layer-wise sparsity, it focuses on unstructured pruning and the number of survival weights of each layer without taking layer-wise computation complexity (e.g. FLOPs) into account. [10] presents Soft Threshold Reparameterization (STR) to learn the layer-wise sparsity during training. However the target model complexity constraint (e.g. model size and FLOPs) is implicitly controlled by two hyper-parameters (λ and s_{init}) and cannot be specified, hence users have to tune λ and s_{init} and train multiple models to get the desired target model complexity . Our paper focuses on how to find the N:M scheme of each layer efficiently with the specified model complexity constraint. It avoids the need for a time consuming hyper-parameter search.

3 Method - DominoSearch

3.1 Problem Formulation

Given a pretrained dense model with weight tensors $\mathbf{w}^1, \ldots, \mathbf{w}^i, \ldots, \mathbf{w}^L$ associated with L layers (convolution layers + fully-connected layers), model complexity constraint Sp_t (e.g. model size or FLOPs) and candidate pool N_p of sparse schemes (e.g. [1:8,2:8,4:8,8:8])[3]. The problem can be formulated as :

$$\begin{aligned} \max_{S(\mathbf{w},N)} \quad & \text{Accuracy}\big(S(\mathbf{w},N)\big) \\ s.t. \quad & \text{Complexity}(S(\mathbf{w},N)) \leq Sp_t \\ & N^i \in N_p \end{aligned} \tag{1}$$

Where $S(\cdot)$ denotes the sparse model which can be got with dense weights $\mathbf{w}$ and sparsity ratio N. Complexity$(\cdot)$ can be model size, FLOPs, real latency or even energy consumption depending on the use case and design objectives. The goal is to find optimal layer-wise schemes for whole network. Due to the large discrete design space and the training/fine-tuning cost of evaluating design point (i.e. recovering the accuracy of sparse model), it is usually not possible to hand-craft the optimal solution or do exhaustive search.

Inspired by methods used in mixed-bit search [21, 22], which decouple mixed-bit policy searching and model accuracy recovering. We use a two-step solution : (1) find the layer-wise sparse schemes from candidate pool N_p ; (2) fine-tune/retrain sparse weights with searched sparsity (i.e. N^i) to recover the accuracy.

3.2 Group-wise threshold and sparsity

We first emphasise the difference between layer-wise fined-grained sparsity and layer-wise unstructured sparsity. How to decide the layer-wise sparsity has been actively studied over the recent years [8, 9, 10]. However, these methods mainly focus on unstructured sparsity whose layer-wise sparsity can be decided by a layer-wise threshold because weights in a certain layer are treated individually. By contrast, fine-grained N:M sparsity treats weights from a *group-wise* view (e.g. Figure 1 (c)). As a consequence, we define a group-wise threshold vector T^i and group-wise sparsity vector N_g^i, where i is the layer index. The length of these two vectors is $K = \frac{|\mathbf{w}^i|}{M}$, where $||$ is cardinality (i.e. the number of parameters), M is the group size and K is the number of groups. $|\mathbf{w}^i|$ should be divisible by M.

Group-wise threshold selection: Instead of using a small global value such as $\frac{1}{1000}$ [15] to decide if the weights should be treated as pruned or not, we use a threshold vector T^i that can represent the weight distributions of each group. In order to achieve this, we use the mean of the smallest (absolute value) $\frac{M}{2}$ elements as threshold because it provides a group-wise feature in an efficient way and can mitigate the effect of outliers (e.g. values like 0.0). Note that we use $\frac{M}{2}$ elements because it gives relatively sufficient search space and representativeness, using more elements can give better representativeness because more elements are included but smaller search space because larger threshold value will lead smaller starting N_g^i and vice versa. Figure 3 shows an example of group-wise threshold selection. The larger variance of T^i indicates that we need a group-wise threshold. On the other hand, a group-wise threshold will lead to group-wise sparsity. Column vector N_g^i encodes the number of kept weights of each group, from which we can infer the sparsity of this layer (i.e. N:M = 3:4). In practice, we can relax the requirement by using the *Supermajority* voting rule. Specifically, as long as the ratio of the number of x in N_g^i reaches a pre-defined voting ratio v_r, the layer sparsity N^i can be inferred as x.

