RETHINKING THE VALUE OF NETWORK PRUNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Network pruning is widely used for reducing the heavy inference cost of deep models in low-resource settings. A typical pruning algorithm is a three-stage pipeline, i.e., training (a large model), pruning and fine-tuning. During pruning, according to a certain criterion, redundant weights are pruned and important weights are kept to best preserve the accuracy. In this work, we make several surprising observations which contradict common beliefs. For all the six state-of-the-art pruning algorithms we examined, fine-tuning a pruned model only gives comparable or even worse performance than training that model with randomly initialized weights. For pruning algorithms which assume a predefined target network architecture, one can get rid of the full pipeline and directly train the target network from scratch. Our observations are consistent for a wide variety of pruning algorithms with multiple network architectures, datasets, and tasks. Our results have several implications: 1) training a large, over-parameterized model is not necessary to obtain an efficient final model, 2) learned "important" weights of the large model are not necessarily useful for the small pruned model, 3) the pruned architecture itself, rather than a set of inherited "important" weights, is what leads to the efficiency benefit in the final model, which suggests that some pruning algorithms could be seen as performing network architecture search; we further show that it is possible to use the sparsity patterns from the pruned network to design efficient models.

1 INTRODUCTION

Over-parameterization is a widely-recognized property of deep neural networks (Denton et al., 2014; Ba & Caruana, 2014), which leads to high computational cost and high memory footprint for inference. As a remedy, *network pruning* (LeCun et al., 1990; Hassibi & Stork, 1993; Han et al., 2015; Molchanov et al., 2016; Li et al., 2017) has been identified as an effective technique to improve the efficiency of deep networks for applications with limited computational budget. A typical procedure of network pruning consists of three stages: 1) train a large, over-parameterized model (sometimes there are pretrained models available), 2) prune the trained large model according to a certain criterion, and 3) fine-tune the pruned model to regain the lost performance.



Figure 1: A typical three-stage network pruning pipeline.

Generally, there are two common beliefs behind this pruning procedure. First, it is believed that starting with training a large, over-parameterized network is important (Luo et al., 2017; Carreira-Perpinán & Idelbayev, 2018), as it provides a high-performance model (due to stronger representation & optimization power) from which one can safely remove a set of redundant parameters without significantly hurting the accuracy. Therefore, this is usually believed, and reported to be superior to directly training a smaller network from scratch (Li et al., 2017; Luo et al., 2017; He et al., 2017b; Yu et al., 2018) – a commonly used baseline approach. Second, both the pruned architecture *and* its associated weights are believed to be essential for obtaining the final efficient model (Han et al., 2015). Thus most existing pruning techniques choose to *fine-tune* a pruned model instead of training it from scratch. The preserved weights after pruning are usually considered to be critical, as how to accurately select the set of important weights is a very active research topic in the literature (Molchanov et al., 2016; Li et al., 2017; Luo et al., 2017; He et al., 2017b; Liu et al., 2017; Suau et al., 2018).

In this work, we show that both of the beliefs mentioned above are not necessarily true. Based on an extensive empirical evaluation of state-of-the-art pruning algorithms on multiple datasets with multiple network architectures, we make two surprising observations. First, for pruning algorithms with predefined target network architectures (Figure 2), directly training the small target model from random initialization can achieve the same, if not better, performance, as the model obtained from the three-stage pipeline. In this case, starting with a large model is not necessary and one could instead directly train the target model from scratch. Second, for pruning algorithms without a predefined target network, training the pruned model from scratch can also achieve comparable or even better performance than fine-tuning. This observation shows that for these pruning algorithms, what matters is the obtained architecture, instead of the preserved weights, despite training the large model is required to find that target architecture. However, in both cases, if a pretrained large model is already available, pruning and fine-tuning from it can still greatly save the training time to obtain the efficient model. The contradiction between our results and those reported in the literature might be explained by less carefully chosen hyper-parameters, data augmentation schemes and unfair computation budget for evaluating this baseline approach.

Our results advocate a rethinking of existing network pruning algorithms. It seems that the over-parameterization during the first-stage training is not as beneficial as previously thought. Also, inheriting weights from a large model is not necessarily optimal, and might trap the pruned model into a bad local minimum, even if the weights are considered "important" by the pruning criterion. Instead, our results suggest that the value of automatic pruning algorithms may lie in identifying efficient structures and performing implicit architecture search, rather than selecting "important" weights. In Section 5, we verify this hypothesis through carefully designed experiments, and show the patterns in the pruned model could provide design guidelines for efficient architectures.

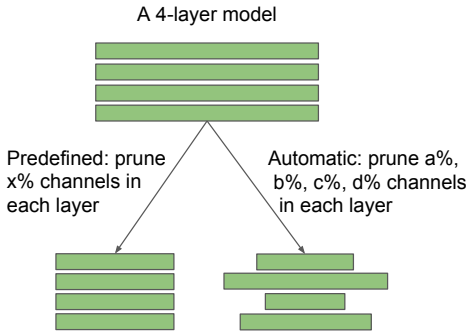


Figure 2: Difference between predefined and non-predefined (automatically discovered) target architectures. The sparsity x is user-specified, while a, b, c, d are determined by the pruning algorithm.

The rest of the paper is organized as follows: in Section 2, we introduce background and some related works on network pruning; in Section 3, we describe our methodology for training the pruned model from scratch; in Section 4 we experiment on various pruning methods and show our main results for both pruning methods with predefined or automatically discovered target architectures; in Section 5, we argue that the value of automatic pruning methods indeed lies in searching efficient network architectures, and show the pruned architectures can provide transferable efficient design guidelines, as supported by experiments; in Section 6 we discuss some implications and conclude the paper.

2 BACKGROUND

Recent success of deep convolutional networks (Girshick et al., 2014; Long et al., 2015; He et al., 2016; 2017a) has been coupled with increased requirement of computation resources. In particular, the model size, memory footprint, the number of computation operations (FLOPs) and power usage are major aspects inhibiting the use of deep neural networks in some resource-constrained settings. Those large models can be infeasible to store, and run in real time on embedded systems. To address this issue, many methods have been proposed such as low-rank approximation of weights (Denton et al., 2014; Lebedev et al., 2014), weight quantization (Courbariaux et al., 2016; Rastegari et al., 2016), knowledge distillation (Hinton et al., 2014; Romero et al., 2015) and network pruning (Han et al., 2015; Li et al., 2017), among which network pruning has gained notable attention due to their competitive performance and compatibility.

One major branch of network pruning methods is individual weight pruning, and it dates back to Optimal Brain Damage (LeCun et al., 1990) and Optimal Brain Surgeon (Hassibi & Stork, 1993), which prune weights based on Hessian of the loss function. More recently, Han et al. (2015) proposes to prune network weights with small magnitude, and this technique is further incorporated into the "Deep Compression" pipeline (Han et al., 2016b) to obtain highly compressed models. Srinivas

& Babu (2015) proposes a data-free algorithm to remove redundant neurons iteratively. However, one drawback of these *non-structured* pruning methods is that the resulting weight matrices are sparse, which cannot lead to compression and speedup without dedicated hardware/libraries (Han et al., 2016a).

In contrast, *structured* pruning methods prune at the level of channels or even layers. Since the original convolution structure is still preserved, no dedicated hardware/libraries are required to realize the benefits. Among structured pruning methods, channel pruning is the most popular, since it operates at the most fine-grained level while still fitting in conventional deep learning frameworks. Some heuristic methods include pruning channels based on their corresponding filter weight norm (Li et al., 2017) and average percentage of zeros in the output (Hu et al., 2016). Group sparsity is also widely used to smooth the pruning process after training (Wen et al., 2016; Alvarez & Salzmann, 2016; Lebedev & Lempitsky, 2016; Zhou et al., 2016). Liu et al. (2017) and Ye et al. (2018) impose sparsity constraints on channel-wise scaling factors during training, whose magnitudes are then used for channel pruning. Huang & Wang (2018) uses a similar technique to prune coarser structures such as residual blocks. He et al. (2017b) and Luo et al. (2017) minimizes next layer's feature reconstruction error to determine which channels to keep. Similarly, Yu et al. (2018) optimizes the reconstruction error of the final response layer and propagates a "importance score" for each channel. Molchanov et al. (2016) uses Taylor expansion to approximate each channel's influence over the final loss and prune accordingly. Suau et al. (2018) analyzes the intrinsic correlation within each layer and prune redundant channels. Chin et al. (2018) proposes a layer-wise compensate filter pruning algorithm to improve commonly-adopted heuristic pruning metrics. He et al. (2018a) proposes to allow pruned filters to recover during the training process. Lin et al. (2017); Wang et al. (2017) prune certain structures in the network based on the current input.

