Differential Model Scaling using Differential Topk

Anonymous Authors[1]

Abstract

Over the past few years, as large language models have ushered in an era of intelligence emergence, there has been an intensified focus on scaling networks. Currently, many network architectures are designed manually, often resulting in sub-optimal configurations. Although Neural Architecture Search (NAS) methods have been proposed to automate this process, they suffer from low search efficiency. This study introduces *Differential Model Scaling (DMS)*, increasing the efficiency for searching optimal width and depth in networks. DMS can model both width and depth in a direct and fully differentiable way, making it easy to optimize. We have evaluated our DMS across diverse tasks, ranging from vision tasks to NLP tasks and various network architectures, including CNNs and Transformers. Results consistently indicate that our DMS can find improved structures and outperforms state-of-the-art NAS methods. Specifically, for image classification on ImageNet, our DMS improves the top-1 accuracy of EfficientNet-B0 and Deit-Tiny by 1.4% and 0.6%, respectively, and outperforms the state-of-the-art zero-shot NAS method, ZiCo, by 1.3% while requiring only 0.4 GPU days for searching. For object detection on COCO, DMS improves the mAP of Yolo-v8-n by 2.0%. For language modeling, our pruned Llama-7B outperforms the prior method with lower perplexity and higher zero-shot classification accuracy. We will release our code in the future.

1. Introduction

In recent years, large models such as GPTs (Radford et al., 2018) and ViTs (Dosovitskiy et al., 2020) have showcased outstanding performance. Notably, the emergent intelligence of GPT4 (OpenAI, 2023) has underscored the importance of scaling networks as a critical pathway toward achieving artificial general intelligence (AGI). To support this scaling process, we introduce a general and potent method to determine the optimal width and depth of a network during its scaling.

Currently, the structure design of most networks still relies on human expertise. It typically demands significant resources to tune structural hyperparameters, making it challenging to pinpoint the optimal structure. Meanwhile, *Neural Architecture Search (NAS)* methods have been introduced to automate network structure design. We classify NAS methods into two categories based on their search strategies: *stochastic search methods* (Xie et al., 2022; Liu et al., 2022; Tan & Le, 2019) and *gradient-based methods* (Liu et al., 2018a; Wan et al., 2020; Guo et al., 2021a).

The stochastic search methods involve sampling numerous sub-networks to compare performance. However, these methods are limited to low search efficiency due to the sample-evaluate cycle, leading to reduced performance and increased search costs.

Unlike stochastic search methods, gradient-based methods employ gradient descent to optimize structural parameters, enhancing their efficiency and making them more adept at balancing search costs with ultimate performance. However, a significant challenge persists: *how to model structural hyperparameters in* ***a direct and differentiable manner***. Prior methods have struggled to meet this challenge, resulting in diminished performance and increased costs. Specifically, we group prior methods into three categories based on their modeling strategies: (1) *multiple element selection*, (2) *single number selection*, and (3) *gradient estimate topk*. Specifically, when searching for the number of channels in a convolutional layer, multiple element selection methods (Li et al.; Guo et al., 2021b) model the channel number as multiple selections of channels, as shown in Figure 1 (a.1). They introduce a much larger search space of element combinations. Single number selection methods (Wan et al., 2020) model the channel number as a single selection from multiple numbers, as shown in Figure 1 (a.2). It ignores the order relationship among these numbers. Gradient estimate topk approaches (Guo et al., 2021a; Gao et al., 2022; Ning et al., 2020) attempt to model width and depth directly, as

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

shown in Figure 1 (a.3). However, they are not differentiable, necessitating the development of different gradient estimation methods. As a result, these methods lack stability and are difficult to optimize.

Regrettably, all the above strategies fall short of modeling structural hyperparameters in a clear-cut and fully differentiable fashion. To address the aforementioned challenge, we introduce *a fully differentiable topk operator*, which can seamlessly model depths and widths in a direct and differential manner. Notably, each differential topk operator has a single learnable parameter, representing either a depth or width structural hyperparameter. It can be optimized based on guidance from both task loss and resource constraint loss. Our method stands out in terms of high optimization efficiency when contrasted with existing gradient-based approaches.

Based on our differential topk, we develop a *Differential Model Scaling (DMS)* algorithm to search for networks' optimal width and depth. To validate the efficacy and efficiency of our approach, we rigorously tested it across various tasks, including vision tasks and NLP tasks, and different architectures, including CNNs and Transformers. Thanks to the high search efficiency of our differential topk, DMS achieves better performance or much lower search costs than prior SOTA methods.

Overall, our contributions are as follows:

- We introduce a differential topk operator, which is easy to optimize as it can model structural hyperparameters in a direct and differentiable manner.
- We develop a Differential Model Scaling (DMS) algorithm based on our differential topk to search for networks' optimal width and depth.
- We evaluate our DMS across various tasks and architectures. For example, DMS outperforms the state-of-the-art zero-shot NAS method, ZiCo, by 1.3% while requiring only 0.4 GPU days for searching. DMS costs fewer than a fraction of dozens of the search costs of the one-shot NAS method, ScaleNet, and the multi-shot NAS method, ModelAmplification with comparable performance. Besides, our method is a widely applicable method, which improves the mAP of Yolo-v8-n by 2.0% on COCO and improves the zero-shot classification accuracy of pruned Llama-7B.

2. Related Work

The *width* and *depth* of networks are critical aspects of model architecture design. A multitude of methodologies have been proposed to automate this process, notably Neural Architecture Search (NAS) (Zoph & Le, 2016; Liu et al., 2018a) and *model structure pruning* (Li et al., 2020; Li et al.). NAS algorithms typically aim to design models automatically from scratch, while model structure pruning approaches focus on compressing pretrained models to enhance their efficiency. Despite their contrasting methodologies, both approaches contribute to the search for model structure.

These search methods can generally be categorized into two groups based on their search strategies: stochastic search methods (Zoph & Le, 2016; Xie et al., 2022; Liu et al., 2022) and gradient-based methods (Liu et al., 2018a; Guo et al., 2021a). In the following sections, we will introduce these methods and compare them with ours.

2.1. Stochastic Search Methods

Stochastic search methods usually operate through a cyclical process of sampling and evaluation. At each step, they sample models with different structures and then evaluate them. This strategy is versatile as it can handle both contiguous and discrete search spaces. However, a significant downside is its low search efficiency, leading to high resource consumption and suboptimal performance. Specifically, stochastic search-based methods can be divided into three groups: *multi-shot NAS*, *one-shot NAS*, and *zero-shot NAS*. Multi-shot NAS (Tan & Le, 2019; Liu et al., 2022) requires the training of multiple models, which is time-consuming. For instance, EfficientNet (Tan & Le, 2019) uses over 1714 TPU days for searching. One-shot NAS (Xie et al., 2022; Cai et al., 2019) requires training a large supernet, which is also resource-intensive. For example, ScaleNet (Xie et al., 2022) uses 379 GPU days for training a supernet. Zero-shot NAS (Li et al., 2023; Lin et al., 2021) reduces the cost by eliminating the need to train any model. However, its performance has not yet met the desired standard.

2.2. Gradient-based Methods

Gradient-based structure search methods (Liu et al., 2018a; Guo et al., 2021a) employ gradient descent to explore the structure of models. Generally, these methods are more efficient than their stochastic search counterparts. The critical aspect of gradient-based methods is how to use learnable parameters to model structural hyperparameters and compute their gradients. Ideally, the learnable parameters should directly model structural hyperparameters, and their gradients should be computed in a fully differentiable manner. However, prior methods have struggled to meet these two criteria in modeling the width and depth of networks. We group them into three categories: (1) multiple element selection, (2) single number selection, and (3) gradient estimate topk. The first two categories model structural hyperparameters indirectly, while the third category is not differentiable and requires gradient estimation.

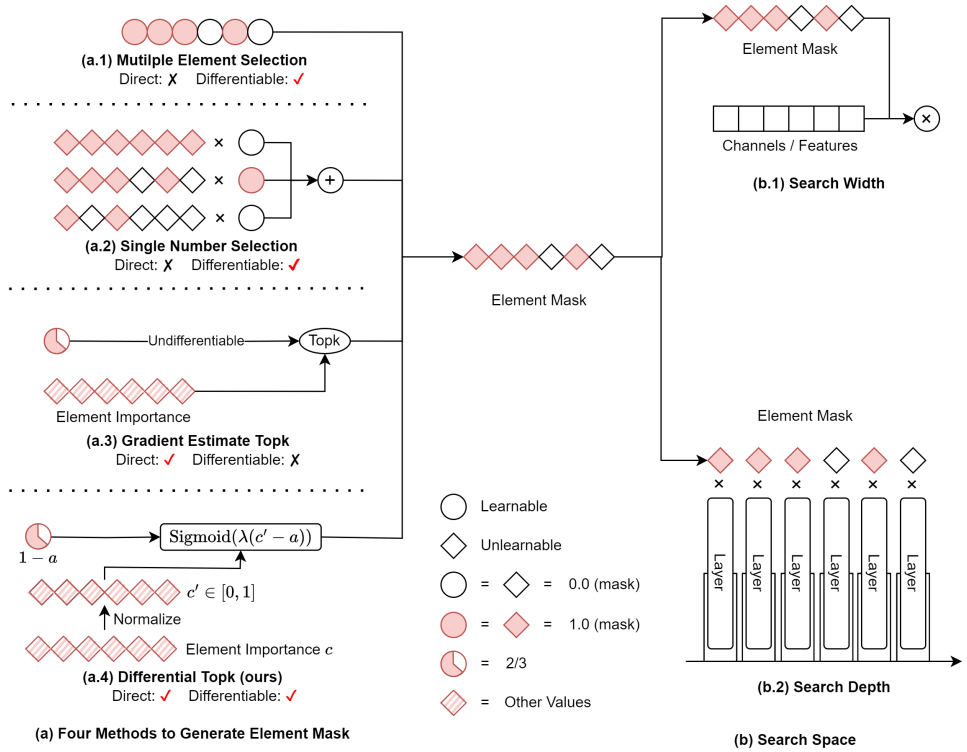


Figure 1. Different Gradient-based Modeling Strategies for Width and Depth. For all strategies, they use learnable parameters to generate an element mask to select width elements or depth elements. SubFigure **(a)** illustrates four methods to generate the element mask, while **(b)** shows how the mask is used to search width and depth. **(a.1)** Multiple Element Selection: The element count is transformed into a multiple-element selection. **(a.2)** Single Number Selection: The element count is transformed into a selection from multiple numbers. **(a.3)** Gradient Estimate Topk: The element count is directly modeled yet non-differentiable. **(a.4)** Our Differential Topk: The element count is directly modeled and is fully differentiable. "Direct" means that the learnable parameters directly model the structural hyperparameters, while "Differentiable" means that the gradient of the learnable parameters can be computed in a fully differentiable manner.