With above-mentioned group-wise definitions, we can reformulate $\mathbf{w}^i$ and T^i as $\mathbf{w}_k^i$ and T_k^i, where k is the group index. We use the reformulated version in the next sections.

[3] Without losing generation, we use N equal to power-of-two as search space.

$$\overset{\mathbf{w}^i}{\begin{bmatrix} 0.0104 & 0.0114 & 0.0020 & 0.0061 \\ 0.0212 & 0.0748 & 0.0368 & 0.0898 \\ 0.0854 & 0.1751 & 0.0406 & 0.0450 \\ 0.0896 & 0.0169 & 0. & 0.0177 \end{bmatrix}} -> \overset{T^i}{\begin{bmatrix} 0.0041 \\ 0.0290 \\ 0.0428 \\ 0.0085 \end{bmatrix}} -> \overset{N_g^i}{\begin{bmatrix} 3 \\ 3 \\ 3 \\ 3 \end{bmatrix}}$$

Figure 3: An illustrative example of dense weight matrix $\mathbf{w}^i$ and corresponding threshold vector T^i. The values are taken from random groups in **the first layer of ResNet18**. N_g^i encodes the number of elements larger than the threshold of each group. Blue highlights values larger than threshold.

3.3 Explore the search space via layer-wise weight penalty

Before introducing our search algorithm, we consider the standard DNNs training process with Stochastic Gradient Descent (SGD) update rule.

$$\mathbf{w}(t+1) := \mathbf{w}(t) - \xi_t g(\mathbf{w}(t)) \tag{2}$$

with learning rate ξ_t and gradients $g(\mathbf{w})$ of $\mathbf{w}$. To obtain a sparse model, we need to apply a mask $\mathbf{m}$ $\in \{0,1\}$ to weights $\mathbf{w}(t)$, resulting in a sparse model $\widetilde{\mathbf{w}}(t) := \mathbf{w}(t) \odot \mathbf{m}$, where $\odot$ denotes entry-wise product. The mask can dynamically change during training iteration t and depends on the weights $\mathbf{w}_t$. With group-wise and layer-wise dimensions, we can reformulate mask $\mathbf{m}$ as

$$\mathbf{m}_k^i(t) := \mathbf{w}_k^i(t) > T_k^i \tag{3}$$

where i is the layer index, k is the group index and T_k^i is the group-wise threshold introduced in Section 3.2. Mask $\mathbf{m}_k^i(t)$ is inferred by Boolean Operator $>$, where $\mathbf{m}$ = 1 means kept and $\mathbf{m}$ = 0 means pruned.

A popular approach is to apply extra regularization penalty to introduce more sparsity, including L_1 norm and L_2 norm [12, 15]. We periodically exploit the L_2 norm [15] which can hold the group-wise feature, to find the layer-wise sparsity. Thus the update rule can be formulated as:

$$\mathbf{w}_k^i(t+1) := \mathbf{w}_k^i(t) - \xi_t(g(\widetilde{\mathbf{w}}_k^i(t)) + p(t) * \eta_i(t) * \widetilde{\mathbf{w}}_k^i(t)) \tag{4}$$

$$p(t) = \begin{cases} 1, & \text{if } t \mod k_p = 0. \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

where $\eta_i(t)$ is the ***layer-wise*** penalty factor to control the strength of penalty applied to each layer which will be introduced in Section 3.4. Inspired by [15], we apply a **periodical** weight penalty which will drive the kept weights towards the threshold gradually and leave k_p iterations as buffer time for adjusting to the applied penalty. The layer-wise sparsity N^i and model complexity decreases during the search phase. When the model complexity reaches the search constraint Sp_t, we terminate the searching process and fine-tune the weights based on searched layer-wise fine-grained sparsity schemes. Algorithm 3.1 shows the search algorithm.

3.4 Layer-wise penalty factor

Modern DNNs tend to have many layers with non-uniform distributions of both parameter redundancy and complexity contribution. Intuitively, we want to encourage layers with larger redundancy and higher complexity to achieve a higher sparse ratio (i.e. decrease N) faster. This can be achieved by controlling the strength of the weight penalty of each layer. For layer-wise computational complexity (e.g. FLOPs) factor it can be calculated using

$$c_i = \frac{f_i * s_i}{\max_{i=1}^{i=L}(f_i * s_i)} \tag{6}$$

where f_i is the dense FLOPs of layer i and $s_i = N_v^i / M$ is the current sparsity.

