Practical Network Acceleration with Tiny Sets

Guo-Hua Wang Jianxin Wu
National Key Laboratory for Novel Software Technology
Nanjing University, Nanjing, China
wangguohua@lamda.nju.edu.cn, wujx2001@nju.edu.cn

Abstract

Network compression is effective in accelerating the inference of deep neural networks, but often requires finetuning with all *the training data to recover from the accuracy loss. It is impractical in some applications, however, due to data privacy issues or constraints in compression time budget. To deal with the above issues, we propose a method named* PRACTISE *to accelerate the network with tiny sets of training images. By considering both the pruned part and the unpruned part of a compressed model,* PRACTISE *alleviates layer-wise error accumulation, which is the main drawback of previous methods. Furthermore, existing methods are confined to few compression schemes, have limited speedup in terms of latency, and are unstable. In contrast,* PRACTISE *is stable, fast to train, versatile to handle various compression schemes, and achieves low latency. We also propose that dropping entire blocks is a better way than existing compression schemes when only tiny sets of training data are available. Extensive experiments demonstrate that* PRACTISE *achieves much higher accuracy and more stable models than state-of-the-art methods.*

1. Introduction

Convolutional neural networks (CNNs) have achieved remarkable success, but suffer from high computational costs. To accelerate the networks, many network compression methods have been proposed, such as network pruning [11, 14, 16], network decoupling [4, 13] and network quantization [2, 5].

After compression, however, a network's accuracy usually drops by a large margin, hence finetuning with *all the training data* to recover the performance is needed. However, to preserve privacy and/or to achieve fast deployment, only scarce training data may be available in many scenarios. Figure 1 illustrates these real-world scenarios. A customer often asks the algorithmic provider to accelerate their CNN models, but due to privacy concerns, the whole training data cannot be accessed. Only the raw model and a few

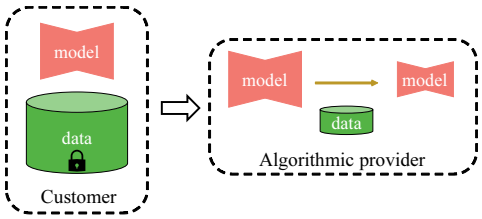

Figure 1. Illustration of network acceleration with a tiny training set in some real-world scenarios.

training examples are presented to the algorithmic provider. In some cases, not a single data point is to be provided. The algorithmic engineers need to collect some out-of-domain training data by themselves. Hence, to learn or tune a deep learning model with only very few data (i.e., a tiny training set) is emerging as a critical problem to be solved.

However, the pruned network will overfit easily if it is finetuned by a tiny set of images. In order to solve this problem, recent works [1, 12] propose to transfer knowledge from the raw network to the pruned one layer-by-layer. However, they ignore the pruned part and require the remaining part to produce feature maps consistent with that of the raw model in the same layer. Simply ignoring the pruned part may *accumulate layer-wisely errors* [3] and deteriorate latter layers, and result in poor performance. Instead, we consider the pruned and remaining parts together and use the next layer's feature to guide the training for pervious layers, which alleviates the error accumulation impact and achieves better performance than previous methods.

Another issue of existing methods is they rely on unstructured pruning, filter-level pruning or network decoupling. These compression schemes *cannot get a high acceleration ratio* on real computing devices (e.g., *both* GPU and CPU) and are thus not practical for deployment. Instead, we propose a versatile method, which can prune with different schemes, including filter-level pruning, *block-level* pruning and network decoupling. In terms of both larger latency (wall-clock timing) reduction and smaller accuracy drop, we are the first to argue that *dropping blocks is signif-*

icantly better than other compressions schemes when only scarce training data are available.

In this paper, we propose PRACTISE, namely Practical network acceleration with tiny sets of images, to effectively train a pruned network with scarce data. The advantages of PRACTISE can be summarized by four words: accurate, versatile, fast and stable.

• **Accurate.** Compared with existing methods, PRACTISE achieves higher accuracy with the same amount of training data. Note that PRACTISE can surpass previous works even *without using the image labels*. The pruned network trained by PRACTISE can be further improved by finetuning using these labels. PRACTISE can even work with out-of-domain training data.

• **Versatile.** Unlike previous methods with only limited applicability, *most CNN architectures* can be pruned and trained by PRACTISE. In this paper, we will apply PRACTISE on ResNet-34, ResNet-50, VGG-16 and MobileNetV2. Other networks can be dealt with in a similar manner.

• **Fast.** The meaning of fast contains two aspects: *inference latency and compression speed*. We are the first to advocate block-level pruning [24] when only tiny sets are available, which directly drops blocks of layers in a network. Dropping blocks can both accelerate the network's inference and achieve higher accuracy compared with other pruning schemes. When considering the compression process, dropping blocks is very fast, because the dropped blocks are predefined and do not need any extra computations to determine the compressed parts (such as which channels to be thrown away). PRACTISE trains the pruning network fast: It took less than 10 minutes when the amount of training data was less than 1000.

• **Stable.** When training the network with scarce data, stability is very important. We expect the pruned models achieve similar accuracy when trained with *different* training subsets. PRACTISE reduces the variance of accuracy and is significantly more stable than previous methods.

2. Related Works

Network compression aims at removing redundant computations in a network. Structured pruning is more preferred than non-structured pruning, because it reduces latency on general GPUs and CPUs. Filter-level pruning is a typical structure pruning approach, where filters in convolutional layers can be chosen to be removed by different criteria, including the ℓ_1-norm [11], the scales of coefficients in batch-normalization (BN) layer [14], the reconstruction error in the next layer [16], etc. When dealing with residual connections, CURL [15] prunes channels both inside and outside the residual connection. DBP [24] directly drops entire blocks to get a higher acceleration ratio. However, these methods rely on finetuning with a large-scale training set to recover the accuracy loss, or need a sophisticated data augmentation strategy to alleviate the overfitting problem. Instead, this paper studies how to finetune the pruned network with tiny training sets.

Another approach is to decouple the computation in a network. A regular convolution layer can be decoupled into the combination of a depthwise convolution and a pointwise convolution [4], or two convolution layers with smaller kernels [13]. Note that these methods can work in the data-free setting. However, accuracy of compressed networks will drop by a large margin, and they also rely on finetuning with all the training data.

Recently, FSKD [12] and CD [1] were proposed for network compression with few-shot training samples. FSKD recovers each layer by making the un-pruned part of the student's feature maps to mimic the teacher's. It ignores the pruned part, hence will layer-wisely accumulate errors. CD reduces the layer-wisely accumulated errors by cross distillation between the raw and the pruned networks. However, many hyperparameters need to be tuned in CD, which is not practical in the few-shot setting. The proposed PRACTISE method avoids layer-wise error accumulation by distilling information in the next feature map and have no hyperparameter to tune. In addition, existing works are based on unstructured pruning, filter-level pruning and network decoupling, which lead to poor acceleration ratios on real computing devices.