Our work is also related to some recent studies on the characteristics of pruning algorithms. Mittal et al. (2018) shows that random channel pruning (Anwar & Sung, 2016) can perform on par with a variety of more sophisticated pruning criteria, demonstrating the plasticity of network models. Anonymous (2019) hypothesizes that certain connections, together with their randomly initialized weights, is particularly effective for training, and a pruning algorithm can help find such sub-networks. Zhu & Gupta (2018) shows that training a small-dense model cannot achieve the same accuracy as a pruned large-sparse model with identical memory footprint. In this work, we reveal a different and rather surprising characteristic of network pruning methods: fine-tuning the pruned model with inherited weights is not better than training it from scratch; the resulting pruned architecture is what brings the benefit and could even provide design guidelines for efficient models.

3 METHODOLOGY

In this section, we describe in detail our methodology for training a small target model from scratch.

Target Pruned Architectures. We first divide network pruning methods into two categories. In a pruning pipeline, the target pruned model's architecture can be determined by either human (i.e., predefined) or the pruning algorithm (i.e., automatic) (see Figure 2).

When human predefines the target architecture, a common criterion is the ratio of channels to prune in each layer. For example, we may want to prune 50% channels in each layer of VGG. In this case, no matter which specific channels are pruned, the pruned target architecture remains the same, because the pruning algorithm only *locally* prunes the least important 50% channels in each layer. In practice, the ratio in each layer is usually selected through empirical studies or heuristics.

When the target architecture is automatically determined by a pruning algorithm, it is usually based on a pruning criterion that *globally* compares the importance of structures (e.g., channels) across layers. Examples include Liu et al. (2017), Huang & Wang (2018), Molchanov et al. (2016) and Suau et al. (2018). Non-structured weight pruning (Han et al., 2015) also falls into this category, where the sparsity patterns are determined by the magnitude of trained weights. Note that it is possible to adapt a method with predefined architectures to an automatic method, through appropriate modifications.

Datasets, Network Architectures and Pruning Methods. In the network pruning literature, CIFAR-10, CIFAR-100 (Krizhevsky, 2009), and ImageNet (Deng et al., 2009) datasets are the de-facto benchmarks, while VGG (Simonyan & Zisserman, 2015), ResNet (He et al., 2016) and DenseNet (Huang et al., 2017) are the common network architectures. We evaluate three pruning

methods with predefined target architectures, Li et al. (2017), Luo et al. (2017), He et al. (2017b) and three which automatically discover target models, Liu et al. (2017), Huang & Wang (2018), Han et al. (2015). For the first five methods, we evaluate using the same (target model, dataset) pairs as presented in the original paper to keep our results comparable. For the last one (Han et al., 2015), we use the aforementioned architectures instead, since the ones in the original paper are no longer state-of-the-art. On CIFAR datasets, we run each experiment with 5 random seeds, and report the mean and standard deviation of the accuracy.

Training Budget. One crucial question is how long should we train the small pruned model from scratch? Naively training for the same number of epochs as we train the large model might be unfair, since the small pruned model requires significantly less computation for one epoch. Alternatively, we could compute the floating point operations (FLOPs) for both the pruned and large models, and choose the number of training epoch for the pruned model that would lead to the same amount of computation as training the large model.

In our experiments, we use **Scratch-E** to denote training the small pruned models for the same epochs, and **Scratch-B** to denote training for the same amount of computation budget (on ImageNet, if the pruned model saves more than $2\times$ FLOPs, we just double the number of epochs for training Scratch-B, which amounts to less computation budget than large model training). One may argue that we should instead train the small target model for fewer epochs since it typically converges faster. However, in practice we found that increasing the training epochs within a reasonable range is rarely harmful. We hypothesize that this is because smaller models are less prone to over-fitting.

Implementation. In order to keep our setup as close to the original paper as possible, we use the following protocols: 1) If a previous pruning method's training setup is publicly available, e.g. Liu et al. (2017) and Huang & Wang (2018), we adopt the original implementation; 2) Otherwise, for simpler pruning methods, e.g., Li et al. (2017) and Han et al. (2015), we re-implement the three-stage pruning procedure and achieve similar results to the original paper; 3) For the remaining two methods (Luo et al., 2017; He et al., 2017b), the pruned models are publicly available but without the training setup, thus we choose to re-train both large and small target models from scratch. Interestingly, the accuracy of our re-trained large model is higher than what is reported in the original paper[1]. In this case, to accommodate the effects of different frameworks and training setups, we report the relative accuracy drop from the unpruned large model.

In all these implementations, we use standard training hyper-parameters and data-augmentation schemes. For random weight initialization, we adopt the scheme proposed in He et al. (2015). For results of models fine-tuned from inherited weights, we either use the released models from original papers (case 3 above) or follow the common practice of fine-tuning the model using the lowest learning rate when training the large model (Li et al., 2017; He et al., 2017b). For CIFAR, training/fine-tuning takes 160/40 epochs. For ImageNet, training/fine-tuning takes 90/20 epochs. The code to reproduce the results and the trained models are available at this anynomous link: `https://drive.google.com/open?id=1HB_1FphsWtbuMAdgHbODSLdAMnj3B6TM`.

4 Experiments

In this section we present our experimental results comparing training pruned models from scratch and fine-tuning from inherited weights, for methods with predefined and automatically discovered target architectures. We also include an experiment on transfer learning from image classification to object detection.

4.1 Predefined Target Architectures

L_1-norm based Filter Pruning (Li et al., 2017) is one of the earliest work on filter/channel pruning for convolutional networks. In each layer, a certain percentage of filters with smaller L_1-norm will be pruned. Table 1 shows our results. The Pruned Model column shows the list of predefined target models (see (Li et al., 2017) for configuration details on each model). We observe that in each row, scratch-trained models achieve at least the same level of accuracy as fine-tuned models, with

[1]This could be due to the difference in the deep learning frameworks: we used Pytorch (Paszke et al., 2017) while the original papers used Caffe (Jia et al., 2014)

Scratch-B slightly higher than Scratch-E in most cases. On ImageNet, both Scratch-B models are better than the fine-tuned ones by a noticeable margin.

Dataset	Model	Unpruned	Pruned Model	Fine-tuned	Scratch-E	Scratch-B
CIFAR-10	VGG-16	93.63 ($\pm$0.16)	VGG-16-A	93.41 ($\pm$0.12)	93.62 ($\pm$0.11)	**93.78** ($\pm$0.15)
	ResNet-56	93.14 ($\pm$0.12)	ResNet-56-A	92.97 ($\pm$0.17)	92.96 ($\pm$0.26)	**93.09** ($\pm$0.14)
			ResNet-56-B	92.67 ($\pm$0.14)	92.54 ($\pm$0.19)	**93.05** ($\pm$0.18)
	ResNet-110	93.14 ($\pm$0.24)	ResNet-110-A	93.14 ($\pm$0.16)	**93.25** ($\pm$0.29)	93.22 ($\pm$0.22)
			ResNet-110-B	92.69 ($\pm$0.09)	92.89 ($\pm$0.43)	**93.60** ($\pm$0.25)
ImageNet	ResNet-34	73.31	ResNet-34-A	72.56	72.77	**73.03**
			ResNet-34-B	72.29	72.55	**72.91**

Table 1: Results (accuracy) for L_1-norm based filter pruning (Li et al., 2017). "Pruned Model" is the model pruned from the large model. Configurations of Model and Pruned Model are both from the original paper.

ThiNet (Luo et al., 2017) greedily prunes the channel that has the smallest effect on the next layer's activation values. As shown in Table 2, for VGG-16 and ResNet-50, both Scratch-E and Scratch-B can almost always achieve better performance than the fine-tuned model, often by a significant margin. The only exception is Scratch-E for VGG-Tiny, where the model is pruned very aggressively from VGG-16 (FLOPs reduced by 15$\times$), and as a result, drastically reducing the training budget for Scratch-E. The training budget of Scratch-B for this model is also 7 times smaller than the original large model, yet it can achieve the same level of accuracy as the fine-tuned model.