Algorithm 3.1 DominoSearch

Input: Complexity constraints Sp_t, dense weights of each layer $\mathbf{w}_k^1, \ldots, \mathbf{w}_k^l$. Dataset D. Learning rate ξ. Interval K_c. Voting ratio v_r.
Init: current model complexity Sp_c = 0, iterations t = 0, layer-wise penalty factor η_i.
while True **do**
  for $\mathbf{w}_k^i$ in $\mathbf{w}_k^1, \ldots, \mathbf{w}_k^L$ **do**
    $\mathbf{w}_k^i(t+1) := \mathbf{w}_k^i(t) - \xi_t(g(\widetilde{\mathbf{w}}_k^i(t)) + p(t) * \eta_i(t) * \widetilde{\mathbf{w}}_k^i(t))$ ▷ Eq. 4
    if t % K_c == 0 **then**
      $\mathbf{m}_k^i(t) := \mathbf{w}_k^i(t) > T_k^i$ ▷ Eq. 3
      N^i := vote(sum($\mathbf{m}_k^i(t)$, axis=1),v_r) ▷ Sec. 3.2
      if $N^i \in N_p^i$ **then**
        $N_v^i = N^i$ ▷ N_v^i tracks $N^i \in N_p^i$
        Sp_c = Complexity(N_v^i) ▷ Compute current model complexity
        if $Sp_c \geq Sp_t$ **then**
          Terminate search process and go to **Output**.
        end if
        Update layer-wise penalty factor η_i ▷ Sec. 3.4
      end if
    end if
  end for
end while
Output Layer wise sparse schemes N_v^i

In order to estimate the layer-wise redundancy, we use the Erdos-Renyi kernel (ERK) [9]. ERK is a heuristic method to decide the layer-wise unstructured sparsity given an overall sparsity target (i.e., model size) which does not require design space exploration or hyper-parameter search. However, ERK cannot be directly applied to select the layer-wise N:M schemes because 1) ERK gives layer-wise sparsity in the continuous domain while N:M sparsity is discrete. 2) Layer-wise sparsity decided by ERK only depends on the overall sparsity target and layers' configuration (e.g. kernel size, input/output channels). It can not reflect layer-wise complexity (e.g., FLOPs) which depends on input tensor/image size. Thus, we use the layer-wise sparsity generated by ERK as a layer-wise redundancy guidance which can be formulated as:

$$r_i = \frac{e_i - s_i}{\max_{i=1}^{i=L} |e_i - s_i|} \tag{7}$$

where e_i is the sparsity of layer i decided by ERK. Note that r_i can be negative if the current sparsity (s_i) is larger than the sparsity estimated by ERK. With r_i and c_i, the joint ***layer-wise*** penalty factor η_i is defined as:

$$\eta_i = \beta_1 * c_i + \beta_2 * r_i. \tag{8}$$

where β_1 and β_2 are weighted coefficients (β_1+β_2 = 1). When searching under model size constraint, we set β_1 = 0.5 , β_2 = 0.5 and use β_1 = 0.8 , β_2 = 0.2 for searching under FLOPs constraint. Weighted coefficients β_1 and β_2 are used to balance the effect of FLOPs which is dependent on input tensor/image size and model size which is not dependent on input tensor/image size, we tend to use higher β_1 when the complexity constraint is FLOPs because it can assign layers with higher FLOPs higher penalty factor. An ablation study of ***layer-wise*** penalty factor can be found in Section 4.4.

4 Experimental Results

Networks and datasets: We demonstrate the advantages of layer-wise fine-grained N:M sparsity via various networks on the large-scale ImageNet dataset [23] including heavy ResNets [13] and compact RegNets [14]. We also demonstrate the effectiveness of our method on a smaller dataset – CIFAR100 with ResNet56 [13].

Training settings: We first apply DominoSearch (DS) to find the layer-wise sparse schemes from pre-trained dense models (Algorithm 3.1), and fine-tune/retrain sparse weights on pre-trained dense weights with searched layer-wise schemes to recover the accuracy. Detailed settings can be found in the Supplementary Material.