Knowledge distillation (KD) was introduced in [8], which uses a large capacity network (the teacher) as extra supervision to train a student network. It can improve the generalization of the student and can alleviate overfitting, hence is widely used in finetuning compressed networks. Recently, [18, 23] argue that mimicking features in the network is a better way to train the student. In this paper, we find that it is beneficial to apply KD in preventing overfitting caused by the tiny training sets.

3. The Proposed Method

Now we will introduce how PRACTISE prune and train networks with only tiny training sets. The network will be trained layer by layer. First, we introduce how to train one layer with filter-level pruning in Section 3.1. Next, Section 3.2 and Section 3.3 explain how to train VGG-like networks and networks with residual connections, respectively. Finally, our block-level pruning strategy will be introduced in Section 3.4.

3.1. Train one layer

Figure 2 illustrates how to train one layer with filter-level pruning. The left, middle and right parts show the raw layer (teacher), the pruning layer (student) and the pruned layer, respectively. A layer is pruned by reducing the number of its feature map's channels (from 64 to N). To reduce the

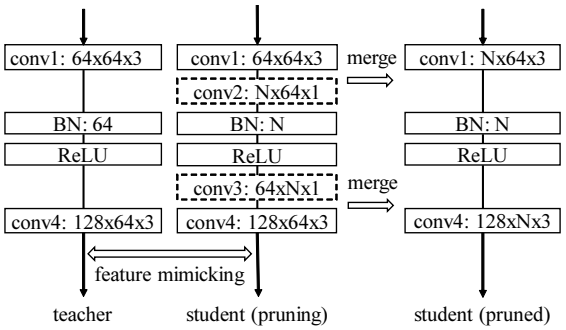

Figure 2. Illustration of training one layer. The three numbers in each conv. are the number of output and input channels, and the kernel size, respectively. The number in a BN layer represent the number of channels.

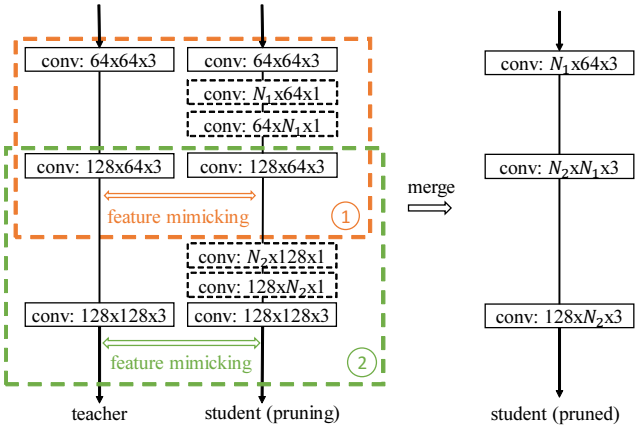

Figure 3. Illustration of training two layers. BN and ReLU layers are omitted for simplicity. This figure is best viewed in color.

size of the feature map, two convolutional layers (conv2 and conv3) with kernel size 1×1 are added before and after it, respectively. To recover the accuracy loss of the pruned model, the ideal situation is: the next feature map will be unchanged after pruning. Motivated by this, we force the student to mimic the teacher's next feature map. To this end, the mean squared error is used and the optimization problem is defined as

$$\min_{\text{conv2,BN,conv3}} \quad \|f_t - f_s\|_F^2 \,, \tag{1}$$

$$s.t. \quad f_t = \text{conv4}(\text{ReLU}(\text{BN}(\text{conv1}(x)))) \,, \tag{2}$$

$$f_s = \text{conv4}(\text{conv3}(\text{ReLU}(\text{BN}(\text{conv2}(\text{conv1}(x)))))) \,, \tag{3}$$

where x denotes the input feature map. Note that this optimization problem is nonlinear, and there is no closed-form solution. Hence, gradient descent is used to train the model. And due to the scarce data, updating all parameters will result in over-fitting easily. So, the parameters in conv1 and conv4 are fixed, while the parameters in conv2 and conv3 are trainable. The statistics of BN layers are also updated. After training, conv2 and conv3 can be merged into conv1 and conv4, respectively.

Compared with previous methods, ours has several advantages. First, our design of the entire pruning framework avoids layer-wise error accumulation by considering both the pruned part and the unpruned part of the compressed model. Previous methods [1, 12] only focus on the unpruned parts of each layer and ignore the pruned part, where they train the pruned network after pruning and cannot utilize the information in the pruned parts. This leads to layer-wise error accumulation, which is an important drawback of FSKD. PRACTISE can make use of all filters by linear combinations via the added 1×1 convolution. And inspired by ThiNet [16], we aim to prune the feature map so as not to affect the *next* feature map. If the next feature map manage to be unchanged after pruning, the compression is guaranteed not to lose any accuracy and so that avoids layer-wise error accumulation.

Second, our algorithm is stable, which will achieve similar performance with different tiny training sets. It is because PRACTISE only trains the inserted 1×1 conv., and the original convolutional layers are not updated. It is thus less prone to overfitting.

Third, our algorithm is versatile, which can deal with different pruning schemes. Figure 2 shows structured filter-level pruning. But our algorithm can work with unstructured pruning [27], network decoupling [4, 13] and more. All these schemes can be considered as pruning networks layer-by-layer, and 1×1 conv. can be inserted and merged in each pruning layer. More details of unstructured pruning with PRACTISE can be found in Section 4.4. To reduce latency on real computing devices, we advocate the adoption of block-level pruning [24], and more details will be introduced in Section 3.4.

Finally, our algorithm does *not* need labels. It can be applied on *unsupervised and out-of-domain* data, too.

3.2. Train the whole network

The whole network is trained layer by layer. Figure 3 illustrates an example of training two layers. In the first step, we focus on pruning the first feature map and keep other layers unchanged. Two convolutional layers with kernel size 1×1 are added before and after the first feature map, respectively. For example, in Figure 3 BN and ReLU are between the $N_1 \times 64 \times 1$ and $64 \times N_1 \times 1$ conv. These two layers are trained by mimicking the next layer's feature map. Once finishing training, both 1×1 layers are merged ($N_1 \times 64 \times 1$ merged into $64 \times 64 \times 3$, and $64 \times N_1 \times 1$ merged into $128 \times 64 \times 3$).

In the second step, similarly, two 1×1 conv. layers are

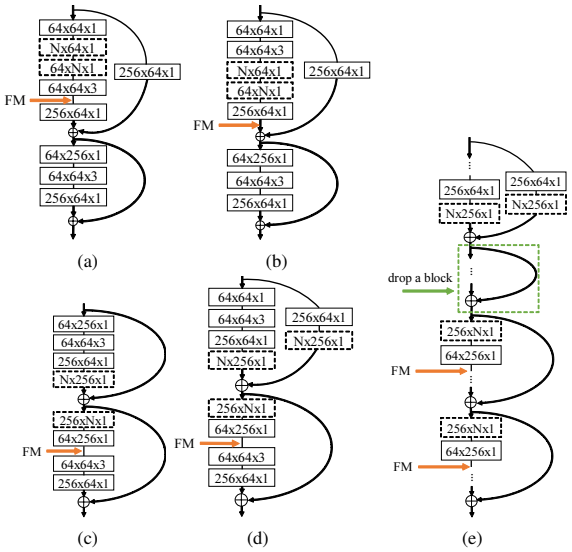

Figure 4. Illustration of training a ResNet with the bottleneck structure. "FM" means feature mimicking. (a) and (b) show how to prune the first and second feature map in the bottleneck, respectively. (c) and (d) show how to prune the feature map between residual blocks with or without the downsample layer, respectively. (e) shows how to train the network when dropping an entire block. This figure needs to be viewed in color.

added, trained by mimicking the third feature map, and then merged into the 3×3 conv. layers. Finally, we will get the network with the structure as the right part of Figure 3.

