DSA: More Efficient Budgeted Pruning via Differentiable Sparsity Allocation

Xuefei Ning[1⋆][0000−0003−2209−8312], Tianchen Zhao[2∗][0000−0002−2071−7514], Wenshuo Li[1][0000−0001−5638−2114], Peng Lei[2][0000−0001−7422−0258], Yu Wang[1][0000−0001−6108−5157], and Huazhong Yang[1][0000−0003−2421−353X]

[1] Department of Electronic Engineering, Tsinghua University
[2] Department of Electronic Engineering, Beihang University
foxdoraame@gmail.com, ztc16@buaa.edu.cn, yu-wang@tsinghua.edu.cn

Abstract. Budgeted pruning is the problem of pruning under resource constraints. In budgeted pruning, how to distribute the resources across layers (i.e., sparsity allocation) is the key problem. Traditional methods solve it by discretely searching for the layer-wise pruning ratios, which lacks efficiency. In this paper, we propose Differentiable Sparsity Allocation (DSA), an efficient end-to-end budgeted pruning flow. Utilizing a novel *differentiable pruning process*, DSA finds the layer-wise pruning ratios with *gradient-based optimization*. It allocates sparsity in continuous space, which is more efficient than methods based on discrete evaluation and search. Furthermore, DSA could work in a *pruning-from-scratch* manner, whereas traditional budgeted pruning methods are applied to pre-trained models. Experimental results on CIFAR-10 and ImageNet show that DSA could achieve superior performance than current iterative budgeted pruning methods, and shorten the time cost of the overall pruning process by at least 1.5× in the meantime.

Keywords: Budgeted pruning, Structured pruning, Model compression

1 Introduction

Convolutional Neural Networks (CNNs) have demonstrated superior performances in computer vision tasks. However, CNNs are computational and storage intensive, which poses significant challenges on the NN deployments under resource constrained scenarios. Model compression techniques [21, 16] are proposed to reduce the computational cost of CNNs. Moreover, there are situations (e.g., deploying the model onto certain hardware, meeting real-time constraints) under which the resources (e.g., latency, energy) of the compressed models must be restricted under certain budgets. Budgeted pruning is introduced for handling these situations.

As shown in Fig. 1, the budgeted pruning problem could be divided into two sub-tasks: to decide how many channels to keep for each layer (i.e., sparsity

⋆ Both authors contributed equally to this work.

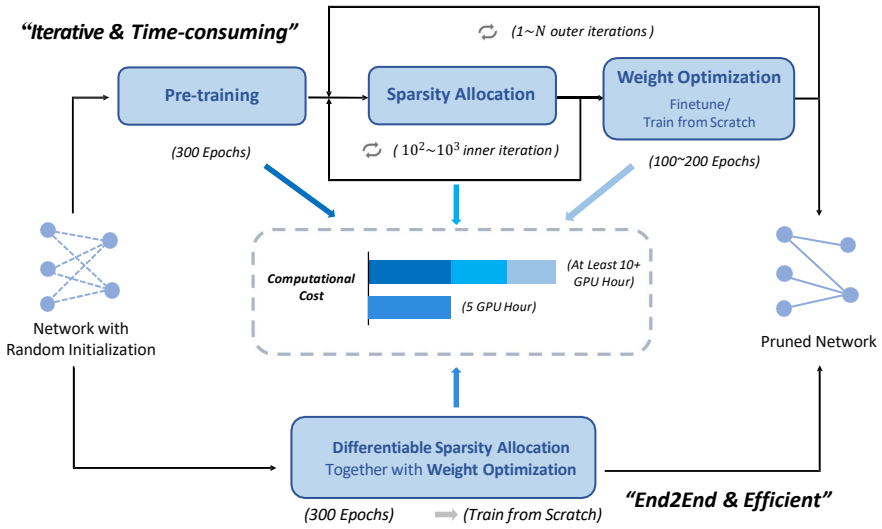


Fig. 1. Workflow comparison of the iterative pruning methods [23, 7, 14, 15] and DSA

allocation) and to acquire proper weights (i.e., weight optimization). Recent work [17] observes that once the pruned structure is acquired, the compressed model can achieve similar accuracies no matter it is trained from scratch or fine-tuned from the weights inherited from the original model. Therefore, sparsity allocation is the key problem for budgeted pruning.

To solve the sparsity allocation problem, the majority of methods [23, 7, 14, 15] adopt an "iterative pruning flow" scheme. The workflow of these methods involves three stages: pre-training, sparsity allocation, and finetuning, as shown in Fig. 1. These methods conduct the sparsity allocation through a discrete search, which contains hundreds of search-evaluation iterations. For each candidate allocation, a time-consuming approximate evaluation is needed. Also, the discrete search in the large search space lacks sample efficiency. Moreover, these methods need to be applied to the pre-trained models, and model pre-training costs much computational effort. As a result, the overall iterative pruning flow is not efficient.

In order to improve the efficiency of budgeted pruning, we propose DSA, an end-to-end pruning flow. Firstly, DSA can work in a "pruning-from-scratch" manner, thus eliminating the cumbersome pre-training process (see Sec. 5.3). Secondly, DSA optimizes the sparsity allocation in continuous space with a gradient-based method, which is more efficient than methods based on discrete evaluation and search.

For applying the gradient-based optimization for allocating sparsity, we should make the evaluation of the validation accuracy and the pruning process differentiable. For the validation performance evaluation, we use the validation loss as a

differentiable surrogate of the validation accuracy. For the pruning process, we propose a probabilistic differentiable pruning process as a replacement. In the differentiable pruning process, we soften the non-differentiable hard pruning by introducing masks sampled from the probability distributions controlled by the pruning ratio. The differentiable pruning process enables the gradients of the task loss, w.r.t. the pruning ratios to be calculated. Utilizing the task loss's gradients, DSA obtains the sparsity allocation under the budget constraint following the methodology of the Alternating Direction Method of Multipliers (ADMM) [1].

The contributions of this paper are as follows.

- DSA uses *gradient-based optimization* for sparsity allocation under budget constraints, and works in a *pruning-from-scratch* manner. DSA is more efficient than iterative pruning methods.
- We propose a novel *differentiable pruning process*, which enables the gradients of the task loss w.r.t. the pruning ratios to be calculated. The gradient magnitudes align well with the layer-wise sensitivity, thus providing an efficient way of measuring the sensitivity (See Sec. 5.3). Due to this property of the gradients, DSA can attribute higher keep ratios to more sensitive layers.
- We give a topological grouping procedure to handle the topological constraints that are introduced by skip connections and so on, thus the resulting model keeps the original connectivity.
- Experimental results on CIFAR-10 and ImageNet demonstrate the effectiveness of DSA. DSA consistently outperforms other iterative budgeted pruning methods with at least 1.5× speedup.