4.1 ResNet18/50 with model size and FLOPs as constraints

ResNet18: Table 1 shows that layer-wise sparse ResNet18 consistently outperforms the uniform baseline with equivalent model size or FLOPs. Figure 4 visualizes the layer-wise sparse FLOPs distributions under two search targets and dense parameters distributions. For FLOPs-constraint search, the shallower layers especially the first layer are allocated with higher sparsity (Figure 4 (b)) because of their larger FLOPs contributions caused by larger input feature size. On the other hand, shallower layers have relatively smaller contributions to model size (i.e. less parameters, Figure 4 (c)), resulting in relatively lower sparsity when using model size constraint (Figure 4 (a)). This observation supports the necessity of using layer-wise penalty factor η_i which is able to capture the information of FLOPs distribution from a global view and assign corresponding layers (shallower layers in this case) with relatively higher penalty factor.

Table 1: Layer-wise sparse ResNet18 vs uniform baseline with equivalent model size or FLOPs. Dense model is downloaded from Pytorch pre-trained model zoo.

Method	#Params	FLOPs	Top1 Acc	Structure
ResNet18	11.7M	1814M	69.8	Dense
SR-STE [5]	1.46M	227M	66.65	2:16
Equal model size				
DS(**ours**)	1.46M	329M	68.76(+2.1)	Mixed N:16
Equal FLOPs				
DS(**ours**)	1.29M	227M	67.98(+1.3)	Mixed N:16

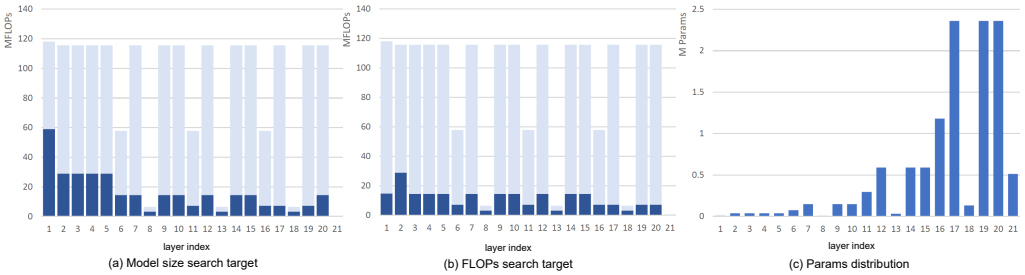

Figure 4: Layer-wise statistics of ResNet18: x-axis is layer index i (layer 8, 13, 18 are *Conv2d* with 1 x 1 kernel and layer 21 is fully-connected layer) and y-axis denotes FLOPs and Parameters. (a)/(b) shows the distributions of layer-wise FLOPs under model size/FLOPs search target, where dark/light means kept/pruned partition. (c) shows the layer-wise distribution of Parameters.

ResNet50: Table 2 shows the results on ResNet50. Layer-wise fine-grained sparsity can consistently outperform its uniform counterparts for both accuracy-sparsity (i.e. model size) and accuracy-FLOPs trade-off. When comparing to the SOTA layer-wise unstructured sparsity [10], which is believed to be the upper-bound of Neural Network pruning with respect of accuracy-sparsity trade-off, our layer-wise fine-grained models can achieve competitive result given the same model size. Although it has higher FLOPs than unstructured sparsity (1.14G v.s. 766M), we emphasize that fine-grained sparsity maintains ***regular*** sparse pattern which is crucial for getting deployment gains (e.g. latency). Furthermore, our layer-wise fine-grained sparsity is significantly better than structured sparsity with respect to accuracy-FLOPs trade-off, which encourages a promising system design research line – efficient supports of flexible N:M sparse patterns. Considering the preliminary success of Nvidia Ampere Sparse Tensor Core [4], we believe the design overhead of supporting flexible N:M sparse patterns is affordable and worthwhile.

Table 2: Compare ResNet50 on ImageNet dataset with related work. (+) denotes the top1-accuracy gain compared to the uniform counterparts with equivalent model size or FLOPs. (+) compares with unstructured sparsity. Reported numbers are either taken from the corresponding literature or obtained by training with open-source code [5]. Dense model is downloaded from Pytorch pre-trained model zoo.