3.3. Train the ResNet

Figure 4 illustrates how to train ResNet by our algorithm. Figures 4a and 4b show how to prune the first and second feature map in a bottleneck structure, respectively. It is similar to training one layer in Figure 2. When pruning the feature map between two blocks, the downsample layer matters. When the shortcut branch does not downsample (as in Figure 4c), it only needs to add two 1×1 conv. before and after the feature map, respectively. But if the shortcut branch does downsample (as in Figure 4d), an extra 1×1 conv. is added after this downsample layer. All added 1×1 conv. are trained by mimicking the feature map in the next layer, and merged after pruning.

Note that previous method [1] only prunes the filters in the bottleneck and leaves the downsample layer unchanged. PRACTISE can deal with the network whose channels inside and outside the residual connection are pruned.

3.4. Prune the network by removing blocks

Previous methods [1, 12] for few-shot pruning are based on unstructured or filter level pruning and network decoupling. However, these pruning schemes' acceleration are limited. We *focus on reducing the latency* on real computing devices and *advocate block-level pruning*. That is, we directly remove several convolutional layers in VGG-like networks or remove several residual blocks in ResNet-like networks. We will show that with only tiny training sets, networks pruned by dropping blocks is both faster and more accurate than other pruning schemes.

Figure 4e shows how we prune blocks with residual-connections. A network consists of several stages, and feature maps in each stage have the same size and channel numbers. In each stage, the first block increases the feature map's channel number and decrease its spatial resolution. The next block will be dropped by our algorithm. In each stage, we remove one block at most. The pruning strategy of VGG-like model is similar to that of ResNet. We directly remove the second convolutional layer in each stage.

Note that this pruning strategy is simple but effective. Most CNNs can adopt this strategy and are pruned without any extra computation. That means the dropped blocks are determined once the structure of a CNN is known. Although this strategy may not find the optimal blocks to drop, it have surpassed the filter-level pruning and network decomposition (cf. Section 4), which shows the advantage of block-level pruning and supports our conclusions well. Finding optimal blocks to drop under the constraint with only a tiny training set may need a more sophisticated algorithm which needs extra computation. And more importantly, such an algorithm may be unstable in practice, because of the small training set.

After pruning, the pruned network is trained stage by stage. Due to the existence of the shortcut connection, the feature map at a dropped block can directly affect the remaining blocks. Hence, 1×1 conv. layers are added before the first convolutional layers in each block. And we make all feature maps after the first convolutional layers mimic the raw feature maps. VGG-like model does not have shortcut connections, and the feature map at a dropped block only affects the next layer. Then, we only need to make the next layer's feature map consistent with the raw feature map.

Another benefit of block-level pruning is the training process is fast compared with filter-level pruning. If we remove three blocks in ResNet, it only needs to optimize the feature mimicking problem three times, which is far less than that in filter-level pruning. The training usually finished in 10 minutes when adopting block-level pruning in our experiments.

3.5. Finetune the network

After the model has been pruned, it is usually finetuned by minimizing the cross entropy loss ($\mathcal{L}_c$). All parameters are updated by back propagation. Finetuning is an important strategy to boost the pruned model's performance in many pervious methods. But, pruned models easily over-

Scheme	Dropped block	#Params (M)	MACs (G)	Top-1	Top-5	GPU (ms)	CPU (s)
ResNet-34		21.80	3.66	73.31	91.42	162	6.60
ResNet-34-A	1.2	21.72	3.43	45.77	69.40	148	6.03
ResNet-34-B	1.2, 2.2	21.43	3.20	23.56	43.44	140	5.72
ResNet-34-C	1.2, 2.2, 3.2	20.25	2.97	12.41	27.07	133	5.15
ResNet-34-50% [1]		19.25	2.80	14.80	30.19	145	5.21
ResNet-50		25.56	4.09	76.13	92.86	374	12.13
ResNet-50-A	1.2	25.49	3.87	71.61	90.57	342	11.21
ResNet-50-B	1.2, 2.2	25.21	3.65	57.24	80.85	324	10.65
ResNet-50-C	1.2, 2.2, 3.2	24.09	3.43	48.32	72.48	312	10.20
ResNet-50-DW-PW [4]		18.11	2.89	1.13	3.55	407	12.85
ResNet-50-LRD [13]		16.65	2.66	37.99	62.07	371	10.63

Table 1. Pruning schemes on ImageNet. GPU latency was tested on a single GeForce RTX 3090 with batch size 512, and CPU latency on a single-thread Intel Xeon Gold 5220R with batch size 64.

fit with tiny training sets. Inspired by recent works on knowledge distillation [8, 18, 23], we finetune the pruned network by making the pruned network mimicking the raw network's features in the penultimate layer (i.e., after the global pooling and before the final fully connected layer). The loss function is

$$\mathcal{L} = \mathcal{L}_c + \beta \mathcal{L}_{mse} \,, \tag{4}$$

where $\mathcal{L}_c$ and $\mathcal{L}_{mse}$ denote the cross entropy loss for classification and the mean squared error for mimicking features, respectively, and β is a hyperparameter to balance the two terms. We simply set $\beta = 100$ in *all* experiments.

4. Experimental Results

In this section, we evaluate the performance of PRACTISE. ResNet-34 [6], ResNet-50, VGG-16 [21] and MobileNetV2 [20] will be pruned on ImageNet [19], while ResNet-56 and VGG-16 will be pruned on CIFAR-10 [10]. Then, to test the generalization ability of PRACTISE, we will prune ResNet-50 with out-of-domain data. Finally, ablation studies will analyze the stability of our algorithm and the effect of omitting the finetuning of some front layers. All experiments were conducted with PyTorch [17].

Implementation details. To train the network with the loss function in Equation 1, we use Adam [9] with batch size 64 and learning rate 0.0001. Note that if the number of training images is less than 64, the batch size will equal to that number. Each layer is trained for 1000 epochs and we early stop if the training loss increases for 10 epochs. After layer-wise training, the network is finetuned with the loss in Equation 4 ($\beta = 100$). We use SGD with batch size 256 and learning rate 0.0001 to finetune the network for 100 epochs. We do not adopt any learning rate schedules for simplicity. For fair comparison, we use the data argumentation strategy supplied by PyTorch official examples, and the resolutions used in ImageNet and CIFAR-10 are 224×224 and 32×32, respectively. Layer-wise training and finetuning use the same tiny training set.