2 Related Work

2.1 Structured Pruning

Structured pruning intends to introduce structured sparsity into the NN models. SSL [21] removes structured subsets of weights by applying group lasso regularization and magnitude pruning. Some studies [16, 4] add regularization on the batch normalization (BN) scaling parameters γ instead of the convolution weights. These methods focus on seeking the trade-off between model sparsity and performance via designing regularization terms. Since these methods allocate sparsity through regularizing the weights, the results are sensitive to the hyperparameters.

There are also studies that target at choosing which filters to prune, given the layer-wise pruning ratios. ThiNet [18] utilizes the information from the next layer to select filters. FPGM [6] exploits the distances to the geometric median as the importance criterion. These methods focus on exploring intra-layer filter importance criteria, instead of handling the inter-layer sparsity allocation.

2.2 Budgeted Pruning

To amend the regularization based methods for budgeted pruning, MorphNet [4] alternates between training with L_1 regularization and applying a channel multiplier. However, the uniform expansion of width neglects the different sensitivity

of different layers and might fail to find the optimal resource allocation strategy under budget. To explicitly control the pruning ratios, ECC [22] updates the pruning ratios according to the energy consumption model. The pruning process is modeled as discrete constraints tying the weights and pruning ratios, and this constrained optimization is handled using a proximal projection. In our work, the pruning process is relaxed into a probabilistic and differentiable process, which enables the pruning ratios to be directly instructed by the task loss.

Other methods view the budgeted pruning problem as a discrete search problem, in which the sparsity allocation and finetuning are alternatively conducted for multiple stages. In each stage, a search-evaluation loop is needed to decide the pruning ratios. For the approximate evaluation, a hard pruning procedure and a walk-through of the validation dataset are usually required. As for the search strategy, NetAdapt [23] empirically adjusts the pruning ratios, while ensuring a certain resource reduction is achieved; AMC [7] employs reinforcement learning to instruct the learning of a controller, and uses the controller to sample the pruning ratios; AutoCompress [14] uses simulated annealing to explore in the search space; MetaPruning [15] improves the sensitivity analysis by introducing a meta network to generate weights for pruned networks. Apart from these methods that search for the layer-wise pruning ratio, LeGR [2] searches for appropriate affine transformation coefficients to calculate the global importance scores of the filters. These methods can guarantee that the resulting models meet the budget constraint, but the discrete search process is inefficient and requires a pre-trained model.

In contrast, DSA (Differentiable Sparsity Allocation) is an end-to-end pruning flow that allocates the inter-layer sparsity with a gradient-based method, which yields better performance and higher efficiency. Moreover, DSA works in a "pruning from scratch" manner, saving the cost of pre-training the model. The comparison of various properties across pruning methods is summarized in Table. 1.

3 Problem Definition

For budgeted pruning, denoting the budget as $B_{\mathcal{F}}$, the weight as W, the optimization problem of the keep ratios $\mathcal{A} = \{\alpha^{(k)}\}_{k=1,\cdots,K}$ (1 - pruning ratio) of K layers can be written as:

$$\begin{aligned} \mathcal{A}^* &= \arg\max_{\mathcal{A}} \mathrm{Acc}_v(W^*(\mathcal{A}), \mathcal{A}) \\ \text{s.t.} \quad W^*(\mathcal{A}) &= \arg\min_{W} L_t(W, \mathcal{A}) \\ \mathcal{F}(\mathcal{A}) &\le B_{\mathcal{F}}, \quad 0 \le \mathcal{A} \le 1 \end{aligned} \tag{1}$$

where Acc_v is the validation accuracy, and L_t is the training loss, and $\mathcal{F}(\mathcal{A})$ is the consumed resource corresponding to the keep ratios $\mathcal{A}$.

To solve this optimization problem, existing iterative methods [23, 7, 14] conduct sensitive analysis in each stage, and use discrete search methods to adjust $\mathcal{A}$. The common assumption adopted is that in each stage, $\mathrm{Acc}_v(\hat{W^*}(\mathcal{A}), \mathcal{A})$

Table 1. Comparison of structured pruning methods. **Headers**: The "budget control" column indicates whether the method could ensure the resulting model to satisfy the budget constraint; The "from scratch" column indicates whether the method could be applied to random initialized models rather than pre-trained ones; The "performance instruction" column describes how the task performance instructs the sparsity allocation, "indirect" means that the task performance instructs the sparsity allocation only indirectly through weights (e.g., magnitude pruning); The "gen. perf." column indicates whether the generalization performance guides the pruning process

Methods	budget control	from scratch	end-to-end	performance instruction	gen. perf.
SSL [21]	**no**	yes	yes	indirect	no
NetAdapt [23]	yes	no	no	discrete evaluation	yes
AMC [7]	yes	no	no	discrete evaluation	yes
MetaPruning [15]	yes	no	no	discrete evaluation	yes
ECC [22]	yes	no	yes	indirect	no
Ours	yes	yes	yes	differentiable	yes

should be correlated to $\mathrm{Acc}_v(W^*(\mathcal{A}), \mathcal{A})$, in which $\hat{W}^*(\mathcal{A})$ is approximated using the current weights (e.g., threshold-based pruning, local layer-wise least-square fitting), instead of finding $W^*(\mathcal{A})$ by finetuning.

4 Method

Since the validation accuracy Acc_v in Eq. 1 is not differentiable, we use the validation loss L_v as a differentiable surrogate of Acc_v. Then, the objective function becomes $\mathcal{A}^* = \arg\min_{\mathcal{A}} L_v(W^*(\mathcal{A}), \mathcal{A})$ in the differentiable relaxation of the budgeted pruning problem in Eq. 1. As for the inner optimization of $W^*(\mathcal{A})$, we adapt the weights to the changes of the structure by adopting similar bi-level optimization as in DARTS [13]. The high-order gradients $\frac{\partial W^*(\mathcal{A})}{\partial \mathcal{A}}$ are ignored, thus $\frac{\partial L_v(W^*(\mathcal{A}),\mathcal{A})}{\partial \mathcal{A}} \approx \frac{\partial L_v(W,\mathcal{A})}{\partial \mathcal{A}}$.

The workflow of DSA is shown in Alg. 1 and Fig. 2. First, DSA groups the network layers according to the topological constraints (Sec. 4.2), and assigned a keep ratio for each group of layers. The sparsity allocation problem is to decide the proper keep ratios $\mathcal{A}$ for the K groups. The optimization of the keep ratios $\mathcal{A}$ is conducted with gradient-based method in the *continuous* space. We apply an ADMM-inspired optimization method (Sec. 4.3) to utilize the gradients of both the task and budget loss to find a good sparsity allocation α^* that meets the budget constraint. Note that to enable the task loss's gradients to flow back to the keep ratios α, we design a novel differentiable pruning process (Sec. 4.1).

