RETHINKING THE VALUE OF NETWORK PRUNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Network pruning is popular in reducing the heavy computation requirement of deep convolutional networks. The typical procedure involves training an large model, pruning some weights based on certain criteria, and finally fine-tuning the smaller pruned model to regain accuracy. In this work we show a rather surprising fact that, for structured pruning (e.g., pruning channels, layers), the small pruned model can be trained from random initialization to achieve the same level of accuracy, given the same or even less total training budget. This phenomenon holds for various structured pruning methods, on multiple network architectures and datasets. The results suggest structured over-parametrization of deep networks is not necessary during training. Conversely we found for non-structured weight-level pruning, the smaller sparse model cannot recover the accuracy if trained from scratch. We also discuss some implications for practice and research on network pruning.

1 INTRODUCTION

The over-parameterization of current neural networks is widely recognized (Denton et al., 2014; Ba & Caruana, 2014), and methods for pruning network weights have been increasingly popular for network compression and accleration, which typically use a "training-pruning-finetuning" procedure to get a smaller but still accurate model. However, an assumption that is implicitly made in these systems is that the network's over-parameterization is necessary or helpful *during training*, and should be removed for model compression only after training, rather than before training. In other words, inheriting part of the weights (which are usually considered "important" weights) in a trained large-capacity model, is better than training a small model from random initialization. These weights trained in the large model, are conceived to provide a good initialization for the pruned model and should be somehow preserved, which is the reason why the fine-tuning process usually uses a very small learning rate (Han et al., 2016; Li et al., 2017).

In this work, we show that **training the small model from random initialization can achieve the same level of accuracy as the model fine-tuned from pruning a large model, on all channel pruning algorithms we evaluated on**. This phenomenon holds for multiple network architectures and datasets, including a case where we transfer the weights from ImageNet classification to object detection. Moreover, this process can be done with equal or less training budget, with a default hyper-parameter setting. Therefore, channel-level over-parameterization during training might not be necessary as previously perceived. Despite it is often believed that not all computation in a trained model are of equal importance, the weights selected to be kept by the pruning algorithm may not be inherently more "important", as the accuracy could be recovered from random initialization. In Section 4, we further this support this point by elaborating that **some pruning methods could be treated as searching architecture rather than selecting important weights**.

As a simple alternative to obtain an efficient model, training from scratch has several benefits over network pruning methods: first, since the model is smaller, we can train the model using less GPU memory and possibly faster than training the original large unpruned model; second, there is no need to implement the pruning criterion and procedure, which could be cumbersome (e.g., pruning and fine-tuning layer by layer (Luo et al., 2017)) and sometimes even need to be customized for different network architectures (Li et al., 2017; Liu et al., 2017); third, we can avoid tuning some tricky hyper-parameters involved, e.g., the sparsity regularization coefficients, the fine-tuning learning rate. Practically, if we need an efficient model for deployment, training it from scratch is a very strong baseline to consider. We'll discuss some cases where pruning methods are more applicable later.

Interestingly, for weight-level pruning (methods that prune individual weights rather than channels), we found that training the pruned sparse model from scratch cannot generally achieve the same accuracy. This disparity suggests the redundancy inside a convolutional kernel might be fundamentally different than that among convolutional kernels. We'll provide further analysis in Section 5.

The rest of the paper is organized as follows: in Section 2, we briefly introduce background on network pruning; in Section 3, we experiment on four recent channel pruning methods to show our main results; in Section 4, we provide a more detailed analysis of this effect; in Section 5, we show that our conclusion does not generally hold for weight level pruning; finally in Section 6 we discuss the implications and draw the conclusion.

2 BACKGROUND

With the success of deep convolutional networks in a variety of vision tasks (Girshick et al.; Long et al., 2015; He et al., 2016; 2017a), their computation resource requirements have been increasing. 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-constraint settings. Those large models can hardly be affordable to store, and run in real time on embedded systems. Other than network pruning, methods for lowering the computation burden include 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), etc.