With tiny training sets, PRACTISE finetunes the pruning network fast. It took less than 10 minutes when the amount of training data was less than 1000. So PRACTISE can be used for fast deployment and/or quickly verifying the compressed model. Next, we are mainly concerned about the trade-off between the accuracy and the model's latency.

4.1. ImageNet results

Table 1 summarizes ResNet models and compression schemes used in our experiments. ResNet-34 and ResNet-50 are official releases of torchvision.[1] ResNet-A, -B and -C are pruned by dropping blocks according to Section 3.4. Note that ResNet has four stages, and ResNet-A removes the second block in the first stage (denoted as block 1.2). ResNet-B and -C are generated in a similar fashion by dropping 2 and 3 blocks, respectively. ResNet-34-50% is pruned by removing filters in the basic-block following previous work [1]. To compare with previous methods fairly, we directly use the same model and train it by our algorithm. ResNet-50-DW-PW is pruned by decoupling a regular conv. into a depthwise plus a pointwise conv. [4]. ResNet-50-LRD is compressed via Low-Rank Decomposition [13]. All 3×3 conv. are decomposed by PCA [25] and the components where the PCA energy ratio is greater than 0.4 are reserved.

We test the running speed of all compressed networks on both GPU and CPU. When running on GPU, networks pruned by dropping filters and decoupling are accelerated only slightly. ResNet-34-50% is slower than ResNet-34-B, while ResNet-50-DW-PW and ResNet-50-LRD are even slower than ResNet-50-A. Note that ResNet-50-DW-PW is even slower than the raw ResNet-50. When running on CPU, ResNet-34-50% is still slower than ResNet-34-C, and

[1] https://pytorch.org/vision/stable/models.html

Model	Method	50	100	500	1-shot
ResNet-34-50% [1]	L1-norm [11]	72.94 ± 0.00	72.94 ± 0.00	72.94 ± 0.00	72.94 ± 0.00
	BP	83.18 ± 1.86	84.32 ± 1.29	85.34 ± 0.89	85.76 ± 0.73
	FSKD [12]	82.53 ± 1.52	84.58 ± 1.13	86.67 ± 0.78	87.08 ± 0.76
	FitNet [18]	86.86 ± 1.81	87.12 ± 1.63	87.73 ± 0.96	87.66 ± 0.84
	ThiNet [16]	85.67 ± 1.57	85.54 ± 1.39	86.97 ± 0.89	87.42 ± 0.76
	CP [7]	86.34 ± 1.24	86.38 ± 1.37	87.41 ± 0.80	88.03 ± 0.66
	CD [1]	87.42 ± 1.69	87.73 ± 1.17	88.60 ± 0.82	88.40 ± 0.61
	baseline	81.52 ± 0.42	84.60 ± 0.28	87.83 ± 0.04	88.60 ± 0.04
	PRACTISE	87.79 ± 0.09	87.88 ± 0.09	88.82 ± 0.06	89.00 ± 0.07
	PRACTISE + FT	$\mathbf{87.86 \pm 0.12}$	$\mathbf{88.23 \pm 0.10}$	$\mathbf{89.06 \pm 0.11}$	$\mathbf{89.35 \pm 0.07}$
ResNet-34-C	baseline	81.75 ± 0.39	85.10 ± 0.13	88.62 ± 0.12	89.46 ± 0.06
	PRACTISE	87.65 ± 0.19	88.31 ± 0.04	89.04 ± 0.04	89.26 ± 0.04
	PRACTISE + FT	$\mathbf{88.38 \pm 0.07}$	$\mathbf{88.94 \pm 0.03}$	$\mathbf{89.53 \pm 0.08}$	$\mathbf{89.80 \pm 0.03}$
ResNet-34-B	baseline	85.40 ± 0.53	87.54 ± 0.28	89.99 ± 0.09	90.49 ± 0.05
	PRACTISE	89.20 ± 0.08	89.43 ± 0.09	89.82 ± 0.05	89.99 ± 0.01
	PRACTISE + FT	$\mathbf{90.09 \pm 0.09}$	$\mathbf{90.25 \pm 0.07}$	$\mathbf{90.55 \pm 0.03}$	$\mathbf{90.70 \pm 0.04}$
ResNet-34-A	baseline	87.79 ± 0.33	89.40 ± 0.19	90.88 ± 0.05	91.02 ± 0.03
	PRACTISE	90.11 ± 0.16	90.51 ± 0.12	90.80 ± 0.04	90.80 ± 0.04
	PRACTISE + FT	$\mathbf{90.92 \pm 0.04}$	$\mathbf{91.01 \pm 0.04}$	$\mathbf{91.03 \pm 0.03}$	$\mathbf{91.10 \pm 0.03}$

Table 2. Top-5 validation accuracy (%) on ImageNet for pruning ResNet-34. The results of PRACTISE were run by five times with different sampled tiny training sets. The Top-5 accuracy of the original ResNet-34 is 91.42%.

Model	Method	50	100	500	1-shot
ResNet-50-DW-PW	baseline	1.12	24.03	55.23	88.52
	PRACTISE	89.85	**90.30**	91.08	91.19
	PRACTISE + FT	**89.98**	90.28	**91.16**	**91.53**
ResNet-50-LRD	baseline	88.97	89.43	90.60	90.90
	PRACTISE	89.95	**90.19**	**90.77**	90.84
	PRACTISE + FT	**90.04**	90.14	**90.77**	**91.06**
ResNet-50-C	baseline	82.51	84.00	86.65	87.33
	PRACTISE	88.66	89.20	89.66	89.68
	PRACTISE + FT	**89.94**	**90.39**	**91.05**	**91.14**
ResNet-50-B	baseline	86.47	87.43	89.52	89.90
	PRACTISE	90.75	90.98	91.25	91.43
	PRACTISE + FT	**91.59**	**91.70**	**92.07**	**92.21**
ResNet-50-A	baseline	91.65	91.48	91.74	91.84
	PRACTISE	91.98	92.12	92.50	92.46
	PRACTISE + FT	**92.62**	**92.65**	**92.66**	**92.69**

Table 3. Top-5 validation accuracy (%) on ImageNet for pruning ResNet-50. The Top-5 accuracy of the original ResNet-50 is 92.86%.

both ResNet-50-DW-PW and ResNet-50-LRD are slower than ResNet-50-C.

Tables 2 and 3 summarize the Top-5 validation accuracy on ImageNet for pruning ResNet-34 and ResNet-50, respectively. Top-1 results can be found in the appendix. Following previous work, we randomly sample 50, 100 and 500 data from the training set to train the pruning network. 1-shot denotes that the training set consists of 1 instance per class, which results in 1000 training samples in total. All experimental results of other methods are cited from [1]. "baseline" denotes directly finetuning the pruned network with the loss in Equation 4. "PRACTISE" represents networks trained by our algorithm without using the data labels. We further finetune the network with the labels and denote as "PRACTISE + FT".

