NETWORK PRUNING SPACES

Anonymous authors
Paper under double-blind review

ABSTRACT

Network pruning techniques, including weight pruning and filter pruning, reveal that most state-of-the-art neural networks can be accelerated without a significant performance drop. This work focuses on filter pruning which enables accelerated inference with any off-the-shelf deep learning library and hardware. We propose the concept of *network pruning spaces* that parametrize populations of subnetwork architectures. Based on this concept, we explore the structure aspect of subnetworks that result in minimal loss of accuracy in different pruning regimes and arrive at a series of observations by comparing subnetwork distributions. We conjecture through empirical studies that there exists an optimal FLOPs-to-parameter-bucket ratio related to the design of original network in a pruning regime. Statistically, the structure of a winning subnetwork guarantees an approximately optimal ratio in this regime. Upon our conjectures, we further refine the initial pruning space to reduce the cost of searching a good subnetwork architecture. Our experimental results on ImageNet show that the subnetwork we found is superior to those from the state-of-the-art pruning methods under comparable FLOPs.

1 INTRODUCTION

Large neural networks are usually preferred as they exhibit better generalization capability than small ones. In many model families such as ResNets (He et al., 2016) and MobileNets (Howard et al., 2017), large networks consistently achieve higher accuracy and are superior to small ones. Although large networks have revolutionized many fields such as computer vision and natural language processing, they suffer from significant inference costs in practice, especially when used with embedded sensors or mobile devices. For many practical applications, computational efficiency and small network sizes are crucial factors in addition to performance.

Recent works on network pruning reveal that most state-of-the-art models are over-parameterized (Li et al., 2017; Frankle & Carbin, 2019). We can remove weights (Frankle & Carbin, 2019), filters (Li et al., 2017; Luo et al., 2017) and other structures (Meng et al., 2020) from large networks without inducing significant performance drop by network pruning techniques. Such strategies reduce the resource demands of neural network inference, including storage requirements, energy consumption and latency, and thus increase runtime efficiency. This work focuses on filter pruning, which prunes the original network to a slimmer subnetwork. Unlike weight pruning approaches that lead to irregular sparsity patterns and require specialized libraries or hardware for computational speedups, filter pruning enables accelerated inference with any off-the-shelf deep learning library and hardware.

In the literature, a well-established paradigm is to train a large network to completions, prune the network according to some heuristics and retrain the pruned subnetwork to recover the accuracy loss (Renda et al., 2020). Despite the effectiveness of this *prune-then-retrain* paradigm, existing filter pruning methods have one major limitation: their outcome is merely a single pruning recipe tuned to a specific setting[1] and may fail to generalize to new settings. For example, we find that some filter pruning methods cannot produce subnetworks that outperform a regular subnetwork with an uniform pruning ratio in some extreme pruning regimes where we reduce more than 90% FLOPs of a network. Moreover, the prune-then-retrain paradigm does not deepen our understanding about the reason why these recipes produced by pruning approaches work well in specific regimes.

Instead of producing the best single pruning recipe like existing works (Li et al., 2017; He et al., 2018; Li et al., 2020), we explore the general principles that can help us understand and refine prun-

[1]In this work, we refer to the combination of pruning ratios for the layers in a network as *pruning recipe*.

ing algorithms. Inspired by Radosavovic et al. (2019; 2020), we present *network pruning spaces* in this work. Given a network, its network pruning space is a large population of subnetwork architectures produced by pruning recipes. This is different from architecture search spaces (Zoph & Le, 2017; Tan & Le, 2019) and network design spaces (Radosavovic et al., 2019; 2020), which aim to produce model families with good generalization from scratch. Specifically, we start with a well-designed network architecture and sample subnetworks from its pruning space, which gives rise to a subnetwork distribution. Based on this distribution, we present a tool to analyze the pruning space. With a constraint on pruning recipes, we divide the initial pruning space into several subspaces to explore the structure aspect of winning subnetworks and reduce the cost of searching such a subnetwork under different settings. The implementation is still filter pruning in common, but elevated to the population level and guided via distribution estimates following Radosavovic et al. (2019).

In our empirical studies, the core observations on network pruning spaces are simple but interesting: (*a*) There exists an optimal FLOPs-to-parameter-bucket ratio in a pruning regime; (*b*) Subnetworks with the optimal FLOPs-to-parameter-bucket ratios will be winning ones; (*c*) The limitation of performance in a pruning regime is predictable by a function of the optimal FLOPs-to-parameter-bucket ratio. With these observations, we return to examine some subnetworks produced by existing pruning methods (Luo et al., 2017; Li et al., 2017). We find that these subnetworks with good performance match our observations (*a* and *b*). Moreover, our Observation *c* is consistent with Rosenfeld et al. (2021) in weight pruning, suggesting that we are able to use an empirical function to make trade-offs between efficiency and accuracy when pruning filters in a network.

The contributions of this work are threefold:

- We present the new concept of network pruning spaces that parametrize populations of subnetwork architectures in the field of filter pruning. With this concept, we empirically study the structure aspect of winning subnetworks in different pruning regimes and explore general pruning principles.
- Based on our experimental results, we make a series of conjectures that help us understand filter pruning principles. With these conjectures, we refine the initial pruning space with a constraint on FLOPs-to-parameter-bucket ratio to reduce the cost of searching a winning subnetwork.
- We analytically show that the limitation of performance in a pruning regime is predictable, suggesting that we can build an empirical tool for reasoning trade-offs between efficiency and accuracy in filter pruning.