4.1 Differentiable Pruning

Pruning Process Relaxation In the traditional pruning process of a particular layer, given the keep ratio α, a subset of channels are chosen according to

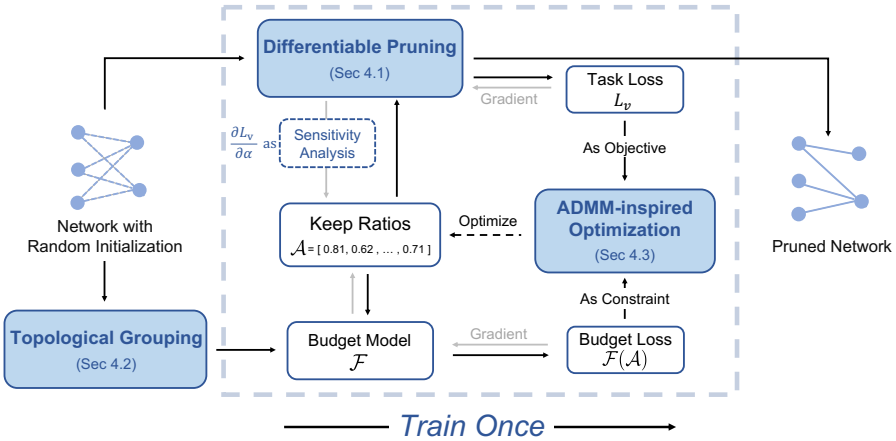


Fig. 2. DSA workflow. For feasible deployment, we first group the network layers according to the topological constraints (Sec. 4.2), and a keep ratio will be attributed to each topological group. The budget model $\mathcal{F}$ is also generated for measuring the budget loss $\mathcal{F}(\mathcal{A})$. Then, in the sparsity allocation procedure, the weights are updated using the task loss on the training dataset. And the keep ratios decision (i.e., inter-layer sparsity allocation) is conducted on the validation dataset using gradient-based steps (Sec. 4.3). Note that the differentiable characteristic of the pruning process (Sec. 4.1) enables the task loss's gradients to flow back to the keep ratios $\mathcal{A}$

the channel-wise importance criteria $b_i \in \mathbb{R}^+, i = 1, \cdots, C$ (e.g., the L1 norm of the convolutional weights). In contrast, we use a probabilistic relaxation of the "hard" pruning process where channel i has the probability p_i to be kept. Then, channel-wise masks are sampled from the Bernoulli distribution of p_i: $m_i \sim \text{Bernoulli}(p_i), i = 1, \cdots, C$. The pruning process is illustrated in Fig. 3.

The channel-wise keep probability $p_i = f(\alpha, b_i)$ is computed using α. Due to the probabilistic characteristics of the pruning process, the proportion of the actual kept channels might deviate from α. We should make the expectation of the actual kept channels $E[\sum_{i=1}^{C} m_i] = \sum_{i=1}^{C} p_i = \sum_{i=1}^{C} f(\alpha, b_i)$ equal to αC, which we denote as the "expectation condition" requirement.

What is more, as we need a "hard" pruning scheme eventually, this probabilistic pruning process should become deterministic in the end. Defining the inexactness as $\mathcal{E} = E[|\sum_i m_i - \alpha C|^2] = \sum_i \text{Var}[m_i] = \sum_i p_i(1 - p_i)$, a proper control on the inexactness is desired such that the inexactness $\mathcal{E}$ can reach 0 at the end of pruning.

The choice of f is important to the stability and controllability of the pruning process. An S-shaped function family w.r.t. b, $f(b; \beta) : \mathbb{R}^+ \to (0, 1)$, parametrized by at least two parameters is required, so that we can control the inexactness and satisfy the expectation condition at the same time. We choose the sigmoid-log function $f(b_i, \beta_1, \beta_2) = \text{Sigmoid} \circ \text{Log}(b_i) = \frac{1}{1+(\frac{b_i}{\beta_1})^{-\beta_2}}, \quad \beta_1, \beta_2 > 0$. This

Algorithm 1 DSA: Differentiable sparsity allocation

1: Run topological grouping, get K topological groups, and the budget model $\mathcal{F}$
2: $\mathcal{A} = \mathbf{1}_K$
3: **while** $\mathcal{F}(\mathcal{A}) > B_{\mathcal{F}}$ **do**
4: Update the keep ratios $\mathcal{A}$, auxiliary and dual variables following Eq. 8 in Sec. 4.3
5: Update weights W with SGD:
$W_T = W_{T-1} - \eta_w \frac{\partial L_t}{\partial W}|_{W_T}$
6: **end while**
7: **return** the pruned network, $\mathcal{A}$

function family has the desired property that, β_1 and β_2 could be used to control the expected keep ratio $E[\sum_{i=1}^{C} m_i]$ and the inexactness $\mathcal{E}$ in a relatively independent manner. 1) In our work, β_2 is a parameter that follows an increasing schedule. As β_2 approaches infinity, the inexactness $\mathcal{E}$ approaches 0, and β_1 becomes the hard pruning threshold of this layer. 2) $\beta_1 = \beta_1(\alpha)$ is a function of α. It has the interpretation of the soft threshold for the base importance score. It is calculated by solving the implicit equation of expectation condition:

$$g(\beta_1) = \frac{1}{C} E[\sum_{i=1}^{C} m_i] - \alpha = \frac{1}{C} \sum_{i=1}^{C} f(b_i, \beta_1, \beta_2) - \alpha = 0 \tag{2}$$

Since $g(\beta_1)$ is monotonically decreasing, the root $\beta_1(\alpha)$ could be simply and efficiently found (e.g., with bisection or Newton methods).

To summarize, utilizing the differentiable pruning process, the forward process of the k-th layer can be written as

$$\begin{aligned} & y^{(k)}(w, y^{(k-1)}; \alpha) = m \odot \text{Conv-BN-ReLU}(w, y^{(k-1)}) \\ \text{s.t.} \quad & m_i \sim \text{Bernoulli}(p_i), \quad \sum_{i=1}^{C} p_i = \sum_{i=1}^{C} f(\alpha, b_i) = \alpha C, \quad i = 1, \cdots, C \end{aligned} \tag{3}$$

where $y^{(k-1)}, w, y^{(k)}$ denote the inputs, weights, outputs of this layer, and the superscript k is omitted for simplicity.