First, using ResNet-34-50%, PRACTISE is consistently better than previous methods: achieving higher accuracy and *much more stable*. Note that CD needs the labels to finetune, but PRACTISE surpasses CD without using the labels. The performance of pruned network can be improved further by finetuning ("PRACTISE + FT"). Second, block-level pruning achieves better accuracy than filter-level pruning. Our ResNet-34-C is consistently better than ResNet-34-50%, and it also runs much faster. Third, PRACTISE is versatile to deal with different pruning schemes. ResNet-50-DW-PW and ResNet-50-LRD can also be boosted by PRACTISE, especially with scarce training data.

Overall, Our method is significantly better than previous methods. First, considering accuracy-latency tradeoff, previous method proposed ResNet-34-50%, while our ResNet-34-B is slightly faster than ResNet-34-50% (cf. Table 1). And the accuracies of ResNet-34-B are higher than that of CD by more than 2% (cf. Table 2), which is a very sig-

Method	ResNet-56-50%			VGG-16-50%		
	1-shot	2-shot	3-shot	1-shot	2-shot	3-shot
L1-norm [11]	80.43 ± 0.00	80.43 ± 0.00	80.43 ± 0.00	14.36 ± 0.00	14.36 ± 0.00	14.36 ± 0.00
BP	84.17 ± 1.55	86.61 ± 1.69	86.86 ± 1.30	49.24 ± 1.76	49.32 ± 1.88	51.39 ± 1.53
FSKD [12]	84.26 ± 1.42	85.79 ± 1.31	85.99 ± 1.29	47.91 ± 1.82	55.44 ± 1.71	61.76 ± 1.39
FitNet [18]	86.85 ± 1.91	87.95 ± 2.13	88.94 ± 1.85	48.51 ± 2.51	71.51 ± 2.03	76.22 ± 1.95
ThiNet [16]	88.40 ± 1.26	88.76 ± 1.18	88.95 ± 1.19	58.06 ± 1.71	72.07 ± 1.68	75.37 ± 1.59
CP [7]	88.53 ± 1.37	88.69 ± 1.09	88.79 ± 0.94	66.03 ± 1.56	75.23 ± 1.49	77.98 ± 1.47
CD [1]	88.42 ± 1.63	89.12 ± 1.57	89.75 ± 1.50	69.25 ± 1.39	80.65 ± 1.47	82.08 ± 1.41
CD*	84.37 ± 0.91	88.89 ± 0.22	89.92 ± 0.35	69.08 ± 1.85	76.98 ± 1.07	$\mathbf{79.74 \pm 0.47}$
PRACTISE	85.40 ± 0.98	87.20 ± 0.48	87.73 ± 0.33	68.72 ± 1.52	75.81 ± 1.09	78.50 ± 0.83
PRACTISE + FT	$\mathbf{88.64 \pm 0.32}$	$\mathbf{89.70 \pm 0.13}$	$\mathbf{89.98 \pm 0.11}$	$\mathbf{70.82 \pm 2.20}$	$\mathbf{77.38 \pm 0.77}$	78.62 ± 0.59

Table 4. Top-1 accuracy (%) on the test set of the CIFAR-10 benchmark with filter-level pruning for ResNet-56 and VGG-16. The model is trained with 1, 2 and 3 randomly sampled training instances per class. We denote by * those methods where we re-run five times using author-provided code and our original networks. The results of our PRACTISE were run by five times. The Top-1 accuracy of the original ResNet-56 and VGG-16 are 93.82% and 93.49%, respectively.

nificant improvement. Second, considering using the same network ResNet-34-50%, PRACTISE is consistently better than CD (cf. Table 2). Note that PRACTISE does *not* use any label while CD needs labels to finetune the network.

In the appendix, we summarize the pruning schemes for VGG-16, and the compression accuracy and speed of these networks on ImageNet. Without shortcut connections, VGG-16 can be accelerated by dropping filters. The experimental results show that PRACTISE has significantly higher accuracy than CD, and again block-level pruning outperforms filter-level pruning.

MobileNetV2 [20] is a lightweight model and popularly applied on mobile devices. It has 7 stages in total, and we remove blocks starting from the third stage and get four pruned models described in the appendix. Our results show that MobileNetV2 can be accelerated by dropping blocks even though it is already a compact model. In the appendix, we report the Top-5 and Top-1 accuracy on the validation set of the ImageNet benchmark for pruning MobileNetV2, respectively. The accuracy of all pruned models can be improved by PRACTISE, too.

4.2. CIFAR-10 results

We also test PRACTISE on the CIFAR-10 benchmark, which is a popular benchmark in previous works. We test the pruned model used in previous works and compare PRACTISE with them on this benchmark. More details about the networks and compression schemes can be found in the appendix. Table 4 presents the experimental results of pruning ResNet-56 and VGG-16. PRACTISE with finetuning achieves the best performance in almost all cases.

Training dataset	50	100	500	1000
FakeData	0.45	0.59	0.69	0.83
Place365 [26]	50.29	53.26	55.11	54.80
ImageNet [19]	54.99	55.23	56.09	56.84
CUB [22]	**64.33**	**66.02**	**66.48**	**67.19**

Table 5. Top-1 accuracy (%) on the test set of the CUB benchmark with ResNet-50-C. The model is trained on different training datasets with 50, 100, 500 and 1000 randomly sampled training images with resolution 224×224. The Top-1 accuracy of the original ResNet-50 and vanilla ResNet-50-C are 79.69% and 44.17%, respectively.

Method	50	100	500	1-shot
CD [1]	69.55	74.21	81.64	83.36
baseline	44.06	59.68	80.18	84.18
PRACTISE	**72.47**	**78.09**	82.52	83.60
PRACTISE + FT	70.83	77.76	**83.28**	**84.81**

Table 6. Top-5 validation accuracy (%) on the ImageNet benchmark for unstructured pruning of ResNet-34. All results were run by three times with different sampled tiny training sets. The Top-5 accuracy of the original ResNet-34 is 91.42%.