2 REVISIT FILTER PRUNING

In this section, we revisit existing techniques in filter pruning and introduce an efficient setting for further exploration. Generally, a pruning method involves instantiating each of the steps in the prune-then-retrain paradigm from a range of choices. The combination of such choices has a major impact on the final performance (Renda et al., 2020). The combinations of pruning implementations vary from each other in accuracy and efficiency. Since we aim to explore general pruning principles that can interpret the behaviors of filter pruning, a qualified and efficient setting is needed. In the following, we assess different pruning implementations and finally introduce a standard setting for conducting empirical studies. Our main experiments use ResNet-50 (He et al., 2016) on CIFAR-10 (Krizhevsky, 2009). The input images are resized to 224×224 such that we only modify the last fully-connected layer in ResNet-50.

2.1 PRUNING FILTERS

We briefly describe the procedure of pruning filters on the convolutional layers. Extending it to other layers is straightforward. Let c_i denote the number of input channels for the ith convolutional layer in a network and h_i/w_i be the height/width of the input feature maps. The convolutional layer transforms the input feature maps $\boldsymbol{x}_i \in \mathbb{R}^{c_i \times h_i \times w_i}$ into the output feature maps $\boldsymbol{x}_{i+1} \in \mathbb{R}^{c_{i+1} \times h_{i+1} \times w_{i+1}}$, which are used as input feature maps for the next convolutional layer, by applying c_{i+1} filters on the c_i input channels.

Each filter $\boldsymbol{F}_{i,j} \in \mathbb{R}^{c_i \times k \times k}$ is composed by c_i 2D kernels $\boldsymbol{K} \in \mathbb{R}^{k \times k}$ and generates one feature map $\boldsymbol{x}_{i+1,j} \in \mathbb{R}^{1 \times h_{i+1} \times w_{i+1}}$ on top of $\boldsymbol{x}_i \in \mathbb{R}^{c_i \times h_i \times w_i}$. All the filters constitute the convolutional

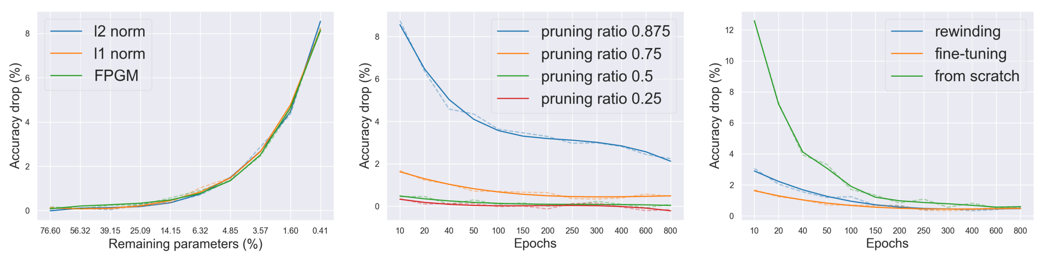


Figure 1: Pruning filters of ResNet-50 on CIFAR-10. (*Left*) Pruning filters of ResNet-50 based on different pruning heuristics, including ℓ_1 norm, ℓ_2 norm and FPGM (He et al., 2019). (*Center*) Experiments on fine-tuning epochs with different pruning ratios. (*Right*) Comparison of fine-tuning, rewinding and retraining from scratch with an uniform pruning ratio of 0.75.

kernel $\boldsymbol{F}_i \in \mathbb{R}^{c_i \times c_{i+1} \times k \times k}$. When m filters are pruned with a ratio $r_i = m/c_{i+1}$, m corresponding feature maps are removed. It reduces m/c_{i+1} of the computational cost for both layers i and $i+1$.

Filter importance. There exist a wide variety of pruning heuristics for evaluating the importance of filters. Figure 1 (left) shows accuracy drop curves of some pruning heuristics when we prune more filters. The difference between these pruning heuristics is small and they have the same trend when pruning filters. We adopt ℓ_2 norm pruning heuristic in this work because of its simplicity.

One-shot pruning. We prune the network to a target size at once. In contrast, the prune-retrain circle can be repeated until a target network size is reached, which is known as iterative pruning. Renda et al. (2020) and our experiments (not shown) suggest that iterative pruning is slightly superior to one-shot pruning in high-pruning-ratio regimes; however, it takes much more time on retraining (about $5\times$). Such an accuracy gain can be ignored considering that one-shot pruning saves a lot of time, especially when we need to study a large population of pruned subnetworks in this work.

2.2 Retraining subnetworks

When the filters of the large network are removed after pruning, accuracy significantly decreases. It is standard to retrain the pruned subnetwork with the remaining weights to recover the original accuracy. We discuss and evaluate conventional retraining techniques in the literature. Our experiments prune the network with different uniform pruning ratios and retrain pruned subnetworks using existing techniques.