Differentiable Instruction by the Task Loss The task loss L can be written as

$$L(\mathcal{A}, W) = E_{x \sim D}[E_{m_i^{(k)} \sim Ber(m; p_i^{(k)})}[\text{CE}(x, y; M, W)]] \tag{4}$$

where D is the training dataset, W denotes the weights, M is the set of masks $\{\{m_i^{(k)}\}_{i=1,\cdots,C^{(k)}}\}_{k=1,\cdots,K}$, $\mathcal{A}$ is the set of keep ratios $\{\alpha^{(k)}\}_{k=1,\cdots,K}$.

To enable differentiable tuning of the layer-wise keep ratios $\mathcal{A}$ instructed by both the task loss and the budget constraint, the major challenge is to derive the task loss's gradients w.r.t. $\alpha^{(k)}$: $\frac{\partial L}{\partial \alpha^{(k)}}$. First, we can calculate the implicit

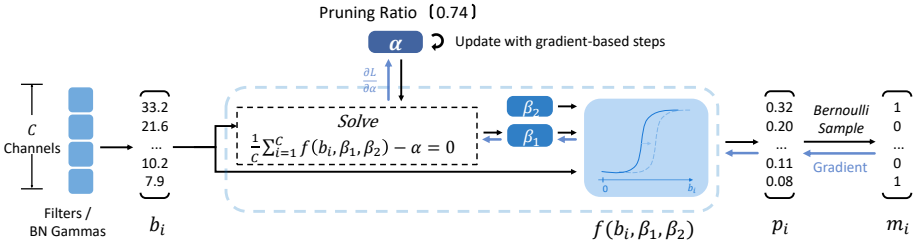


Fig. 3. The illustration of the differentiable pruning process of one layer. Given the base importances b_i and the keep ratio α, the process outputs channel-wise keep probabilities $p_i = f(b_i, \beta_1, \beta_2)$, which satisfy that the expectation condition $\sum_i^C p_i = \alpha C$. Then, the channel-wise masks m_i are sampled

gradient $\frac{\partial \beta_1(\alpha)}{\partial \alpha}$ as:

$$\begin{aligned} \frac{1}{C}\sum_{i=1}^{C} \frac{\partial f(b_i,\beta_1,\beta_2)}{\partial \beta_1}\frac{\partial \beta_1}{\partial \alpha} - 1 = 0 \\ \frac{\partial \beta_1(\alpha)}{\partial \alpha} = \frac{C}{\sum_{i=1}^{C} f'(b_i,\beta_1,\beta_2)} \end{aligned} \tag{5}$$

Then, $\frac{\partial L}{\partial \alpha^{(k)}}$ could be calculated as:

$$\frac{\partial L}{\partial \alpha^{(k)}} = \frac{\partial \beta_1}{\partial \alpha}\sum_{i=1}^{C}\frac{\partial L}{\partial p_i}\frac{\partial p_i}{\partial \beta_1} = C\sum_{i=1}^{C}\frac{\partial L}{\partial p_i}\hat{f}_i'; \quad \hat{f}_i' = \frac{f_i'}{\sum f_i'} \tag{6}$$

where the superscript k is omitted for simplicity and $\frac{\partial L}{\partial p_i}$ could be approximated using Monte-Carlo samples of the reparametrization gradients.

Eq. 6 could be interpreted as: The update of $\alpha^{(k)}$ is instructed using a weighted aggregation of the gradients of the task loss L w.r.t. the keep probabilities of channels $\frac{\partial L}{\partial p_i^{(k)}}$, and the aggregation weights are $f_i', i = 1, \cdots, C^{(k)}$.

4.2 Topological Grouping and Budget Modeling

For plain CNN, we can choose the layer-wise keep ratios $\alpha^{(k)}, k = 1, \cdots, K$ independently. However, for networks with shortcuts (e.g., ResNet), the naive scheme can lead to irregular computation patterns. Our grouping procedure for handling the topological constraints is described in Alg. 2 in the appendix. An example of grouping convolutions in two residual blocks is shown in the appendix.

The $\mathcal{F}(\mathcal{A})$ function models relationship of the keep ratios $\mathcal{A}$ and the resources. Taking FLOPs as an example, $\mathcal{F}(\mathcal{A})$ could be represented as $\mathcal{F}(\mathcal{A}) = \mathcal{A}^T\mathcal{F}_A\mathcal{A} + \mathcal{F}_B^T\mathcal{A}$. For completeness, we summarize the calculation procedure of $\mathcal{F}_A$ and $\mathcal{F}_B$ in Alg. 1 in the appendix. Under budget constraints for resources other than FLOPs, regression models can be fitted to get the corresponding $\mathcal{F}$ model.

Table 2. Pruning results of ResNet-20 and ResNet-56 on CIFAR-10. SSL and MorphNet are re-implemented with topological grouping. Accuracy drops for the referred results are calculated based on the reported baseline in their papers. **Headers**: "TG" stands for Topological Grouping; "FLOPs Budget" stands for the percentage of the pruned models' FLOPs compared to the full model

FLOPs Budget	Method	TG	ResNet-20		ResNet-56	
			FLOPs ratio	Acc. (Acc. Drop)	FLOPs ratio	Acc. (Acc. Drop)
Baseline	Ours		100 %	92.17 %	100 %	93.12 %
75%	SSL [21]	✓	73.8%	91.08% (-1.09%)	69.0%	92.06% (-1.06%)
	Variational* [24]		83.5%	91.66% (-0.41%)	79.7%	92.26% (-0.78%)
	PFEC* [10]	✓	-	-	74.4%	91.31% (-1.75%)
	MorphNet [4]	✓	74.9%	90.64% (-1.53%)	69.2%	91.71% (-1.41%)
	DSA (Ours)	✓	74.0%	**92.10% (-0.07%)**	70.7%	**93.08% (-0.04%)**
50% (×2)	SSL [21]	✓	51.8%†	89.78 % (-2.39%)	45.5%	91.22% (-1.90%)
	MorphNet [4]	✓	47.1%	90.1% (-2.07%)	51.9%†	91.55% (-1.57%)
	AMC* [7]	✓	-	-	50%	91.9% (-0.9%)
	CP* [8]		-	-	50%	91.8% (-1.0%)
	Rethink* [17]		60.0%	91.07% (-1.34%)	50%	93.07% (-0.73%)
	SFP* [5]		57.8%	90.83% (-1.37%)	47.4%	92.26% (-1.33%)
	FPGM* [6]		57.8%	91.09% (-1.11%)	47.4%	**92.93%** (-0.66%)
	LCCL* [3]		64.0%	**91.68%** (-1.06%)	62.1%	92.81% (-1.54%)
	DSA (Ours)	✓	49.7%	91.38% **(-0.79%)**	47.8%	92.91% **(-0.22%)**
33.3% (×3)	SSL [21]	✓	34.6%†	89.06% (-3.11%)	38.1%†	91.32% (-1.80%)
	MorphNet [4]	✓	30.5%	88.72% (-3.45%)	39.7%†	91.21% (-1.91%)
	DSA (Ours)	✓	32.5%	**90.24% (-1.93%)**	32.6%	**92.20% (-0.92%)**

†: These pruned models' FLOPs of SSL and MorphNet are higher than the budget constraints, since these regularization based methods lack explicit control of the resource consumption and even with carefully tuned hyperparameters, the resulting model might still violate the budget constraint.
*: These methods' results are directly taken from their paper, and their accuracy drops are calculated based on their reported baseline accuracies.

