Rethinking the Value of Network Pruning

Anonymous authors
Paper under double-blind review

Abstract

Network pruning is widely used for reducing the heavy computational cost of deep networks. A typical pruning algorithm is a three-stage pipeline, i.e., training (a large model), pruning and fine-tuning, and each of the three stages is considered as indispensable. 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 architecture of the target pruned network, one can completely get rid of the 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 an over-parameterized model is not necessary to obtain an efficient final model, 2) learned "important" weights of the large model are not necessarily helpful 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.

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. 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, 2) prune the trained large model according to a certain criterion, and 3) fine-tune the pruned model to regain the lost performance.

Training
Pruning
Fine-tuning

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), as it provides a high-performance model (due to stronger representation & optimization power) from which one can remove a set of redundant parameters without significant hurting the accuracy. This is usually reported to be superior to directly training a smaller network from scratch. Second, both the pruned architecture *and* its associated weights are believed to be essential for obtaining the final efficient model. 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 (Han et al., 2015; Hu et al., 2016; Li et al., 2017; Liu et al., 2017; Luo et al., 2017; He et al., 2017b; Ye 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 existing pruning algorithms on multiple datasets with multiple network architectures, we make two surprising observations. First, for pruning algorithms with pre-defined 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. This implies that starting with a large model is not necessary. Second, for pruning algorithms without a pre-defined 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. The contradiction between our results and those reported in the literature might be explained by less carefully chosen hyper-parameters, data augmentation scheme and unfair computation budget for training.

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 some pruning algorithms may lie in identifying efficient structures and performing architecture search, rather than selecting "important" weights. We verify this hypothesis through a set of carefully designed experiments described in Section 5.

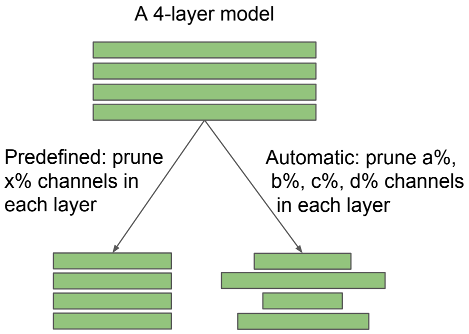
A 4-layer model
Predefined: prune x% channels in each layer
Automatic: prune a%, b%, c%, d% channels in each layer

Figure 2: Difference between pre-defined and non-pre-defined (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 the 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 to show our main results for both pruning methods with pre-defined or automatically discovered target architectures; in Section 5, we argue that the value of some pruning methods indeed lies in searching efficient network architectures, 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.), 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., 2016). 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.

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 idea 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) use 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.

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. 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 no better than training it from scratch.

3 METHODOLOGY

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

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 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.

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 discovered 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. For testing on the ImageNet, the image is first resized so that the shorter edge has length 256, and then center-cropped to be of 224×224.

Training Budget. One crucial question is how long we should 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[1]. 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[2]. 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 (for 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). The code to reproduce the results will be made publicly available.

4 Experiments

In this section we present our experimental results comparing training pruned models from scratch and fine-tuning from inherited weights, for both 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

L1-norm based Channel Pruning (Li et al., 2017) is one of the earliest work on channel pruning for convolutional networks. In each layer, a certain percentage of channels with smaller L1-norm of its filter weights 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. The ImageNet models trained from scratch are even slightly better than the fine-tuned ones. Note that here we train pruned models from scratch for the same number of epochs (Scratch-E), thus the training budget is much less than training the large model.

Dataset	Model	Unpruned	Pruned Model	Fine-tuned	Scratch-E
CIFAR-10	VGG-16	93.63 (±0.16)	VGG-16-A	93.41 (±0.12)	**93.62** (±0.11)
	ResNet-56	93.14 (±0.12)	ResNet-56-A	**92.97** (±0.17)	92.96 (±0.26)
			ResNet-56-B	**92.67** (±0.14)	92.54 (±0.19)
	ResNet-110	93.14 (±0.24)	ResNet-110-A	93.14 (±0.16)	**93.25** (±0.29)
			ResNet-110-B	92.69 (±0.09)	**92.89** (±0.43)
ImageNet	ResNet-34	73.31	ResNet-34-A	72.56	**72.77**
			ResNet-34-B	72.29	**72.55**

Table 1: Results (accuracy) for L1-norm based channel 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.

[1] On ImageNet, if the pruned model saves more than 2× FLOPs, we just double the number of epochs, which amounts to less computation budget than large model training.

[2] 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)

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×), 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	-	−4.93	−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	−7.71	−5.14	−3.95
	76.13	Scratch-E	−5.21	−2.82	−1.71
		Scratch-B	−**4.56**	−**2.23**	−**1.01**

Table 2: Results (accuracy) on ImageNet 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-GAP, −4.93 is relative to 71.03 on the left, which is the reported accuracy of the unpruned large model VGG-16 in the original paper; −4.66 is relative to 71.51 on the left, which is VGG-16's accuracy in our implementation.

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 minimization 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.