Dataset	Unpruned	Strategy	Pruned Model		
ImageNet	VGG-16		VGG-Conv	VGG-GAP	VGG-Tiny
	71.03	Fine-tuned	-1.23	-3.67	-11.61
	71.51	Scratch-E	-2.75	-4.66	-14.36
		Scratch-B	$+$**0.21**	$-$**2.85**	$-$**11.58**
	ResNet-50		ResNet50-30%	ResNet50-50%	ResNet50-70%
	75.15	Fine-tuned	-6.72	-4.13	-3.10
	76.13	Scratch-E	-5.21	-2.82	-1.71
		Scratch-B	$-$**4.56**	$-$**2.23**	$-$**1.01**

Table 2: Results (accuracy) for ThiNet (Luo et al., 2017). Names such as "VGG-GAP" and "ResNet50-30%" are pruned models whose configurations are defined in Luo et al. (2017). To accommodate the effects of different frameworks between our implementation and the original paper's, we compare relative accuracy drop from the unpruned large model. For example, for the pruned model VGG-Conv, -1.23 is relative to 71.03 on the left, which is the reported accuracy of the unpruned large model VGG-16 in the original paper; -2.75 is relative to 71.51 on the left, which is VGG-16's accuracy in our implementation.

Dataset	Unpruned	Strategy	Pruned Model
ImageNet	VGG-16		VGG-16-5x
	71.03	Fine-tuned	-2.67
	71.51	Scratch-E	-3.46
		Scratch-B	$-$**0.51**
	ResNet-50		ResNet-50-2x
	75.51	Fine-tuned	-3.25
	76.13	Scratch-E	-1.55
		Scratch-B	$-$**1.07**

Table 3: Results (accuracy) for Regression based Feature Reconstruction (He et al., 2017b). Pruned models such as "VGG-16-5x" are defined in He et al. (2017b). Similar to Table 2, we compare relative accuracy drop from unpruned large models.

Regression based Feature Reconstruction (He et al., 2017b) prunes channels by minimizing the feature map reconstruction error of the next layer. Different from ThiNet (Luo et al., 2017), this optimization problem is solved by LASSO regression. Results are shown in Table 3. Again, in terms of relative accuracy drop from the large models, scratch-trained models are better than the fine-tuned models.

In summary, for pruning methods with predefined target architectures, training the small models for the same number of epochs as the large model (Scratch-E), is often enough to achieve the same accuracy as models output by the three-stage pipeline. Combined with the fact that the target architecture is predefined, in practice one would prefer to train the small model from scratch directly. Moreover, when provided with the same amount of computation budget (measured by FLOPs) as the large model, scratch-trained models can even lead to better performance than the fine-tuned models.

4.2 AUTOMATICALLY DISCOVERED TARGET ARCHITECTURES

Network Slimming (Liu et al., 2017) imposes L_1-sparsity on channel-wise scaling factors from Batch Normalization layers (Ioffe & Szegedy, 2015) during training, and prunes channels with lower scaling factors afterward. Since the channel scaling factors are compared across layers, this method produces automatically discovered target architectures. As shown in Table 4, for all networks, the small models trained from scratch can reach the same accuracy as the fine-tuned models. More specifically, we found that Scratch-B consistently outperforms (8 out of 10 experiments) the fine-tuned models, while Scratch-E is slightly worse but still mostly within the standard deviation.

Dataset	Model	Unpruned	Prune Ratio	Fine-tuned	Scratch-E	Scratch-B
CIFAR-10	VGG-19	93.53 (±0.16)	70%	93.60 (±0.16)	93.30 (±0.11)	**93.81** (±0.14)
	PreResNet-164	95.04 (±0.16)	40%	94.77 (±0.12)	94.70 (±0.11)	**94.90** (±0.04)
			60%	94.23 (±0.21)	94.58 (±0.18)	**94.71** (±0.21)
	DenseNet-40	94.10 (±0.12)	40%	94.00 (±0.20)	93.68 (±0.18)	**94.06** (±0.12)
			60%	**93.87** (±0.13)	93.58 (±0.21)	93.85 (±0.25)
CIFAR-100	VGG-19	72.63 (±0.21)	50%	72.32 (±0.28)	71.94 (±0.17)	**73.08** (±0.22)
	PreResNet-164	76.80 (±0.19)	40%	76.22 (±0.20)	76.36 (±0.32)	**76.68** (±0.35)
			60%	74.17 (±0.33)	75.05 (± 0.08)	**75.73** (±0.29)
	DenseNet-40	73.82 (±0.34)	40%	**73.35** (±0.17)	73.24 (±0.29)	73.19 (±0.26)
			60%	72.46 (±0.22)	72.62 (±0.36)	**72.91** (±0.34)
ImageNet	VGG-11	70.84	50%	68.62	70.00	**71.18**

Table 4: Results (accuracy) for Network Slimming (Liu et al., 2017). "Prune ratio" stands for total percentage of channels that are pruned in the whole network. The same ratios for each model are used as the original paper.

Sparse Structure Selection (Huang & Wang, 2018) also uses sparsified scaling factors to prune structures, and can be seen as a generalization of Network Slimming. Other than channels, pruning can be on residual blocks in ResNet or groups in ResNeXt (Xie et al., 2017). We examine residual blocks pruning, where ResNet-50 are pruned to be ResNet-41, ResNet-32 and ResNet-26. Table 5 shows our results. On average Scratch-E outperforms pruned models, and for all models Scratch-B is better than both.

Dataset	Model	Unpruned	Pruned Model	Pruned	Scratch-E	Scratch-B
ImageNet	ResNet-50	76.12	ResNet-41	75.44	75.61	**76.17**
			ResNet-32	74.18	73.77	**74.67**
			ResNet-26	71.82	72.55	**73.41**

Table 5: Results (accuracy) for residual block pruning using Sparse Structure Selection (Huang & Wang, 2018). In the original paper no fine-tuning is required so there is a "Pruned" column instead of "Fine-tuned" as before.

Non-structured Weight Pruning (Han et al., 2015) prunes individual weights that have small magnitudes. This pruning granularity leaves the weight matrices sparse, hence it is commonly referred to as non-structured weight pruning. Because all the network architectures we evaluated are fully-convolutional (except for the last fully-connected layer), for simplicity, we only prune weights in convolution layers here. Before training the pruned sparse model from scratch, we re-scale the standard deviation of the Gaussian distribution for weight initialization, based on how many non-zero weights remain in this layer. This is to keep a constant scale of backward gradient signal (He et al., 2015). As shown in Table 6, on CIFAR datasets, Scratch-E sometimes falls short of the fine-tuned results, but Scratch-B is able to perform at least on par with the latter. On ImageNet, we note that sometimes even Scratch-B is slightly worse than fine-tuned result. This is the only case where Scratch-B does not achieve comparable accuracy in our attempts. We hypothesize this could be

due to the task complexity of ImageNet and the fine pruning granularity. Another possible reason is that compared with structured pruning, non-structured pruning significantly changes the weight distribution (more details in Appendix G).

Dataset	Model	Unpruned	Prune Ratio	Fine-tuned	Scratch-E	Scratch-B
CIFAR-10	VGG-19	93.50 (±0.11)	30%	93.51 (±0.05)	**93.71** (±0.09)	93.31 (±0.26)
			80%	93.52 (±0.10)	**93.71** (±0.08)	93.64 (±0.09)
	PreResNet-110	95.04 (±0.15)	30%	95.06 (±0.05)	94.84 (±0.07)	**95.11** (±0.09)
			80%	**94.55** (±0.11)	93.76 (±0.10)	94.52 (±0.13)
	DenseNet-BC-100	95.24 (±0.17)	30%	95.21 (±0.17)	95.22 (±0.18)	**95.23** (±0.14)
			80%	95.04 (±0.15)	94.42 (±0.12)	**95.12** (±0.04)
CIFAR-100	VGG-19	71.70 (±0.31)	30%	71.96 (±0.36)	72.81 (±0.31)	**73.30** (±0.25)
			50%	71.85 (±0.30)	73.12 (±0.36)	**73.77** (±0.23)
	PreResNet-110	76.96 (±0.34)	30%	76.88 (±0.31)	76.36 (±0.26)	**76.96** (±0.31)
			50%	**76.60** (±0.36)	75.45 (±0.23)	76.42 (±0.39)
	DenseNet-BC-100	77.59 (±0.19)	30%	77.23 (±0.05)	77.58 (±0.25)	**77.97** (±0.31)
			50%	77.41 (±0.14)	77.65 (±0.09)	**77.80** (±0.23)
ImageNet	VGG-16	73.37	30%	73.68	72.75	**74.02**
			60%	**73.63**	71.50	73.42
	ResNet-50	76.15	30%	**76.06**	74.77	75.70
			60%	**76.09**	73.69	74.91

Table 6: Results (accuracy) for non-structured pruning (Han et al., 2015). "Prune Ratio" denotes the percentage of parameters pruned in the set of all convolutional weights.