4.3 ADMM-inspired Method For Budgeted Pruning

ADMM is an iterative algorithm framework that is widely used to solve unconstrained or constrained convex optimization problems. Here, we use alternative gradient steps inspired by the methodology of ADMM to solve the constrained non-convex optimization problem.

Substituting the variable $\mathcal{A}$ by $\Theta = \text{Sigmoid}^{-1}(\mathcal{A})$, the $0 \leq \mathcal{A} \leq 1$ constraints are satisfied naturally. By introducing auxiliary variable z and the corresponding dual variable u_2 for the equality constraint $z = \Theta$, the augmented Lagrangian

is:

$$\mathcal{L}(\Theta, z, u_2) = L_v(\Theta) + I(\mathcal{F}(z) \leq B_{\mathcal{F}}) + u_2^T(\Theta - z) + \frac{\rho_2}{2}||\Theta - z||^2 \tag{7}$$

We then minimize the dual lower bound $\max_{u_2} \mathcal{L}(\Theta, z, u_2)$ of $L_v(\Theta)$. Eq. 8 shows the 3 alternative steps in one iteration. The variables with the superscript "$'$" denote the values at the previous time step.

$$\begin{aligned} \Theta &= \arg\min_{\Theta} L_v(\Theta) + u_2^T(\Theta - z') + \frac{\rho_2}{2}||\Theta - z'||^2 \\ z &= \arg\min_{z} u_2^T(\Theta - z) + \frac{\rho_2}{2}||\Theta - z||^2 \quad \text{s.t.} \quad \mathcal{F}(z) \leq B_{\mathcal{F}} \\ u_2 &= u_2' + \rho_2(\Theta - z) \end{aligned} \tag{8}$$

The unconstrained sub-problem for Θ is hard to solve, since $L_v(\Theta)$ is a stochastic objective and W can only be regarded as being constant in a local region. Therefore, in each iteration, we only do one stochastic gradient step on one validation batch for Θ.

To solve the inner problem for the auxiliary variable z with an inequality constraint, we use the standard trick of converting $\mathcal{F}(z) \leq B_{\mathcal{F}}$ to $[\mathcal{F}(z) - B_{\mathcal{F}}]_+ = 0$, and then use gradient descent to solve the min-max optimization of the augmented lagrangian $\mathcal{L}^{(z)}(z, u_1)$ in Eq. 9. In each iteration of the inner optimization, one gradient descent step is applied to z: $z = z' - \eta_z \nabla_z \mathcal{L}^{(z)}(z, u_1)$, and one dual ascent step is applied to u_1: $u_1 = u_1' + \rho_1[\mathcal{F}(\Theta) - B_{\mathcal{F}}]_+$. This optimization is efficient since only z and u_1 need to be updated.

$$\mathcal{L}^{(z)}(z, u_1) = u_1[\mathcal{F}(z) - B_{\mathcal{F}}]_+ \frac{\rho_1}{2}[\mathcal{F}(z) - B_{\mathcal{F}}]^2 + u_2^T(\Theta - z) + \frac{\rho_2}{2}||\Theta - z||^2 \tag{9}$$

The dual variables u_1, u_2 can be interpreted as the regularization coefficients, which are dynamically adjusted according to the constraint violations.

5 Experiments

5.1 Setup

We conduct the experiments on CIFAR-10 and ImageNet. For CIFAR-10, the batch size is 128, and an SGD optimizer with momentum 0.9, weight decay 4e-5 is used to train the model for 300 epochs. The learning rate is initialized to 0.05 and decayed by 10 at epochs 120, 180, and 240. The differentiable sparsity allocation is conducted simultaneously with normal training after 20 epochs of warmup. As for ImageNet, we use an SGD optimizer (weight decay 4e-5, batch size 256) to optimize the models for 120 epochs. The learning rate is 0.1 and decayed by 10 at epochs 50, 80, 110. The first 15 epochs remains plain training without pruning.

In the sparsity allocation process, 10% of the training data are used as the validation data for updating the keep ratios, while 90% are used for tuning the

Table 3. Pruning results on ImageNet. "TG" stands for Topological Grouping

Network	TG	Method	FLOPs Ratio	Top-1 Acc Drop	Top-5 Acc Drop
ResNet18		Baseline	100%	**69.72%**	**89.07%**
		MiL [3]	65.4%	-3.65%	-2.30%
		SFP [5]	60.0%	-3.18%	-1.85%
		FPGM [6]	60.0%	-2.47%	-1.52%
	✓	Ours	60.0%	**-1.11%**	**-0.718%**
ResNet50		Baseline	100%	**76.02%**	**92.86%**
		APG [9]	69.0%	-1.94%	-1.95%
		GDP [12]	60.0%	-2.52%	-1.25%
		SFP [5]	60.0%	-1.54%	-0.81%
		FPGM [6]	60.0%	-1.12%	-0.47%
	✓	Ours	60.0%	**-0.92%**	**-0.41%**
		ThiNet [18]	50.0%	-4.13%	-
		CP [8]	50.0%	-3.25%	-1.40%
		FPGM [6]	50.0%	-2.02%	-0.93%
		PFS [20]	50.0%	-1.60%	-
		Hinge [11]	46.55%	-1.33%	-
	✓	Ours	50.0%	**-1.33%**	**-0.8%**

weights. For the optimization of $\mathcal{A}$, the penalty parameters ρ_1, ρ_2 is set to 0.01, and the scaling coefficient of $L_v(\Theta)$ is 1e+5. z is updated for 50 steps with learning rate 1e-3 in the inner optimization. In practice, to reach the budget faster, we project the gradients of Θ to be nonnegative. After each update of $\mathcal{A}$, the weights are tuned for 20 steps for adaption. After acquiring the budget, the whole training set is used for updating the weights.

In the differentiable pruning process, the L_1 norms of BN scales are chosen as the base importance scores $b_i = |\gamma_i|$. $\beta_1(\alpha)$ is found by solving Eq. 2 with the bisection method. $\beta_2(T)$ follows a increasing schedule: starts at 0.05 and gets multiplied by 1.1 every epoch. As $\beta_2 \to \infty$, the soft pruning process becomes a hard pruning process, and the inexactness $\mathcal{E}$ goes to 0.