4.3. Train with out-of-domain data

PRACTISE trains the pruning model without using the ground-truth labels. Therefore, we can evaluate PRACTISE with out-of-domain training data, that is, finetuning the pruning network by other datasets. As illustrated by Figure 1, this is a common real-world scenario that the algorithmic engineers need to collect some out-of-domain training data due to the customer's privacy policy. Table 5 reports the results. The accuracy of ResNet-50-C can be boosted

Method	50	100	500	1-shot
FT ($\mathcal{L}_c$)	72.70 ± 1.34	77.79 ± 0.83	82.53 ± 0.53	83.67 ± 0.17
FT ($\mathcal{L}_c + \beta\mathcal{L}_{mse}$)	81.75 ± 0.39	85.10 ± 0.13	88.62 ± 0.12	89.46 ± 0.06
PRACTISE	87.65 ± 0.19	88.31 ± 0.04	89.04 ± 0.04	89.26 ± 0.04
PRACTISE + FT ($\mathcal{L}_c$)	85.70 ± 0.45	86.28 ± 0.31	86.76 ± 0.40	86.73 ± 0.10
PRACTISE + FT ($\mathcal{L}_c + \beta\mathcal{L}_{mse}$)	**88.38 ± 0.07**	**88.94 ± 0.03**	**89.53 ± 0.08**	**89.80 ± 0.03**

Table 7. Top-5 validation accuracy (%) on the ImageNet benchmark with ResNet-34-C. The results of our method were run by five times. The Top-5 accuracy of the original ResNet-34 is 91.42%.

by PRACTISE using the original domain (i.e., trained on CUB training images). But, it can also be improved surprisingly by images from ImageNet and Place365. This fact demonstrates the generalization ability of PRACTISE. Compared with ImageNet, Place365 consists of scene pictures, which is further different from CUB. So the accuracy of training on Place365 is lower than that of training on ImageNet. FakeData[2] consists of randomly generated images based on the Gaussian distribution. Because they are not natural images, the pruning network cannot learn any useful information from them.

4.4. Unstructured pruning

To apply PRACTISE on unstructured pruning, the sparse mask M should be generated first by an unstructured pruning method. To compare with CD, we follow their settings and use [27] to generate M. So the pruned weight will be $W \bigodot M$, where W is the unpruned dense tensor. The linear (1×1 conv.) layer can be inserted as $(W * U) \bigodot M$. Note that PRACTISE only optimizes U by feature mimicking. The final weight will be sparse, because M is sparse.

We evaluate our method on ImageNet with ResNet-34. First, we directly run the official scripts to evaluate CD with unstructured pruning scheme. Note that they remove 90% parameters in each convolution layer and we use the same setting for fair comparison. All experiments are run by three times. The experimental results are shown by Table 6.

Although PRACTISE also works well for unstructured pruning, we claim that filter-level and block-level pruning is more practical.

4.5. Ablation studies

We conduct ablative experiments to study the stability of PRACTISE and the finetuning strategy. Table 7 summarizes the results. We find that finetuning by minimizing $\mathcal{L}_c + \beta\mathcal{L}_{mse}$ achieves higher accuracy and is more stable than minimizing $\mathcal{L}_c$ alone. And if the pruning model is first trained by PRACTISE, finetuning can get more stable models than directly from pretrained weights. Although PRACTISE can improve the accuracy, simply finetuned by minimizing $\mathcal{L}_c$ easily result in overfitting with tiny training set. Finetuning with feature mimicking can further boost the pruned network trained by PRACTISE.

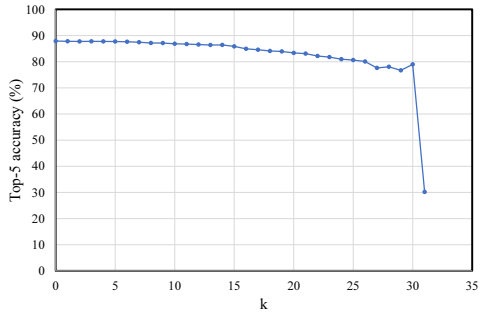

Figure 5. Top-5 validation accuracy (%) on ImageNet with filter-level pruning on ResNet34 and 50 training instances. We freeze the front k layers and train the remaining layers by PRACTISE.

Finally, we study the effect of omitting to finetune some front layers. PRACTISE will finetune the pruned model layer by layer. We freeze the front k layers and finetune the pruned model from the middle layer. Note that PRACTISE will train 31 layers in total if $k = 0$. Figure 5 summarizes the experimental results. The accuracy will decrease as k increases. And the rate of decreasing for the latter layers is larger than that of the front layers. This observations shows that it is more important to train the latter layers.

5. Conclusions and Future Work

In this paper, we proposed PRACTISE, an algorithm to train the pruning network with tiny sets of training images. PRACTISE is advantageous because it achieved higher accuracy than existing methods and was more stable than previous works. Furthermore, it can deal with different pruning schemes and is fast for deployment. We studied different model compression schemes and we are the first to advocate drop entire blocks, because when there are only tiny training sets for network compression and acceleration, it is faster in both training and inference, and is more stable than existing schemes (filter-level pruning, decoupling, etc.)

In the future, we plan to integrate PRACTISE learning and the subsequent finetuning into a coherent step, with the

[2] https://pytorch.org/vision/stable/datasets.html#fakedata

goal to (potentially) compress a network without accuracy loss when the compression ratio is not high and only tiny training sets are available. We also plan to extend PRACTISE to other vision tasks, including object detection, segmentation and action recognition.

References

[1] Haoli Bai, Jiaxiang Wu, Irwin King, and Michael Lyu. Few shot network compression via cross distillation. In *AAAI*, volume 04, pages 3203–3210, 2020. 1, 2, 3, 4, 5, 6, 7, 11

[2] Ron Banner, Itay Hubara, Elad Hoffer, and Daniel Soudry. Scalable methods for 8-bit training of neural networks. In *NeurIPS 31*, pages 5151–5159, 2018. 1

[3] Xin Dong, Shangyu Chen, and Sinno Jialin Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. In *NeurIPS 30*, pages 4857–4867, 2017. 1

[4] Jianbo Guo, Yuxi Li, Weiyao Lin, Yurong Chen, and Jianguo Li. Network decoupling: From regular to depthwise separable convolutions. In *BMVC*, page 248, 2018. 1, 2, 3, 5

[5] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *ICLR*, pages 1–14, 2016. 1

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016. 5

[7] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *ICCV*, pages 1389–1397, 2017. 6, 7

[8] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. 2, 5

[9] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, pages 1–15, 2015. 5

[10] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009. 5

[11] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016. 1, 2, 6, 7

[12] Tianhong Li, Jianguo Li, Zhuang Liu, and Changshui Zhang. Few sample knowledge distillation for efficient network compression. In *CVPR*, pages 14639–14647, 2020. 1, 2, 3, 4, 6, 7

[13] Shaohui Lin, Rongrong Ji, Chao Chen, Dacheng Tao, and Jiebo Luo. Holistic CNN compression via low-rank decomposition with knowledge transfer. *IEEE TPAMI*, 41(12):2889–2905, 2018. 1, 2, 3, 5

[14] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *ICCV*, pages 2736–2744, 2017. 1, 2

[15] Jian-Hao Luo and Jianxin Wu. Neural network pruning with residual-connections and limited-data. In *CVPR*, pages 1458–1467, 2020. 2

[16] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. ThiNet: A filter level pruning method for deep neural network compression. In *ICCV*, pages 5058–5066, 2017. 1, 2, 3, 6, 7

[17] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. PyTorch: An imperative style, high-performance deep learning library. In *NeurIPS 32*, pages 8026–8037, 2019. 5

[18] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. FitNets: Hints for thin deep nets. In *ICLR*, pages 1–13, 2015. 2, 5, 6, 7

[19] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet large scale visual recognition challenge. *IJCV*, 115(3):211–252, 2015. 5, 7