Multiple element selection methods (Li et al.) model the number of elements as multiple selections from elements (e.g., channel selection), as shown in Figure 1 (a.1). Similarly, Single number selection methods (Wan et al., 2020) model element quantity as a single choice from multiple numbers, as shown in Figure 1 (a.2). Both them model structural hyperparameters in indirect and inaccurate ways and introduce much more learnable structural parameters, making optimization hard. Naturally, They result in low performance.

Gradient estimate topk approaches (Guo et al., 2021a; Gao et al., 2022; Ning et al., 2020) attempt to model width and depth directly, as shown in Figure 1 (a.3). However, they are not differentiable, necessitating the development of different gradient estimation methods. As a result, these methods lack stability and are also difficult to optimize.

To improve the optimization efficiency for structure search, we introduce a new differential topk that can model width and depth directly and is fully differentiable. We compare our method and these search methods that are mentioned above in Section 4.1, The results show that our method is much more efficient and effective.

3. Method

In this section, we will detail our Differential Model Scaling (DMS) in two steps. First, we introduce our differential topk, which models structural hyperparameters directly in a fully differentiable manner. Second, we explain how to use our differential topk to construct our DMS algorithm.

3.1. Differential Top-k

Suppose there is a structural hyperparameter denoted by k, representing the number of elements, such as k channels in a convolutional layer or k residual blocks in a network stage. k has a maximal value of N. We use $\boldsymbol{c} \in \mathbb{R}^N$ to represent the *element importance*, where a larger value indicates a higher importance. The objective of our differential topk is to output a soft mask $\boldsymbol{m} \in [0,1]^N$ to indicate the selected elements with top k importance scores.

Our topk operator uses a learnable parameter a as a threshold to select elements whose importance values are larger than a. a is able to model number of elements k directly, as k can be seen as a function of a, where $k = sum_{i=1}^{N} 1[c_i > a]$. $1[A]$ is an indicator function, which equals 1 if the A is true and 0 otherwise. We use c_i to represent the importance of the i-th element. We denote our topk as a function f as follows:

$$m_i = f(a) \approx \begin{cases} 1 & \text{if } c_i > a \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

In prior methods, f is usually a piecewise function, which is not smooth and not differentiable, and the gradient of a is computed by estimation. We argue the biggest challenge to employing a fully differentiable f with respect to a is that the channel importance is distributed unevenly. Specifically, uneven distribution causes the importance difference between two neighboring elements, ordered by importance value, to vary significantly. Supposed a is updated by a fixed value in each iteration, when the difference is large, a lot of steps are needed for a to go across these two elements. When the difference is small, a can cross many elements in one step. Therefore, optimizing a in a fully differentiable manner is too hard when element importance is uneven.

To address this challenge, we employ an *importance normalization* process to forcefully convert the unevenly distributed importance to evenly distributed values, making the topk function smooth and easy to optimize in a differentiable way. To sum up, our differential topk has two steps: importance normalization and *soft mask generation*.

3.1.1. Importance Normalization

We normalize all element importance by mapping them to evenly distributed values from 0 to 1, based on the following:

$$c'_i = \frac{1}{N} \sum_{j=1}^{N} 1[c_i > c_j]. \tag{2}$$

The normalized element importance is denoted by $\boldsymbol{c}'$. $1[A]$ is the same indicator function as above. Any two elements in $\boldsymbol{c}$ are supposed to be different, which is usually the case in practice. Notably, although $\boldsymbol{c}'$ is evenly distributed from 0 to 1, $\boldsymbol{c}$ can follow any distribution.

Intuitively, c'_i indicates the portion of $\boldsymbol{c}$ values smaller than c_i. Besides, the learnable threshold a also becomes meaningful, representing the pruning ratio of elements. k can be computed by $k = \lfloor (1-a)N \rceil$, where $\lfloor \rceil$ is a round function. a is limited to the range of $[0, 1]$, where $a = 0$ indicates no pruning and $a = 1$ indicates pruning all elements. Our ablation study in Appendix A.3.1 demonstrates that the importance normalization is critical to the performance of our topk.

3.1.2. Soft Mask Generation

After the normalization, it's easy to generate the soft mask $\boldsymbol{m}$ using a smooth and differentiable function based on the relative size of pruning ratio a and normalized element importance $\boldsymbol{c}'$.

$$m_i = f(a) = \text{Sigmoid}(\lambda(\boldsymbol{c}'_i - a)) = \frac{1}{1 + e^{-\lambda(\boldsymbol{c}'_i - a)}}. \tag{3}$$

We add a hyperparameter λ to control the degree of approximation from Equation 3 to a hard mask generation function. When λ tends to infinity, Equation (3) approaches a hard mask generation function. We usually set λ to N. Because when $c'_i > a + 3/N$ or $c'_i < a - 3/N$, $|(m_i - \lfloor m_i \rceil)| < 0.05$. It means that except for the six elements whose importance values are around the pruning ratio, the masks of other elements are close to 0 or 1, where the approximation error is less than 0.05. Therefore, $\lambda = N$ is sufficient to approximate a hard mask generation function for our topk.

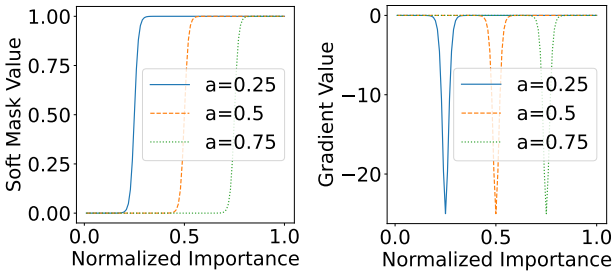
The forward and backward graph of Equation 3 are shown in Figure 2 (a) and Figure 2 (b), respectively. It can be observed that 1) Our topk models the number of elements k directly using the learnable pruning ratio a, and it generates a polarized soft mask $\boldsymbol{m}$ to simulate the pruned model perfectly during forward. 2) Our differential topk is fully differentiable and is able to be optimized stably. The gradient of a with respect to m_i is $\frac{\partial m_i}{\partial a} = -\lambda(1 - m_i)m_i$. Our topk intuitively detects the gradient of the mask in the fuzzy area with $0.05 < m_i < 0.95$. Note, Figure 2 (b) illustrates the value of $\frac{\partial m_i}{\partial a}$, which is not the total gradient of a. The total gradient of a is $\sum_{i=1}^{N} \frac{\partial task\_loss}{\partial m_i} \frac{\partial m_i}{\partial a} + \frac{\partial resource\_loss}{\partial a}$.

3.1.3. Element Evaluation

As we do not limit the distribution of element importance, element importance can be quantified through various methods, such as L1-norm (Li et al., 2016), among others. In our approach, we implement *Taylor importance* (Molchanov et al., 2019) in a *moving average* manner as follows:

$$c_i^{t+1} = c_i^t \times decay + (m_i^t \times g_i)^2 \times (1 - decay). \tag{4}$$

Here, t represents the training step. g_i is the gradient of m_i with respect to training loss. $Decay$ refers to the decay rate.



(a) Forward (b) Backward

Figure 2. Forwad and Backward Graph of Our Differential Topk. We set maximal element number $N = \lambda = 100$, pruning ratio $a \in \{0.25, 0.5, 0.75\}$. The x-axis represents the normalized element importance c_i'. **(a)** demonstrates the forward process, where the y-axis represents the soft mask m_i. **(b)** illustrates the backward process, where the y-axis represents the gradient of a with respect to m_i, $\frac{\partial m_i}{\partial a}$.

The initial value of c_i^0 is set to zero, and the decay rate is set to 0.99. Note that the importance of elements is not updated by gradient descent. By leveraging Taylor importance, we can efficiently and stably estimate the importance of elements. We conduct an ablation study on the importance evaluation methods in Section 5.1, which shows that Taylor importance is enough to achieve good performance.

3.2. Differential Model Scaling

Relying on our differential topk, we develop Differential Model Scaling (DMS) to optimize the width and depth of networks. Our DMS has three pipeline variants based on that of training-based model pruning, as shown in Table 1.

DMS$_{\text{p}}$ is the standard training-based model pruning pipeline, it consists of pretrain stage, search stage, and retrain stage. The pretrain stage is used to pretrain a supernet. It usually costs a lot of time and resources. In the search stage, we search for the optimal width and depth of the supernet under a specific resource constraint. Due to the high search efficiency of our method, the search stage only uses about 1/10 or fewer of the epochs of retraining. In the retrain stage, we retrain the searched model. We use this pipeline when comparing with SOTA pruning methods.

DMS$_{\text{np}}$ is our default and most-used pipeline in our paper. The high costs of pretrain stage, which may make up most of the total costs, is a big obstacle to the practical application of NAS and pruning methods. To overcome this problem, we discard the pretrain stage from the DMS$_{\text{p}}$and directly search from a randomly initialized supernet. According to our ablation study in Appendix A.3.2, DMS$_{\text{np}}$surpluses DMS$_{\text{p}}$by increasing the supernet size on both performance and efficiency. DMS$_{\text{np}}$makes our method maintain high performance and is more efficient than other NAS methods.

Table 1. Three Pipelines We Used in Our Paper.