Network weight pruning 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. Han et al. (2015) proposes to prune network weights with small magnitude. Srinivas & Babu (2015) proposes a data-free algorithm to remove redundant neurons iteratively. However, these methods prune individual weights and make the weight matrix sparse, which only translates to compression and speedup at the presence of special hardware/libraries.

In contrast, structured pruning methods usually require no dedicated libraries to achieve the benefit, since they prune on a coarser granularity and the original convolution structure is still preserved. Among structured pruning methods, channel pruning is of the dominating popularity, since it is the most fine-grained pruning 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) imposes a sparsity constraint on channel-wise scaling factors during training, which act as agents for channel selection. He et al. (2017b) and Luo et al. (2017) solve the optimization of minimizing next layer's feature reconstruction error to determine which channels to be kept. 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.

There are some works investigating simple alternative approaches to those pruning algorithms. Anwar & Sung (2016) proposes to randomly prune channels. Mittal et al. (2018) further shows random pruning can perform on par with a variety of other pruning criteria, demonstrating the plasticity of network models. Changpinyo et al. (2017) train channel-wise sparse networks from scratch by randomly deactivating connections before training. Zhu & Gupta (2018) shows that training a small-dense model cannot achieve the same accuracy as large-sparse pruned model with identical memory footprint. While Zhu & Gupta (2018) experiment on a weight-level pruning strategy, our main conclusion is for channel pruning. We further demonstrate that training the large-sparse model from scratch cannot achieve the same accuracy either.

Dataset	Model	Baseline	Pruned	Scratch
CIFAR-10	VGG-16-A	93.62 (±0.16)	−0.22 (±0.12)	**−0.01** (±0.11)
	ResNet-56-A	93.14 (±0.12)	**−0.17** (±0.17)	−0.18 (±0.26)
	ResNet-56-B		**−0.47** (±0.14)	−0.60 (±0.19)
	ResNet-110-A	93.14 (±0.24)	+0.00 (±0.16)	**+0.11** (±0.29)
	ResNet-110-B		−0.45 (±0.09)	**−0.25** (±0.43)
ImageNet	ResNet-34-A	73.31	−0.75	**−0.54**
	ResNet-34-B		−1.02	**−0.76**

Table 1: Results for Pruning filters (Li et al., 2017)

Dataset	Model	Baseline	Prune Ratio	Pruned	Scratch	Longer Scratch +fine-tune
CIFAR-10	VGG-19		70%	93.80 (±0.11)	93.56 (±0.13)	93.72 (±0.14)
	ResNet-164		40%	95.07 (±0.12)		
	DenseNet-40		40%	94.41 (±0.15)	94.06 (±0.18)	
			60%	94.24 (±0.13)	93.79 (±0.14)	
CIFAR-100	VGG-19		50%	73.41 (±0.32)	73.22 (±0.34)	73.38 (±0.22)
	ResNet-164					
	DenseNet-40		40%	74.12 (±0.22)	74.09 (±0.20)	
			60%	73.81 (±0.41)	73.24 (±0.27)	
ImageNet	VGG-11	70.84	50%	−2.22	**−0.84**	-

Table 2: Results for Network Slimming (Liu et al., 2017)

3 TRAINING FROM SCRATCH

In this section, we show our results on several channel level pruning methods. We show that under the training strategy which guarantees equal or less training budget, training from scratch can match the accuracy of pruning plus fine-tuning.

Previous work has shown that training from scratch can't achieve the accuracy of prune plus finetuning. They usually use the same number of epochs to train the small model from scratch. However, we argue that the conventional training schedule may not be fair for training from scratch. The pruned model has less parameters and flops than the base model. So the principle for training the small model from scratch should be that the small model should be trained with equal or less training budget. In this paper, we use the FLOPs of the neural network as the measurement of training budget. In other words, we should train the small model for more epochs propotional to the FLOPs reduction of the small model compared to the base model.

For all channel pruning methods we verify in this paper, if the pruned model is not released to public, we implement the pruning method. For deep learning framework, we use Pytorch in all our experiments. Therefore, if the released model is in Caffe, we use the relative difference in accuracy between the pruned model and the base model for comparison. For experiments on CIFAR-10 and CIFAR-100 datasets, we average over 5 runs and report the results.

