PAPER

Pruning Ratio Optimization with Layer-Wise Pruning Method for Accelerating Convolutional Neural Networks

Koji KAMMA^{†a)}, Member, Sarimu INOUE[†], Nonmember, and Toshikazu WADA[†], Member

SUMMARY Pruning is an effective technique to reduce computational complexity of Convolutional Neural Networks (CNNs) by removing redundant neurons (or weights). There are two types of pruning methods: holistic pruning and layer-wise pruning. The former selects the least important neuron from the entire model and prunes it. The latter conducts pruning layer by layer. Recently, it has turned out that some layer-wise methods are effective for reducing computational complexity of pruned models while preserving their accuracy. The difficulty of layer-wise pruning is how to adjust pruning ratio (the ratio of neurons to be pruned) in each layer. Because CNNs typically have lots of layers composed of lots of neurons, it is inefficient to tune pruning ratios by human hands. In this paper, we present Pruning Ratio Optimizer (PRO), a method that can be combined with layer-wise pruning methods for optimizing pruning ratios. The idea of PRO is to adjust pruning ratios based on how much pruning in each layer has an impact on the outputs in the final layer. In the experiments, we could verify the effectiveness of PRO.

key words: pruning, pruning ratio optimizer, PRO

1. Introduction

In various Computer Vision tasks, Convolutional Neural Networks (CNNs) have been showing record-breaking performance. In large scale image recognition task, some CNN models, such as ResNet [1], outperform humans. Although, is it possible to use those models in resource-limited embedded systems? At this point, we need strong computational resources to use large models in real-time applications. For future applications, it is important to accelerate inference of large models without significant accuracy degradation.

A major approach for producing a fast CNN model is to conduct pruning on a pretrained large model. Pruning is a technique to reduce computational complexity of pretrained CNNs by removing redundant neurons (or weights). Up to now, a lot of pruning methods have been developed [2]–[9].

There are two types of pruning: holistic pruning [2], [6], [10]–[13] and layer-wise pruning [3]–[5], [7], [8], [14]. Holistic pruning methods are designed to select the neurons to be pruned from the entire model. On the other hand, layer-wise methods select neurons to be pruned in each layer separately. As we mention in Sect. 2, layer-wise methods have turned out to be more effective and theoretically sound than holistic methods.

When we conduct layer-wise pruning, it is important to

tune pruning ratio (the ratio of neurons to be pruned) in each layer of the models in order to preserve their accuracy well. However, it is difficult to tune pruning ratios, because it is not trivial how redundant those layers are. Besides, in order to check if the current pruning ratios are proper or not, we have to retrain the pruned model and evaluate the accuracy, which is time-consuming.

This problem can be eased by using the pruning methods that conduct *reconstruction* [3], [14]. These methods do not only prune neurons but also reconstruct the pruned ones' outputs by tuning the remaining neurons' weights. Therefore, these methods can reduce redundancy of the pruned models while preserving the accuracy well even without re-training. By using these methods, we can tune the pruning ratios efficiently.

Then, the next problem is how to optimize pruning ratios. Because CNN models generally have many layers composed of dozens to thousands of neurons (or channels), the search space is vast. Therefore, it is obviously not efficient to tune pruning ratios by human hands. We need an efficient way of optimizing pruning ratios.

In this paper, we present Pruning Ratio Optimizer (PRO). PRO optimizes pruning ratios in a greedy fashion based on the error in the final layer of the model. The idea of PRO is to repeat the following steps until the model becomes fast enough: 1) Select the layer where pruning will have the smallest impact on the outputs in the final layer, and 2) Prune some neurons in that layer. After repeating these procedures several times, pruning ratio in each layer will be properly tuned.

The rest of this paper are structured as follows. The related works are reviewed in Sect. 2, the proposed method is explained in Sect. 3, the experiments are in Sect. 4, and we conclude the discussions in Sect. 5.

2. Related Works

We review the works related to pruning and a prior work that aims to optimize pruning ratios. Other types of CNN compression methods are also briefly reviewed.

2.1 Pruning Methods

The pruning methods can be divided into two groups: the group of holistic methods [2], [6], [10]–[13] and the group of layer-wise methods [3]–[5], [7], [8], [14].