Pipeline	Pretrain	Search	Retrain
DMS$_{\text{np}}$	N	All weights	Y
DMS$_{\text{p}}$	Y	All weights	Y
DMS$_{\text{p-}}$	Y	Only structural parameters	N

DMS$_{\text{p-}}$ is used to compare different search methods extremely quickly. Compared with DMS$_{\text{p}}$, it only optimizes the structural parameters and does not retrain the searched models. Take advantage of existing pretrained supernets, it also outputs reasonable results. Besides, it only takes hundreds of iterations, costing less than 10 minutes on a single RTX3090, to search for a model.

Search Space: As shown in Figure 1 (b), our search space encompasses both the width and depth of networks, which are the most critical structural hyperparameters for model scaling. To represent these dimensions, we use our differential topk. The width in networks typically covers the channel dimension in convolutional layers, the feature dimension in fully connected layers, and so on. Regarding depth, we focus on networks with residual connections and search the number of blocks in each stage. Specifically, We incorporate the soft masks of differential topk into residual connections, allowing each block to be represented as $x_{i+1} = x_i + f(x_i) \times m_i$.

Besides, for a structural hyperparameter x, we search it in the range of $[1, x_{max}]$ with a step of 1, while most of prior NAS methods (Cai et al., 2019; Chen et al., 2021) search it in the range of $[x_{min}, x_{max}]$ with a large step, like 32. Their search spaces have been a pretty good sub-space of ours, designed by human experts. However, our search space is more general and costs the least human effort. We conduct an ablation study on search spaces in Appendix A.3.6. The results show that our method can achieve better performance on our fine-grained search space, which is harder to search, than prior methods on coarse-grained search space, which is easier to search. More details about our search space are provided in Appendix A.1.1.

Resource Constraint Loss To ensure that a network adheres to specific *resource constraints*, we incorporate an additional component into the optimization process, termed the "*resource constraint loss*". Consequently, the aggregate loss function is:

$$loss = loss_{task} + \lambda_{resource} \times loss_{resource}. \quad (5)$$

$$loss_{resource} = \begin{cases} \log(\frac{r_c}{r_t}) & \text{if } r_c > r_t \\ 0 & \text{otherwise} \end{cases}. \quad (6)$$

Here, $loss_{task}$ denotes the *task loss*. $loss_{resource}$ repre-

sents the additional resource constraint loss, and the term $\lambda_{resource}$ acts as its weighting factor. r_c symbolizes the current level of resource consumption, which is calculated based on the learnable parameters of differential topk operators. r_t denotes the targeted level of resource consumption and is user-specified. As our topk is fully differentiable, the learnable structural parameters can be optimized under the guidance of both task loss and resource constraint loss. More details about our resource constraint loss are provided in Appendix A.1.2. MAC constraint loss is used in most experiments. We use latency constraint loss in Appendix A.3.3, and use the number of parameters constraint loss in Appendix A.2.4. They show that our method is compatible with various resource constraints.

4. Experiment

We applied our method to rigorous evaluations across various tasks, including vision and NLP tasks, and architectures, including CNNs and Transformers. We focus on search costs and performance. The search cost associated with a model is divided into two distinct components: the *public cost* that costs once for all searched models for a method, like supernet pretraining, and the *private cost* that costs for each model. Notably, our method consistently outperforms both baseline models and prior NAS methods, highlighting its superior performance and adaptability.

4.1. Comparison with Different Search Methods

To demonstrate the higher search efficiency of our method, we first compared it with different search methods in the same setting. For simplicity and fairness, we utilize "$\text{DMS}_{\text{p-}}$" pipeline for all methods in this section, which loads pretrained weights. As all methods load pretrained weights, we don't count pretrain costs in search costs in this section. All experiments in this section are conducted on ImageNet (Deng et al., 2009).

4.1.1. Comparison with Gradient-based Methods

Firstly, we compare our method with other gradient-based methods, including the multiple element selection (MES), single number selection (SNS), and gradient estimate topk (GET). We use ResNet-50 as our supernet and search for models with about 3G MACs. The search stage costs 800 iterations for all methods. The results are shown in Table 2.

Our method achieves the best performance among all gradient-based methods by a large margin. Compared with the multiple element selection and single number selection, our method uses much fewer (less than 1/250) learnable parameters, making it easier to optimize. Compared with the gradient estimate topk, our parameters get more accurate gradients, achieving much better performance. These results demonstrate our "direct and differentiable" manner is much more efficient than other gradient-based methods.

Table 2. Comparison with other Gradient-based Methods: "MES" means multiple element selection, "SNS" means single number selection, and "GET" means gradient estimate topk. "N" means the number of learnable structural parameters. We use "$\text{DMS}_{\text{p-}}$" pipeline for all methods in this table.

Method	Top-1 (%)	Macs (G)	N
MES (Herrmann et al., 2020)	55.5	3.2	11468
SNS (Wan et al., 2020)	58.6	3.4	11468
GET (Yao et al., 2021)	61.6	3	41
$\text{DMS}_{\text{p-}}$(ours)	**70.7**	3	41

Table 3. Comparison with Evolutionary Algorithm: "EA" means evolutionary algorithm. Only EA + predictor needs a public search cost, which is used to train an accuracy predictor. The unit for search cost is GPU hours. We use the pipeline of "$\text{DMS}_{\text{p-}}$" in this table.

Method	Top-1 (%)	Macs (G)	Cost Public+Private
EA + predictor (Cai et al., 2019)	78.2	0.45	40 + 0
EA (Cai et al., 2019)	78.2	0.45	0 + 1.1
$\text{DMS}_{\text{p-}}$(ours)	78.2	0.45	**0 + 0.05**

4.1.2. Comparison with Evolutionary Algorithm

We additionally compare our method with an *evolutionary algorithm (EA)*, a typical stochastic search method. We utilize the supernet of OFA (Cai et al., 2019) and search for models with 0.45G MACs. The results are shown in Table 3. Our DMS just consumes less than 1/20 of the search cost of the evolutionary algorithm without any performance degradation. It reveals that our method is more efficient than the evolutionary algorithm under the guidance of gradients.

4.2. Comparision with SOTA NAS Methods

Then, we compare our method with SOTA NAS methods on ImageNet. We choose EfficientNet models as baselines and search for optimal configurations in terms of their width and depth. EfficientNet (Tan & Le, 2019) is a widely accepted baseline for NAS research (Xie et al., 2022; Liu et al., 2022). We use our default pipeline "DMS_{np}", which does not load pretrained weights, in this section, Therefore, the public search cost for our method is zero.

The performance of these searched models and their search costs are presented in Table 4. According to the level of

Table 4. Experiments on EfficientNet. We compare our DMS with other NAS methods on EfficientNet variants. We divide all NAS methods into two groups, including low-search-cost methods and high-search-cost methods. Our method outperforms low-search-cost methods by a large margin with similar search costs, and it uses much fewer search costs than high-search-cost methods achieving comparable or better performance. The unit of search cost is TPU days for EfficientNet and GPU days for other models. "Ratio" stands for the ratio of the search cost of the model to that of our corresponding DMS model. ‡ means the model is trained with distillation. How to obtain these search costs is detailed in Appendix A.4.2. Note we use our default "DMS_{np}" pipeline, which does not load pretrained supernets, in this table.

Model	NAS Type	Top-1 (%)	MACs (G)	Params (M)	Search Cost Public + Private		Ratio	Cost Level
JointPruning‡ (Guo et al., 2021a)	Gradient	77.3	0.34	/	0 +	8	2.5×	Low
DMS_{np}-EN-350‡ (ours)	Gradient	**78.5**	0.35	5.6	**0 +**	**3.2**	**1×**	
Zen-score‡ (Lin et al., 2021)	ZeroShot	78.0	0.41	5.7	0 +	0.5	1.3×	
ZiCo‡ (Li et al., 2023)	ZeroShot	78.1	0.45	/	0 +	0.4	1×	
DMS_{np}*-EN-450‡ (ours)	Gradient	**79.4**	0.45	6.5	**0 +**	**0.4**	**1×**	
ScaleNet-EN-B0 (Xie et al., 2022)	OneShot	77.5	0.35	4.4	379 +	1.6	119×	High
DMS_{np}-EN-350 (ours)	Gradient	**78.0**	0.35	5.6	**0 +**	**3.2**	**1×**	
EfficientNet-B0 (Tan & Le, 2019)	MultiShot	77.1	0.39	5.3	1714 +	0	536×	
DMS_{np}-EN-B0 (ours)	Gradient	**78.5**	0.39	6.2	**0 +**	**3.2**	**1×**	
EfficientNet-B1 (Tan & Le, 2019)	MultiShot	79.1	0.69	7.8	1714 +	0	296×	
ScaleNet-EN-B1 (Xie et al., 2022)	OneShot	79.9	0.80	7.4	379 +	3.7	66×	
MA-EN-B1 (Liu et al., 2022)	MultiShot	79.9	0.68	8.8	> 124 +	131	> 44×	
DMS_{np}-EN-B1 (ours)	Gradient	**80.0**	0.68	8.9	**0 +**	**5.8**	**1×**	
EfficientNet-B2 (Tan & Le, 2019)	MultiShot	80.1	1.0	9.2	1714 +	0	245×	
MA-EN-B2 (Liu et al., 2022)	MultiShot	80.9	1.0	9.3	> 124 +	192	> 45×	
DMS_{np}-EN-B2 (ours)	Gradient	**81.1**	1.1	9.6	**0 +**	**7.0**	**1×**	

their search costs, we divide all compared NAS methods into two groups: *low-search-cost methods* and *high-search-cost methods*. Low-search-cost methods usually contain gradient-based and zero-shot NAS methods. This category only needs several GPU days, which are fewer than that of training the searched model itself. high-search-cost methods usually contain multi-shot and one-shot NAS methods, which usually cost more than one hundred GPU days (much over than training the searched model itself).