Fine-tuning. The first common retraining technique is fine-tuning, which retrains pruned subnetworks for a specified number of epochs t using a small learning rate lr (Molchanov et al., 2019; Renda et al., 2020). In this work, we use a cosine annealing learning rate schedule (Loshchilov & Hutter, 2017) with an initial $lr = 0.01$ and empirically study the influence of t in the field of filter pruning. As shown in Figure 1 (center), in low-pruning-ratio regimes, it is sufficient to fine-tune pruned subnetworks for a few epochs (*e.g.*, 50 epochs under an uniform pruning ratio of 0.25). In high-pruning-ratio regimes, where more than 90% parameters are removed after pruning, subnetworks require more epochs for fine-tuning to recover the original accuracy.

Learning rate rewinding. Renda et al. (2020) introduce a retraining technique, namely learning rate rewinding, which retrains the pruned networks using the original schedule from last t epochs of training. During retraining, the learning rate in rewinding is always larger than that used in fine-tuning. In this work, we adopt a slightly different rewinding implementation, which starts from the initial learning rate lr in training and retrains pruned subnetworks for t epochs with warming up for 5 epochs. Our implementation is similar to learning rate restarting (Le & Hua, 2021) with a cosine annealing schedule in practice.[2] Figure 1 (right) shows the experimental results in a high-pruning-ratio regime. Comparison between rewinding and fine-tuning shows that their performance is almost the same after retraining 200 epochs. However, in a low-epoch retraining regime, fine-tuning is more efficient than rewinding for filter pruning.

[2]We consider learning rate rewinding and learning rate restarting as the same type of retraining technique, as both of them rely on a large learning rate.

From scratch. We also conduct experiments that retrain pruned subnetworks from scratch, where the parameters of a subnetwork are re-initialized after pruning. The original learning rate schedule is adopted during retraining. As shown in Figure 1 (right, green line), retraining from scratch requires more epochs to reduce accuracy drop compared to fine-tuning and rewinding. When retrained for enough epochs (*e.g.*, 600 or 800 epochs), retraining from scratch is able to produce comparable performance, which is consistent with Liu et al. (2019b).[3] Moreover, our experimental results suggest that it is unfair to compare retraining techniques under different epochs, since all of them benefit from more epochs.

Discussion. In this subsection, we empirically study existing retraining techniques, including *fine-tuning*, *rewinding* and *retraining from scratch*. Our experimental results show that these retraining techniques can achieve similar performance if we retrain pruned subnetworks with enough epochs. However, fine-tuning for a few epochs is a more efficient choice among them as our focus is not on recovering the original accuracy as much as possible in the following. Finally, we introduce a standard setting for further exploration. Given a pruning recipe, the standard setting prunes filters by their ℓ_2 norms in a one-shot manner, and fine-tunes pruned subnetworks with $lr = 0.01$ for t epochs.

3 Network pruning spaces

In this section, we introduce *network pruning spaces*, which are inspired by the concept of network design spaces (Radosavovic et al., 2019). Given a trained network f_θ, a network pruning space $\mathbb{S}$ comprises a large population of subnetworks $\{f_{\theta_{\text{sub}}}\}$ pruned from f_θ. The initial pruning space $\mathbb{S}$ is loosely defined by a constraint on FLOPs. We refer to the constraint on FLOPs as c_{flops}, which is a relative value calculated by $c_{\text{flops}} = \text{FLOPs}(f_{\theta_{\text{sub}}})/\text{FLOPs}(f_\theta)$ in this work. In practice, we use a range $c_{\text{flops}} \pm \delta$ ($\delta = 0.002$) for efficiency, as it is difficult to search recipes with a fixed constraint.

Based on the specified pruning heuristics (*e.g.*, our standard setting), a subnetwork $f_{\theta_{\text{sub}}}$ in $\mathbb{S}$ is generated by a pruning recipe, which consists of pruning ratios $\boldsymbol{r} = \{r_1, r_2, \dots, r_N\}$ for N layers. We sample a number of pruning recipes from $\mathbb{S}$, giving rise to a subnetwork distribution, and analyze the pruning space.

Tools for pruning spaces. We adopt accuracy drop empirical distribution functions (EDFs) (Radosavovic et al., 2019; 2020) to analyze the pruning space. Given a set of n subnetworks with accuracy drops $\{e_i\}$, the accuracy drop EDF is given by:

$$F(e) = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}[e_i < e], \tag{1}$$

where $\mathbf{1}$ is the indicator function. $F(e)$ represents the fraction of subnetworks with accuracy drop less than e after retraining.