Method	#Params	FLOPs	Top1 Acc	Structure	Uniform
ResNet50	25.6M	4.09G	76.2	Dense	-
GMP[24]	5.1M	818M	75.6	Unstructured	U(niform)
DSR[25]	5.1M	1.23G	71.6	Unstructured	L(ayer-wise)
DNW[26]	5.1M	818M	76.2	Unstructured	U
RigL[9]	5.1M	1.68G	75.1	Unstructured	L
STR[10]	5.2M	766M	76.2	Unstructured	L
DS(**ours**)	5.0M	1.14G	***76.7***(+0.5)	Mixed N:16	L
IncReg[27]	-	1.37G	71.1	Structured	U
IE[28]	-	1.34G	71.7	Structured	L
Greg-2[15]	-	1.34G	73.9	Structured	L
GDP[29]	-	1.57	70.93	Structured	L
META[30]	-	1.0	74.4	Structured	L
SR-STE[5]	6.4M	1.02G	76.5	4:16	U
SR-STE[5]	3.2M	511M	74.4	2:16	U
SR-STE[5]	1.6M	256M	70.7	1:16	U
SR-STE[5]	1.6M	256M	71.5	2:32	U
Equal model size					
DS(**ours**)	3.2M	731M	75.7(+1.3)	Mixed N:16	L
DS(**ours**)	1.6M	392M	73.5(+2.0)	Mixed N:32	L
Equal FLOPs					
DS(**ours**)	3.3M	511M	75.4 (+1.0)	Mixed N:16	L
DS(**ours**)	1.7M	256M	73.4 (+1.9)	Mixed N:32	L

4.2 RegNet with FLOPs constraint

We further study our search method on compact model – RegNet. RegNet [14] is the state-of-the-art compact network architecture searched by NAS from 10^{18} candidates. Table 3 shows that layer-wise sparse RegNetX3.2G outperforms its uniform sparse counterpart by 1.0% and dense smaller RegNetX400M by 3.9% given equivalent FLOPs. The results demonstrates that our method can effectively find the layer-wise sparsity even if the dense model is designed to be compact.

Table 3: Compare sparse RegNets and dense RegNets with various FLOPs. (+) denotes the accuracy gain compared to dense model with same FLOPs.

Network(Method)	FLOPs	Top1 Acc	Structure
RegNetX400M	400M	72.3	Dense
RegNetX800M	800M	74.9	Dense
RegNetX1.6G	1.6G	76.8	Dense
RegNetX3.2G	3.2G	78.2	Dense
Sparse RegNetX3.2G (SR-STE)	400M	75.2	2:16
Sparse RegNetX3.2G (Ours)	400M	76.2(+1.0) (+3.9)	Mixed N:16

4.3 ResNet56 on CIFAR100

In addition to the ImageNet dataset, Table 4 shows that layer-wise N:M sparsity can significantly improve the accuracy on the smaller CIFAR100 dataset under both model size and FLOPs constraints.

Table 4: ResNet56+CIFAR100: Comparison between uniform sparsity and layer-wise sparsity. Each setting is run for 3 times, mean and std top-1 accuracy are reported. Baseline dense ResNet56: 74.8% top1-acc, 0.85M #Params, 127M FLOPs.

Model Complexity	0.21M#Params	0.11M#Params	32M FLOPs	16M FLOPs
Acc. uniform	$72.33_{\pm 0.37}$	$71.08_{\pm 0.27}$	$72.33_{\pm 0.37}$	$71.08_{\pm 0.27}$
Acc. layer-wise	$74.31_{\pm 0.34}$	$73.26_{\pm 0.24}$	$74.00_{\pm 0.18}$	$72.79_{\pm 0.25}$
Acc. gain	+1.98	+2.18	+1.67	+1.71

4.4 An ablation study: The advantage of *layer-wise* penalty factor

In order to study the effectiveness of ***layer-wise*** penalty factor η_i proposed in section 3.4, we consider a constant penalty factor (1.0) on all layers as baseline. Table 5 shows that layer-wise schemes searched with η_i consistently outperform the counterparts without η_i. The gap is significant under complexity with small FLOPs (i.e. 16M). In order to achieve such a low FLOPs without η_i, both shallow and deep layers are extensively pruned no matter their contributions of FLOPs, resulting in very high sparsity (92.2%). By contrast, with the guidance of η_i, the sparsity is only 89.0% when achieving equivalent FLOPs. Additional experiments on ImageNet can be found in Supplementary Material.

Table 5: ResNet56+CIFAR100: Ablation study of layer-wise penalty factor η_i.