The holistic methods are designed for comparing the

Manuscript received April 27, 2021.

Manuscript revised August 25, 2021.

Manuscript publicized September 29, 2021.

[†]The authors are with Wakayama University, Wakayama-shi, 640–8510 Japan.

a) E-mail: kammakoji@gmail.com

DOI: 10.1587/transinf.2021EDP7096

importance of neurons/weights in the whole model simultaneously and removing the least important one [2], [6], [10]–[13]. However, many methods in this group rely on heuristic criteria. For example, the method presented in [12] selects the neurons based on L1 norm of the weights connected to them. Although, it has not been proven that the neurons having small weights do not contribute to model performance significantly. For another example, Optimal Brain Damage (OBD) [2] evaluates neuron importance by using Hessian of the cost function over the outputs of each neuron. Because it is not realistic to compute the whole Hessian due to large computational cost, OBD computes only its diagonals, assuming that the non-diagonals are zero. This is not a reasonable assumption, because the weights are obviously dependent on each other. It is also a weakness of OBD that they ignore higher than third derivative terms. There is a method that uses the first derivative information of the loss function to evaluate the neuron importance [15]. However, because of the nonlinearity between the weight values and the loss, only the first derivative information is obviously not sufficient for evaluating neuron importance. In some other works, the first and/or the second derivative information are used for evaluating neuron importance [16], [17], that are also theoretically weak because of the same reasons mentioned above.

On the other hand, some works have offered the layer-wise methods that have been more successful [3]–[5], [7], [8], [14], [18]. These methods select the neurons to be pruned based on layer-wise error. Therefore, their optimization problem can be much simpler than those of the holistic methods. For example, the problem of neuron selection can be formulated as a simple Lasso regression problem [3], [8]. When we use these methods, we need to adjust pruning ratios layer by layer.

Some layer-wise methods perform *reconstruction* for preserving the original layer-wise outputs [3], [4], [14]. In the reconstruction step, the weights connected to the remaining neurons are updated based on least squares method so that the pruned neurons' outputs are reconstructed. Therefore, we can reduce the number of neurons to some degree, while preventing accuracy degradation.

By using the layer-wise methods with reconstruction, we can judge if a certain pruning ratio is proper or not without time-consuming retraining. However, because CNN models typically have lots of layers, it is difficult to tune pruning ratios by human hands. We need an efficient way of optimizing pruning ratios.

He et al. proposed AutoML Model Compression (AMC), a pruning ratio optimization method based on Reinforcement Learning (RL) [19]. They show that the accuracy can be preserved better by using AMC than tuning the pruning ratios by human hands. However, AMC has lots of RL-related hyper-parameters and tuning them is difficult as well. Therefore, it is desired to develop a method that is easier to handle.

2.2 Other Types of Acceleration Methods for CNNs

Here in below, we briefly review existing approaches for compressing pretrained CNN models. These methods can be optionally combined with pruning methods to achieve further acceleration.

(1) Factorization

The most fundamental method in the factorization group is presented in [20]. They apply singular value decomposition to large weight matrix, and approximate it by the product of small matrices by discarding the components with small singular values. This results in reducing the parameters with small sacrifice of accuracy. For example, assume that $m \times n$ matrix is approximated by the product of $m \times o$ matrix and $o \times n$ matrix. If $o \ll m, n$, the number of the parameters reduces from mn to $(m+n)o$. Some other methods [21], [22] also belong to this group. The drawback is that the factorized model will have more layers where the computation of forward propagation has to be done sequentially, therefore, the inference may not be accelerated.

(2) Sparsification

Sparsification methods make the weight tensors sparse by training the models with L1 regularization [8], [23]–[26]. The theoretical weakness of sparsification is that L1 regularization shifts the global minimum of the cost function and sacrifices the model performance. Besides, the sparsified models need special hardware and libraries for executing the computations only on non-zero elements of the weight tensors.

(3) Quantization

The methods in this group reduce the redundancy of each bit-wise operation [11], [27], [28], e.g. changing the floating point precision from 32-bit to 8-bit. The quantized models require special hardware and libraries to be deployed.