4.3 Transfer Learning to Object Detection

We have shown that the small pruned model can be trained from scratch to match the accuracy of the fine-tuned model in classification tasks. To see whether this phenomenon would also hold for transfer learning to other vision tasks, we evaluate the L_1-norm based pruning method (Li et al., 2017) on the PASCAL VOC object detection task, using Faster-RCNN (Ren et al., 2015).

Object detection frameworks usually require transferring model weights pre-trained on ImageNet classification, and one can perform pruning either before or after the weight transfer. More specifically, the former could be described as "train on classification, prune on classification, fine-tune on classification, transfer to detection", while the latter is "train on classification, transfer to detection, prune on detection, fine-tune on detection". We call these two approaches Prune-C (classification) and Prune-D (detection) respectively, and report the results in Table 7. With a slight abuse of notation, here Scratch-E/B denotes "train the small model on classification, transfer to detection", and is different from the setup of detection without ImageNet pre-training as in Shen et al. (2017).

Dataset	Model	Unpruned	Pruned Model	Prune-C	Prune-D	Scratch-E	Scratch-B
PASCAL VOC 07	ResNet-34	71.69	ResNet34-A	71.47	70.99	71.64	**71.90**
			ResNet34-B	70.84	69.62	**71.68**	71.26

Table 7: Results (mAP) for pruning on detection task. The pruned models are chosen from Li et al. (2017). Prune-C refers to pruning on classifcation pre-trained weights, Prune-D refers to pruning after the weights are transferred to detection task. Scratch-E/B means pre-training the pruned model from scratch on classification and transfer to detection.

For this experiment, we adopt the code and default hyper-parameters from Yang et al. (2017), and use PASCAL VOC 07 trainval/test set as our training/test set. For backbone networks, we evaluate ResNet-34-A and ResNet-34-B from the L_1-norm based channel pruning (Li et al., 2017), which are pruned from ResNet-34. Table 7 shows our result, and we can see that the model trained from scratch can surpass the performance of fine-tuned models under the transfer setting.

Another interesting observation from Table 7 is that Prune-C is able to outperform Prune-D, which is surprising since if our goal task is detection, directly pruning away weights that are considered unimportant for detection should presumably be better than pruning on the pre-trained classification models. We hypothesize that this might be because pruning early in the classification stage makes

the final model less prone to being trapped in a bad local minimum caused by inheriting weights from the large model. This is in line with our observation that Scratch-E/B, which trains the small models from scratch starting even earlier at the classification stage, can achieve further performance improvement.

5 NETWORK PRUNING AS ARCHITECTURE SEARCH

While we have shown that the inherited weights in the pruned architecture are not better than random, the pruned architecture itself turns out to be what brings the efficiency benefits. In this section, we demonstrate through empirical studies that the value of automatic network pruning algorithms (Figure 2) actually lies in searching efficient architectures. Note that the connection between network pruning and architecture learning has also been made in some prior works (Han et al., 2015; Liu et al., 2017; Gordon et al., 2018).

Parameter Efficiency of Pruned Architectures. In Figure 3(left), we compare the parameter efficiency of architectures obtained by an automatic channel pruning method (Network Slimming (Liu et al., 2017)) with a naive predefined pruning strategy that uniformly prunes the same percentage of channels in each layer. All architectures are trained from random initialization for the same number of epochs. We see that the architectures obtained by Network Slimming are more parameter efficient, as they could achieve the same level of accuracy using 5× fewer parameters than uniformly pruning architectures. For non-structured weight pruning (Han et al., 2015), we conducted a similar experiment shown in Figure 3(right). Here we uniformly sparsify all individual weights at a fixed probability, and the architectures obtained this way are much less efficient than the pruned architectures.

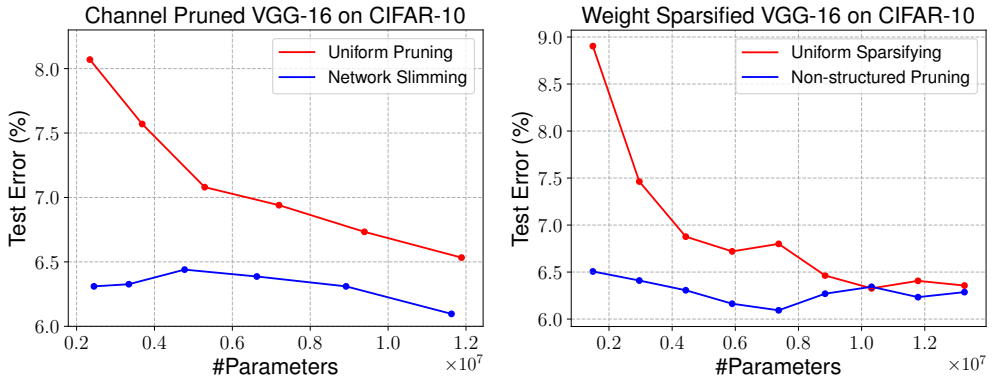


Figure 3: Pruned architectures obtained by different approaches, all *trained from scratch*, averaged over 5 runs. Architectures obtained by automatic pruning methods (*Left:* Network Slimming (Liu et al., 2017), *Right:* Non-stuctured weight pruning (Han et al., 2015)) have better parameter efficiency than uniformly pruning channels or sparsifying weights in the whole network.

Layer	Width	Width*	Layer	Width	Width*
1	64	39.0±3.7	8	512	217.3±6.6
2	64	64.0±0.0	9	512	120.0±4.4
3	128	127.8±0.4	10	512	63.0±1.9
4	128	128.0±0.0	11	512	47.8±2.9
5	256	255.0±1.0	12	512	62.0±3.4
6	256	250.5±0.5	13	512	88.8±3.1
7	256	226.0±2.5	Total	4224	1689.2

Table 8: Network architectures obtained by pruning 60% channels on VGG-16 (in total 13 conv-layers) using Network Slimming. Width and Width* are number of channels in the original and pruned architectures, averaged over 5 runs.

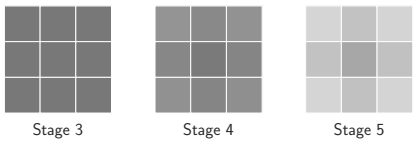


Figure 4: The average sparsity pattern of all 3×3 convolutional kernels in certain layer stages in a non-structured pruned VGG-16. Darker color means higher probability of weight being kept.

We also found the channel/weight pruned architectures exhibit very consistent patterns (see Table 8 and Figure 4). This suggests the original large models may be redundantly designed for the task and the pruning algorithm can help us improve the efficiency. Combined with the results presented in Section 4, we hypothesize that the value of automatic pruning methods actually lies in the resulting architecture rather than the inherited weights.

Generalizable Design Principles from Pruned Architectures. Given that the automatically discovered architectures tend to be parameter efficient, one may wonder: can we derive generalizable principles from them on how to design a better architecture? To answer this question, we conduct experiments on Network Slimming and non-structured pruning with various network architectures.

For Network Slimming, we use the average number of channels in each layer stage (layers with the same feature map size) from pruned architectures to construct a new set of architectures, and we call this approach "Guided Pruning"; for non-structured pruning, we analyze the sparsity patterns (Table 4) in the pruned architectures, and apply them to construct a new set of sparse models, which we call "Guided Sparsifying". The results are shown in Figure 5 and Figure 6. It can be seen that for both Network Slimming (Figure 5 left and Figure 6) and non-structured pruning (Figure 5 right), guided design of architectures (green) can perform on par with pruned architectures (blue).