Compared with Low-Search-Cost Methods: Our method also belongs to low-search-cost methods. Compared with this category, our method achieves better performance with similar or even lower search costs. For example, our method outperforms JointPruning (Guo et al., 2021a) by 1.2% with 2/5 of its search cost. It also outperforms zero-shot NAS methods, ZiCo and Zen-score, by a margin of 1.3% and 1.4%, respectively. This result demonstrates that our method achieves significant performance improvements than previous low-search-cost NAS methods.

Compared with High-Search-Cost Methods: Compared with this category, our method spends much fewer search costs but still achieves comparable or even better performance. For example, Compared with EfficientNet, our searched models, DMS_{np}-EN-B0, B1, and B2, have improved performance by 1.4%, 0.9%, and 1.0%, respectively. Remarkably, DMS also achieves over 100 times search cost savings in the search process. our method outperforms ScaleNet by 0.5% and 0.1% on EfficientNet-B0 and B1, respectively, with less than 1/50 of the search cost of it. Our DMS also outperforms ModelAmplification (MA) by 0.1% and 0.2% on EfficientNet-B1 and B2, respectively, with less than 1/40 of the search cost of it. This result proves the high search efficiency of our method.

For high-search-cost methods, their search costs usually come from huge public costs, like supernet training. Even if we average their public cost over four variants (the number of variants used by most papers), they still cost over 20 times our total search cost. Except for the drawback of high search costs, the high-public-cost methods make it impossible to search a large model. For example, if a target model needs a month to train from scratch, like a LLM, the public search process of high-search-cost methods may take over a year, which is unacceptable for practical applications. While our method only needs a few days to search for a model, making it possible to search for a large model.

Through these experiments, we demonstrate that our method is more suitable for real-world applications, as it achieves comparable or even better performance than SOTA NAS methods with low search costs. It makes it easy to be inserted into the practical pipeline of model development. We compare our method with more NAS methods in Appendix A.2.1 and draw this table as accuracy vs MACs plots and a search cost vs accuracy plot in Figure 3.

Table 5. Comparison with SOTA Pruning Methods. The supernet is ResNet-50 for all methods. We use "DMS_p" pipeline, which loads pretrained weights as other methods, in this table.

Method	MACs (G)	Top-1 (%)
ResNet-50	4.1	76.5
LFPC (He et al., 2020)	1.6	74.46
GReg2 (Wang et al., 2020)	1.6	74.93
CC (Li et al., 2021)	1.5	74.54
TPP (Wang & Fu, 2022)	1.6	75.12
DMS_p(ours)	1.6	**75.53**

4.3. Comparison with SOTA pruning methods

Our method can also be applied as a model structure pruning method. We compare our method with SOTA pruning methods in Table 5. Following prior structure pruning methods, we utilize "DMS_p" pipeline, which loads pretrained weights, and only prune width.

Compared with SOTA pruning methods, our DMS achieves the best performance though we do not employ complicated importance evaluation methods like others. This is because of the strong search ability of our differential topk.

More experiments: Except for the above experiments, we also applied our method to more architectures and more tasks. The architectures include ResNet (He et al., 2016), MobileNetV2 (Sandler et al., 2018) and Deit (Touvron et al., 2021) (detailed in Appendix A.2.2). The tasks include object detection (detailed in Appendix A.2.3) and language modeling (detailed in Appendix A.2.4). Our method improves yolov8-n by 2.0% on COCO and improves pruned Llama-7B on several language modeling benchmarks. These results show that our method is highly versatile, rather than just being a method for one task or one architecture.

5. Ablation Study

5.1. Ablation Study on Element Importance Metrics

Here, we compare different element importance metrics, including Index metric, SNIP (Lee et al., 2018), Fisher (Liu et al., 2021), and Taylor (Molchanov et al., 2019). We use the pipeline of "DMS_{np}", which does not load pretrained weights, in this section. We search for 1G models on ResNet-50 and compare the performance after retraining.

Index metric is the simplest metric, which assigns importance value according to the index of elements statically. SNIP, Fisher, and Taylor are more complicated metrics, using gradients and activations to update the importance of elements with a moving average. The results are shown in Table 6. Index metric works poorly as its static strategy. SNIP, Fisher, and Taylor work better than the index metric and are comparable with each other for our method.

Table 6. Ablation Study on Element Importance Metric. We use the pipeline of "DMS_{np}" in this table.

Element Importance	Top-1 (%)
Index metric	72.3
SNIP (Lee et al., 2018)	73.0
Fisher (Liu et al., 2021)	**73.2**
Taylor w/o moving average	72.5
Taylor (Molchanov et al., 2019)	73.1

We further compare Taylor importance with and without moving average. The results are shown in Table 6. It can be seen that moving average can improve the performance of Taylor importance, because the moving average can smooth the importance of elements, making it more stable.

Therefore we just apply Taylor importance as our default metric, as it's more widely used in prior works (Humble et al., 2022; Molchanov et al., 2019).

More Ablation: There are more ablation studies in the Appendix A.3: (1) Ablation study on importance normalization, in Appendix A.3.1, demonstrate that importance normalization is significant for our method. (2) Ablation study on pretraining and supernet sizes, in Appendix A.3.2, shows that our DMS_{np}, which does not load pretrained weights, outperforms DMS_p, which loads pretrained weights, on efficiency and efficacy by increasing supernet size. (3) Experiments with latency constraint, in Appendix A.3.3, shows that our method is also compatible with latency constraints. (4) Ablation study on search time, in Appendix A.3.4, shows that our method is efficient in terms of search time. (5) Ablation study on our hyperparameters, in Appendix A.3.5, shows that our method has few hyperparameters that need to be tuned for each model, and they are easy to set. (6) Ablation study on search spaces, in Appendix A.3.6, shows that our method is compatible with other search spaces. Besides, our method can achieve better performance on our fine-grained search space, which is harder to search, than prior methods on coarse-grained search space.

6. Conclusion

In this paper, we introduce a novel model scaling method termed *Differential Model Scaling (DMS)*. Compared with prior NAS methods, our DMS has three advantages. (1) DMS is efficient for searching, which makes it easy to use. (2) DMS also achieves high performance, comparable with SOTA NAS methods. (3) DMS is universal and is compatible with various tasks and architectures. In conclusion, our DMS is a highly efficient and versatile method for model scaling, which is suitable for real-world applications.

References

Cai, H., Gan, C., Wang, T., Zhang, Z., and Han, S. Once-for-all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791*, 2019.

Chen, M., Peng, H., Fu, J., and Ling, H. Autoformer: Searching transformers for visual recognition. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 12270–12280, 2021.

Clark, C., Lee, K., Chang, M.-W., Kwiatkowski, T., Collins, M., and Toutanova, K. Boolq: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*, 2019.

Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C., and Tafjord, O. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Contributors, M. Openmmlab model compression toolbox and benchmark. https://github.com/open-mmlab/mmrazor, 2021.

Contributors, M. MMYOLO: OpenMMLab YOLO series toolbox and benchmark. https://github.com/open-mmlab/mmyolo, 2022.

Contributors, M. Openmmlab's pre-training toolbox and benchmark. https://github.com/open-mmlab/mmpretrain, 2023.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Fang, G., Ma, X., Song, M., Mi, M. B., and Wang, X. Depgraph: Towards any structural pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 16091–16101, 2023.

Frankle, J. and Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

Gao, S., Huang, F., Zhang, Y., and Huang, H. Disentangled differentiable network pruning. In *European Conference on Computer Vision*, pp. 328–345. Springer, 2022.

Guo, J., Liu, J., and Xu, D. Jointpruning: Pruning networks along multiple dimensions for efficient point cloud processing. *IEEE Transactions on Circuits and Systems for Video Technology*, 32(6):3659–3672, 2021a.

Guo, Y., Yuan, H., Tan, J., Wang, Z., Yang, S., and Liu, J. Gdp: Stabilized neural network pruning via gates with differentiable polarization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 5239–5250, 2021b.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

He, Y., Ding, Y., Liu, P., Zhu, L., Zhang, H., and Yang, Y. Learning filter pruning criteria for deep convolutional neural networks acceleration. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 2009–2018, 2020.

Herrmann, C., Bowen, R. S., and Zabih, R. Channel selection using gumbel softmax. In *European Conference on Computer Vision*, pp. 241–257. Springer, 2020.

Humble, R., Shen, M., Latorre, J. A., Darve, E., and Alvarez, J. Soft masking for cost-constrained channel pruning. In *European Conference on Computer Vision*, pp. 641–657. Springer, 2022.

Jocher, G., Chaurasia, A., and Qiu, J. YOLO by Ultralytics, January 2023. URL https://github.com/ultralytics/ultralytics.

Lee, N., Ajanthan, T., and Torr, P. H. Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340*, 2018.

Li, B., Wu, B., Su, J., and Wang, G. Eagleeye: Fast sub-net evaluation for efficient neural network pruning. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16*, pp. 639–654. Springer, 2020.

Li, G., Yang, Y., Bhardwaj, K., and Marculescu, R. Zico: Zero-shot nas via inverse coefficient of variation on gradients. *arXiv preprint arXiv:2301.11300*, 2023.

Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

Li, Y., Zhao, P., Yuan, G., Lin, X., Wang, Y., and Chen, X. Pruning-as-search: Efficient neural architecture search via channel pruning and structural reparameterization.

Li, Y., Lin, S., Liu, J., Ye, Q., Wang, M., Chao, F., Yang, F., Ma, J., Tian, Q., and Ji, R. Towards compact cnns via collaborative compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6438–6447, 2021.

Lin, M., Wang, P., Sun, Z., Chen, H., Sun, X., Qian, Q., Li, H., and Jin, R. Zen-nas: A zero-shot nas for high-performance image recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 347–356, 2021.

Liu, C., Han, K., Xiao, A., Nie, Y., Zhang, W., and Wang, Y. Network amplification with efficient macs allocation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1933–1942, 2022.

Liu, H., Simonyan, K., and Yang, Y. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018a.

Liu, L., Zhang, S., Kuang, Z., Zhou, A., Xue, J.-H., Wang, X., Chen, Y., Yang, W., Liao, Q., and Zhang, W. Group fisher pruning for practical network compression. In *International Conference on Machine Learning*, pp. 7021–7032. PMLR, 2021.