Dataset	Unpruned	Strategy	Pruned Model
ImageNet	VGG-16		VGG-16-2x
	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**

Table 3: Results (accuracy) for Regression based Feature Reconstruction (He et al., 2017b). Pruned models such as "VGG-16-2x" are defined in He et al. (2017b). Similar to Table 2, we compare relative accuracy drop from unpruned large 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 L1-sparsity on channel-wise scaling factors 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 model, 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**	-

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. For two models Scratch-E is better, but on ResNet-32 the pruned model is better. On average Scratch-E still outperforms pruned models.

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

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. Here we show that in most cases, the pruned sparse model trained from scratch could match the accuracy of the fine-tuned model as well. 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, even for non-structured pruning methods, both Scratch-E and Scratch-B are still able to match the performance of the fine-tuned models in most cases (with Scratch-B being slightly better).

4.3 Transfer Learning to Detection Task

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 L1-norm based pruning method (Li et al., 2017) on the PASCAL VOC object detection task, using the Faster-RCNN framework (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 nota-

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	ResNet-50	76.15	30%	**76.06**	74.77	75.70

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

tion, here Scratch-E 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
PASCAL VOC 07	ResNet-34	71.69	ResNet34-A	71.47	70.99	**71.64**
			ResNet34-B	70.84	69.62	**71.68**

Table 7: Results 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 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 L1-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, which trains the small models from scratch starting even earlier at the classification stage, is able to 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.

Parameter Efficiency of the Target Architectures. In Figure 3(left), we compare the parameter effciency 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× less parameters than uniformly pruning architectures. For non-structured weight pruning (Han et al., 2015), we conducted a similar experiment shown in Figure 3(middle). Here we uniformly sparsify all individual weights at a fixed probability, and the architectures obtained this way are much less efficient than the pruned

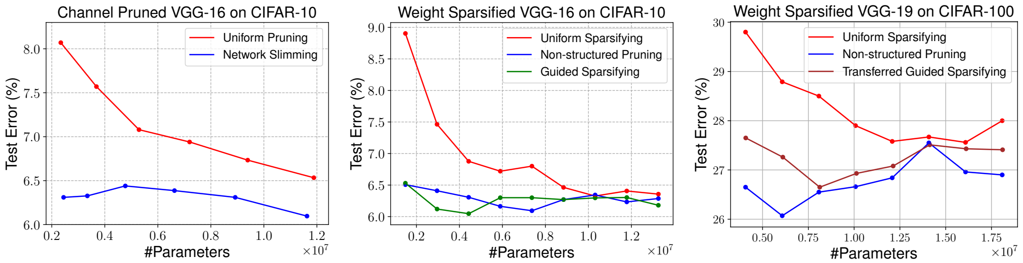

Figure 3: Pruned architectures obtained by different approaches, all trained from scratch, averaged over 5 runs. *Left:* Architectures obtained by a channel pruning method (Network Slimming (Liu et al., 2017)) has better parameter efficiency than uniformly pruning channels in all layers. *Middle:* Guided sparsifying can achieve same parameter efficiency as pruned architectures using (Han et al., 2015), which is better than uniform sparsifying. *Right:* Sparsity patterns could be transferred to another architecture and dataset to help achieve better parameter efficiency.

architectures. We also found the pruned architectures exhibit very consistent patterns in different runs. 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, we analyzed the sparsity patterns induced by non-structured pruning (Han et al., 2015), and the results are illustrated in Figure 4(middle), where each 3×3 square corresponds to a 3×3 kernel and the color indicates the probability for that weight to be kept (darker means higher). We apply these sparsity patterns to construct a new set of sparse models, whose parameter efficiency curve is shown in Figure 3(middle) as "guided sparsity". This set of sparsity patterns is obtained on VGG-16 trained CIFAR-10 dataset using different pruning ratios. Interestingly, these sparsity patterns can generalize to a different architecture on a different dataset, namely VGG-19 on CIFAR-100 as shown in Figure 3(right), where the "Transferred Guided Sparsity" curve is the parameter efficiency we obtained after transferring such sparsity patterns. It is slightly worse than architectures directly pruned on VGG-19 and CIFAR-100 by non-structured pruning, but is significantly better than uniform sparsifying.

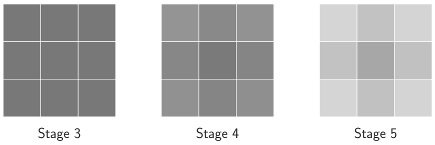

Figure 4: The average sparsity pattern of 3×3 convolutional kernels in some stages of layers in a pruned VGG-16. Darker color means higher probability of weight being kept.

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 when the goal includes 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 are useful, in particular when a pre-trained large model is given and when little or no training budget is available, or if 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. Pruning and fine-tuning is much faster than training from scratch in these cases.

In summary, our experiments have shown that training a small pruned model from scratch can almost always achieve the same or higher level of accuracy than a model fine-tuned from inherited weights. This changed our understanding of over-parameterization, and the effectiveness of inheriting weights. We further demonstrated the value of automatic pruning algorithms could be regarded as searching efficient architectures.

REFERENCES

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

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.

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.

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

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

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.

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

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.

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.

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. 2017.

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

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.

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

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. 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.