3.1 The STD pruning spaces

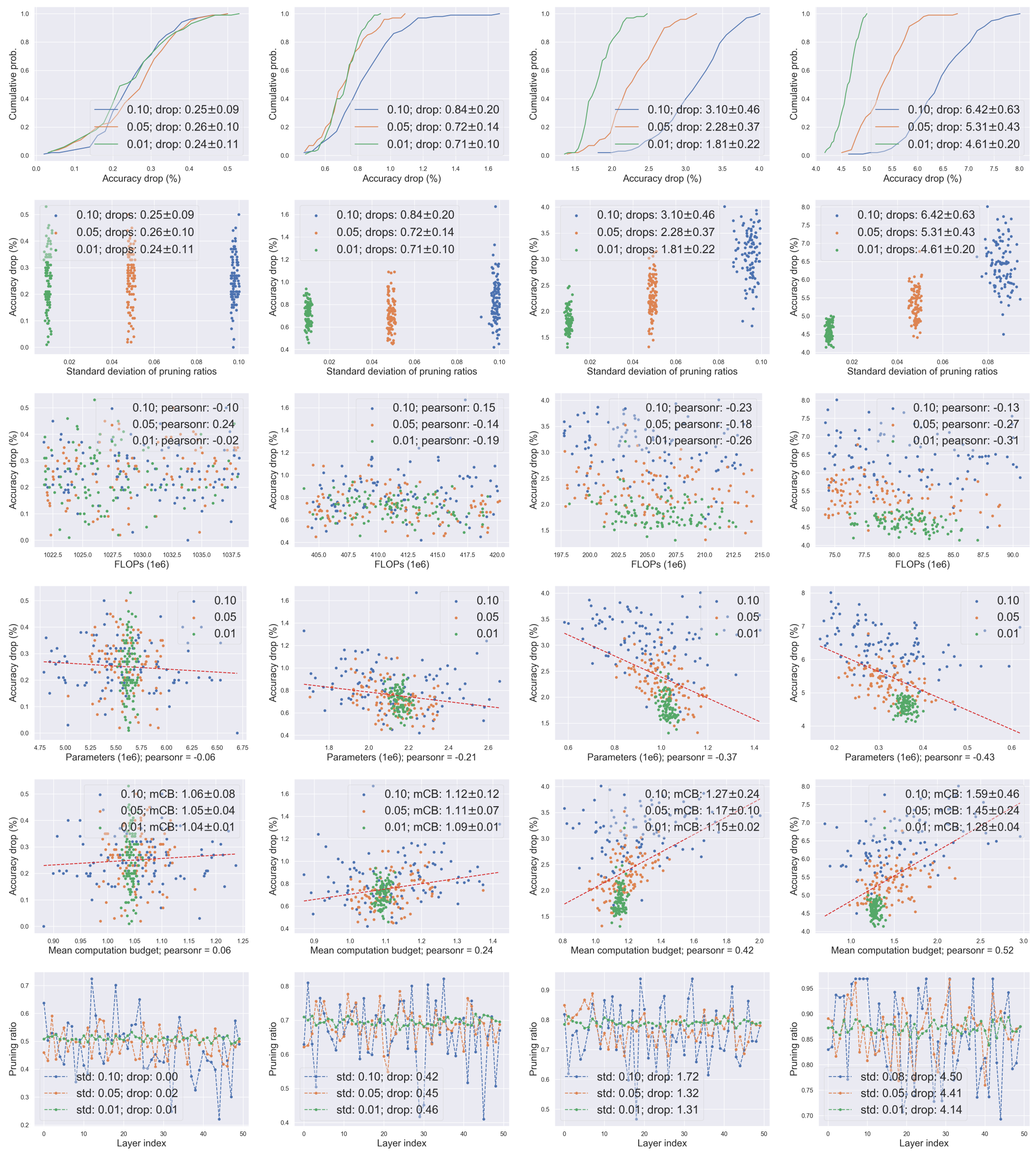
We start with the initial pruning space $\mathbb{S}$. Any recipe generation strategy, such as evolutionary algorithm and reinforcement learning (He et al., 2018), can be used when sampling subnetworks. Random sampling is the most straightforward one, which randomly generates N real numbers from a given range $[0, R]$ as a pruning recipe $\boldsymbol{r}$ (Li et al., 2020).[4]

Although a random recipe that meets the requirement c_{flops} can be a valid one, we observe an interesting problem when sampling subnetworks from $\mathbb{S}$ following Li et al. (2020). This random sampling process is not effective enough, taking many times for a valid recipe, as we have a compact constraint (*i.e.*, small δ). It is much easier to collect enough recipes when we start with an uniform pruning ratio and modify $\{r_i\}$ with N small random numbers. Therefore, we take a refinement step and come to `STD` pruning spaces.

We additionally use a constraint c_{std} on the standard deviation of the recipe $\boldsymbol{r}$ and refer to the pruning spaces with this constraint as `STD` pruning spaces. By controlling the standard deviation of $\boldsymbol{r}$, we

[3]In our experiments, the others outperform retraining from scratch with a less accuracy drop of 0.3.

[4]R is the largest pruning ratio applied to a layer.



(a) $c_{\text{flops}} = 0.25$ (b) $c_{\text{flops}} = 0.1$ (c) $c_{\text{flops}} = 0.05$ (d) $c_{\text{flops}} = 0.02$

Figure 2: STD pruning spaces under $c_{\text{flops}} = \{0.25, 0.1, 0.05, 0.02\}$. In the figure, we study three STD pruning spaces, including STD-0.1, STD-0.05 and STD-0.01 spaces. (*Row 1*) Accuracy drop EDFs on STD spaces. (*Row 2*) Standard deviation of $\boldsymbol{r}$ distribution. (*Row 3*) Remaining FLOPs distribution. (*Row 4*) Remaining parameters distribution. (*Row 5*) Mean computation budget distribution. (*Row 6*) The winning pruning recipe on each STD pruning space. Note that the scales of accuracy drop under different c_{flops} varies.

divide the initial space $\mathbb{S}$ into STD subspaces. In this work, we mainly study three STD pruning spaces with $c_{\text{std}} = \{0.1, 0.05, 0.01\}$, denoted as STD-0.1, STD-0.05 and STD-0.01 spaces respectively.