Liu, Z., Sun, M., Zhou, T., Huang, G., and Darrell, T. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*, 2018b.

Ma, X., Fang, G., and Wang, X. Llm-pruner: On the structural pruning of large language models. *arXiv preprint arXiv:2305.11627*, 2023.

Marcus, M., Santorini, B., and Marcinkiewicz, M. A. Building a large annotated corpus of english: The penn treebank. 1993.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Molchanov, P., Mallya, A., Tyree, S., Frosio, I., and Kautz, J. Importance estimation for neural network pruning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11264–11272, 2019.

Moons, B., Noorzad, P., Skliar, A., Mariani, G., Mehta, D., Lott, C., and Blankevoort, T. Distilling optimal neural networks: Rapid search in diverse spaces. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 12229–12238, 2021.

Ning, X., Zhao, T., Li, W., Lei, P., Wang, Y., and Yang, H. Dsa: More efficient budgeted pruning via differentiable sparsity allocation. In *European Conference on Computer Vision*, pp. 592–607. Springer, 2020.

OpenAI. Gpt-4 technical report, 2023.

Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., et al. Improving language understanding by generative pre-training. 2018.

Sakaguchi, K., Bras, R. L., Bhagavatula, C., and Choi, Y. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018.

Tan, M. and Le, Q. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pp. 6105–6114. PMLR, 2019.

Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., and Le, Q. V. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 2820–2828, 2019.

Taori, R., Gulrajani, I., Zhang, T., Dubois, Y., Li, X., Guestrin, C., Liang, P., and Hashimoto, T. B. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca, 2023.

Touvron, H., Cord, M., Douze, M., Massa, F., Sablayrolles, A., and Jégou, H. Training data-efficient image transformers & distillation through attention. In *International conference on machine learning*, pp. 10347–10357. PMLR, 2021.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Wan, A., Dai, X., Zhang, P., He, Z., Tian, Y., Xie, S., Wu, B., Yu, M., Xu, T., Chen, K., et al. Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 12965–12974, 2020.

Wang, H. and Fu, Y. Trainability preserving neural structured pruning. *arXiv preprint arXiv:2207.12534*, 2022.

Wang, H., Qin, C., Zhang, Y., and Fu, Y. Neural pruning via growing regularization. *arXiv preprint arXiv:2012.09243*, 2020.

Wightman, R. Pytorch image models. https://github.com/rwightman/pytorch-image-models, 2019.

Wightman, R., Touvron, H., and Jégou, H. Resnet strikes back: An improved training procedure in timm. *arXiv preprint arXiv:2110.00476*, 2021.

Xie, J., Su, X., You, S., Ma, Z., Wang, F., and Qian, C. Scalenet: Searching for the model to scale. In *European Conference on Computer Vision*, pp. 104–120. Springer, 2022.

Yao, Z., Wu, X., Ma, L., Shen, S., Keutzer, K., Mahoney, M. W., and He, Y. Leap: Learnable pruning for transformer-based models. *arXiv preprint arXiv:2105.14636*, 2021.

Zhuang, T., Zhang, Z., Huang, Y., Zeng, X., Shuang, K., and Li, X. Neuron-level structured pruning using polarization regularizer. *Advances in neural information processing systems*, 33:9865–9877, 2020.

Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

A. Appendix

A.1. More Details about DMS

A.1.1. Search Space

Our search space encompasses both the width and depth of networks, which are the most critical structural hyperparameters for model scaling.

The width in networks typically covers the channel dimension in convolutional layers, the feature dimension in fully connected layers, qkv dimension and the number of heads in attention mechanisms, among others. For convolutional and fully connected layers, we use two distinct differential topk operators to model their respective input and output widths, treating each channel or feature as an individual element. For multi-head attention, we employ a single differential topk to represent the number of heads, treating each head as a separate element.

Specifically, We apply our differential topk to different layers by multiplying masks, output by differential topk operators, with inputs to layers. For convolutional layers, suppose the input is $X \in \mathbb{R}^{B\times C\times H\times W}$, and the mask is reshaped as $m \in \mathbb{R}^{1\times C\times 1\times 1}$, $X \times m$ works as the new input to the layer. For an attention layer, we search the head dims of qkv and the number of heads. Suppose our supernet has H heads and D dims in each head. We have a mask for qk head dim with $m_{qk} \in R^{1\times 1\times 1\times D}$, a mask for v head dim with $m_v \in R^{1\times 1\times 1\times D}$, and a mask for number of heads $m_{head} \in R^{1\times H\times 1\times 1}$. Suppose the sequence length is L, and the qkv for self-attention is $Q, K, V \in R^{B\times H\times L\times D}$. We compute the output of the self-attention by $softmax(\frac{Q'K'^T}{\sqrt{D}})V'$, where $Q' = Q \times m_{qk} \times m_{head}, K' = K \times m_{qk} \times m_{head}, V' = V \times m_v \times m_{head}$.

It is crucial to highlight that there can be channel or feature dependencies within models (Liu et al., 2021; Fang et al., 2023). Interdependent Layers are treated as one group and share the same differential topk. We implemented this using an open-source model compression toolkit MMRazor (Contributors, 2021), which is able to build element dependencies automatically.

Regarding depth, we focus on networks with residual connections. In this context, a residual block can be defined as $x_{i+1} = x_i + f(x_i)$, and contiguous residual blocks are viewed as a network stage. The depth in our approach mainly comprises the number of blocks in each stage. We use a single differential topk for a network stage, with each block functioning as a distinct element. We incorporate the soft masks of differential topk into residual connections, allowing each block to be represented as $x_{i+1} = x_i + f(x_i) \times m_i$. In the context of Transformers, an attention mechanism combined with a feed-forward network (FFN) is considered as one block sharing the same soft mask.

The depth and width structure hyperparameters are trained jointly in our approach. For example, we have a $layer$ and an input x; we use $m_{L_i} \in [0, 1]$ to denote the depth mask and $m_C \in [0, 1]^N$ for the width mask. The forward process is as follows: $y = m_C \times x + m_{L_i} \times layer(m_C \times x)$. After searching, we will prune depth and width according to the depth mask and width mask, respectively.

Besides, as some maximal numbers of elements are small from several to tens, like the number of blocks and attention heads, we increase the λ in the differential topk operators from N to $4N$ to approximate a hard mask generation function better.

For a structural hyperparameter x, we search it, in a fine-grained manner, in the range of $[1, x_{max}]$ with a step of 1. while most of prior NAS methods (Cai et al., 2019; Chen et al., 2021) search it, in a coarse-grained manner, in the range of $[x_{min}, x_{max}]$ with a large step, like 32. Our search method is also comparable with these coarse-grained search spaces. We limit the value of a to $[0, 1 - \frac{x_{min}}{x_{max}}]$ to ensure $x \in [x_{min}, x_{max}]$. We treat contiguous elements in a step as an unit, and each unit share the same element importance and mask. For example, when searching the number of channels in a layer with a step of 32, each 32 channels share the same element importance and mask.

A.1.2. Resource Constrait

Our resource constraint loss is defined as:

$$loss_{resource} = \begin{cases} \log(\frac{r_c}{r_t}) & \text{if } r_c > r_t \\ 0 & \text{otherwise} \end{cases}. \tag{7}$$

$$\tag{8}$$

In this definition, r_c symbolizes the current level of resource consumption, and r_t denotes the targeted level of resource

consumption. If r_c exceeds r_t, a non-zero $loss_{resource}$ is used to compress the model. The value of r_t is user-specified. The value of r_c is calculated based on the learnable parameters of differential topk operators. Take a linear layer as an example, we use f_{in} and f_{out} to represent the number of input and output features, respectively. a_{in} and a_{out} are the learnable parameters of differential topk operators for that layer. We compute different resource consumption as follows:

MAC constraint: $r_c = f_{in} \times a_{in} \times f_{out} \times a_{out} \times batchsize$

The Number of Parameters constraint: $r_c = f_{in} \times a_{in} \times f_{out} \times a_{out}$

Latency constraint: $r_c = latency_{max} \times F(a_{in}, a_{out})$, where $latency_{max}$ is the latency of the layer without pruning. F is a contiguous function to map the pruning ratio a_{in},a_{out} to the ratio of the latency of the pruned layer to the original layer. F can be found by:

$$F = argmin_F(MSE(latency_{pruned(a_{in},a_{out})} - latency_{max} \times F(a_{in}, a_{out}))) \tag{9}$$

$latency_{pruned(a_{in},a_{out})}$ is the latency of the pruned layer with a_{in},a_{out} as the pruning ratio. Take the OFA search space as an example, a_{out} always equals a_{in}, and each block only has three width choices. Hence, we just employ a quadratic function as F, and it works fine.

To enhance stability during training, we gradually reduce r_t to the final target value r_t^{final} throughout the training process by default. For an epoch-based training procedure, r_t is determined by an exponential decay function as shown below:

$$r_t = (\frac{r_t^{final}}{r_{supernet}})^{\frac{e}{e_{max}}} \times r_{supernet}, \quad \text{where } \frac{r_t^{final}}{r_{supernet}} < 1. \tag{10}$$

Here, e denotes the current epoch out of total epochs e_{max}. $r_{supernet}$ is a constant representing the resource demand of the supernet.

Furthermore, since resource consumption can fluctuate significantly with respect to depth, we introduce extra epochs dedicated to optimizing width while maintaining depth constant. By adopting the strategies outlined above, Differential Model Scaling ensures that models adhere to specific resource constraints.

A.2. More Experiments

A.2.1. Comparison with More NAS Methods

We additionally compare our method with more NAS methods, as shown in Table 7. Note this is a rough comparison. As it's hard to compute a precise search cost for some methods, we only group them into two groups, including high-search-cost methods and low-search-cost methods, according to a rough estimation of their search costs. High-search-cost methods costs over 100 GPU days. while others belong to low-search-cost methods.

We draw Table 4 and Table 7 as accuracy vs MACs plots and a search costs vs accuracy plot, as shown in Figure 3. It can be observed that our DMS outperforms low-search-cost methods significantly. DMS achieves much lower search costs than high-search-cost methods, achieving comparable and even higher performance.

