Rethinking the Value of Network Pruning

Zhuang Liu[1]*, **Mingjie Sun**[2]*†, **Tinghui Zhou**[1], **Gao Huang**[2], **Trevor Darrell**[1]
[1]University of California, Berkeley [2]Tsinghua University
`{zhuangl, tinghuiz, trevor}@eecs.berkeley.edu`
`{sunmj15, gaohuang.thu}@gmail.com`

Abstract

Network pruning is widely used for reducing the heavy computational cost of deep models. A typical pruning algorithm is a three-stage pipeline, i.e., training (a large model), pruning and fine-tuning. In this work, we make a rather surprising observation: fine-tuning a pruned model only gives comparable or even worse performance than training that model with randomly initialized weights. 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 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

Network pruning is a commonly used approach for obtaining an efficient neural network. A typical procedure of network pruning consists of three stages: 1) train a large, over-parameterized model, 2) prune the unimportant weights according to a certain criterion, and 3) fine-tune the pruned model to regain accuracy. Generally, there are two common beliefs behind this pruning procedure. First, it is believed that training a large network first is important [1] and the three-stage pipeline can outperform directly training the small model from scratch. Second, both the architectures and the weights of the pruned model are believed to be essential for the final efficient model, which is why most existing pruning methods choose to fine-tune the pruned model instead of training it from scratch. Also because of this, how to select the set of important weights is a very active research topic [1, 2, 3, 4].

In this work, we show that both of the beliefs mentioned above are not necessarily true. We make a surprising observation that directly training the target pruned model from random initialization can achieve the same or better performance as the model obtained from the three-stage pipeline. This means that, for pruning methods with a predefined target architecture (Figure 1), starting with a large model is not necessary and one could instead directly train the target model from scratch. For pruning algorithms with automatically discovered target architectures, what brings the efficiency benefit is the obtained architecture, instead of the inherited weights. Our results advocate a rethinking of existing network pruning algorithms: the preserved "important" weights from the large model are not necessary for obtaining an efficient final model; instead, the value of

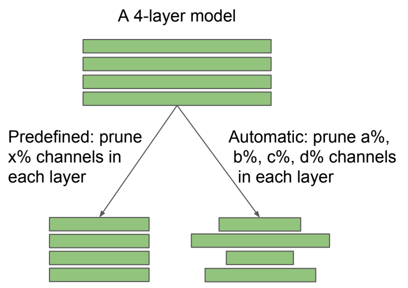

A 4-layer model
Predefined: prune x% channels in each layer
Automatic: prune a%, b%, c%, d% channels in each layer

Figure 1: 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.

*Equal contribution.
†Work done while visiting UC Berkeley.

automatic network pruning methods may lie in identifying
efficient architectures and performing implicit architecture search.

2 Methodology

Target Pruned Architectures. We divide pruning methods by whether the target pruned architecture is determined by either human or the pruning algorithm (see Figure 1). An example of designing predefined target architecture is specifying how many channels to prune in each layer. In contrast, when the target architecture is automatically discovered, the pruning algorithm determines how many channels to prune in each layer, by comparing the importance of channels across layers.

Training from scratch. Because the pruned model requires less computation, it may be unfair to train the pruned model for the same number of epochs as 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 (measured by FLOPs). We use standard training hyper-parameters and data-augmentation schemes. The optimization method used is SGD with Nesterov momentum.

3 Experiments

In this section we present our experimental results comparing training pruned models from scratch with fine-tuning. For method with predefined architectures, we evaluate $L1$-norm based channel pruning method [3]. For method with automatically discovered target architectures, we use Network Slimming [5]. The models, datasets and the number of epochs for fine-tuning are the same as those in the original paper. More results for other pruning methods and transfer learning can be found in Appendix B.

3.1 Predefined Target Architectures

L_1-norm based Channel Pruning [3] is one of the earliest work on channel pruning for convolutional networks. In each layer, a certain percentage of channels with smaller L_1-norm of its filter weights will be pruned. Table 1 shows our results. The Pruned Model column shows the list of predefined target models. We observe that in each row, scratch-trained models achieve at least the same level of accuracy as fine-tuned models, with Scratch-B slightly higher than Scratch-E in most cases. On ImageNet, both Scratch-B models are better than the fine-tuned ones.