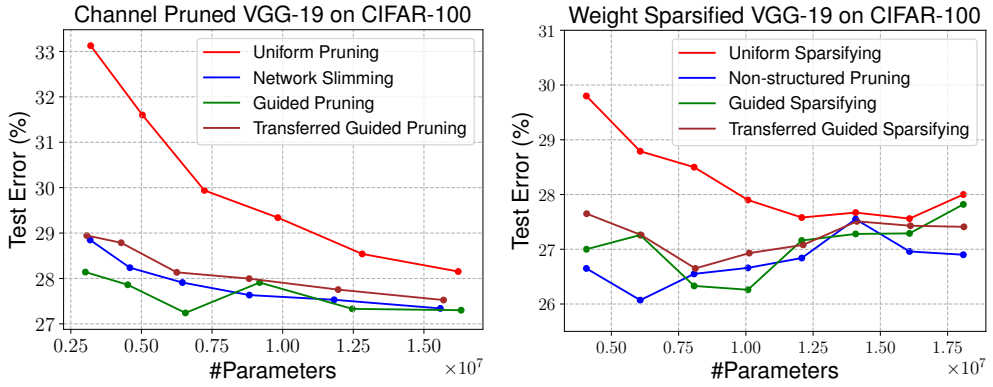


Figure 5: Pruned architectures obtained by different approaches, *all trained from scratch*, averaged over 5 runs. "Guided Pruning/Sparsifying" means using the average sparsity patterns in each layer stage to design the network; "Transferred Guided Pruning/Sparsifying" means using the sparsity patterns obtained by a pruned VGG-16 on CIFAR-10, to design the network for VGG-19 on CIFAR-100. Following the design guidelines provided by the pruned architectures, we achieve better parameter efficiency, even when the guidelines are transferred from another dataset and model.

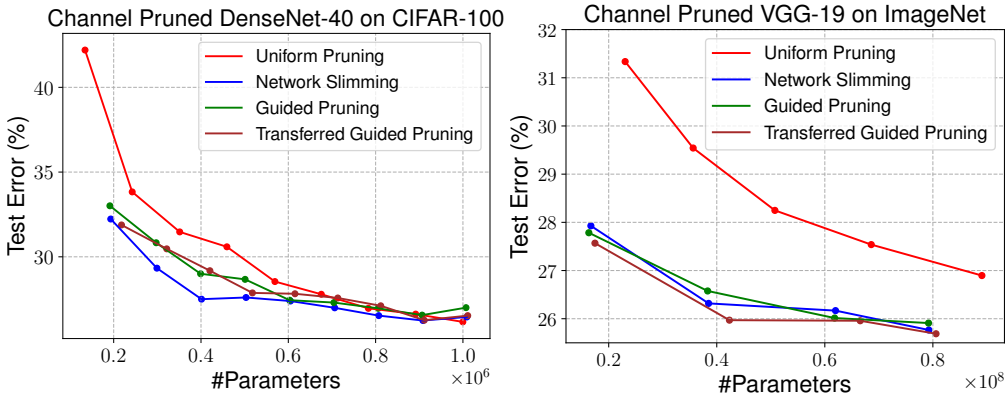


Figure 6: Pruned architectures obtained by Network Slimming (Liu et al., 2017), *all trained from scratch*, averaged over 5 runs. *Left:* Results for DenseNet-40 on CIFAR-100. We transfer the sparsity patterns of pruned DenseNet-76 on CIFAR-100 to design efficient networks. *Right:* Results for VGG-19 on ImageNet. We transfer the sparsity patterns of pruned VGG-11 on ImageNet to design efficient networks.

Interestingly, these guided design patterns can sometimes be transferred to a different architecture and/or a different dataset. In Figure 5, we distill the patterns of pruned architectures from VGG-16

on CIFAR-10 and apply them to design efficient VGG-19 on CIFAR-100. Similarly, in Figure 6, we transfer the average pruned sparsity patterns from DenseNet-76 on CIFAR-100 and VGG-11 on ImageNet respectively to DenseNet-40 on CIFAR-100 and VGG-19 on ImageNet. These sets of architectures are denoted as "Transferred Guided Pruning/Sparsifying". We can observe that they (brown) may sometimes be slightly worse than architectures directly pruned (blue), but are significantly better than uniform pruning/sparsifying (red). In these cases, one does not need to train a large model to obtain an efficient model as well, as transferred design patterns can help us achieve the efficiency directly. However, there also exist cases where the architectures obtained by pruning are not better than uniformly pruned ones, and we defer the results and analysis in Appendix F due to space limit.

Comparison with Traditional Architecture Search Methods. Conventional techniques for network architecture search include reinforcement learning (Zoph & Le, 2017; Baker et al., 2017) and evolutionary algorithms (Xie & Yuille, 2017; Liu et al., 2018a). In each iteration, a randomly initialized network is trained and evaluated to guide the search, and the search process usually requires thousands of iterations to find the goal architecture. In contrast, using network pruning as architecture search only requires a one-pass training, however the search space is restricted to the set of all "sub-networks" inside a large network, whereas traditional methods can search for more variations, e.g., activation functions or different layer orders.

Recently, Gordon et al. (2018) uses a similar pruning technique to Network Slimming (Liu et al., 2017) to automate the design of network architectures; He et al. (2018c) prune channels using reinforcement learning and automatically compresses the architecture. On the other hand, in the network architecture search literature, sharing/inheriting trained parameters (Pham et al., 2018; Liu et al., 2018b) has become a popular approach to accelerate the convergence and reduce the training budget during searching, though it would be interesting to investigate whether training from scratch would sometimes yield better results as we observed in network pruning. We can see that these two areas, namely network pruning and architecture search, share many common traits and start to borrow wisdom from each other.

6 Discussion and Conclusion

We suggest future pruning methods be evaluated on appropriately strong baselines, especially when the target pruned architectures are predefined. In addition to high accuracy, training predefined target models from scratch has the following benefits over conventional network pruning procedures:

- Since the model is smaller, we can train the model using less GPU memory and possibly faster than training the original large model.
- There is no need to implement the pruning criterion and procedure, which sometimes requires fine-tuning layer by layer (Luo et al., 2017) and/or needs to be customized for different network architectures (Li et al., 2017; Liu et al., 2017).
- We avoid tuning additional hyper-parameters involved in the pruning procedure.

Our results support the use of pruning methods for finding efficient architectures or sparsity patterns. This can be done using automatic pruning approaches. In addition, there are still some cases where conventional pruning methods can be much faster than training from scratch:

- When a pre-trained large model is already given and little or no training budget is available.
- There is a need to obtain multiple models of different sizes, in this situation one can train a large model and then prune it by different ratios.

In summary, our experiments have shown that training the small pruned model from scratch can almost always achieve comparable or higher level of accuracy than the model obtained from the typical "training, pruning and fine-tuning" procedure. This changed our understanding about the necessity of over-parameterization for obtaining a final efficient model, and the effectiveness of inheriting weights that are considered important by the pruning criteria. We further demonstrated the value of automatic pruning algorithms could be regarded as finding efficient architectures and providing architecture design guidelines.

REFERENCES

Jose M Alvarez and Mathieu Salzmann. Learning the number of neurons in deep networks. In *NIPS*, 2016.

Anonymous. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *Submitted to International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=rJl-b3RcF7`. under review.

Sajid Anwar and Wonyong Sung. Compact deep convolutional neural networks with coarse pruning. *arXiv preprint arXiv:1610.09639*, 2016.

Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *NIPS*, 2014.

Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. *ICLR*, 2017.

Miguel A Carreira-Perpinán and Yerlan Idelbayev. "Learning-compression" algorithms for neural net pruning. In *CVPR*, 2018.

Ting-Wu Chin, Cha Zhang, and Diana Marculescu. Layer-compensated pruning for resource-constrained convolutional neural networks. *arXiv preprint arXiv:1810.00518*, 2018.

Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *NIPS*, 2014.

Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014.

Ariel Gordon, Elad Eban, Ofir Nachum, Bo Chen, Hao Wu, Tien-Ju Yang, and Edward Choi. Morphnet: Fast & simple resource-constrained structure learning of deep networks. In *CVPR*, 2018.