A.2.2. Image Classification Experiments on More Architectures

To validate the universality of our method across various model architectures, we applied it to different architectures, as shown in Table 8.

Classic CNNs: We validated our method on ResNet (He et al., 2016) and MobileNetV2 (Sandler et al., 2018). Our searched ResNet surpasses ResNet-50 by 1.1%. Furthermore, when the searched ResNet is trained using an enhanced training setting (referred to as rsb-a1 (Wightman et al., 2021)), it also exceeds the corresponding model by 0.9%. Although MobileNetV2 is a lightweight model, our searched version outperforms the original model by a margin of 1.0%.

Transformer: We additionally applied our method to Deit, a one-stage transformer. Our searched models outperform the original model by 0.6%.

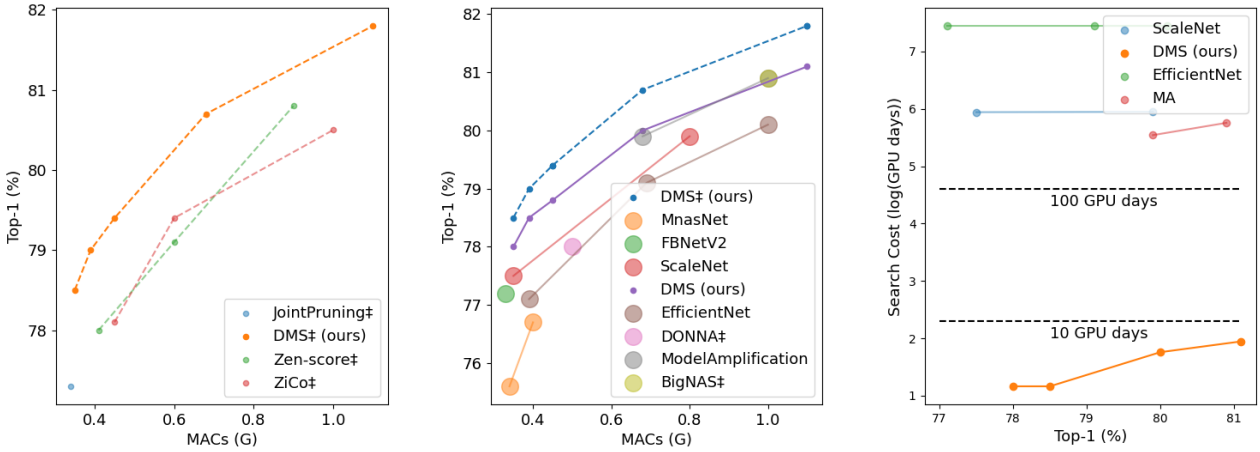
These results demonstrate the strong ability of our DMS to search for the optimal structure of models.

Table 7. Rough Comparison with more NAS methods. As some methods did not report their search cost, we simply use "High" and "Low" to represent the search cost of NAS methods, while "High" for multi-shot NAS methods and one-shot NAS methods, "Low" for gradient-based NAS methods and zero-shot NAS methods. ‡ means the model is trained with distillation. We use the pipeline of "DMS_{np}", which does not load pretrained weights, in this table

Model	NAS Type	Top-1 (%)	MACs (G)	Params (M)	Cost Level
JointPruning‡ (Guo et al., 2021a)	Gradient	77.3	0.34	/	Low
DMS_{np}-EN-350‡ (ours)	Gradient	**78.5**	0.35	5.6	
Zen-score‡ (Lin et al., 2021)	ZeroShot	78.0	0.41	5.7	
DMS_{np}-EN-B0‡ (ours)	Gradient	**79.0**	0.39	6.2	
ZiCo‡ (Li et al., 2023)	ZeroShot	78.1	0.45	/	
DMS_{np}*-EN-450‡ (ours)	Gradient	**79.4**	0.45	6.5	
Zen-score‡ (Lin et al., 2021)	ZeroShot	79.1	0.60	7.1	
ZiCo‡ (Li et al., 2023)	ZeroShot	79.4	0.60	/	
DMS_{np}-EN-B1‡ (ours)	Gradient	**80.7**	0.68	8.9	
ZiCo‡ (Li et al., 2023)	ZeroShot	80.5	1.0	/	
Zen-score‡ (Lin et al., 2021)	ZeroShot	80.8	0.9	19.4	
DMS_{np}-EN-B2‡ (ours)	Gradient	**81.8**	1.1	9.6	
MnasNet-A2 (Tan et al., 2019)	MultiShot	75.6	0.34	4.8	High
FBNetV2-L1 (Wan et al., 2020)	Gradient	77.2	0.33	/	
ScaleNet-EN-B0 (Xie et al., 2022)	OneShot	77.5	0.35	4.4	
DMS_{np}-EN-350 (ours)	Gradient	**78.0**	0.35	5.6	
MnasNet-A3 (Tan et al., 2019)	MultiShot	76.7	0.40	5.2	
EfficientNet-B0 (Tan & Le, 2019)	MultiShot	77.1	0.39	5.3	
DMS_{np}-EN-B0 (ours)	Gradient	**78.5**	0.39	6.2	
DONNA‡ (Moons et al., 2021)	OneShot	78.0	0.50	/	
DMS_{np}*-EN-450 (ours)	Gradient	**78.8**	0.45	6.5	
DMS_{np}*-EN-450‡ (ours)	Gradient	**79.4**	0.45	6.5	
EfficientNet-B1 (Tan & Le, 2019)	MultiShot	79.1	0.69	7.8	
DMS_{np}-EN-B1 (ours)	Gradient	**80.0**	0.68	8.9	
ScaleNet-EN-B1 (Xie et al., 2022)	OneShot	79.9	0.80	7.4	
ModelAmplification-EN-B1 (Liu et al., 2022)	MultiShot	79.9	0.68	8.8	
EfficientNet-B2 (Tan & Le, 2019)	MultiShot	80.1	1.0	9.2	
ModelAmplification-EN-B2 (Liu et al., 2022)	MultiShot	80.9	1.0	9.3	
BigNAS-XL‡ (Liu et al., 2022)	OneShot	80.9	1.0	9.5	
DMS_{np}-EN-B2 (ours)	Gradient	**81.1**	1.1	9.6	
DMS_{np}-EN-B2‡ (ours)	Gradient	**81.8**	1.1	9.6	

Table 8. Experiments on ImageNet with Various Architectures. We searched the models' width and depth and compared them with the original models. We use the pipeline of "DMS_{np}", which does not load pretrained weights, in this table.

Model	Top-1 (%)	MACs (G)	Params (M)
ResNet-50 (He et al., 2016)	76.5	4.1	25.6
DMS-ResNet	**77.6**	4.0	28.4
ResNet-50-rsb-a1 (He et al., 2016)	80.1	4.1	25.6
DMS-ResNet-rsb-a1	**81.0**	4.0	28.4
MobileNetV2 (Sandler et al., 2018)	72.0	0.3	3.4
DMS-MobileNetV2	**73.0**	0.3	5.3
Deit-T (Touvron et al., 2021)	74.5	1.3	5.7
DMS-Deit-T	**75.1**	1.3	6.2



(a) (b) (c)

Figure 3. We draw these three plots based on Table 4 and Table 7. We use larger dot sizes to represent the "High" search cost level and smaller dot sizes to represent the "Low" search cost level. Dashed lines are used to represent the models trained with distillation. **(a) Performance Comparison with Low-Search-Cost methods** It can be seen that our method outperforms these methods significantly. **(b) Performance Comparison with High-Search-Cost methods** Our method achieves comparable or even better performance, while all high-search-cost methods cost more than dozens of times our total search costs. **(c) Search Cost Comparison with High-Search-Cost methods** We compare the search costs of ours and that of high-search-cost methods. We only draw methods with precise search cost estimation. The search costs of unpainted high-search-cost methods are also larger than 100 GPU days. We present search costs on a log scale.

Table 9. Object Detection Experiments on COCO. We use the pipeline of "DMS_{np}", which does not load pretrained weights, in this table.

Model	mAP (%)	MACs (G)	Params (M)
Yolo-v8-n (Jocher et al., 2023)	37.4	4.4	3.2
DMS_{np}-Yolo-v8-n (ours)	**39.4**	4.2	2.7

A.2.3. Object Detection Experiments on COCO

Since the complete end-to-end searching of our differential topk, DMS is a general search method that can be applied to various tasks. We also evaluated DMS for object detection on COCO. We chose Yolo-v8-n (Jocher et al., 2023) as the baseline model and searched for the optimal structure of it. Our searched version betters the original model by 2.0% in box AP, as shown in Table 9.

A.2.4. LLM Experiment

Beyond vision tasks, we extended our method to evaluate its applicability on a large language model (LLM) called Llama (Touvron et al., 2023), as shown in Table 10. Due to resource constraints, we were unable to train an LLM from scratch. Instead, we adopted a "prune and finetune" strategy using the alpaca dataset (Taori et al., 2023). This is similar to our pipeline of "DMS_{p}", but only finetuning rather than retraining. To mitigate overfitting to the alpaca dataset, we used the original model to distill the pruned model both during pruning and the subsequent finetuning process. In alignment with LLMPruner (Ma et al., 2023), we limited our pruning to the heads of self-attentions and the hidden dimensions of the feed-forward networks (FFN) within Llama. The resource constraint is set to 5.47B parameters as LLMPruner. After pruning 20% of the parameters from Llama-7B and comparing it with LLMPruner, our method demonstrated superior performance across various benchmarks. Specifically, we observed reduced perplexity on WikiText2 (Merity et al., 2016) and Pth (Marcus et al., 1993), and higher zero-shot classification accuracy on BoolQ (Clark et al., 2019), WinoGrande (Sakaguchi et al., 2021), ARC-e (Clark et al., 2018), and ARC-c (Clark et al., 2018).