[20] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted residuals and linear bottlenecks. In *CVPR*, pages 4510–4520, 2018. 5, 7

[21] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, pages 1–14, 2015. 5

[22] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. The Caltech-UCSD birds-200-2011 dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011. 7

[23] Guo-Hua Wang, Yifan Ge, and Jianxin Wu. Distilling knowledge by mimicking features. *IEEE TPAMI*, to appear. 2, 5

[24] Wenxiao Wang, Shuai Zhao, Minghao Chen, Jinming Hu, Deng Cai, and Haifeng Liu. DBP: Discrimination based block-level pruning for deep model acceleration. *arXiv preprint arXiv:1912.10178*, 2019. 2, 3

[25] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987. 5

[26] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Antonio Torralba, and Aude Oliva. Places: An image database for deep scene understanding. *arXiv preprint arXiv:1610.02055*, 2016. 7

[27] Michael Zhu and Suyog Gupta. To prune, or not to prune: Exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017. 3, 8

Practical Network Acceleration with Tiny Sets -Supplementary Material-

Model	Method	50	100	500	1-shot
ResNet-34-50%	baseline	58.60 ± 0.52	62.90 ± 0.33	67.57 ± 0.18	68.68 ± 0.06
	PRACTISE	67.47 ± 0.06	67.49 ± 0.13	69.04 ± 0.06	69.34 ± 0.08
	PRACTISE + FT	$\mathbf{67.43 \pm 0.16}$	$\mathbf{67.95 \pm 0.08}$	$\mathbf{69.34 \pm 0.07}$	$\mathbf{69.84 \pm 0.12}$
ResNet-34-C	baseline	58.66 ± 0.62	63.06 ± 0.31	68.32 ± 0.19	69.86 ± 0.04
	PRACTISE	67.10 ± 0.19	68.08 ± 0.15	69.19 ± 0.12	69.52 ± 0.10
	PRACTISE + FT	$\mathbf{68.18 \pm 0.16}$	$\mathbf{68.97 \pm 0.10}$	$\mathbf{69.99 \pm 0.08}$	$\mathbf{70.54 \pm 0.08}$
ResNet-34-B	baseline	63.61 ± 0.74	66.82 ± 0.48	70.65 ± 0.05	71.45 ± 0.08
	PRACTISE	69.41 ± 0.14	69.90 ± 0.22	70.47 ± 0.10	70.77 ± 0.02
	PRACTISE + FT	$\mathbf{70.93 \pm 0.09}$	$\mathbf{71.15 \pm 0.11}$	$\mathbf{71.50 \pm 0.08}$	$\mathbf{71.81 \pm 0.05}$
ResNet-34-A	baseline	66.95 ± 0.63	69.69 ± 0.44	72.42 ± 0.05	72.66 ± 0.10
	PRACTISE	71.15 ± 0.16	71.74 ± 0.19	72.25 ± 0.07	72.37 ± 0.05
	PRACTISE + FT	$\mathbf{72.56 \pm 0.07}$	$\mathbf{72.59 \pm 0.09}$	$\mathbf{72.57 \pm 0.06}$	$\mathbf{72.71 \pm 0.07}$

Table 8. Top-1 validation accuracy (%) on ImageNet for pruning ResNet-34. The results of our method were run by five times. The Top-1 accuracy of the original ResNet-34 is 73.31%.

Model	Method	50	100	500	1-shot
ResNet-50-DW-PW	baseline	0.25	8.33	28.38	68.72
	PRACTISE	70.87	**71.51**	72.62	72.83
	PRACTISE + FT	**70.94**	71.44	**72.80**	**73.19**
ResNet-50-LRD	baseline	68.78	69.42	71.36	72.26
	PRACTISE	**70.41**	**70.84**	**71.87**	72.32
	PRACTISE + FT	70.40	70.76	71.62	**72.54**
ResNet-50-C	baseline	59.30	61.42	65.66	66.64
	PRACTISE	68.62	69.48	70.28	70.24
	PRACTISE + FT	**70.41**	**71.28**	**72.45**	**72.68**
ResNet-50-B	baseline	64.47	66.21	69.89	70.63
	PRACTISE	72.12	72.53	73.21	73.31
	PRACTISE + FT	**73.54**	**73.69**	**74.17**	**74.51**
ResNet-50-A	baseline	73.62	73.12	74.01	73.98
	PRACTISE	74.44	74.87	75.35	75.40
	PRACTISE + FT	**75.38**	**75.54**	**75.56**	**75.71**

Table 9. Top-1 validation accuracy (%) on ImageNet for pruning ResNet-50. The Top-1 accuracy of the original ResNet-50 is 76.13%.

Scheme	Dropped block	#Params (M)	MACs (G)	Top-1	Top-5	GPU (ms)	CPU (s)
VGG-16		138.37	15.47	73.36	91.52	616	30.91
VGG-16-A	1.2	138.33	13.62	0.17	0.67	500	24.66
VGG-16-B	1.2, 2.2	138.18	11.77	0.08	0.49	427	21.32
VGG-16-C	1.2, 2.2, 3.2	137.59	9.92	0.07	0.46	382	18.41
VGG-16-50% [1]		131.32	3.96	0.11	0.51	248	10.77

Table 10. Pruning schemes of VGG-16 on ImageNet. GPU latency was tested on a single GeForce RTX 3090 with batch size 512, and CPU latency on a single-thread Intel Xeon Gold 5220R CPU with batch size 64.

Model	Method	50	100	500	1-shot
VGG-16-50% [1]	baseline	1.29 ± 0.62	2.57 ± 1.30	76.10 ± 3.78	81.00 ± 1.91
	CD*	75.41 ± 0.55	76.76 ± 0.29	79.04 ± 0.31	79.62 ± 0.25
	PRACTISE	$\mathbf{80.36 \pm 0.22}$	81.13 ± 0.41	83.76 ± 1.38	84.99 ± 0.09
	PRACTISE + FT	79.87 ± 0.22	$\mathbf{82.53 \pm 0.28}$	$\mathbf{85.46 \pm 0.07}$	$\mathbf{86.21 \pm 0.12}$
VGG-16-C	baseline	0.83 ± 0.05	1.09 ± 0.20	2.47 ± 0.16	4.40 ± 0.20
	PRACTISE	84.85 ± 0.33	85.51 ± 0.53	86.08 ± 0.50	86.52 ± 0.36
	PRACTISE + FT	$\mathbf{86.08 \pm 0.18}$	$\mathbf{86.45 \pm 0.47}$	$\mathbf{88.31 \pm 0.59}$	$\mathbf{89.34 \pm 0.11}$
VGG-16-B	baseline	0.65 ± 0.05	0.65 ± 0.10	2.34 ± 0.39	68.36 ± 0.53
	PRACTISE	87.04 ± 0.55	87.67 ± 0.48	87.69 ± 0.75	87.54 ± 0.55
	PRACTISE + FT	$\mathbf{88.66 \pm 0.21}$	$\mathbf{89.02 \pm 0.06}$	$\mathbf{89.93 \pm 0.27}$	$\mathbf{90.17 \pm 0.54}$
VGG-16-A	baseline	9.18 ± 1.92	25.92 ± 3.04	75.29 ± 0.68	82.99 ± 0.28
	PRACTISE	88.98 ± 0.57	89.15 ± 0.56	89.87 ± 0.20	90.22 ± 0.09
	PRACTISE + FT	$\mathbf{89.95 \pm 0.28}$	$\mathbf{90.11 \pm 0.34}$	$\mathbf{90.46 \pm 0.47}$	$\mathbf{91.02 \pm 0.24}$

Table 11. Top-5 validation accuracy (%) on ImageNet for pruning VGG-16. We denote by * those methods where we re-run using author-provided code. All results were run by five times with different sampled tiny training subset. The Top-5 accuracy of the original VGG-16 is 91.52%.