Dataset	Model	Unpruned	Pruned Model	Fine-tuned	Scratch-E	Scratch-B
CIFAR-10	VGG-16	93.63 (±0.16)	VGG-16-A	93.41 (±0.12)	93.62 (±0.11)	**93.78** (±0.15)
	ResNet-56	93.14 (±0.12)	ResNet-56-A	92.97 (±0.17)	92.96 (±0.26)	**93.09** (±0.14)
			ResNet-56-B	92.67 (±0.14)	92.54 (±0.19)	**93.05** (±0.18)
	ResNet-110	93.14 (±0.24)	ResNet-110-A	93.14 (±0.16)	**93.25** (±0.29)	93.22 (±0.22)
			ResNet-110-B	92.69 (±0.09)	92.89 (±0.43)	**93.60** (±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 channel pruning [3]. "Pruned Model" is the model pruned from the large model. Names such as 'VGG-16-A' refer to pruned models whose configurations are defined in [3].

3.2 Automatically Discovered Target Architectures

Network Slimming [5] imposes L_1-sparsity regularization on channel-wise scaling factors from Batch Normalization layers [6] 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 2, for all networks, the small models trained from scratch can reach the same accuracy as the fine-tuned models, where Scratch-B consistently outperforms the fine-tuned models. Morever, when we extend the standard training of large model, the above observation still holds.

4 Network Pruning as Architecture Search

In this section, we demonstrate that the value of automatic network pruning methods actually lies in searching efficient architectures. We use Network Slimming [5] as an example automatic method.

Parameter Efficiency of Pruned Architectures. In Figure 2(left), we compare the parameter efficiency of architectures obtained by Network Slimming with a naive predefined pruning strategy

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 2: Results (accuracy) for Network Slimming [5]. "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.

that uniformly prunes the same percentage of channels in all layer. All architectures are trained from random initialization for the same number of epochs without sparsity regularization. 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.

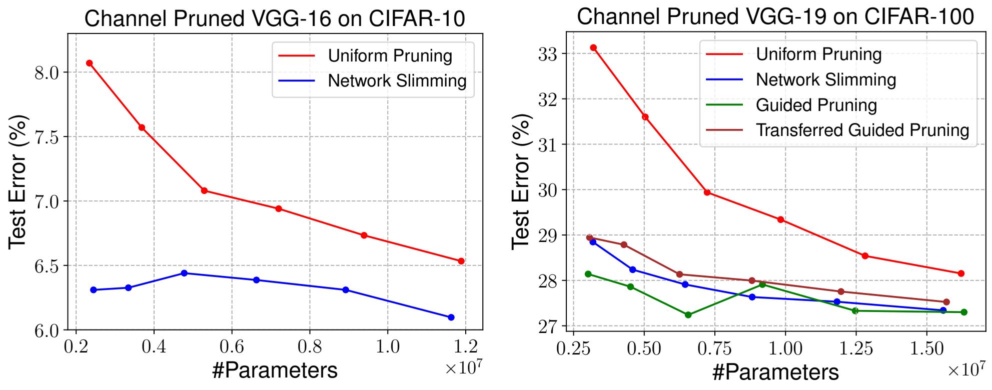


Figure 2: Pruned architectures obtained by Network Slimming [5] and different other design strategies, all trained from scratch. *Left*: Results for VGG-16 on CIFAR-10 , *Right:* Results for VGG-19 on CIFAR-100.

Generalizable Design Principles from Pruned Architectures. Given that the automatically discovered architectures tend to be parameter efficient, here we show a generalizable principle on designing better architectures. We show two design strategies: 1) "Guided Pruning": 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. 2) "Transferred Guided Pruning": use these patterns from a different architecture on a different dataset. Figure 2(right) shows our results. Here for "Transferred Guided Pruning", we use the patterns obtained by a pruned VGG-16 on CIFAR-10, to design the network for VGG-19 on CIFAR-100. We observe that both "Guided Pruning" (green) and "Transferred Guided Pruning" (brown) can both perform on par architectures directly pruned on the VGG-19 and CIFAR-100 (blue), and are significantly better than uniform pruning (red). This means we could distill generalizable design principles from pruned architectures.