Table 10. Experiment on Llama-7B. We pruned Llama-7B using DMS and compared it with LLMPruner. We evaluate the pruned model using perplexity on Wikitext2 and Pth datasets and zero-shot classification accuracy on BoolQ, WinoGrande, ARC-e, and ARC-c datasets. In our results, the symbol "↑" denotes that a larger value is better, while "↓" signifies that a smaller value is preferable. We use our pipeline of "DMS_p", which loads pretrained weights, in this table, due to resource constraints.

Model	Params	Wikitext2 ↓	Pth ↓	BoolQ ↑	WinoGrande ↑	ARC-e ↑	ARC-c ↑
Llama-7B (Touvron et al., 2023)	6.74B	12.62	22.14	76.5	67.01	72.8	41.38
LLM-Pruner-Llama-7B (Ma et al., 2023)	5.47B	17.39	30.2	66.79	64.96	64.06	37.88
DMS_p-Llama-7B (ours)	5.47B	**17.13**	**27.98**	**75.23**	**65.35**	**71.46**	**39.59**

Table 11. Comparison with other Gradient-based Methods: We use "$\text{DMS}_{\text{p-}}$" pipeline for all methods in this table.

Method	Top-1 (%)	MACs (G)
$\text{DMS}_{\text{p-}}$ w/o normalization	0.1	3
$\text{DMS}_{\text{p-}}$(ours)	**70.7**	3

A.3. More Ablaion Study

A.3.1. Ablation Study on Importance Normalization

We compare our method with that of disabling importance normalization (replace Eq 2 by $c_i' = c_i$), as shown in Table 11. We use the pipeline of "$\text{DMS}_{\text{p-}}$" for all methods in this table, and ResNet-50 as our supernet. Without importance normalization, our method drops to an extremely low performance. It's caused by the unevenly distributed importance, which makes our topk function hard to optimize. This result demonstrates the significance of our importance normalization.

A.3.2. Ablation Study on Pretraining and Supernet Sizes

It's widely accepted that both pretraining and increasing supernet size can improve the performance of searched models (Liu et al., 2018b; Frankle & Carbin, 2018). However, both of them also introduce more resource consumption. Here, we compare the influence of conducting pretraining and increasing supernet size on performance and resource consumption. Note, when we load pretrained weights, it means we use "DMS_p" pipeline, and when we do not load pretrained weights, it means we use "DMS_np" pipeline. We search for ResNet models of 1G MACs models with different supernet sizes and compare their performance and resource consumption. The results are shown in Table 12.

For pretraining, searching on a pretrained ResNet-50 surpluses searching on a randomly initialized ResNet-50 by 0.7%. However, pretraining also increases about 10 times the resource consumption. For supernet sizes, searching on a randomly initialized ResNet-152 surpluses searching on a randomly initialized ResNet-50 by 1.5%, and only increases about 3 times the resource consumption. We find increasing supernet size is more efficient than using strong pretrained weights for our method. Therefore, our default pipeline "DMS_np" just searches on randomly initialized supernets, making our method much more efficient than high-search-cost NAS methods. It also makes our method easier to be applied in the real world. This benefit originates from the high search efficiency of our method, making us use much fewer epochs to search a model than that of pretraining a supernet.

A.3.3. Search with Latency Constraint

Except for the MAC constraint, we also conduct experiments with *latency constraint*. We utilize the supernet of OFA (Cai et al., 2019) and search for models under the latency constraint of 3ms on RTX3090. The results are shown in Table 13. Our method costs much less search cost and achieves the same performance as the evolutionary algorithm, again. It demonstrates that our method is applicable under different constraints. The implementation detail of our latency constraint is detailed in Appendix A.1.2.

A.3.4. Ablation Study of Search Time

We assessed the relationship between search time and final performance, as detailed in Table 14. Our method achieves the best performance with 10 epochs. It's about 1/10 of the epochs of retraining the searched model. This proves the high search

Table 12. Ablation Study on Pretraining and Supernet Size. We search for 1G models in these experiments. $Cost_{pretrain}$ is the cost of pretraining a supernet, $Cost_{search}$ is the cost of searching a model. The unit of cost is $GMACs \times epochs$.

Supernet	Pretrain (Pipeline)	$Cost$	Top-1 (%)
ResNet-50	Y (DMS_p)	$410 + 41$	73.8
ResNet-50	N (DMS_{np})	$0 + 41$	73.1
ResNet-101	N (DMS_{np})	$0 + 79$	74.2
ResNet-152	N (DMS_{np})	$0 + 116$	**74.6**

Table 13. Experiment with Latency Constraint: We compare our DMS with an evolutionary algorithm (EA) under the latency constraint of 3ms on RTX3090. Only EA + predictor needs a public cost, which is used to train an accuracy predictor. The unit for search cost is GPU hours, respectively. We use the pipeline of "DMS_{p-}" in this table.

Method	Top-1 (%)	Latency (ms)	Search Cost
EA + predictor (Cai et al., 2019)	78.3	3	40 + 0
DMS_{p-}(ours)	78.3	3	**0 + 0.05**

efficiency of our method, where a few epochs are enough to search for a model. Besides, even though when the search time is only 3 epochs, the performance of the searched model is still better than the human-designed ResNet-18 with fewer MACs.

A.3.5. Ablation Study of Hyperparameters for our Method

We divide the hyperparameters of our method into two categories: fixed hyperparameters and unfixed hyperparameters. Fixed hyperparameters are hyperparameters that are fixed for all models, while unfixed hyperparameters are hyperparameters that needs to be turned for different models.

The fixed hyperparameters include the decay rate for Taylor importance and the temperature λ for our differential topk operator.

Taylor importance (Molchanov et al., 2019) is a well-known method to measure the importance of elements, and the decay of moving average is also widely used in the literature. Therefore, we directly use the decay rate of 0.99 regarding prior works.

Temperature λ of our diffenretial topk. The temperature is used to polarize (Zhuang et al., 2020) the mask of elements. Directly selecting a value that can polarize the mask of elements is enough. Thanks to our importance normalization, the temperature can be directly computed by closed-form, detailed in Section 3.1.2. The temperature λ is set to N for width elements and $4N$ for depth elements and the number of heads in attention mechanisms. N is the number of elements in the corresponding dimension. They work well for all models.

Therefore, we do not conduct an ablation study on these fixed hyperparameters.

The unfixed hyperparameters include the weight of resource constraint loss $\lambda_{resource}$ and the learning rate for structure parameters $lr_{structure}$. They are used to control the update of the structure parameters. The update value of a structure parameter is computed by $lr_{strucutre} \times (g_{task} + \lambda_{resource} \times g_{resource})$, where g_{task} and $g_{resource}$ is the gradient of structure parameters with respect to the task loss and resource constraint loss, Table 15 shows the ablation study results.

Obviously, 1) Smaller $\lambda_{resource}$ is better, as far as the model can reach the target resource constraint. Smaller $\lambda_{resource}$ means that the task loss takes more control of the update of the structure parameter. 2) When $\lambda_{resource}$ is small, the model is not sensitive to the change of $lr_{structure}$. When $\lambda_{resource}$ is large, a relatively large $lr_{structure}$ is better. This is because reaching the target resource constraint quickly can reduce the influence of the resource constraint loss, as resource constraint loss is zero when the model reaches the target resource constraint.

Therefore, the setting of $\lambda_{resource}$ and $lr_{structure}$ is not difficult. We first fix $lr_{structure}$ and turn $\lambda_{resource}$ to a small value and ensure the model can reach the target resource constraint. Then, we turn $lr_{structure}$ to a relatively large value, which makes the model reach the target resource constraint in the first hundreds of iterations. Only observing the resource decrease

Table 14. Ablation Study on Search Time. We use the pipeline of "DMS_{np}", which does not load pretrained weights, in this table.

Search Time	MACs (G)	Top-1 (%)
ResNet-18	1.8	69.9
3 epochs	1	71.6
5 epochs	1	72.9
10 epochs	1	**73.1**
20 epochs	1	72.8

Table 15. Ablation Study on Unfixed Hyperparameters. "/" denotes the model is not able to reach our resource target. We use the pipeline of "DMS_{np}", which does not load pretrained weights, in this table.

$\lambda_{resource}$ \ $lr_{structure}$	5e-2	5e-3	5e-4	5e-5
0.1	/	/	/	/
1	73.0	**73.1**	72.9	/
10	72.5	72.2	72.6	70.9

in the first epoch is enough to set these two hyperparameters.

Compared with other NAS methods, our method uses fewer hyperparameters. For example, ModelAmplification (Liu et al., 2022) must turn at least five hyperparameters for different tasks and models.

A.3.6. Ablation Study on Search Spaces

In prior experiments, except for our own search space, we also apply our method on the search space defined by OFA (Cai et al., 2019) in Section 4.1.2 and Appendix A.3.3. Experiment results demonstrate our method works fine on these search spaces. Here, we further conduct an ablation study about search spaces.

There are two differences between the search space of our method and that of prior NAS methods.

- On CNN models, we only search for width and depth, while prior methods usually also search for resolution and kernel size. We do not search for resolution and kernel size because we want to build a general search method that can be applied to various tasks and architectures. Searching for resolution and kernel size is not necessary for transformers and other tasks except for vision tasks.
- Due to our high search efficiency, our search space is more fine-grained, while prior methods usually implement a coarse-grained search space, containing much fewer sub-networks than ours. Besides, Coarse-grained search spaces also always need more expert knowledge to design, making it hard to build a general search method. Specifically, for a structure hyperparameter x, we directly search it in the range of $[1, x_{max}]$ with step 1, while most prior methods usually search it in the range of $[x_{min}, x_{max}]$ with a minimal step, such as 32 and 64.

Here, we conduct an ablation study based on Autoformer (Chen et al., 2021) search space. Autoformer implements a coarse-grained search space, such as searching embedding size with a minimal size of 192, a maximal size of 240, and a step of 24. We search two models with the exact same coarse-grained search space as Autoformer and a corresponding fine-grained search space. The results are shown in Table 16.