Binarization is a sub-group of quantization. The idea is to fix all weights to either of -1 or 1 (or 0) so that only additions and subtractions are needed for inference. The binarized models can be deployed on general environments. However, as the multipliers are not necessary, it is more ideal to use the dedicated computational environments for additions and subtractions.

3. Strategy for Pruning Ratio Optimization

In this section, we explain Pruning Ratio Optimizer (PRO) and the algorithm we actually implemented.

3.1 Notations

We first define the notations. $\mathcal{M}$ denotes the model to be pruned. $\mathcal{D}$ denotes a dataset used for pruning. K denotes the number of layers. $\mathcal{Z}^k$ denotes the set of neuron indices

in the k -th layer. $\mathbf{x}_i^k$ denotes the i -th neuron's outputs corresponding to $\mathcal{D}$. $\mathbf{w}_i^k$ denotes the weights going from the i -th neuron to the ones in the next layer. $Y^k = \sum_{i \in \mathcal{Z}^k} \mathbf{x}_i^k \mathbf{w}_i^{k\top}$ denotes the layer-wise outputs. p^k denotes the pruning ratio in the k -th layer. f^k denotes FLOPs (the number of floating point multiplications per data) in the k -th layer.

3.2 Pruning Ratio Optimizer (PRO)

PRO is a pruning ratio optimization method that can be combined with the pruning methods with reconstruction. Especially, it is suited to use Reconstruction Error Aware Pruning (REAP), the pruning method we proposed in our previous work [14], because it is the best pruning methods for minimizing layer-wise errors. For another example, Channel Pruning (CP) can be an option as well. We explain the details of these methods in Appendix A and Appendix B.

Although pruning methods with reconstruction can prevent layer-wise error well, it is still important to adjust pruning ratios in order to make accuracy degradation as small as possible. The idea of PRO is to tune the p -s so as to minimize the error in the final layer of the pruned model. The original outputs in the final layer is given by

$$Y^K = \mathcal{M}(\mathcal{D}). \quad (1)$$

Then, we perform pruning in each layer while conducting reconstruction, and get a pruned model $\mathcal{M}'$. The error in the final layer is given by

$$\Delta Y^K = \mathcal{M}'(\mathcal{D}) - Y^K. \quad (2)$$

We tune the p -s to minimize $\|\Delta Y^K\|_F^2$ and to reduce the FLOPs to the target value.

Because it is difficult to solve this optimization problem analytically, we solve it in a greedy fashion. We repeatedly select the layer where pruning will have the smallest impact on the outputs in the final layer, and prune some neurons in that layer. The procedures of PRO can be described as follows.

Step 1) Draw a curve of FLOPs reduction at each layer and the error in the final layer, as shown in Fig. 1 (a). For this purpose, we try pruning with several pruning ratios and observe the error in the final layer. Here, FLOPs reduction is computed by $p^k f^k$.

Step 2) Select the layer where the most FLOPs can be reduced at a certain cost of error (t_{err}), and perform pruning in the selected layer.

Step 3) If enough FLOPs have been reduced, terminate computation. Otherwise, go to Step 1).

At the end, the pruning ratio in each layer will be optimized.

3.3 Strategy for More Efficient Optimization

The problem of PRO is the large computational cost for Step 1). Drawing precise curves of error and FLOPs reduction such as Fig. 1 (a) is computationally intensive. Thus, we draw rough curves such as Fig. 1 (b) in the following scheme.

Step 1-a) Set p^k to a few values (e.g. $p^k = 0, 0.25, 0.5, 0.75$), and compute $\|\Delta Y^K\|_F^2$ corresponding to each p^k .

Step 1-b) Plot $\|\Delta Y^K\|_F^2$ and corresponding FLOPs reduction ($p^k f^k$), and interpolate between the dots, as shown in Fig. 1 (b).