5 Discussion and Conclusion

In practice, for methods with predefined target architectures, training from scratch is more computationally efficient and saves us from implementing the pruning procedure and tuning the additional hyper-parameters. Still, pruning methods are useful when a pretrained large model is already given, in this case fine-tuning is much faster. Also, obtaining multiple models of different sizes can be done quickly by pruning from a large model 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 previous belief that over-parameterization is necessary for obtaining a final efficient model, and the understanding on 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

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

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

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

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

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

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

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

[8] 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.

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

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

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

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

[13] 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.

[14] 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.

[15] 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*, pages 1026–1034, 2015.

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

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

[18] 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.

[19] 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.

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

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

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

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

[24] 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.

[25] 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*, 2018.

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

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

Appendix

A Experiment Setups

Here we provide additional details on our experiment setups.

Datasets, Network Architectures and Pruning Methods. In the network pruning literature, CIFAR-10, CIFAR-100 [7], and ImageNet [8] datasets are the de-facto benchmarks, while VGG [9], ResNet [10] and DenseNet [11] are the common network architectures. We evaluate three pruning methods with predefined target architectures, [1, 3, 4], and three which automatically discover target models, [2, 5, 12]. 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 [2], 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.

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. [5] and [12], we adopt the original implementation; 2) Otherwise, for simpler pruning methods, e.g., [2, 3], we re-implement the three-stage pruning procedure and achieve similar results to the original paper; 3) For the remaining two methods [1, 4], 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[3]. In this case, to accommodate the effects of different frameworks and training setups, we report the relative accuracy drop from the unpruned large model.

For random weight initialization, we adopt the scheme proposed in [15]. 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 [3, 4]. The code to reproduce the results and trained ImageNet models are publicly available in the following anonymous link: `https://drive.google.com/open?id=1HB_1FphsWtbuMAdgHbODSLdAMnj3B6TM`.

B Additional Experiment Results

The results of two pruning methods are already shown in the extended abstract, here in appendix we provide results of the remaining four pruning methods. We also include an experiment on transfer learning from image classification to object detection.

B.1 Predefined Target Architectures

ThiNet [1] greedily prunes the channel that has the smallest effect on the next layer's activation values. As shown in Table 3, 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.

Regression based Feature Reconstruction [4] prunes channels by minimizing the feature map reconstruction error of the next layer. Different from ThiNet [1], this optimization problem is solved by LASSO regression. Results are shown in Table 4. 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.

[3]This could be due to the difference in the deep learning frameworks: we used Pytorch [13] while the original papers used Caffe [14]

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 3: Results (accuracy) for ThiNet [1]. Names such as "VGG-GAP" and "ResNet50-30%" are pruned models whose configurations are defined in [1]. 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 4: Results (accuracy) for Regression based Feature Reconstruction [4]. Pruned models such as "VGG-16-5x" are defined in [4]. Similar to Table 3, we compare relative accuracy drop from unpruned large models.

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.

B.2 Automatically Discovered Target Architectures

Sparse Structure Selection [12] also imposes sparsity regularization on the scaling factors during training 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 [16]. We examine residual blocks pruning, where ResNet-50 are pruned to be ResNet-41, ResNet-32 and ResNet-26. Table 7 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 [12]. 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 [2] 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 [15]. 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.

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 [2]. "Prune Ratio" denotes the percentage of parameters pruned in the set of all convolutional weights.