On the same supernet size, no matter the fine-grained search space or the coarse-grained search space, our method achieves better performance than Autoformer with less than 1/10 search costs. It proves our high search efficiency. Besides, we find coarse-grained search yields better performance than fine-grained search space for our methods. This is because human-designed coarse-grained search space has been a pretty good sub-space of the fine-grained search space by the search conducted by human experts. The coarse-grained search space was updated by many human experts in different research projects, the "search costs" of this "human searching stage" may take a lot of time and resources.

Table 16. Ablation Study on Search Space. We compare the performance with Autoformer with the exact same search space. We use the pipeline of "DMS_{np}", which does not load pretrained weights, in this table.

Method	Search Space	Supernet Size (G)	Top-1 (G)	Search Cost (GPU days)
Autoformer-T (Chen et al., 2021)	Carse-Grained	2.2	74.7	> 25
DMS_{np}(ours)	Fine-Grained	2.2	74.8	2
DMS_{np}(ours)	Fine-Grained	6.1	75.1	5.5
DMS_{np}(ours)	Carse-Grained	2.2	**75.2**	2

Besides, we find increasing the supernet size makes the fine-grained search space achieve comparable performance with the coarse-grained search space. It demonstrates bigger supernet size contains better subnets, which can be searched by our method.

In this paper, as we want to build a general search method with the least human labor, we still use the fine-grained search space as our default search space.

A.4. Implementation Details

A.4.1. Detail of Training Setting

In general, given a baseline model and a training setting, we enlarge the baseline model as our supernet and decrease the number of epochs of the training setting as our searching setting. We list details of our experiment setting as shown below.

EfficientNet: For all DMS_{np}-ES variants, we pruned the supernets over a span of 30 epochs. For those DMS_{np}-ES variants with MACs fewer than 0.5G, the pruning was conducted from EfficientNet-B4, using an input size of 224. Meanwhile, for DMS_{np}-EN-B1 and B2, the pruning was initiated from EfficientNet-B7. The input sizes for DMS_{np}-EN-B1 and B2 were 256 and 288, respectively. Subsequently, the DMS_{np}-EN variants were retrained using the corresponding training scripts of EfficientNet available in the Timm library (Wightman, 2019).

ResNet: We pruned the ResNet over ten epochs, starting from the ResNet-152 model. After pruning, the ResNet was retrained utilizing the MMPretrain (Contributors, 2023) training settings. This encompasses the foundational setting with a step learning scheduler and the rsb-a1 configuration.

MobileNetV2: To search for the ideal structure for MobileNet, we commenced by enlarging MobileNetV2 before pruning. Specifically, all channel numbers were expanded by 1.5 times, and the number of blocks in each stage was doubled. The pruning process for MobileNetV2 spans 30 epochs. Subsequent to this, the architecture was retrained employing the MMPretrain training settings.

Deit: We enhanced the depth of the Deit-small model, moving from 12 to 16, to serve as the supernet. The pruning for Deit was conducted with 30 epochs, including 20 epochs as a warmup phase. After pruning, we retrained the model using MMPretrain combined with the Swin training setting.

Yolo-v8 We used Yolo-v8 with deepen factor of 0.5 and widen factor of 0.5 as our supernet, while the original Yolo-v8-n has deepen factor of 0.33 and widen factor of 0.26. We used the training setting of Yolo-v8-n to train the supernet and pruned it over 30 epochs. The experiment of Yolo-v8 was conducted based on MMYolo (Contributors, 2022).

A.4.2. Detail of Search Cost Estimation

In this section, we delve into the specifics of how we estimate the search costs for other NAS methods as outlined in Table 4. The search cost of a searched model is divided into two parts: the public part and the private part. The public part is conducted for all sub-models, while the private part pertains to a specific sub-model.

EfficientNet: EfficientNet searches for common scaling strategies across all variants, thus incurring no private search cost. The public search cost estimate for EfficientNet is sourced directly from the ScaleNet paper (Xie et al., 2022).

ScaleNet (Xie et al., 2022): The ScaleNet paper explicitly presented their search cost, which includes a public cost of 379 GPU days and a private cost of 106 GPU days for several sub-models, totaling 21G MACs. We compute the private search

cost for a sub-model based on the ratio of its MACs to the overall 21G MACs.

ModelAmplification (Liu et al., 2022): As a multi-shot NAS method, ModelAmplification requires training multiple models. For all sub-models, it utilizes a public proxy dataset and a proxy training script. Approximately 2007 epochs are expended to examine the proxy dataset, and an additional 2963 epochs are used for the proxy training script, leading to a total of 4970 epochs. During the model search phase, for a variant with 390M MACs, ModelAmplification trains about 390 models per iteration. Assuming a ten-fold iteration search per model, this results in roughly 3000 epochs. By benchmarking the training time of EfficientNet-B0 on A100, we determine that 100 epochs require about 2.5 GPU days. As a result, the public search cost for ModelAmplification is at least 144 GPU days, while the private cost for the 390M MACs variant is 75 GPU days. We linearly scale the search costs of different variants based on their MACs.

JointPruning (Guo et al., 2021a): As a gradient-based pruning method, JointPruning presumably employs a supernet and training script analogous to ours. We deduce its search cost based on the number of pruning epochs. JointPruning paper indicates that a quarter of the total training epochs is earmarked for model searching. In contrast, we utilize at most a tenth of the total epochs for this purpose. Hence, the search cost for JointPruning is 2.5 times that of ours.

A.5. Visualization of Searched Model Structure

In Figure 4, a visualization is provided to delineate the structural intricacies of our searched DMS_{np}-EN-B0 in comparison to EfficientNet-B0. A distinct observation that stands out is the depth of our DMS_{np}-EN-B0. It possesses 8 more inverted residual blocks than its EfficientNet counterpart. Furthermore, when we delve deeper into the channel distribution across different stages, it becomes evident that our DMS_{np}-EN-B0 has undergone significant structural modifications, veering away from the traditional blueprint of EfficientNet-B0. Such distinct differences underscore the fine-grained adaptability of our method, emphasizing its capability to recalibrate and refine models in a way that they are acutely tailored to the task.

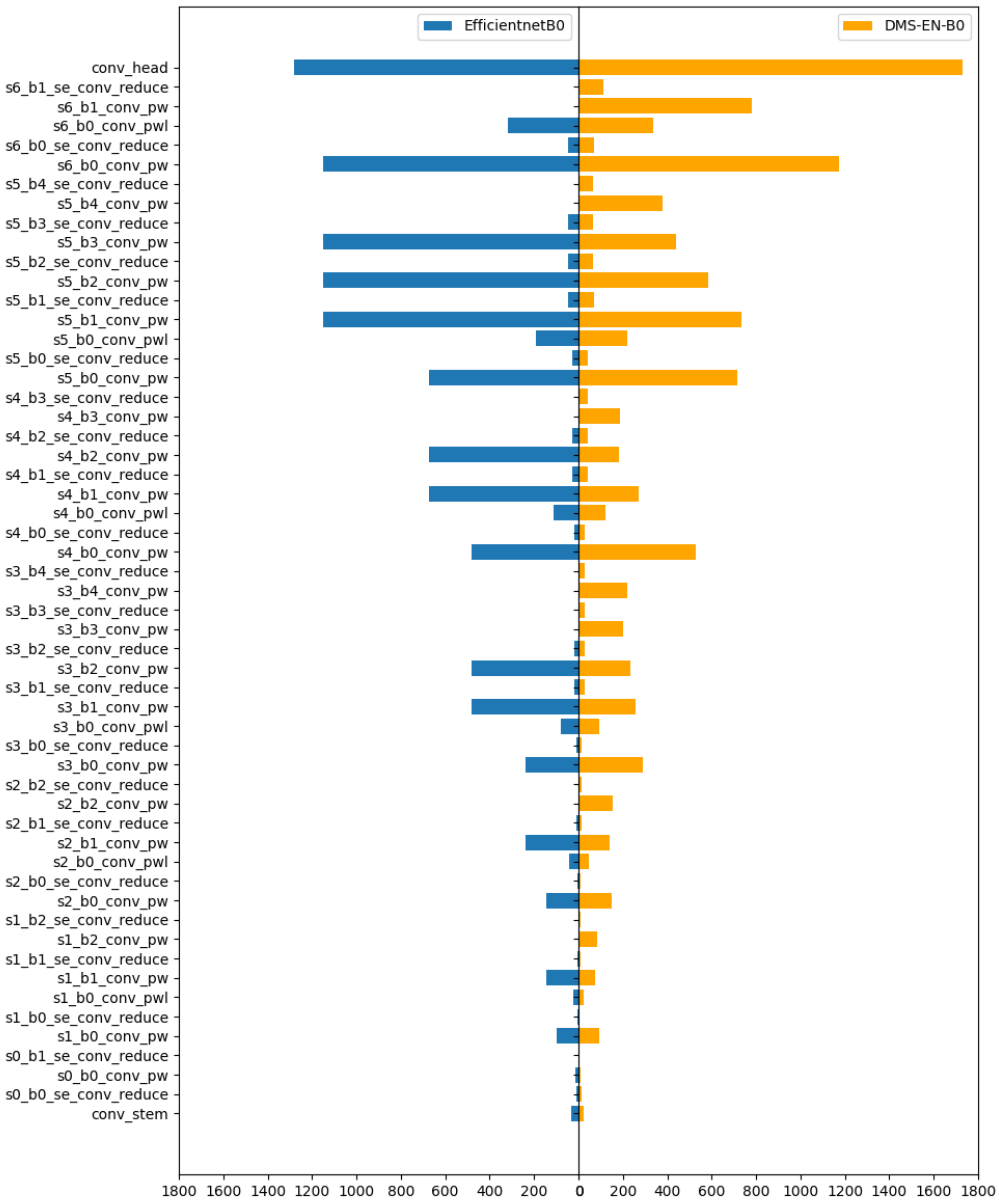


Figure 4. Visualization of Our Searched Structure. The x-axis represents the layers' width (channels/features), while the y-axis represents the layers. As DMS_{np}-EN-B0 has more layers than EfficientNet-B0, the width of extra layers for EfficientNet-B0 are seen as 0.