Fig. 1 (a) An illustration of the idea of PRO. In each layer, we try pruning with various pruning ratios and observe the error in the final layer, in order to draw a curve of FLOPs reduction and error. Then, we set error threshold t_{err} , select the layer where the most FLOPs can be reduced at the cost of error of t_{err} (In this case, *layer 1* will be selected.), and eventually perform pruning in the selected layer. We repeat these procedures many times until inference with the pruned model becomes fast enough. (b) The strategy for efficient layer selection. Drawing precise curves is computationally intensive, as it repeatedly requires us to conduct pruning and feed images into the pruned model for observing the error in the final layer. We set p^k , the pruning ratio in the k -th layer, to a few values (In this example, $p^k = 0, 0.25, 0.5, 0.75$), conduct pruning, and observe the error of the final layer. Then, we perform linear interpolation between observed points.

3.4 Algorithm Details

The algorithm that we have implemented is described in Algorithm 1.

In Algorithm 1, we first compute original Y^K (line 2) so that the error $\|\Delta Y^K\|_F^2$ can be computed in the subsequent steps.

Line 4–12 describe the procedures of Step 1). Note that the parameter P is a set of pruning ratio values that we use in Step 1-a). In line 8, we conduct pruning (and reconstruction) based on the x^k -s and Y^k with REAP, CP, or other layer-wise pruning methods.

Line 13–19 correspond to Step 2). We need some additional explanations about this part. The point here is how to set proper value to t_{err} . With too small t_{err} , FLOPs reduction in each layer corresponding to t_{err} will be close to zero, and we cannot prune any neuron. With too large t_{err} , all the neurons in the selected layer will be pruned. In order to avoid these undesired situations, we induce a threshold t_{flops} for FLOPs reduction at each iteration. We first set a very small value to t_{err} (line 13), and select the layer where we can reduce the most FLOPs at the cost of error t_{err} (line 15). If the FLOPs reduction in the selected layer is smaller than t_{flops} , we increase t_{err} a little (line 17) and repeat layer selection (go back to line 14).

Basically, once we prune some neurons in the selected layer, we go back to line 3 and recompute the x -s at line 5. However, in most cases, pruning a small number of neu-

rons in a layer does not significantly change them in subsequent layers because we conduct reconstruction. Therefore, in practice, we (optionally) select more than one layers at line 15. This allows us to set larger value to t_{flops} and to finish pruning ratio optimization efficiently.

In this algorithm, the controlled parameters are P , n , t_{err} , t_{flops} , ϵ . Among these, n and t_{flops} have to be adjusted carefully, because they may affect the accuracy degradation and optimization time significantly. Ideally, we should set n to 1 and t_{flops} to values small enough, e.g. 1/10,000 of the total FLOPs of the original model. However, in order to finish pruning ratio optimization in considerable time, we have to increase t_{flops} and n . Based on some experimental studies, we found out that it is good to set t_{flops} to 1/10 to 1/100 of the total FLOPs, and n to 1/2 to 1/4 of the number of target layers. For the other parameters, $P = \{0, 0.25, 0.5, 0.75\}$, $t_{err} = 10^{-10}$, and $\epsilon = 1.1$ are good in most cases.

4. Experiments

We evaluated PRO with several benchmark datasets and several models. We implemented PRO with Python 3.6.9 and Pytorch 1.0.0. The experiments were done on Intel Core-i9 9900K CPU and NVIDIA Titan RTX GPU. For inference time measurements, we used NVIDIA Jetson Xavier NX, a single board computer designed for neural network inference.

4.1 VGG16 on ImageNet

We first conducted experiments with VGG16 model [29] and ImageNet dataset [30].

4.1.1 Setups

(1) Dataset

ImageNet is a major dataset for image recognition. It has approximately 1.28M training images and 50K test images, and each of them belongs to either of 1,000 classes. All the images were resized so that the shorter side would become 256. Then, 224×224 random crop was applied to the training images, and center crop was applied to the test images. Random horizontal flip was applied to only training images.

For pruning, we used randomly sampled 5,000 images from training dataset. We fed these 5,000 images into the model in order to compute the x -s and the Y -s in Algorithm 1, and performed pruning.

(2) Model

VGG16 [29] is a classification model that is often used for evaluating pruning methods. It has 13 convolutional layers that account for 99% of computational complexity, and 3 fully connected layers that account for 90% of parameters (model size). As we aimed to accelerate inference, we conducted pruning on 12 prunable convolutional layers, and did not conduct pruning in fully connected layers.