3.1 PRUNING BASED ON L1-NORM (LI ET AL., 2017)

This is the first work on channel-level pruning for convolutional networks. It uses the l1-norm of the filter to decide which filters to prune from the convolutional weight. The argument is that if the l1-norm of a filter is small then on average the weights in this filter will be small, therefore leading to small activation. The authors use sensitivity analysis of single layer pruning to design the small model.

Network	Baseline	Strategy	Pruned
VGG-16	71.03	fine-tune	−2.67
	71.51	scratch	−3.46
		longer scratch	**−0.51**
ResNet-50	75.51	fine-tune	−3.25
	76.13	scratch	**−1.55**
		longer scratch	-

Table 3: Results for channel pruning (He et al., 2017b)

Network	Baseline	Strategy	Model		
VGG-16	VGG-16		VGG-Conv	VGG-GAP	VGG-Tiny
	71.03	fine-tune	-	−4.93	−11.61
	71.51	scratch	−2.75	−4.66	−14.36
		longer scratch	**+0.21**	**−2.85**	**−11.58**
ResNet-50	ResNet-50		ResNet50-30%	ResNet50-50%	ResNet50-70%
	75.15	fine-tune	−7.71	−5.14	−3.95
	76.13	scratch	−5.21	−2.82	−1.71
		longer scratch	**−4.56**	**−2.23**	**−1.01***

Table 4: Imagenet result for ThiNet (Luo et al., 2017). We report the accuracy under the following data augmentation: resize shorter size to 256 and center crop to 224×224.

Table 1 shows our result. In both CIFAR-10 and ImageNet datasets, we use the standard training schedule for training the small model from scratch. We can see that with standard training, the small model is able to achieve competitive performance to the fine-tuned model.

3.2 Network Slimming (Liu et al., 2017)

The main idea of this work is that it introduces sparsity on the channel-wise scaling factor during training and then during pruning, it uses the magnitude of the channel scaling factor to prune the convolutional filters. We note that in finetuning process, the authors use the restart learning rate schedule.

Table 2 shows our result. For all networks, the small models trained from scratch are able to reach the same accuracy as the fine-tuning accuracy.

3.3 Channel Pruning (He et al., 2017b)

Channel Pruning (He et al., 2017b) is a reconstruction based method which uses the idea of minimizing the reconstruction error to select the important filters. Like Li et al. (2017), the authors use sensitivity analysis to design the small model.

Table 3 shows our result. For the fine-tuned results, we use the released models. For a fair comparison, we use the relative difference here. VGG-16 here does not include Batch Normalization, still, we are able to match the accuracy of the fine-tuned model by doubling the epochs of standard training schedule.

3.4 ThiNet (Luo et al., 2017)

ThiNet (Luo et al., 2017) is also a reconstruction based method. Different from He et al. (2017b), the authors use a uniform pruning ratio for each layer during pruning.

Table 4 summarizes our result. As we can see, for VGG and ResNet, training the small model from scratch can achieve the same or even better performance than the fine-tuned model.

Model	Pruning Task	Fine-tune	Scratch
ResNet34-A	Classification	71.47	71.64
	Detection	70.99	
ResNet34-B	Classification	70.84	71.68
	Detection	69.62	

Table 5: Results for pruning on detection task. The pruned models are chosen from Li et al. (2017).

Network	Baseline	Ratio	Pruned	Longer Scratch
VGG		30%		
		80%		
ResNet		30%		
		80%		
DenseNet-40		30%		
		80%		
DenseNet-BC-100		30%		
		80%		

Table 6: Results for weight-level pruning.

3.5 Transferability on Detection Task

Although we have shown that the small model can be trained from scratch to match the accuracy of the fine-tuned model in the original image classification task, it still remains unknown how the scratch trained model performs in the detection task. Currently the state-of-the-art object detection model relies on ImageNet pretraining. Here we show that the small model trained from scratch can perform equally well to the fine-tuned model when transferred to object detection task.

There are two possible ways to perform pruning on the task of object detection. The first way is to prune the model trained on classification task, then finetune the pruned model on the detection task. The second way is to directly prune the backbone network of the trained detection model and then finetune the pruned model.