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *NIPS*, 2015.

Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. Eie: efficient inference engine on compressed deep neural network. In *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*, 2016a.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *ICLR*, 2016b.

Babak Hassibi and David G Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *NIPS*, 1993.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

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

Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *ICCVs*, 2017a.

Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. Soft filter pruning for accelerating deep convolutional neural networks. In *IJCAI*, 2018a.

Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. Soft filter pruning for accelerating deep convolutional neural networks. *https://github.com/he-y/soft-filter-pruning*, 2018b.

Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *ICCV*, 2017b.

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*, 2018c.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *NIPS Workshop*, 2014.

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

Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *CVPR*, 2017.

Zehao Huang and Naiyan Wang. Data-driven sparse structure selection for deep neural networks. *ECCV*, 2018.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *ACM Multimedia*, 2014.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

Vadim Lebedev and Victor Lempitsky. Fast convnets using group-wise brain damage. In *CVPR*, 2016.

Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan Oseledets, and Victor Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *ICLR*, 2014.

Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *NIPS*, 1990.

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

Ji Lin, Yongming Rao, Jiwen Lu, and Jie Zhou. Runtime neural pruning. In *NIPS*, 2017.

Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. *ICLR*, 2018a.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018b.

Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *ICCV*, 2017.

Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015.

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

Deepak Mittal, Shweta Bhardwaj, Mitesh M Khapra, and Balaraman Ravindran. Recovering from random pruning: On the plasticity of deep convolutional neural networks. *arXiv preprint arXiv:1801.10447*, 2018.

Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS Workshop*, 2017.

Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *ICML*, 2018.

Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *ECCV*, 2016.

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *NIPS*, 2015.

Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. *ICLR*, 2015.

Zhiqiang Shen, Zhuang Liu, Jianguo Li, Yu-Gang Jiang, Yurong Chen, and Xiangyang Xue. Dsod: Learning deeply supervised object detectors from scratch. In *ICCV*, 2017.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *ICLR*, 2015.

Suraj Srinivas and R Venkatesh Babu. Data-free parameter pruning for deep neural networks. *BMVC*, 2015.

Xavier Suau, Luca Zappella, Vinay Palakkode, and Nicholas Apostoloff. Principal filter analysis for guided network compression. *arXiv preprint arXiv:1807.10585*, 2018.

Xin Wang, Fisher Yu, Zi-Yi Dou, and Joseph E Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. *arXiv preprint arXiv:1711.09485*, 2017.

Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *NIPS*, 2016.

Lingxi Xie and Alan L Yuille. Genetic cnn. In *ICCV*, 2017.

Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *CVPR*, 2017.

Jianwei Yang, Jiasen Lu, Dhruv Batra, and Devi Parikh. A faster pytorch implementation of faster r-cnn. *https://github.com/jwyang/faster-rcnn.pytorch*, 2017.

Jianbo Ye, Xin Lu, Zhe Lin, and James Z Wang. Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. *ICLR*, 2018.

Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S Davis. Nisp: Pruning networks using neuron importance score propagation. In *CVPR*, 2018.

Hao Zhou, Jose M Alvarez, and Fatih Porikli. Less is more: Towards compact cnns. In *ECCV*, 2016.

Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *ICLR Workshop*, 2018.

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

APPENDIX

A RESULTS ON SOFT FILTER PRUNING (HE ET AL., 2018A)

Soft Filter Pruning (SFP) (He et al., 2018a) prunes filters every epoch during training but also keeps updating the pruned filters, i.e., the pruned weights have the chance to be recovered. In the original paper, SFP can either run upon a random initialized model or a pretrained model. It falls into the category of predefined methods (Figure 2). Table 9 shows our results without using pretrained models and Table 10 shows the results with a pretrained model. We use authors' code (He et al., 2018b) for obtaining the results. It can be seen that Scratch-E outperforms pruned models for most of the time and Scratch-B outperforms pruned models in nearly all cases. Therefore, our observations also hold on soft filter pruning.

Dataset	Model	Unpruned	Prune Ratio	Pruned	Scratch-E	Scratch-B
CIFAR-10	ResNet-20	92.41 (±0.12)	10%	92.00 (±0.32)	**92.22** (±0.15)	92.13 (±0.10)
			20%	91.50 (±0.30)	91.62 (±0.12)	**91.67** (±0.15)
			30%	90.78 (±0.15)	90.93 (±0.10)	**91.07** (±0.23)
	ResNet-32	93.22 (±0.16)	10%	93.28 (±0.05)	**93.42** (±0.40)	93.08 (±0.13)
			20%	92.50 (±0.17)	92.68 (±0.20)	**92.96** (±0.11)
			30%	92.02 (±0.11)	92.37 (±0.12)	**92.56** (±0.06)
	ResNet-56	93.80 (±0.12)	10%	93.77 (±0.07)	93.42 (±0.40)	**93.98** (±0.21)
			20%	93.14 (±0.42)	93.44 (±0.05)	**93.71** (±0.14)
			30%	93.01 (±0.09)	93.19 (±0.20)	**93.57** (±0.12)
			40%	92.59 (±0.14)	92.80 (±0.25)	**93.07** (±0.25)
	ResNet-110	93.77 (±0.23)	10%	93.60 (±0.50)	**94.21** (±0.39)	94.13 (±0.37)
			20%	93.63 (±0.44)	93.52 (±0.18)	**94.29** (±0.18)
			30%	93.26 (±0.37)	93.70 (±0.16)	**93.92** (±0.13)
ImageNet	ResNet-34	73.92	30%	71.83	71.67	**72.97**
	ResNet-50	76.15	30%	74.61	74.98	**75.56**

Table 9: Results (accuracy) for Soft Filter Pruning (He et al., 2018a) without pretrained models.

Dataset	Model	Unpruned	Prune Ratio	Pruned	Scratch-E	Scratch-B
CIFAR-10	ResNet-56	93.80 (±0.12)	30%	93.51 (±0.26)	**94.45** (±0.30)	93.77 (±0.25)
			40%	93.10 (±0.34)	**93.84** (±0.16)	93.41 (±0.08)
	ResNet-110	93.77 (±0.23)	30%	93.46 (±0.19)	93.89 (±0.17)	**94.37** (±0.24)

Table 10: Results (accuracy) for Soft Filter Pruning (He et al., 2018a) using pretrained models.

B EXPERIMENTS ON THE LOTTERY TICKET HYPOTHESIS (ANONYMOUS, 2019)

The lottery ticket hypothesis (Anonymous, 2019) conjectures that inside the large network, a sub-network together with their initialization makes the training particularly effective, and together they are termed the "winning ticket". In this hypothesis, the original initialization of the sub-network (before large model training) is needed for it to achieve competitive performance when trained in isolation. This is shown in their Figure 3, where training the sub-network with randomly re-initialized weights performs worse than training it with the original initialization inside the large network. In contrast, our work does not require reuse of the original initialization of the pruned model, and shows that random initialization is enough for the pruned model to achieve competitive performance.

We note that the experiments in Anonymous (2019) are conducted on very small plain networks (<6 layers), and use Adam with a small learning rate, which could not lead to high-performance on common benchmarks such as CIFAR-10. Our experiments are on modern network architectures

(e.g., ResNet and DenseNet) and follow the standard training practice (e.g., use SGD with momentum and a large initial learning rate of 0.1) in both the image classification literature (He et al., 2016; Huang et al., 2017) and the original setting of the pruning algorithms we evaluate. Moreover, Anonymous (2019) tests the hypothesis on non-structured weight pruning, while we consider six pruning methods on various levels of structures.

Dataset	Model	Unpruned	Prune Ratio	Lottery Ticket	Random Init
CIFAR-10	VGG-19	93.50 (±0.11)	30%	**93.69** (±0.13)	93.63 (±0.16)
			80%	93.58 (±0.15)	**93.65** (±0.19)
	PreResNet-110	95.04 (±0.15)	30%	94.89 (±0.14)	**94.97** (±0.10)
			80%	**93.87** (±0.15)	93.79 (±0.17)
CIFAR-100	VGG-19	71.70 (±0.31)	30%	**72.57** (±0.58)	**72.57** (±0.23)
			50%	**72.75** (±0.22)	72.31 (±0.19)
	PreResNet-110	76.96 (±0.34)	30%	76.41 (±0.15)	**76.60** (±0.10)
			50%	**75.61** (±0.12)	75.48 (±0.17)

Table 11: Experiments on the lottery ticket hypothesis (Anonymous, 2019) with non-structured weight pruning (Han et al., 2015). 'Lottery Ticket" refers to training the pruned models with the original initialization as in Anonymous (2019). "Random Init" refers to training the pruned models with weights randomly re-initialized, as in all other experiments in this paper.