We focus on exploring the structure aspect of winning subnetworks that have the best performance after retraining. With our baseline setting, we sample $n = 100$ subnetworks from STD pruning spaces and fine-tune these subnetworks for $t = 50$ epochs. Although better performance can be achieved when we retrain subnetworks for more epochs, we argue that fine-tuning for 50 epochs is more efficient for most settings.

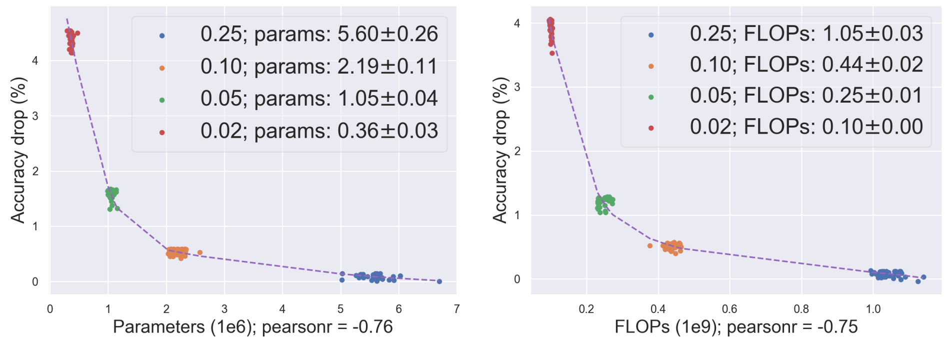


Figure 3: We present winning subnetworks on all STD spaces with c_{flops} and c_{params} respectively. (*Left*) Parameters distribution of winning subnetworks under $c_{\text{flops}} = \{0.25, 0.1, 0.05, 0.02\}$. (*Right*) FLOPs distribution of winning subnetworks under $c_{\text{params}} = \{0.25, 0.1, 0.05, 0.02\}$.

Medium-pruning-ratio regime. Figure 2a (Row 1) illustrates accuracy drop EDFs for three STD pruning spaces in a medium-pruning-ratio regime. As a baseline, the pruned subnetwork with an uniform pruning ratio of 0.5 reduces 74.1% FLOPs (corresponds to $c_{\text{flops}} = 0.259$) from the original network and has an accuracy drop of 0.3% after fine-tuning for 50 epochs. On each STD space, we can easily find many subnetworks (about a fraction of 60%) that outperform the baseline subnetwork.

The winning subnetworks in this regime have significantly different recipes (Row 6) and all of them lead to pruned subnetworks without any accuracy drop.[5] From accuracy drop EDFs and standard deviation of $\boldsymbol{r}$ (Row 2), the cost to find such a good subnetwork on each STD space is similar, suggesting that there is no difference between STD spaces in medium-pruning-ratio regimes.