Table 5 shows the result of pruning on PASCAL VOC 2007 and 2012. For pruning on ImageNet, we finetune the model on detection dataset for 7 epochs with initial learning rate 0.001 decreased by 10 at epoch 4. For pruning on detection, we finetune the model for 4 epochs with learning rate 0.0001. The scratch experiment is the same for both pruning methods. From the results, we can see that the model trained from scratch can match the performance of finetuned model in terms of transferability.

3.6 Non-structured weight pruning

Han et al. (2015) prunes individual weights that have small magnitudes. Here we show that for this non-structured weight-level pruning strategy, in general, the pruned model trained from scratch can't match the accuracy of the fine-tuned model.

Table 6 shows our for weight level pruning. For simplicity and direct comparison with pruning convolution channels, for the experiments, we only prune the weights in the convolutional layers. We note that in training the small model from scratch, the initialization may differ from the base model because some weights are removed. So we use the average number of input channels or output channels to compute the initializations.

4 Architecture search

Our results show that channel pruning methods are useful in that they can help us find new architectures. For Network Slimming (Liu et al., 2017), it is essentially searching efficient architecture. To verify that Network Slimming is doing architecture search, we compare it with uniform pruning method. Here uniform pruning means that each convolutional layer has a uniform pruning rate. Fig-

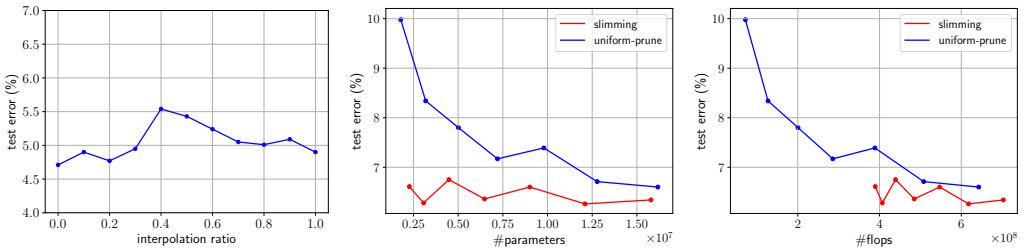

Figure 1: *Left:* Alpha experiment for ResNet164-pruned-0.4 in Network Slimming (Liu et al., 2017) *Middle and Right:* Comparison of accuracy between the VGG network searched by Network Slimming (Liu et al., 2017) and the network searched by uniform pruning with respect to both number of parameters and flops.

ure 1a shows our result, where we show the relationship between the accuracy and the number of parameters, also flops of the pruned models.

5 Analysis

5.1 Fine-tuning learning rate

Fine-tuning the pruned network is the crucial step for the pruned network to regain the accuracy. There are two strategies in finetuning the pruned network: 1. finetune with small learning rate for a few epochs; 2. finetune with restart learning rate schedule. Finetuning with restart learning rate usually takes more epochs but can help the pruned network to escape local minimum. In this section, we show by experiments that fine-tuning by restart the learning rate schedule is better than fine-tuning with low learning rate without restart. Table 7 shows our result. We choose three models on two datasets from Li et al. (2017) and compare the performance between fine-tuning with restart learning rate and small learning rate.

Dataset	Network	Scratch	Small-lr	Restart-lr
CIFAR-10	ResNet-110-A	93.34	93.48	93.82
	VGG-16-A	93.56	93.48	93.79
ImageNet	ResNet-34-B	72.55	72.25	73.12

Table 7: Results on fine-tuning learning rate

Figure 2: Fine-tuning restart experiment.

Furthermore, we show that the weight of the pruned model influences the convergence speed. We use Network Slimming (Liu et al., 2017) to show our result. We compare three ways of pruning: 1. prune the channels with the largest channel-scaling factors; 2. random prune channels; 3. prune the channels with the smallest channel-scaling factors. We keep the resulting architecture to be the same so that the only difference is the weight. Our result shows that the weights which correspond to the first case actually leads to faster convergence speed.