Dataset	Model	Unpruned	Pruned Model	Lottery Ticket	Random Init
CIFAR-10	VGG-16	93.63 (±0.16)	VGG-16-A	**93.62** (±0.09)	93.60 (±0.15)
	ResNet-56	93.14 (±0.12)	ResNet-56-A	92.72 (±0.10)	**92.75** (±0.26)
			ResNet-56-B	92.78 (±0.23)	**92.90** (±0.27)
	ResNet-110	93.14 (±0.24)	ResNet-110-A	**93.21** (±0.09)	**93.21** (±0.21)
			ResNet-110-B	93.15 (±0.12)	**93.37** (±0.29)

Table 12: Experiments on the lottery ticket hypothesis (Anonymous, 2019) with L_1-norm based filter pruning (Li et al., 2017). "Lottery Ticket" refers to training the pruned models with the original initialization as in Anonymous (2019). "Random Init" refers to training the pruned models with weights randomly re-initialized, as in all other experiments in this paper.

We investigate whether the original initialization is needed to achieve competitive performance in a standard setting, by comparing the models trained with the original initialization ("winning ticket") and that trained from randomly re-initialized weights. We show our results for non-structured weight pruning (Han et al., 2015) in Table 11 and L_1-norm based filter pruning (Li et al., 2017) in Table 12. It can be seen that using the original initialization as in (Anonymous, 2019) does not provide any advantage over random initialization in a standard setting. Note that the authors of (Anonymous, 2019) also report in their Appendix D, that the "winning ticket" cannot be found on a ResNet-18 using the original large learning rate used in ResNet (He et al., 2016). The reason why the original initialization is helpful in the setting of Anonymous (2019) might be that the used learning rate is relatively small, thus the weights of the final trained model are not changed to a large degree from the original initialization.

C Aggressively Pruned Models

It would be interesting to see whether our observation still holds if the model is very aggressively pruned, since they might not have enough capacity to be trained from scratch and achieve decent accuracy. Here we provide results using Network Slimming (Liu et al., 2017). From Table 13, it can be seen that when the pruned ratio is large, training from scratch is better than fine-tuned models by an even larger margin, and in this case fine-tuned models are significantly worse than the unpruned accuracy. Note that in Table 2, the VGG-Tiny model we evaluated for ThiNet (Luo et al., 2017) is also a very aggressively pruned model (FLOPs reduced by 15× and parameters reduced by 100×).

Dataset	Model	Unpruned	Prune Ratio	Fine-tuned	Scratch-E	Scratch-B
CIFAR-10	PreResNet-164	95.04 (±0.16)	80%	91.76 (±0.38)	93.21 (±0.17)	**93.49** (±0.20)
			90%	82.06 (±0.92)	87.55 (±0.68)	**88.44** (±0.19)
	DenseNet-40	94.10 (±0.12)	80%	92.64 (±0.12)	93.07 (±0.08)	**93.61** (±0.12)
CIFAR-100	DenseNet-40	73.82 (±0.34)	80%	69.60 (±0.22)	71.04 (±0.36)	**71.45** (±0.30)

Table 13: Results (accuracy) for Network Slimming (Liu et al., 2017) when the models are significantly pruned. "Prune ratio" stands for total percentage of channels that are pruned in the whole network. Larger ratios are used than the original paper of Liu et al. (2017).

D FINE-TUNING EPOCHS

Generally, pruning algorithms use fewer epochs for fine-tuning than training the large model (Li et al., 2017; He et al., 2017b; Luo et al., 2017). For example, L_1-norm based filter pruning (Li et al., 2017) uses 164 epochs for training on CIFAR-10 datasets, and only fine-tunes the pruned networks for 40 epochs. This is due to that mostly small learning rates are used for fine-tuning to better preserve the weights from the large model. Here we experiment with fine-tuning for more epochs (e.g., for the same number of epochs as Scratch-E) and show it does not bring noticeable performance improvement.

Dataset	Model	Pruned Model	Fine-tune-40	Fine-tune-80	Fine-tune-160	Scratch-E
CIFAR-10	VGG-16	VGG-16-A	93.40 (±0.12)	93.45 (±0.06)	93.45 (±0.08)	**93.62** (±0.11)
	ResNet-56	ResNet-56-A	**92.97** (±0.17)	92.92 (±0.15)	92.94 (±0.16)	92.96 (±0.26)
		ResNet-56-B	92.68 (±0.19)	92.67 (±0.14)	**92.76** (±0.16)	92.54 (±0.19)
	ResNet-110	ResNet-110-A	93.14 (±0.16)	93.12 (±0.19)	93.04 (±0.22)	**93.25** (±0.29)
		ResNet-110-B	92.69 (±0.09)	92.75 (±0.15)	92.76 (±0.16)	**92.89** (±0.43)

Table 14: "Fine-tune-40" stands for fine-tuning 40 epochs and so on. Scratch-E models are trained for 160 epochs. We observe that fine-tuning for more epochs does not help improve the accuracy much, and models trained from scratch can still perform on par with fine-tuned models.

We use L_1-norm filter pruning (Li et al., 2017) for this experiment. Table 14 shows our results with different number of epochs for fine-tuning. It can be seen that fine-tuning for more epochs gives negligible accuracy increase and sometimes small decrease, and Scratch-E models are still on par with models fine-tuned for enough epochs.

E EXTENDING THE STANDARD TRAINING SCHEDULE

In our experiments, we use the standard training schedule for both CIFAR (160 epochs) and ImageNet (90 epochs). Here we show that our observation still holds after we extend the standard training schedule. We use L_1-norm based filter pruning (Li et al., 2017) for this experiment. Table 15 shows our results when we extend the standard training schedule of CIFAR from 160 to 300 epochs. We observe that scratch trained models still perform at least on par with fine-tuned models.

Dataset	Model	Unpruned	Pruned Model	Fine-tuned	Scratch-E	Scratch-B
CIFAR-10	ResNet-110	93.82 (±0.32)	ResNet-110-A	93.75 (±0.24)	93.80 (±0.15)	**94.10** (±0.12)
			ResNet-110-B	93.36 (±0.28)	93.75 (±0.16)	**93.90** (±0.17)

Table 15: Results for L_1-norm filter pruning (Li et al., 2017) when the training schedule of the large model is extended from 160 to 300 epochs.

F ADDITIONAL RESULTS FOR ARCHITECTURE SEARCH

Continuing on section 5, we show some cases where the pruned architectures or guided pruning/sparsifying are not better than architectures obtained by uniform pruning/sparsifying in Figure 7. The blue and green curves are not noticeably better than the red curve. This phenomenon happens

more likely on modern architectures like ResNet and DenseNet. When we investigate the sparsity patterns of those pruned architectures (shown in Table 16, Table 17 and Table 18), we find that they exhibit near-uniform sparsity patterns across stages, which might be the reason why it can only perform on par with uniform pruning. In contrast, for VGG the pruned sparsity patterns can always beat the uniform ones as shown in Figure 3 and Figure 5. We also show the sparsity patterns of VGG pruned by Network Slimming (Liu et al., 2017) in Table 19, and they are rather far from uniform. Compared to ResNet and DenseNet, we can see that VGG's redundancy is rather inbalanced across layer stages, while modern networks' redundancy is shared more evenly. Network pruning techniques can help us identify the redundancy better in the former case.