Algorithm 1

- 1: **Input:** $\mathcal{M}$: model, $\mathcal{D}$: dataset, P : a set of values that denote pruning ratios, n : the number of selected layers for pruning in each iteration, t_{err} : threshold for the error in the final layer, t_{flops} : threshold for FLOPs reduction, ϵ : scaling rate for tuning t_{err} .
 - 2: Feed $\mathcal{D}$ into $\mathcal{M}$ to compute Y^K .
 - 3: **while** The FLOPs of the pruned model is not small enough **do**
 - 4: **for** $k = 1, \dots, K - 1$ **do**
 - 5: Feed $\mathcal{D}$ into $\mathcal{M}$ to compute $\{x_i^k | i \in \mathcal{Z}^k\}$ and Y^k .
 - 6: **for** $p^k \in P$ **do**
 - 7: $\mathcal{M}' \leftarrow \mathcal{M}$.
 - 8: Perform pruning with pruning ratio p^k on the k -th layer of $\mathcal{M}'$, based on $\{x_i^k | i \in \mathcal{Z}^k\}$ and Y^k .
 - 9: Compute corresponding $\|\Delta Y^k\|_F^2$.
 - 10: **end for**
 - 11: Plot $\|\Delta Y^k\|_F^2$ and corresponding FLOPs reduction ($p^k f^k$), and perform linear interpolation, as shown in Fig. 1.
 - 12: **end for**
 - 13: $t'_{err} \leftarrow t_{err}$.
 - 14: **while** the FLOPs reduction in the selected layer(s) is smaller than t_{flops} **do**
 - 15: Select n layer(s) with the largest FLOPs reduction when $\|\Delta Y^k\|_F^2 = t'_{err}$.
 - 16: Compute pruning ratio p'^k in each selected layer corresponding to the error of t'_{err} .
 - 17: $t'_{err} \leftarrow \epsilon t'_{err}$.
 - 18: **end while**
 - 19: Perform pruning on the selected layer(s) of $\mathcal{M}$, with pruning ratio p'^l in the l -th layer, where l is the index of selected layer.
 - 20: **end while**
-

Table 1 VGG16 on ImageNet. The top-5 accuracy is reported (the greater, the better). In this table, “rt” stands for “retraining”, “Inf. time” represents inference time per image, “uniform” means that the pruning ratio was set to the same value for all the layers. Just for reference, the result of AMC&CP at $\times 0.2$ FLOPs shown in [19] is -1.4% degradation after retraining, which we could not replicate in our experiments.

	FLOPs	Acc. before rt	Acc. after rt	Time for optim.	Inf. time
Baseline	$\times 1$	89.5%		-	40.5 msec
PRO & CP	$\times 0.203$	73.6%	87.8%	71,840 sec	17.3 msec
AMC & CP	$\times 0.219$	49.7%	85.8%	35,181 sec	17.3 msec
uniform & CP	$\times 0.212$	45.1%	86.0%	2,099 sec	16.6 msec
PRO & REAP	$\times 0.200$	80.5%	88.2%	78,026 sec	15.8 msec
AMC & REAP	$\times 0.208$	69.9%	86.9%	40,328 sec	15.1 msec
uniform & REAP	$\times 0.212$	56.2%	87.1%	4,061 sec	16.5 msec

We took a pretrained model provided by [29]’s authors[†], converted it from Caffe model to Pytorch model, and performed pruning. Its top-5 accuracy on ImageNet validation dataset is 89.5%.

Note that the baseline accuracy of VGG16 varies across literatures (e.g. 89.9% in [19], 88.4% in [4], 89.6% in [6]). This might be because of the difference of frameworks (e.g. Caffe and Pytorch), framework version difference, or other unknown experimental setups.

(3) Pruning

The baseline method is AMC [19]. AMC is a RL-based method for pruning ratio optimization. Basically, AMC is combined with CP [3], and we assume that PRO is combined with REAP [14]. For fair comparison of PRO and AMC, we also evaluated their comprehensive combinations (including PRO & CP, AMC & REAP). In addition, we applied REAP and CP with uniform pruning ratio settings in all the layers, without pruning ratio optimization methods.