5.2 Interpolation between fine-tuning and training from scratch

In this section, we show that the weights of the pruned model is not necessary to training the small model from scratch. We interpolate the weights of the pruned model with random initializations. Here is our experiment design. We take the weights W of a pruned model and also random initializations R of the model. Then we interpolate two weights with different ratio α: $\alpha \cdot W + (1-\alpha) \cdot R$.

Then for each interpolated weight under a ratio, we first train the model with number of epochs $\alpha \cdot K$ where K is the number of epochs computed according the flops reduction of the pruned model. Then we finetune each model with restart learning rate schedule. We compare the final accuracy under different interpolation ratio α.

Figure 1b 1c shows our result. We can see that under all α the accuracy remains the same level. This verifies our claim that the weights of the pruned model is not necessary to training the small model from scratch.

6 Discussion and Conclusion

Based on our experiments, we discuss several possible implications and give our suggestions to future research related to network compression.

6.1 The over-parameterization in networks

Our results give us a deeper understanding of the redundancy of convolutional networks. We'll summarize our main findings here:

- The redundancy in number of channels does not provide a benefit for compressing the neural networks. However, the weight-level redundancy inside kernel is indeed helpful.
- Inheriting weights from a pruned large model may not be beneficial. It may cause the model to be trapped in a local minimum, which could be escaped by using a large learning rate initially.
- Pruning methods that automatically select target model architecture, essentially works in a similar spirit as network architecture search.

6.2 Network pruning in engineering practice

Training a smaller model from scratch is a strong baseline. As our experiments show, training the small model from scratch can almost always achieve same or higher accuracy. It can sometimes save a great amount of effort in implementation/hyper-parameter tuning, require cheaper computation, and should sometimes be considered first.

Use cases of channel pruning methods. Despite of training from scratch has some advantages, we list several cases where channel pruning methods are likely to be useful.

- The goal includes finding an efficient architecture. This can be done using automatic approaches like Molchanov et al. (2016) and Liu et al. (2017) or by informed sensitivity analysis (Li et al., 2017; He et al., 2017b).
- A pre-trained large model is given, especially when little or no training budget is available. Pruning and fine-tuning could be much faster than training another model from scratch in this case, if a small learning rate is used, as shown in Section 4.1.
- Need to obtain multiple models of different sizes quickly. In this situation one can train a large model and then prune it by different ratios. The models predictions will also be correlated with each other, which may be desired in some cases.

6.3

Ensure a fair comparison with baselines. Evaluating the baseline of training from scratch is not a new proposal, but it could easily be carried in a unfair setting unintentionally. In this case, training the small model for the same number of epochs as the large one is arguably not fair, as it requires substantially less training budget. As another example, enabling data augmentation, might not be necessary during fine-tuning if already deployed when training the large model, but as we found, is necessary for training from scratch as a baseline.

Identify the source of gain. The fact that we can often obtain high compression rate while maintaining the same level of accuracy, is often accredited to the soundness of the pruning criteria.

Our experiments, provides evidence that it might come from the inherent robustness against shrinking of neural networks. The large models are usually designed to optimize the accuracy without much consideration on efficiency, thus they are possibly on the "diminishing return" phase of the performance-size curve, where shrinking size by half while losing 1% accuracy should not have been a surprise. Other methods for compressing networks, might also need to pay attention to this issue.

In summary, through experiments we have found training the pruned model from scratch is a strong baseline versus typical channel pruning methods. Based on this observation, we made connections between network pruning methods and network architecture search. We also found the phenomenon does not hold for weight-level pruning. Finally we discussed some conclusions from our experiments.

ACKNOWLEDGMENTS

Use unnumbered third level headings for the acknowledgments. All acknowledgments, including those to funding agencies, go at the end of the paper.

REFERENCES

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

Sajid Anwar and Wonyong Sung. Coarse pruning of convolutional neural networks with random masks. 2016.

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

Soravit Changpinyo, Mark Sandler, and Andrey Zhmoginov. The power of sparsity in convolutional neural networks. *arXiv preprint arXiv:1702.06257*, 2017.

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.

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*.

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. 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.

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.

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

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

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.