Effects of Sparsity Regularization. Some methods [5, 12] use sparsity regularization during the training of large, over-parameterized models, to smooth the following pruning process, and in fine-tuning, no such sparsity is imposed. In all of our experiments presented above, no sparsity is used for training the pruned models from scratch. Here we analyze the effects of using sparsity regularization (or not) for Network Slimming [5]. Table 7 shows the results when all training procedures are with sparsity regularization. We can see that Scratch-B are still able to be on par with the fine-tuned models. Table 8 shows the results when we do not use sparsity regularization in any training procedures, including large model training. We can see that when no sparsity is induced, Scratch-B can also consistently achieve comparable results with the fine-tuned models.

Dataset	Model	Unpruned	Prune Ratio	Fine-tuned	Scratch-E	Scratch-B
CIFAR-10	VGG-19	93.59 (±0.19)	70%	93.51 (±0.23)	93.36 (±0.23)	93.62 (±0.10)
	DenseNet-40	94.00 (±0.17)	40%	94.01 (±0.23)	93.76 (±0.06)	93.91 (±0.23)
			60%	93.86 (±0.12)	93.62 (±0.22)	93.93 (±0.20)

Table 7: Results for training with both with sparsity for VGG-19 and DenseNet-40 on CIFAR-10.

Dataset	Model	Unpruned	Prune Ratio	Fine-tuned	Scratch-E	Scratch-B
CIFAR-10	DenseNet-40	94.10 (±0.12)	40%	94.00 (±0.12)	93.79 (±0.19)	94.07 (±0.14)
			60%	93.77 (±0.12)	93.73 (±0.20)	94.01 (±0.11)

Table 8: Results for training without sparsity for DenseNet-40 on CIFAR-10. We found a whole layer in VGG-19 is pruned when we try to prue 70% channels, thus the network gets destroyed and we do not have the results here.

B.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 [3] on the PASCAL VOC object detection task, using Faster-RCNN [17].

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 9. 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 [18].

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 9: Results (mAP) for pruning on detection task. The pruned models are chosen from **(author?)** [3]. 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.

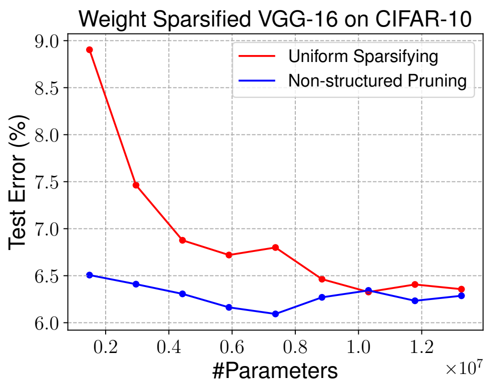
For this experiment, we adopt the code and default hyper-parameters from [19], and use PASCAL VOC 07 trainval/test set as our training/test set. For backbone networks, we evaluate ResNet-34-A and ResNet-34-B from the L_1-norm based channel pruning [3], which are pruned from ResNet-34. Table 9 shows our 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 9 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.

C Network Pruning as Architecture Search

Here we provide results that complements Section 4 for non-structured weight pruning [2]. The networks and datasets used are the same as those used in Section 4. We also discuss the relation with conventional architecture search methods at the end.

Figure 3(left) shows our results on parameter efficiency of architectures obtained by non-structured pruning. Here “Uniform Sparsifying” means uniformly sparsifying individual weights in the network at a fixed probability. It can be seen that pruned architectures are more parameter-efficient than uniform sparsified architectures.



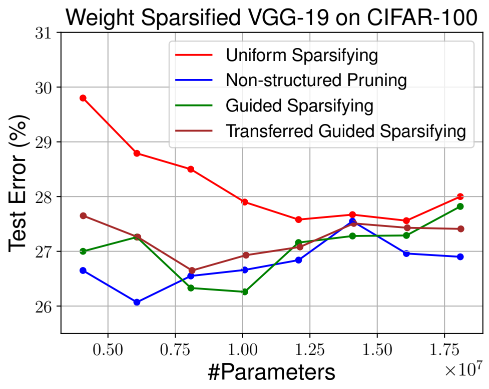


Figure 3: Pruned architectures obtained by non-structured pruning [2] and different design strategies. *Left:* Results for VGG-16 on CIFAR-10, *Right:* Results for VGG-19 on CIFAR-100.