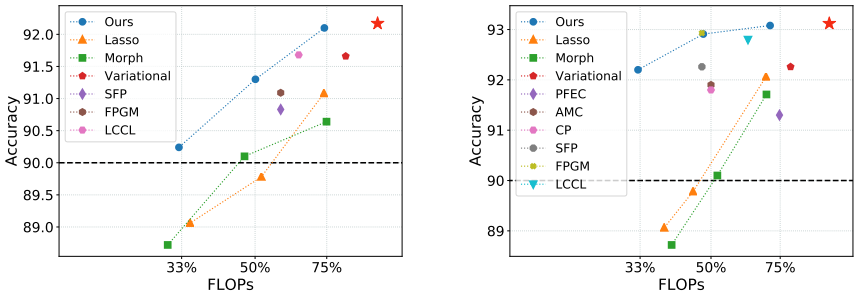
5.2 Results on CIFAR-10 and ImageNet

On CIFAR-10, for SSL [21] and MorphNet [4], the regularization coefficients on the convolution weights or BN scaling parameters are adjusted to meet various budget constraints.

Table. 2 and Fig. 4 show the results of pruning ResNet-20 and ResNet-56 on CIFAR-10. The pruned models obtained by DSA meet the budget constraints with smaller accuracy drops than the baseline methods. Compared with the reg-

ularization based methods (e.g., SSL and MorphNet), due to the explicit budget modeling, DSA guarantees that the resulting models meet different budget constraints, without hyperparameter trials. Compared with the iterative pruning methods (e.g., AMC), DSA allocates the sparsity in a gradient-based way and is more efficient (See Sec. 5.3). We also apply DSA to compress ResNet-18 and VGG-16, and the results are included in Appendix Table. 1 and Fig. 2. It shows that DSA outperforms the recent work based on a discrete search [14]. For ResNet-18, DSA achieves 94.19% versus 93.91% of the baseline with roughly the same FLOPs ratio. For VGG-16, DSA achieves 90.16% with 20.4× FLOPs reduction, which is significantly better than the baseline [14] (88.78% with 14.0× FLOPs reduction).

Table 3 shows the results of applying DSA to prune ResNet-18 and ResNet-50 on ImageNet. The official pre-trained models provided by PyTorch [19] are used as the baseline models. As could be seen, DSA consistently outperforms other methods across different FLOPs ratios and network structure. For example, DSA could achieve a small accuracy drop of 1.11% while keeping 60% FLOPs of ResNet-18, which is significantly better than the baselines. Note that we take the "'from-scratch" results of FPGM, since DSA also works in the "pruning-from-scratch" manner.

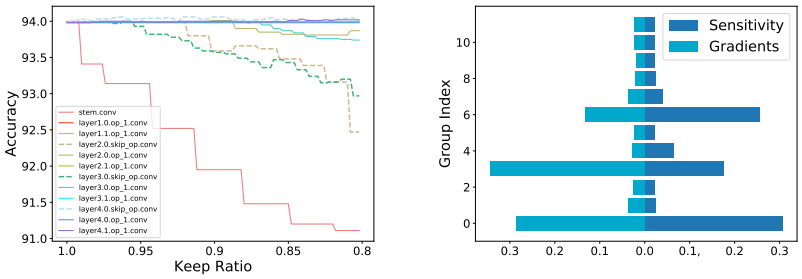


(a) ResNet-20 (b) ResNet-56

Fig. 4. Pruning results on CIFAR-10

5.3 Analysis and Discussion

Computational Efficiency [17] observes that once the pruned structure is acquired, the compressed model could achieve similar accuracy no matter it is trained from scratch or fine-tuned from the inherited weights. Therefore, starting with an over-parameterized pre-trained model might not be necessary for pruning. Unlike most iterative methods for inter-layer sparsity allocation, DSA could work in a "pruning from scratch" manner, without hurting the quality of sparsity allocation.



(a) Sensitivity analysis.

(b) The normalized magnitudes for layer-wise sensitivity and task loss's gradients in the first epoch.

Fig. 5. The alignment between sensitivity analysis and gradient magnitudes of ResNet-18 on CIFAR-10. The magnitudes are normalized by $\hat{v} = \text{softmax}(v/\text{std}(v))$

As shown in Fig. 1, traditional budgeted pruning consists of 3 stages: pre-training, sparsity allocation, and finetuning. The iterative pruning methods conduct hundreds of search-evaluation steps for sparsity allocation, e.g., AMC takes about 3 GPU hours for pruning ResNet-56 on CIFAR-10 [20]. After learning the structure, the finetuning stage takes 100˜200 epochs for CIFAR-10 (60 epochs for ImageNet), which accounts for about 2˜3 GPU hours for ResNet-56 on CIFAR-10 and 150 GPU hours for ResNet-18 on ImageNet. Moreover, these two stages should be repeated for multiple rounds to achieve the maximum pruning rates [23, 14], thus further increase the computational costs by several times. What's more, these methods need to be applied to the pre-trained models, and the pre-training stage takes about 300 and 120 epochs for models on CIFAR-10 and ImageNet. To summarize, the 3 stages can take up to 10 GPU hours for ResNet-56 on CIFAR-10, and 450 GPU hours for ResNet-18 on ImageNet. In contrast, the sparsity allocation in DSA is carried out in a more efficient gradient-based way, without the need of the pre-trained models. The extra cost of the sparsity allocation is small, since all the ADMM updates can be merged into the optimization of weights, and are conducted only once every tens of weight optimization steps. The whole DSA flow runs for 300 and 120 epochs on CIFAR-10 and ImageNet (5/300 GPU hours), thus speed up the overall pruning process by about 1.5×.