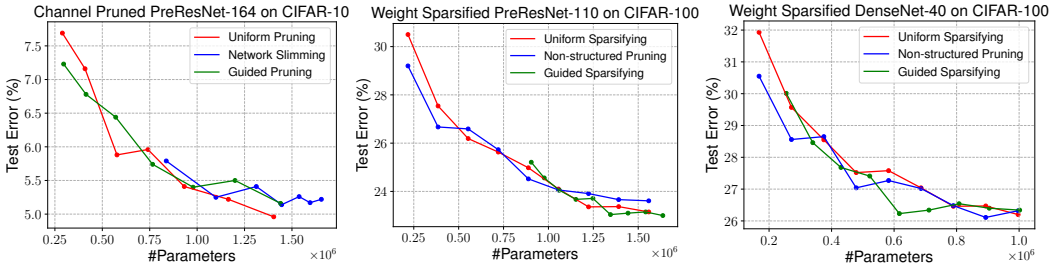


Figure 7: Pruned architectures obtained by different approaches, *all trained from scratch*, averaged over 5 runs. *Left:* Results for PreResNet-164 pruned on CIFAR-10 by Network Slimming (Liu et al., 2017). *Middle* and *Right*: Results for PreResNet-110 and DenseNet-40 pruned on CIFAR-100 by non-structured weight pruning (Han et al., 2015).

	10%	20%	30%	40%	50%	60%	70%
Stage 1	0.879	0.729	0.557	0.484	0.421	0.349	0.271
Stage 2	0.959	0.863	0.754	0.651	0.537	0.428	0.320
Stage 3	0.889	0.798	0.716	0.610	0.507	0.403	0.301

Table 16: Sparsity patterns of PreResNet-164 pruned on CIFAR-10 by Network Slimming shown in Figure 7 (left) under different prune ratio. The top row denotes the total prune ratio. The values denote the ratio of channels to be kept. We can observe that for a certain prune ratio, the sparsity patterns are close to uniform (across stages).

	10%			50%			90%		
Stage 1	0.905	0.905	0.909	0.530	0.561	0.538	0.129	0.171	0.133
	0.900	0.912	0.899	0.559	0.588	0.551	0.166	0.217	0.176
	0.903	0.913	0.902	0.532	0.563	0.547	0.142	0.172	0.163
Stage 2	0.906	0.911	0.906	0.485	0.523	0.503	0.073	0.102	0.085
	0.912	0.911	0.915	0.508	0.529	0.525	0.099	0.114	0.111
	0.911	0.916	0.912	0.502	0.529	0.519	0.080	0.113	0.096
Stage 3	0.901	0.904	0.900	0.454	0.475	0.454	0.043	0.059	0.048
	0.885	0.891	0.889	0.409	0.420	0.415	0.032	0.033	0.035
	0.898	0.903	0.902	0.450	0.468	0.458	0.042	0.055	0.046

Table 17: Average sparsity patterns of 3×3 kernels of PreResNet-110 pruned on CIFAR-100 by non-structured weight pruning shown in Figure 7 (middle) under different prune ratio. The top row denotes the total prune ratio. The values denote the ratio of weights to be kept. We can observe that for a certain prune ratio, the sparsity patterns are close to uniform (across stages).

G WEIGHT DISTRIBUTION

We show the weight distribution of unpruned large models, fine-tuned pruned models and scratch-trained pruned models, for two pruning methods: Network Slimming (Liu et al., 2017) and non-structured weight pruning (Han et al., 2015). We choose DenseNet-40 and CIFAR-10 for visualization and compare the weight distribution of unpruned models, fine-tuned models and scratch-trained

	10%			50%			90%		
Stage 1	0.861	0.856	0.858	0.507	0.495	0.510	0.145	0.129	0.142
	0.843	0.844	0.851	0.484	0.486	0.479	0.123	0.115	0.126
	0.850	0.854	0.857	0.509	0.490	0.511	0.136	0.131	0.147
Stage 2	0.907	0.905	0.906	0.498	0.487	0.499	0.099	0.088	0.100
	0.892	0.888	0.892	0.442	0.427	0.444	0.064	0.043	0.065
	0.907	0.906	0.905	0.497	0.485	0.493	0.095	0.082	0.098
Stage 3	0.897	0.901	0.899	0.470	0.475	0.472	0.060	0.060	0.064
	0.888	0.890	0.889	0.433	0.437	0.435	0.040	0.040	0.042
	0.898	0.900	0.899	0.473	0.477	0.473	0.060	0.061	0.063

Table 18: Average sparsity patterns of 3×3 kernels of DenseNet-40 pruned on CIFAR-100 by non-structured weight pruning shown in Figure 7 (right) under different prune ratio. The top row denotes the total prune ratio. The values denote the ratio of weights to be kept. We can observe that for a certain prune ratio, the sparsity patterns are close to uniform (across stages).

	10%	20%	30%	40%	50%	60%
Stage 1	0.969	0.914	0.883	0.875	0.844	0.836
Stage 2	1.000	1.000	1.000	1.000	1.000	1.000
Stage 3	0.991	0.975	0.966	0.957	0.947	0.947
Stage 4	0.861	0.718	0.575	0.446	0.312	0.258
Stage 5	0.871	0.751	0.626	0.486	0.352	0.132

Table 19: Sparsity patterns of VGG-16 pruned on CIFAR-10 by Network Slimming shown in Figure 3 (left) under different prune ratio. The top row denotes the total prune ratio. The values denote the ratio of channels to be kept. For each prune ratio, the latter stages tend to have more redundancy than earlier stages.

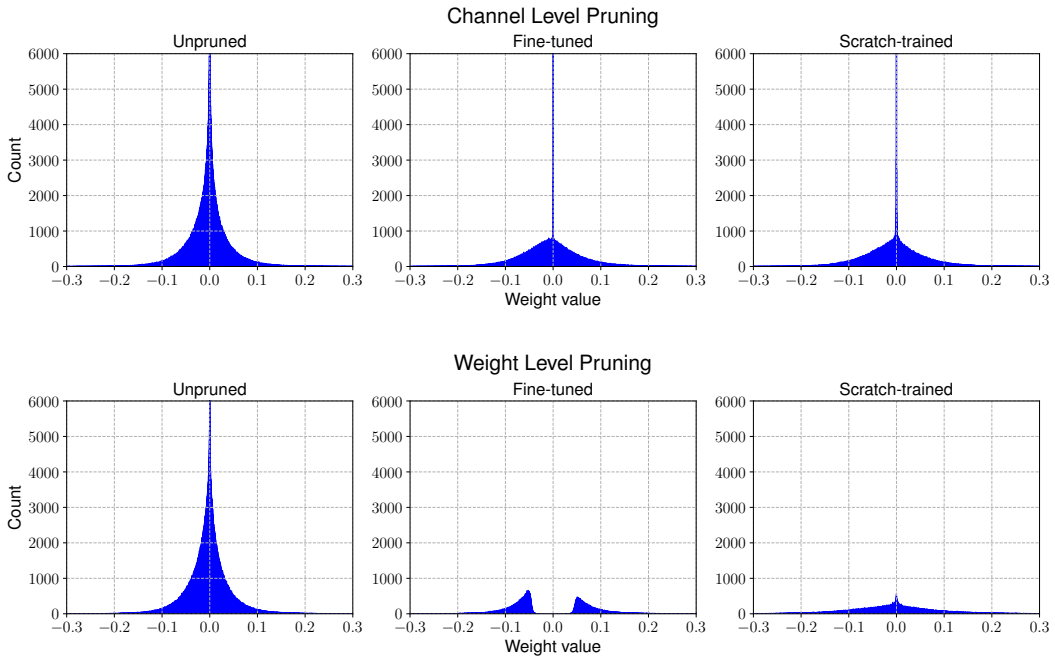


Figure 8: Weight distribution of convolutional layers for different pruning methods. We use DenseNet-40 and CIFAR-10 for this visualization. We compare the weight distribution of unpruned models, fine-tuned models and scratch-trained models. *Top*: Results for Network Slimming Liu et al. (2017). *Bottom*: Results for non-structured weight pruning Han et al. (2015).

models. For Network Slimming, the prune ratio is chosen to be 60%. For non-structured weight pruning, the prune ratio is chosen to be 80%. Figure 8 shows our result. We can see that the weight distribution of fine-tuned models and scratch-trained pruned models are different from the unpruned large models – the weights that are close to zero are much fewer. This seems to imply that there are less redundant structures in the found pruned architecture, and support the effect of architecture learning for automatic pruning methods.

For non-structured weight pruning, the fine-tuned model also has significantly different weight distribution from the scratch-trained model – it has nearly no near-zero values. This might be a reason why on ImageNet sometimes scratch trained models cannot achieve the accuracy of the fine-tuned models, as shown in Table 6.