Figure 3(right) shows our results for architectures obtained by other different design strategies. For "Guided Sparsifying" and "Transferred Guided Sparsifying", we use the average sparsity patterns of 3×3 kernel in each layer stage to design new structures. Similar to Network Slimming, both "Guided Sparsifying" (green) and "Transferred Guided Sparsifying" (brown) are significantly more parameter-efficient than uniform sparsifying (red).

Comparison with Traditional Architecture Search Methods. Conventional techniques for network architecture search include reinforcement learning [20, 21] and evolutionary algorithms [22, 23]. 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, [24] uses a similar pruning technique to Network Slimming [5] to automate the design of network architectures; [25] prune channels using reinforcement learning and automatically compresses the architecture. On the other hand, in the network architecture search literature, sharing/inheriting trained parameters [26, 27] 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.

D Weight Distribution

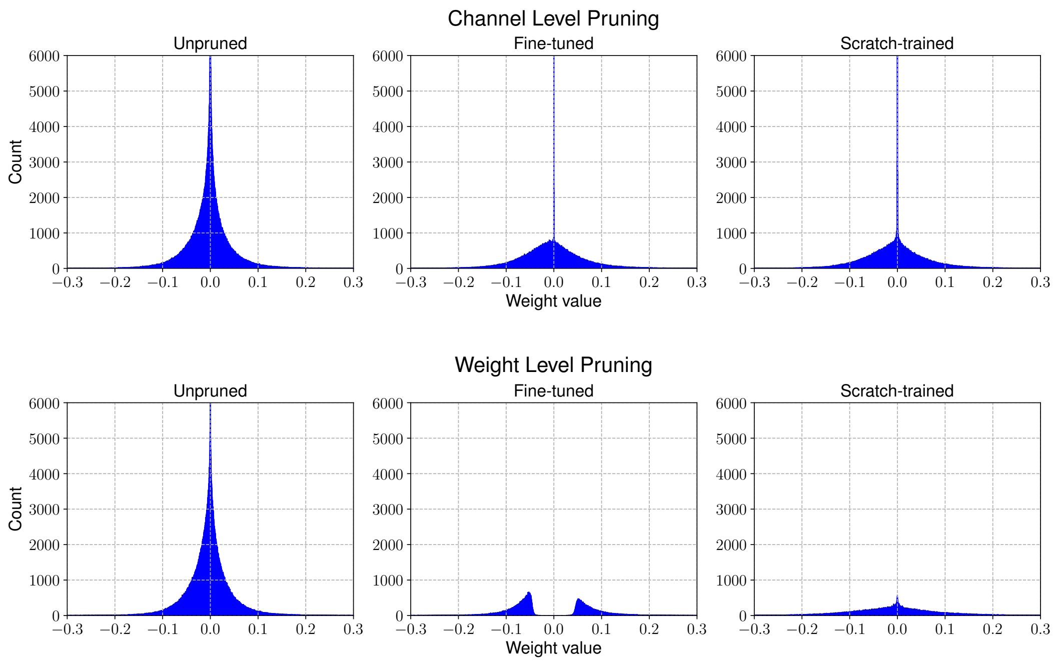


Figure 4: Weight distribution of convolutional layers for different pruning methods. We use DenseNet-40 and CIFAR-10 dataset. We compare the weight distribution of unpruned models, fine-tuned models and scratch-trained models. *Top*: Results for Network Slimming [5]. *Bottom*: Results for non-structured weight-level pruning [2].

We show the weight distribution of unpruned large models, fine-tuned pruned models and scratch-trained pruned models, for two pruning methods: Network Slimming [5] and non-structured weight level pruning [2]. We choose DenseNet-40 and CIFAR-10 for visualization and compare the weight distribution of unpruned models, fine-tuned models and scratch-trained models. For Network Slimming, the prune ratio is chosen to be 60%. For non-structured weight level pruning, the prune ratio is chosen to be 80%. Figure 4 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 search for automatic pruning methods.