Model Complexity	0.21M#Params	0.11M#Params	32M FLOPs	16M FLOPs
Acc. with η_i	$74.31_{\pm 0.34}$	$73.26_{\pm 0.24}$	$74.00_{\pm 0.18}$	$72.79_{\pm 0.25}$
Acc. w/o η_i	$73.80_{\pm 0.20}$	$72.95_{\pm 0.19}$	$73.71_{\pm 0.14}$	$70.39_{\pm 0.19}$
Acc. drop	-0.51	-0.31	-0.29	-2.40

5 Conclusions

This paper presents a novel and efficient framework, *DominoSearch*, together with the proposed ***layer-wise*** penalty factor, to find the layer-wise fine-grained N:M sparsity from dense weights which can significantly improve the accuracy compared to its uniform counterparts with equivalent model size or FLOPs. We justify the advantages of the layer-wise N:M sparsity on CIFAR100 and ImageNet datasets with various networks. The results of ResNet50 on ImageNet show that our layer-wise N:M sparsity is competitive to layer-wise unstructured sparsity which is believed to the upper-bound of neural pruning with respect to the accuracy-sparsity trade-off. We hope that our contributions can encourage and draw attention to flexible fine-grained N:M hardware-software system design/research.

6 Limitations

The major limitation of our work is that we did not benchmark deployment gains such as latency and throughput. To the best of our knowledge, Nvidia Ampere Tensor Core is the only commercially available architecture that supports fine-grained N:M sparse computation efficiently. However, it only supports limited options for fixed sparsity (i.e. 2:4 for float and 4:8 for integer). Thus, following the common practice of related work we compared to, we used theoretical computational complexity – FLOPs to estimate the performance gains.

We also acknowledge that the experiments are mainly focusing on CNN models and image classification datasets. Other models like Transformer [31] and MLP [32] have not been discussed and are left for future work.

7 Potential negative societal impact

We acknowledge that the proposed DNN optimization framework could make some negative applications (e.g. unintended surveillance) easier for deployment.

References

[1] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. "A Survey of Quantization Methods for Efficient Neural Network Inference". In: *arXiv preprint arXiv:2103.13630* (2021).

[2] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Xiaojiang Chen, and Xin Wang. "A comprehensive survey of neural architecture search: Challenges and solutions". In: *arXiv preprint arXiv:2006.02903* (2020).

[3] Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. "Knowledge distillation: A survey". In: *International Journal of Computer Vision* (2021), pp. 1–31.

[4] Ronny Krashinsky and Olivier Giroux et al. *Nvidia Ampere Sparse Tensor Core*. URL: `https://developer.nvidia.com/blog/nvidia-ampere-architecture-in-depth/`.

[5] Aojun Zhou, Yukun Ma, Junnan Zhu, Jianbo Liu, Zhijie Zhang, Kun Yuan, Wenxiu Sun, and Hongsheng Li. "Learning N:M Fine-grained Structured Sparse Neural Networks From Scratch". In: *International Conference on Learning Representations*. 2021.

[6] *Nvidia Automatic SParsity*. URL: `https://github.com/NVIDIA/apex/tree/e2083df5eb96643c61613b9df48dd4eea6b07690/apex/contrib/sparsity`.

[7] Ziheng Wang. "SparseRT: Accelerating Unstructured Sparsity on GPUs for Deep Learning Inference". In: *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*. 2020, pp. 31–42.

[8] Jaeho Lee, Sejun Park, Sangwoo Mo, Sungsoo Ahn, and Jinwoo Shin. "Layer-adaptive Sparsity for the Magnitude-based Pruning". In: *International Conference on Learning Representations*. 2021.

[9] Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. "Rigging the lottery: Making all tickets winners". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 2943–2952.

[10] Aditya Kusupati, Vivek Ramanujan, Raghav Somani, Mitchell Wortsman, Prateek Jain, Sham Kakade, and Ali Farhadi. "Soft Threshold Weight Reparameterization for Learnable Sparsity". In: *Proceedings of the International Conference on Machine Learning*. 2020.

[11] Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. "Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science". In: *Nature communications* 9.1 (2018), pp. 1–12.

[12] Song Han, Jeff Pool, John Tran, and William Dally. "Learning both Weights and Connections for Efficient Neural Network". In: *Advances in Neural Information Processing Systems*. 2015.