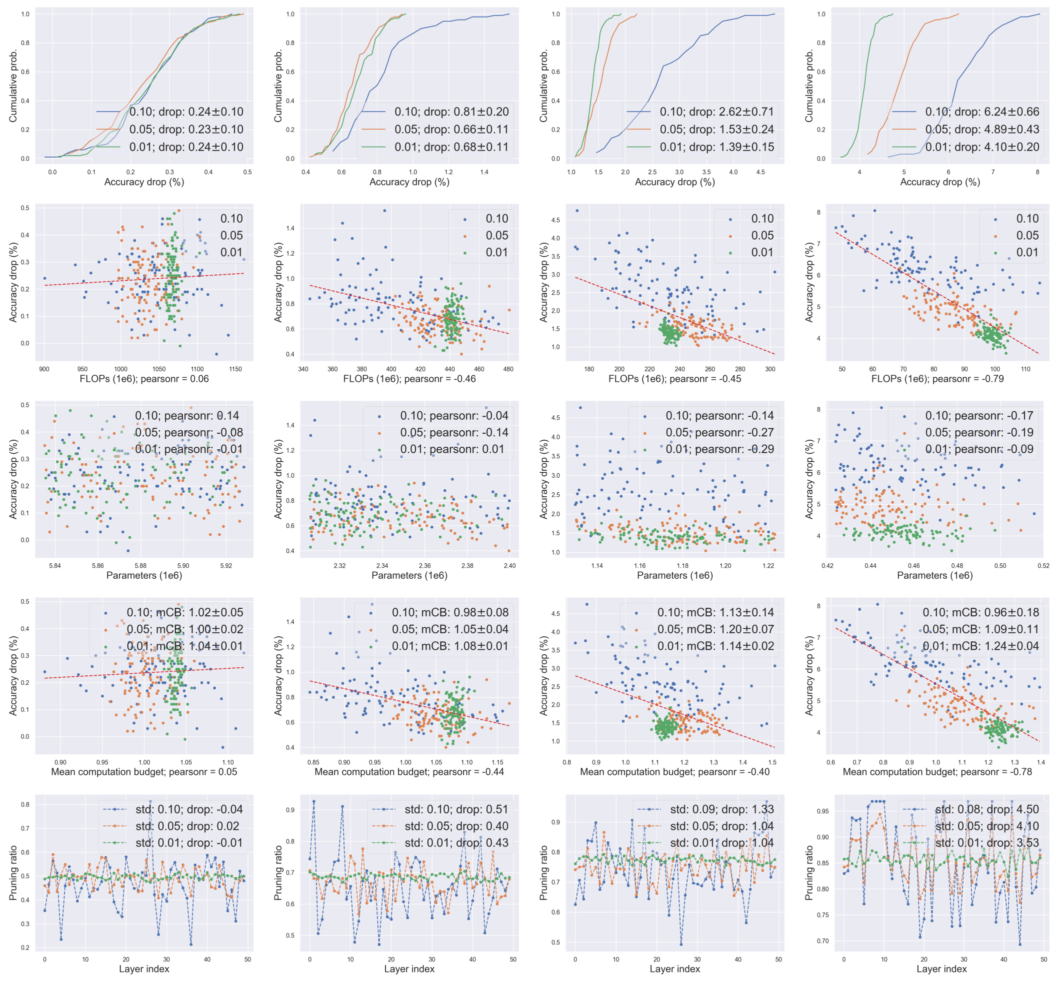
High-pruning-ratios regime. Next, we present STD spaces in high-pruning-ratio regimes, where more than 90% FLOPs are reduced after pruning. In Figure 2b, 2c and 2d (Row 1) we find that the accuracy drop EDFs for STD spaces progressively differ from each other. From $c_{\text{flops}} = 0.1$ to $c_{\text{flops}} = 0.02$, STD-0.01 spaces become higher than STD-0.05 and STD-0.1 spaces consistently. In other words, STD-0.01 space has a higher fraction of better subnetworks at every accuracy drop threshold when we prune more filters. This clear qualitative difference suggests that the cost to find a good subnetwork on STD-0.01 spaces is less than the costs on others. Moreover, under an extreme constraint such as $c_{\text{flops}} = 0.02$, we can easily find many subnetworks on STD-0.01 space that outperform the winning one on STD-0.1 space. There might exist a winning subnetwork on STD-0.1 space that is comparable to the best subnetworks on others, as we only sample $n = 100$ recipes in the experiments. However, it does not alter our conclusion that STD-0.01 space is a more efficient network pruning space across pruning regimes, especially in high-pruning-ratio regimes.

According to our experimental results, in these pruning regimes, there seems to be no shared pattern from the perspective of winning subnetwork structures. However, we observed an interesting trend that good pruning recipes always have a small standard deviation, although those with a large standard deviation can also produce good subnetworks with a small probability. Upon this discovery, one might ask: *What makes these recipes with a small standard deviation outperform others?*

Distribution comparison. In Figure 2 we present FLOPs distribution (Row 3) and parameter bucket distribution (Row 4). A pattern emerges: the amount of remaining parameters of subnetworks on STD-0.01 spaces is much closer to a certain number than those on other STD spaces in each pruning regime. Since we produce pruning recipes $\boldsymbol{r}$ with a constraint on FLOPs, we have the following conjecture:

Conjecture 1. *(**a**) Given a target FLOPs reduction, there exists an optimal parameter bucket for pruning filters; (**b**) Subnetworks that have the optimal parameter bucket will be the winning ones on $\mathbb{S}$; (**c**) A good structure of pruned subnetwork guarantees an approximately optimal parameter bucket under its FLOPs.*

[5]A small loss of 0.02% is ingored here, as better performance can be achieved when retrained for more epochs.



(a) $c_{\text{params}} = 0.25$ (b) $c_{\text{params}} = 0.1$ (c) $c_{\text{params}} = 0.05$ (d) $c_{\text{params}} = 0.02$

Figure 4: `STD` pruning spaces under $c_{\text{params}} = \{0.25, 0.1, 0.05, 0.02\}$. In the figure, we study three `STD` pruning spaces, including `STD-0.1`, `STD-0.05` and `STD-0.01` spaces with a constraint on parameter bucket. (*Row 1*) Accuracy drop EDFs on `STD` spaces. (*Row 2*) Remaining FLOPs distribution. (*Row 3*) Remaining parameters distribution. (*Row 4*) Mean computation budget distribution. (*Row 5*) The winning pruning recipe on each `STD` pruning space.

With Conjecture 1, we present the best 40 subnetworks on all `STD` spaces under each c_{flops} in Figure 3 (left). Note that the 40 winning subnetworks have different standard deviations of $\boldsymbol{r}$, which empirically proves our Conjecture 1c.