Rationality of the Differentiable Sparsity Allocation In DSA, the task loss's gradient w.r.t. layer-wise pruning ratios directly guides the budget allocation. To see whether the gradient magnitudes align well with the local sensitivity, we conduct an empirical sensitivity analysis for ResNet-18 on CIFAR-10. We prune each layer (topological group) independently with different pruning ratios according to the L_1 norm, and show the test accuracy in Fig. 5(a). Although

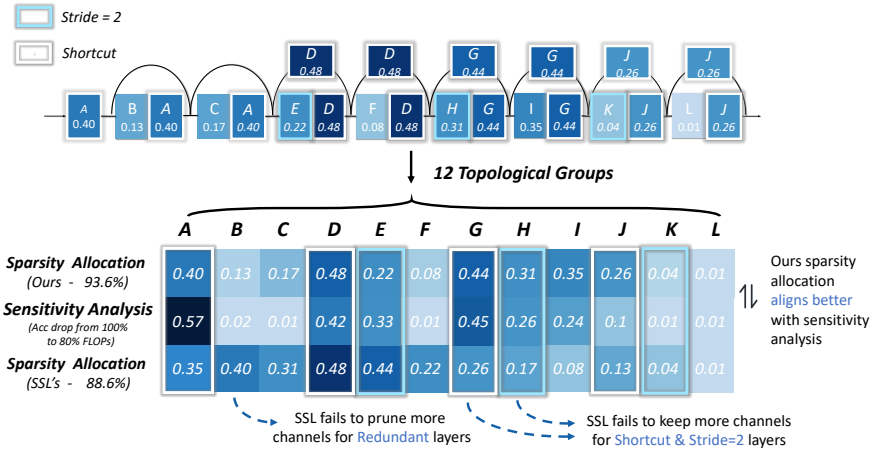


Fig. 6. The comparison between the normalized sensitivity, and the sparsity allocation of DSA and SSL [21] for ResNet-18 on CIFAR-10

this sensitivity analysis is heuristic and approximate, the accuracy drop could be interpreted as the local sensitivity for each group. Fig. 5(b) shows the normalized magnitudes of the task loss's gradients and the sensitivity of the layer-wise sparsity. We can see that these two entities align well, giving evidence that the task loss's gradient indeed encodes the relative layer-wise importance.

Fig. 6 presents the sparsity allocation (FLOPs budget 25%) for ResNet-18 on CIFAR-10 obtained by DSA and SSL [21]. We can see that the sparsity allocation of DSA coordinates better with sensitivity analysis. The results show that the first layer for primary feature extraction should not be pruned too aggressively (group A), so do shortcut layers that are responsible for information transmission across stages (groups A, D, G, J). The strided convolutions are relatively more sensitive, and more channels should be kept (groups E, H). In conclusion, DSA obtains reasonable sparsity allocation that matches empirical knowledge [15, 17], with lower computational cost than iterative pruning methods.

6 Conclusion

In this paper, we propose Differentiable Sparsity Allocation (DSA), a more efficient method for budgeted pruning. Unlike traditional discrete search methods, DSA optimizes the sparsity allocation in a *gradient-based way*. To enable the gradient-based sparsity allocation, we propose a novel *differentiable pruning process*, and verify that the magnitudes of the gradients w.r.t. the keep ratios align well with the layer-wise sensitivity. Experimental results show that DSA could achieve superior performance than iterative pruning methods, with significantly lower training costs.

References

1. Boyd, S., Parikh, N., Chu, E., Peleato, B., Eckstein, J., et al.: Distributed optimization and statistical learning via the alternating direction method of multipliers. Foundations and Trends® in Machine learning **3**(1), 1–122 (2011)
2. Chin, T.W., Ding, R., Zhang, C., Marculescu, D.: Towards efficient model compression via learned global ranking. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (2020)
3. Dong, X., Huang, J., Yang, Y., Yan, S.: More is less: A more complicated network with less inference complexity. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 5840–5848 (2017)
4. Gordon, A., Eban, E., Nachum, O., Chen, B., Wu, H., Yang, T.J., Choi, E.: Morphnet: Fast & simple resource-constrained structure learning of deep networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 1586–1595 (2018)
5. He, Y., Kang, G., Dong, X., Fu, Y., Yang, Y.: Soft filter pruning for accelerating deep convolutional neural networks. arXiv preprint arXiv:1808.06866 (2018)
6. He, Y., Liu, P., Wang, Z., Hu, Z., Yang, Y.: Filter pruning via geometric median for deep convolutional neural networks acceleration. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 4340–4349 (2019)
7. He, Y., Lin, J., Liu, Z., Wang, H., Li, L.J., Han, S.: Amc: Automl for model compression and acceleration on mobile devices. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 784–800 (2018)
8. He, Y., Zhang, X., Sun, J.: Channel pruning for accelerating very deep neural networks. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 1389–1397 (2017)
9. Huang, Z., Wang, N.: Data-driven sparse structure selection for deep neural networks. In: Proceedings of the European conference on computer vision (ECCV). pp. 304–320 (2018)
10. Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H.P.: Pruning filters for efficient convnets. ArXiv **abs/1608.08710** (2016)
11. Li, Y., Gu, S., Mayer, C., Gool, L.V., Timofte, R.: Group sparsity: The hinge between filter pruning and decomposition for network compression. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (June 2020)
12. Lin, S., Ji, R., Li, Y., Wu, Y., Huang, F., Zhang, B.: Accelerating convolutional networks via global & dynamic filter pruning. In: IJCAI. pp. 2425–2432 (2018)
13. Liu, H., Simonyan, K., Yang, Y.: Darts: Differentiable architecture search. arXiv preprint arXiv:1806.09055 (2018)
14. Liu, N., Ma, X., Xu, Z., Wang, Y., Tang, J., Ye, J.: Autocompress: An automatic dnn structured pruning framework for ultra-high compression rates (2019)
15. Liu, Z., Mu, H., Zhang, X., Guo, Z., Yang, X., Cheng, K.T., Sun, J.: Metapruning: Meta learning for automatic neural network channel pruning. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 3296–3305 (2019)
16. Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., Zhang, C.: Learning efficient convolutional networks through network slimming. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 2736–2744 (2017)
17. Liu, Z., Sun, M., Zhou, T., Huang, G., Darrell, T.: Rethinking the value of network pruning. arXiv preprint arXiv:1810.05270 (2018)

18. Luo, J.H., Wu, J., Lin, W.: Thinet: A filter level pruning method for deep neural network compression. In: Proceedings of the IEEE international conference on computer vision. pp. 5058–5066 (2017)
19. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: Pytorch: An imperative style, high-performance deep learning library. In: Advances in Neural Information Processing Systems 32, pp. 8024–8035. Curran Associates, Inc. (2019), http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf
20. Wang, Y., Zhang, X., Xie, L., Zhou, J., Su, H., Zhang, B., Hu, X.: Pruning from scratch. ArXiv **abs/1909.12579** (2019)
21. Wen, W., Wu, C., Wang, Y., Chen, Y., Li, H.: Learning structured sparsity in deep neural networks. In: Advances in neural information processing systems. pp. 2074–2082 (2016)
22. Yang, H., Zhu, Y., Liu, J.: Ecc: Platform-independent energy-constrained deep neural network compression via a bilinear regression model. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 11206–11215 (2019)
23. Yang, T.J., Howard, A., Chen, B., Zhang, X., Go, A., Sandler, M., Sze, V., Adam, H.: Netadapt: Platform-aware neural network adaptation for mobile applications. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 285–300 (2018)
24. Zhao, C., Ni, B., Zhang, J., Zhao, Q., Zhang, W., Tian, Q.: Variational convolutional neural network pruning. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2780–2789 (2019)

Appendices for DSA: More Efficient Budgeted Pruning via Differentiable Sparsity Allocation

1 Topological Grouping and Budget Model $\mathcal{F}$

An example of grouping convolutions in two consecutive residual blocks is shown in Fig. 1. All the incoming connections of the normal convolutions are removed, and then the convolutions in each connected component belong to the same topological group (i.e., share the same keep ratio $\alpha^{(k)}$ and masks $m^{(k)}_{i=1,\cdots,C}$).