[13] 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*. 2016, pp. 770–778.

[14] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. "Designing network design spaces". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 10428–10436.

[15] Huan Wang, Can Qin, Yulun Zhang, and Yun Fu. "Neural Pruning via Growing Regularization". In: *arXiv preprint arXiv:2012.09243* (2020).

[16] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. "Learning structured sparsity in deep neural networks". In: *arXiv preprint arXiv:1608.03665* (2016).

[17] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. "Pruning filters for efficient convnets". In: *International Conference on Learning Representations*. 2017.

[18] 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: *arXiv preprint arXiv:2008.13006* (2020).

[19] Itay Hubara, Brian Chmiel, Moshe Island, Ron Banner, Seffi Naor, and Daniel Soudry. "Accelerated Sparse Neural Training: A Provable and Efficient Method to Find N: M Transposable Masks". In: *arXiv preprint arXiv:2102.08124* (2021).

[20] Mingzhu Shen, Feng Liang, Chuming Li, Chen Lin, Ming Sun, Junjie Yan, and Wanli Ouyang. *Once Quantized for All: Progressively Searching for Quantized Efficient Models*. 2020. arXiv: `2010.04354 [cs.CV]`.

[21] Zhaowei Cai and Nuno Vasconcelos. "Rethinking differentiable search for mixed-precision neural networks". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 2349–2358.

[22] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. "Haq: Hardware-aware automated quantization with mixed precision". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 8612–8620.

[23] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: `10.1007/s11263-015-0816-y`.

[24] Trevor Gale, Erich Elsen, and Sara Hooker. "The state of sparsity in deep neural networks". In: *arXiv preprint arXiv:1902.09574* (2019).

[25] Hesham Mostafa and Xin Wang. "Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 4646–4655.

[26] Mitchell Wortsman, Ali Farhadi, and Mohammad Rastegari. "Discovering neural wirings". In: *arXiv preprint arXiv:1906.00586* (2019).

[27] Huan Wang, Xinyi Hu, Qiming Zhang, Yuehai Wang, Lu Yu, and Haoji Hu. "Structured pruning for efficient convolutional neural networks via incremental regularization". In: *IEEE Journal of Selected Topics in Signal Processing* 14.4 (2019), pp. 775–788.

[28] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. "Importance estimation for neural network pruning". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 11264–11272.

[29] Shaohui Lin, Rongrong Ji, Yuchao Li, Yongjian Wu, Feiyue Huang, and Baochang Zhang. "Accelerating Convolutional Networks via Global & Dynamic Filter Pruning." In: *IJCAI*. 2018, pp. 2425–2432.

[30] Zechun Liu, Haoyuan Mu, Xiangyu Zhang, Zichao Guo, Xin Yang, Kwang-Ting Cheng, and Jian Sun. "Metapruning: Meta learning for automatic neural network channel pruning". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 3296–3305.

[31] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *arXiv preprint arXiv:2010.11929* (2020).

[32] Ilya Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, et al. "MLP-Mixer: An all-MLP Architecture for Vision". In: *arXiv preprint arXiv:2105.01601* (2021).

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] Experiments section 4 supports the main claims we made in abstract 1. layer-wise fine-grained outperforms the uniform counterpart 2. Ablation study supports the effectiveness of proposed layer-wise penalty factor in section 3.4 .
   (b) Did you describe the limitations of your work? [Yes] See section 6.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See section 7. Our work focuses on the generic DNN optimization which has no obvious or direct relations to societal impacts. The impacts depend on the end applications.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] We will provide our source code in the supplemental material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] We describe the training details in the Supplemental Material.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] and [No] Experiments on large-scale ImageNet dataset are computationally expensive so we did not run multiple times, but we did report standard deviations and averages for CIFAR100 experiments.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] We describe training resources and estimate the total amount of compute in the Supplementary Material.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] See section 4.
   (b) Did you mention the license of the assets? [No] But the licenses of datasets and code can be found in corresponding website/Github repositories.
   (c) Did you include any new assets either in the supplemental material or as a URL? [No] We did not use new assets, we conducted experiments based on published materials.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [No] Based on the licenses, we can use them for research purpose without asking for permission.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No] We used well-known CIFAR100 and ImageNet datasets. To the best of our knowledge, these datasets do not have these issues.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]