Constraint on parameter bucket. Consider a simple question: *What kind of subnetworks will be winning ones under a constraint on parameter bucket?* Next, we present `STD` pruning spaces with a constraint on parameter bucket c_{params} in Figure 4. We observe a consistent trend that `STD-0.01` space is higher than others in high-pruning-ratio regimes (*e.g.*, $c_{\text{params}} = \{0.05, 0.02\}$). Similarly, in Figure 3 (right), we present the best 40 subnetworks on all `STD` spaces under each c_{params} and have the following conjecture:

Conjecture 2. *(**a**) Given a target parameter bucket reduction, there exists an optimal FLOPs for pruning filters; (**b**) Subnetworks that have the optimal FLOPs will be winning ones on $\mathbb{S}$; (**c**) A good structure of pruned subnetwork guarantees an approximately optimal FLOPs under its parameter bucket.*

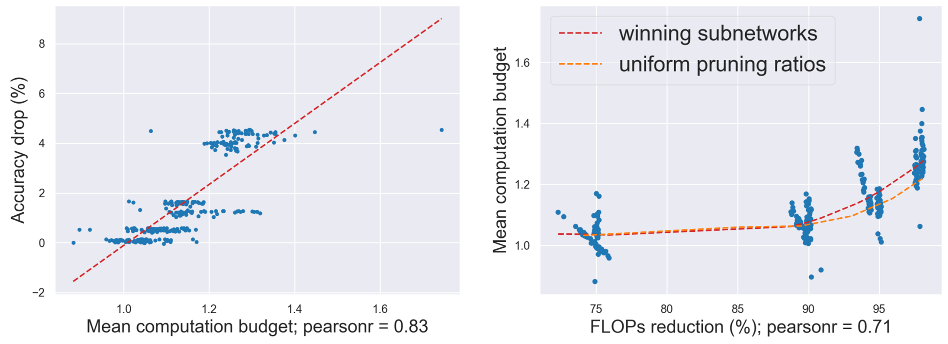


Figure 5: (*Left*) Sort winning subnetworks under different constraints (c_{flops} and c_{params}) by their accuracy drops. Each one is the best on all `STD` spaces under a constraint. (*Right*) We present the best 40 subnetworks on all `STD` spaces under each constraint.

Based on Conjectures 1 and 2, we believe that the winning subnetworks in different pruning regimes have optimal configurations of *(FLOPs, parameter bucket)*, which are achieved by their pruning recipes. To further reduce the degrees of freedom, we introduce a tool that represents the FLOPs-to-parameter-bucket ratio compared to the original network, namely *mean computation budget* (mCB):

$$\text{mCB} = \frac{\text{FLOPs}(f_{\theta_{\text{sub}}})/\text{Params}(f_{\theta_{\text{sub}}})}{\text{FLOPs}(f_{\theta})/\text{Params}(f_{\theta})}. \tag{2}$$

In Figure 2 (Row 5) and Figure 4 (Row 4) we present mCB distribution of subnetworks with c_{flops} and c_{params} respectively. With this tool, we combine Conjectures 1 and 2 into one and have the following conjecture:

Conjecture 3. *(**a**) In a pruning regime, there exists an optimal mean computation budget for pruning filters; (**b**) Subnetworks that have the optimal mean computation budget will be the winning ones on $\mathbb{S}$; (**c**) A good structure of pruned subnetwork guarantees an approximately optimal mean computation budget in this pruning regime.*

Taking the design of the original network as the optimal configuration of *(FLOPs, parameter bucket)*, we argue that the optimal mCBs in many pruning regimes are roughly equivalent to 1.0. In Figure 5 (right), we present mCB distribution of winning subnetworks when FLOPs reduction increases. Statistically, our experimental results show that the optimal mCB for a pruning regime should be a dynamic range and the optimal mCB range is around 1.0 in most pruning regimes.

Moreover, the optimal mCB tends to increase when we reduce more FLOPs by pruning. We draw a fitted curve of optimal mCB in Figure 5 (right, red dashed line) and the curve rises significantly when the reduction is above 90%. We also draw a curve (orange dashed line) presenting the mCB of subnetworks with uniform pruning ratios. These two curves are close in most regimes, suggesting that an uniform recipe is a good starting point when we search winning pruning recipes. That might be the reason why uniform pruning ratios outperform some pruning methods in our extremely pruning experiments.

In Figure 3, we observed that the optimal parameter bucket, the optimal FLOPs and the final performance of winning subnetworks seemed to be predictable. This observation is consistent with Rosenfeld et al. (2021), in which a scaling law is developed to estimate the error after weight pruning. Therefore, we have another conjecture:

Conjecture 4. *The limitation of performance in each pruning regime can be predicted by a functional form.*[6]

In Figure 5 (left), we present the relationship between the mCBs of winning subnetworks and their accuracy drops. We observed that the limitation of performance could be predicted by a function of the optimal mCB in a pruning regime. Since the goal of network pruning is to increase efficiency

[6]The limitations of performance and the curves in Figure 3 can be further refined, as we only retrain subnetworks for 50 epochs.

Table 1: Pruning ResNet-50 on ImageNet. † denotes our re-implementation.