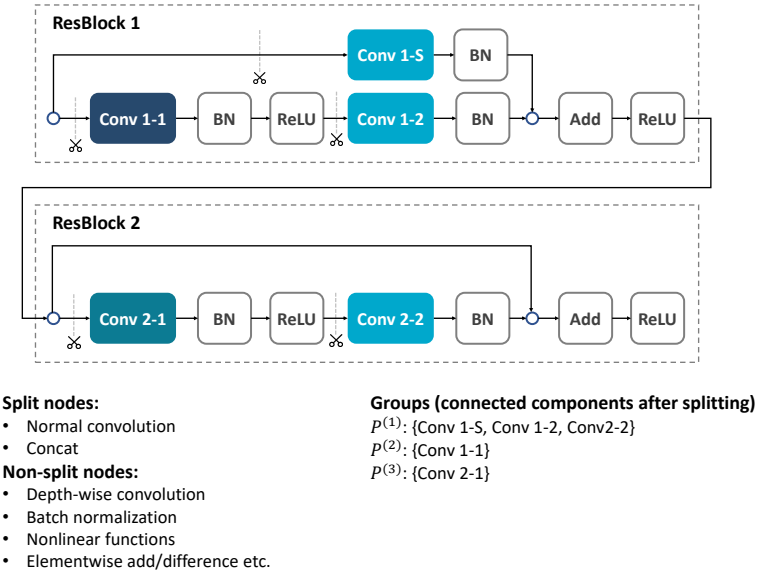


Fig. 1. An example of the topological grouping procedure

A budget model $\mathcal{F}$ is needed for measuring the resource consumption $\mathcal{F}(\mathcal{A})$ corresponding to the sparsity allocation $\mathcal{A}$. Taking FLOPs as an example, $\mathcal{F}(\mathcal{A})$ could be represented as $\mathcal{F}(\mathcal{A}) = \mathcal{A}^T\mathcal{F}_A\mathcal{A} + \mathcal{F}_B^T\mathcal{A}$. We summarize the calculation procedure of $\mathcal{F}_A$ and $\mathcal{F}_B$ in Alg. 3.

Algorithm 2 Topological grouping procedure

1: Construct the computational directed acyclic graph G
2: Removing all the incoming connections at split nodes (operation nodes that is not a channel-wise operation): normal convolution, concat operation.
NOTE: Non-split nodes include all channel-wise operations: depthwise convolution, element-wise add, ReLU, batch normalization, etc.
3: Find the connected components $\{P^{(k)}\}_{k=1,\cdots,K}$. All the convolution layers in each $P^{(k)}$ share the same keep ratio $\alpha^{(k)}$ and masks $m^{(k)}_{i=1,\cdots,C}$

Algorithm 3 Calculation of $\mathcal{F}_A, \mathcal{F}_B$ ($\mathcal{F}$ for FLOPs resource)

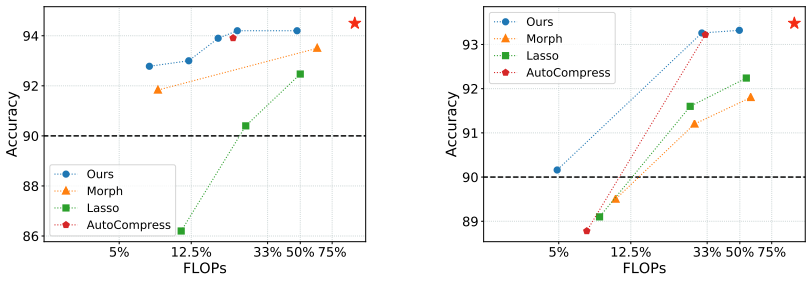
G: the directed acyclic graph of operations
K: number of connected components
$\mathcal{F}_A = \mathbf{0}_{K\times K}, \mathcal{F}_B = \mathbf{0}_K$
Convolution node attributes: 1) C: output channel number; 2) k: the index of the connected components that the convolution node belongs to; 3) kss: kernel spatial size (e.g. $3 \times 3 = 9$); 4) oss: output spatial size (e.g. $16 \times 16 = 256$)
for all convolution node M in G **do**
  if n.type == DEPTHWISE_CONV **then**
    $\mathcal{F}_B[M.k] += 2 \times M.c \times M.\text{kss} \times M.\text{oss}$
  else
    stack = [predecessor(M)]
    while stack **do**
      n = stack.pop()
      if n.type == CONCAT **then**
        for pn in predecessor(n) **do**
          stack.push(pn)
        end for
      else if n.type in ELEMENTWISE_OPs (e.g. ADD, ReLU) **then**
        stack.push(predecessor(n)[0])
      else if n.type == NORMAL_CONV **then**
        $\mathcal{F}_A[M.k, n.k] += 2 \times n.C \times M.C \times M.\text{kss} \times M.\text{oss}$
      else if n.type == INPUT **then**
        $\mathcal{F}_B[M.k] += 2 \times n.C \times M.C \times M.\text{kss} \times M.\text{oss}$
      else
        stack.push(n)
      end if
    end while
  end if
end for
return $\mathcal{F}_A, \mathcal{F}_B$

2 Additional Results

Table. 1 and Fig. 2 shows the results of pruning ResNet-18 and VGG-16 on CIFAR-10.

Table 1. Pruning results of ResNet-18 and VGG-16 on CIFAR-10

Method	ResNet-18 (94.0%)		VGG-16 (93.48%)	
	Acc.	FLOPs ratio	Acc.	FLOPs ratio
AutoCompress [14]	93.91%	4.7× (21.28%)	93.22%	3.1× (32.26%)
	-	-	88.78%	14.0× (7.14%)
DSA (Ours)	**94.19%**	4.5× (22.46%)	**93.26%**	3.2× (30.85%)
	93.90%	**5.7× (17.54%)**	90.16%	**20.4× (4.90%)**



(a) ResNet-18 (b) VGG-16

Fig. 2. Pruning results of ResNet-18, VGG-16 on CIFAR-10