Model	Method	50	100	500	1-shot
VGG-16-50% [1]	baseline	0.31 ± 0.14	0.69 ± 0.37	49.88 ± 3.89	56.27 ± 2.36
	CD*	49.93 ± 0.71	51.57 ± 0.42	54.47 ± 0.34	55.03 ± 0.30
	PRACTISE	$\mathbf{55.74 \pm 0.50}$	56.74 ± 0.32	60.38 ± 2.22	62.39 ± 0.12
	PRACTISE + FT	54.87 ± 0.39	$\mathbf{58.58 \pm 0.31}$	$\mathbf{62.77 \pm 0.25}$	$\mathbf{64.27 \pm 0.15}$
VGG-16-C	baseline	0.17 ± 0.02	0.22 ± 0.05	0.58 ± 0.06	1.06 ± 0.06
	PRACTISE	62.88 ± 0.61	63.88 ± 0.71	64.60 ± 0.87	65.44 ± 0.50
	PRACTISE + FT	$\mathbf{64.61 \pm 0.26}$	$\mathbf{65.17 \pm 0.61}$	$\mathbf{68.06 \pm 0.90}$	$\mathbf{69.61 \pm 0.25}$
VGG-16-B	baseline	0.15 ± 0.02	0.14 ± 0.04	0.55 ± 0.13	42.51 ± 0.67
	PRACTISE	65.74 ± 0.97	66.78 ± 0.73	66.94 ± 1.15	66.63 ± 0.82
	PRACTISE + FT	$\mathbf{68.38 \pm 0.31}$	$\mathbf{69.05 \pm 0.15}$	$\mathbf{70.55 \pm 0.48}$	$\mathbf{70.93 \pm 1.00}$
VGG-16-A	baseline	3.16 ± 0.83	11.08 ± 1.56	49.59 ± 0.76	59.68 ± 0.44
	PRACTISE	68.87 ± 1.00	69.23 ± 0.88	70.57 ± 0.27	71.08 ± 0.11
	PRACTISE + FT	$\mathbf{70.62 \pm 0.43}$	$\mathbf{70.84 \pm 0.52}$	$\mathbf{71.46 \pm 0.81}$	$\mathbf{72.36 \pm 0.43}$

Table 12. Top-1 validation accuracy (%) on ImageNet for pruning VGG-16. We denote by * those methods where we re-run five times using author-provided code. And the results of our method were run by five times. The Top-1 accuracy of the original VGG-16 is 73.36%.

Scheme	Dropped block	#Params (M)	MACs (M)	Top-1	Top-5	GPU (ms)	CPU (s)
MobileNetV2		3.50	300.79	71.88	90.29	126	3.90
MobileNetV2-A	3.2	3.49	289.80	62.30	84.27	121	3.65
MobileNetV2-B	3.2, 4.2	3.44	279.49	44.43	68.36	117	3.52
MobileNetV2-C	3.2, 4.2, 5.2	3.32	256.80	31.95	55.63	111	3.34
MobileNetV2-D	3.2, 4.2, 5.2, 6.2	3.00	241.32	16.85	36.60	108	3.27

Table 13. Block-level pruning schemes of MobileNetV2 on ImageNet. GPU latency was tested on a single GeForce RTX 3090 with batch size 512, and CPU latency on a single-thread Intel Xeon Gold 5220R CPU with batch size 64.

Model	Method	50	100	500	1-shot
MobileNetV2-D	baseline	66.46	72.25	79.79	82.45
	PRACTISE	67.63	68.06	74.06	74.51
	PRACTISE + FT	**68.32**	**68.89**	**80.13**	**82.69**
MobileNetV2-C	baseline	79.63	82.32	85.83	86.94
	PRACTISE	83.03	83.84	84.74	84.66
	PRACTISE + FT	**83.66**	**84.76**	**86.34**	**87.30**
MobileNetV2-B	baseline	85.29	86.35	88.06	88.59
	PRACTISE	86.18	86.42	86.97	87.04
	PRACTISE + FT	**86.98**	**87.44**	**88.12**	**88.76**
MobileNetV2-A	baseline	88.74	88.93	89.49	**89.86**
	PRACTISE	87.93	88.50	88.78	88.80
	PRACTISE + FT	**89.01**	**89.28**	**89.58**	**89.86**

Table 14. Top-5 validation accuracy (%) on ImageNet for block-level pruning MobileNetV2. The Top-5 accuracy of the original MobileNetV2 is 90.29%.

Model	Method	50	100	500	1-shot
MobileNetV2-D	baseline	40.62	46.64	55.54	59.23
	PRACTISE	41.39	42.55	48.86	49.55
	PRACTISE + FT	**42.18**	**43.28**	**56.45**	**59.91**
MobileNetV2-C	baseline	55.96	59.59	64.25	65.91
	PRACTISE	60.52	61.94	62.91	62.80
	PRACTISE + FT	**61.27**	**62.90**	**65.02**	**66.48**
MobileNetV2-B	baseline	64.01	65.54	67.77	68.64
	PRACTISE	65.10	65.79	66.43	66.44
	PRACTISE + FT	**66.19**	**66.95**	**68.03**	**68.91**
MobileNetV2-A	baseline	68.95	69.39	70.23	70.54
	PRACTISE	68.06	68.56	69.19	69.24
	PRACTISE + FT	**69.51**	**69.70**	**70.35**	**70.65**

Table 15. Top-1 validation accuracy (%) on ImageNet for block-level pruning MobileNetV2. The Top-1 accuracy of the original MobileNetV2 is 71.88%.

Scheme	#Params	MACs (M)	Acc (%)	GPU (ms)	CPU (ms)
ResNet-56	853.02 K	125.49	93.82	21	241
ResNet-56-50%	514.69 K	74.25	85.46	18	207
VGG-16	14.99 M	313.47	93.49	18	450
VGG-16-50%	4.53 M	91.11	14.41	9	152

Table 16. Pruning schemes of ResNet-56 and VGG-16 on CIFAR-10. GPU latency was tested on a single GeForce RTX 3090 with batch size 512, and CPU latency on a single-thread Intel Xeon Gold 5220R CPU with batch size 64.