FLOPs zone	method	FLOPs (G)	Top-1 (%)	Top-5 (%)
-	ResNet-50 (He et al., 2016)†	4.12	76.88	93.44
2G	Thinet-50 (Luo et al., 2017)	2.10	74.70	90.02
	EagleEye (Li et al., 2020)	2.00	76.40	92.89
	MetaPruning (Liu et al., 2019a)	2.00	75.40	-
	AutoSlim (Yu & Huang, 2019)	1.00	75.60	-
	HRank (Lin et al., 2020)	2.30	74.98	92.33
	FPGM (He et al., 2019)	1.92	74.83	92.32
	Ours ($c_{\text{flops}} = 0.5$)	2.06	76.90	93.55
1G	ThiNet-30 (Luo et al., 2017)	1.20	72.10	88.30
	EagleEye (Li et al., 2020)	1.00	74.20	91.77
	MetaPruning (Liu et al., 2019a)	1.00	73.40	-
	AutoSlim (Yu & Huang, 2019)	1.00	74.00	-
	HRank (Lin et al., 2020)	1.55	71.98	91.01
	Ours ($c_{\text{flops}} = 0.25$)	1.03	74.96	92.51

while maintaining accuracy, such an empirical function helps a lot when one is making a trade-off. Moreover, we believe that one might find a winning subnetwork without any accuracy drop in many pruning regimes, where the optimal mCB ranges are around 1.0.[7]

4 RESNET-50 ON IMAGENET

In this section, we prune ResNet-50 on ImageNet (Russakovsky et al., 2015) with $c_{\text{flops}} = 0.5$ and $c_{\text{flops}} = 0.25$. Since ImageNet is a large-scale dataset that takes more time for one epoch than CIFAR-10, we first generate pruning recipes and then fine-tune subnetwork candidates for 5 epochs. The top-5 ranking candidates will be retrained with a complete schedule. Specifically, we fine-tune subnetworks with an initial learning rate $lr = 0.01$ for 100 epochs on ImageNet.

In the experiments, we produce pruning recipes with a refined pruning space upon the proposed Conjecture 3. We empirically use a constraint on mCB $c_{\text{mCB}} = 1.0 \pm 0.1$ instead of a constraint on the standard deviation of recipes $\boldsymbol{r}$ for efficiency. This is because all `STD` spaces have a similar distribution in such a pruning regime, according to our analysis for CIFAR-10. Note that the excepted parameter bucket range is decided by c_{mCB} when a target FLOPs reduction is set. We generate $n = 300$ subnetwork candidates and finally keep the top-5 ranking candidates. In Table 1, we compare the winning subnetwork to the state-of-the-art pruning methods. The comparison shows that our analysis for network pruning spaces guides us to better pruning results.

5 CONCLUSION AND DISCUSSION

In this work, we introduce network pruning spaces for exploring general pruning principles. Inspired by (Radosavovic et al., 2019; 2020), we focus on the structure aspect of subnetwork architectures by comparing populations of subnetworks, instead of producing the best single pruning recipe. Although our empirical studies do not lead to a common pattern from the perspective of architecture, we observe that there exists an optimal mean computation budget in a pruning regime. Moreover, our observations suggest that the limitations of performance in different pruning regimes might be predictable by a function of the optimal mean computation budget. Our work empirically provides insight into the existence of such a functional form that approximates the accuracy drops and characterizes filter pruning.

[7]We argue that the limitation is related to the dataset size. Our experiments on ImageNet fail to find such a subnetwork without any accuracy drop when we reduce more than 75% FLOPs.

REFERENCES

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *ICLR*, 2019.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pp. 770–778, 2016.

Yang He, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi Yang. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *CVPR*, pp. 4340–4349, 2019.

Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *ECCV*, pp. 784–800, 2018.

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Alex Krizhevsky. Learning multiple layers of features from tiny images. *Tech Report*, 2009. URL `https://www.cs.toronto.edu/~kriz/cifar.html`.

Duong Hoang Le and Binh-Son Hua. Network pruning that matters: A case study on retraining variants. In *ICLR*, 2021.

Bailin Li, Bowen Wu, Jiang Su, and Guangrun Wang. Eagleeye: Fast sub-net evaluation for efficient neural network pruning. In *ECCV*, pp. 639–654. Springer, 2020.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *ICLR*, 2017.

Mingbao Lin, Rongrong Ji, Yan Wang, Yichen Zhang, Baochang Zhang, Yonghong Tian, and Ling Shao. Hrank: Filter pruning using high-rank feature map. In *CVPR*, pp. 1529–1538, 2020.

Zechun Liu, Haoyuan Mu, Xiangyu Zhang, Zichao Guo, Xin Yang, Kwang-Ting Cheng, and Jian Sun. Metapruning: Meta learning for automatic neural network channel pruning. In *ICCV*, pp. 3296–3305, 2019a.

Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *ICLR*, 2019b.

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. In *ICLR*, 2017.

Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *ICCV*, pp. 5058–5066, 2017.

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

Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *CVPR*, pp. 11264–11272, 2019.

Ilija Radosavovic, Justin Johnson, Saining Xie, Wan-Yen Lo, and Piotr Dollár. On network design spaces for visual recognition. In *ICCV*, pp. 1882–1890, 2019.

Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. In *CVPR*, pp. 10428–10436, 2020.

Alex Renda, Jonathan Frankle, and Michael Carbin. Comparing rewinding and fine-tuning in neural network pruning. In *ICLR*, 2020.

Jonathan S Rosenfeld, Jonathan Frankle, Michael Carbin, and Nir Shavit. On the predictability of pruning across scales. In *ICML*, pp. 9075–9083. PMLR, 2021.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *IJCV*, 115(3):211–252, 2015.

Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *ICML*, pp. 6105–6114. PMLR, 2019.

Jiahui Yu and Thomas Huang. Autoslim: Towards one-shot architecture search for channel numbers. *arXiv preprint arXiv:1903.11728*, 2019.

Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017.