As shown in Algorithm 1, we have to set several hyperparameters of PRO. In these experiments, we set them as follows: $n = 3$, $t_{err} = 10^{-10}$, $t_{flops} = 2 \times 10^8$ (For reference, original VGG16 has 1.547×10^{10} FLOPs.), $P = \{0, 0.25, 0.5, 0.75\}$, and $\epsilon = 1.1$.

Regarding to AMC, we could not find some important experimental information in [19]. In order to be fair, we evaluated AMC by ourselves using the source codes provided by [19]’s authors^{††}. This Github repository lacks the function for pruning convolutional layers with larger than 1×1 kernel shape, which must have been used for conducting the experiments in [3]. Therefore, we implemented the missing part by ourselves. We also found some fatal bugs in their code and fixed them.

(4) Fine-tuning after pruning

The pruned models were fine-tuned for 10 epochs with 10^{-4} learning rate. The momentum was set to 0.9, the mini-batch size was set to 128, and the dropout rate in the fully connected layers was set to 0.5. For the rest of the training setups, we followed [29].

4.1.2 Results

The results are summarized in Table 1, and the discussions are as follows.

(1) Comparison to the case of uniform pruning ratios

Compared to the case of uniform pruning ratios, we could make the accuracy degradation much smaller by using PRO. For example, the accuracy degradation was 9% by using PRO & REAP while the accuracy dropped by 33.3% with uniform pruning ratios, at approximately $\times 0.2$ FLOPs ratio, before retraining.

The accuracy of the pruned model after retraining was also better when using PRO. By using PRO, the pruning ratio for each layer has been optimized. Therefore, we can preserve the accuracy of the pruned models well, which means that we can start retraining with the models that have been less damaged.

(2) Comparison to AMC

We then discuss the comparison of PRO and AMC. As shown in Table 1, PRO & CP could outperform AMC & CP significantly. PRO suffers 15.9% accuracy degradation at $\times 0.203$ FLOPs ratio without retraining, while AMC suffers 39.8% degradation at $\times 0.219$ FLOPs ratio. After retraining, PRO still suffers smaller degradation than AMC by 2.0%. This trend is consistent with the cases of using REAP for pruning.

(3) Analyses on optimized pruning ratios

Figure 2 shows the pruning ratio in each layer. The trend of both PRO and AMC is that the pruning ratios are higher in the layers closer to the input side and are lower in the layers closer to the output side.

A remarkable observation is that PRO does not prune a lot in *Conv5-1* and *Conv5-2* layers, while AMC does. Actually, it is known that these layers are not redundant and pruning them leads to significant degradation, as reported in [3], [4]. PRO could successfully figure out that these layers should not be pruned and eventually set zero or very low pruning ratios to them. On the other hand, AMC pruned a lot in these layers, which ended up in significant degradation.

We also investigated how the error in the final layer

[†]https://www.robots.ox.ac.uk/~vgg/research/very_deep/

^{††}<https://github.com/mit-han-lab/amc>

Fig. 2 Results of pruning ratio optimization for VGG16. Both PRO and AMC tend to set higher pruning ratio to the layers on the input side and lower pruning ratio to the layers on the output side. The difference is that PRO does not prune Conv5-1 and Conv5-2 layers a lot, while AMC does. As reported in several literatures, such as [3], [4], pruning these layers leads to significant degradation. And our PRO successfully avoids pruning these layers.

Fig. 3 (a) Relationship of the error in the final layer and pruning ratio in each layer. (b) Relationship of the error in the final layer and FLOPs reduction in each layer, which we actually use for selecting the layer to be pruned.

responses to the pruning ratio in each layer. We used REAP to prune *Conv1-1*, *Conv2-1*, *Conv3-1*, *Conv4-1*, and *Conv5-1* layers, with various pruning ratios, and observed the error in the final layer. The result is shown in Fig. 3.

Figure 3(a) shows the relationship of the error in the final layer and the pruning ratio in each layer, and Fig. 3(b) shows a similar graph with FLOPs reduction in the horizontal axis. We can see clearly different trends between the layers. In *Conv5-1*, the error increases more rapidly than the other layers. Thus, by observing the relationship of pruning ratio (FLOPs reduction) and the error in the final layer directly, we can get the insight that we should not perform pruning a lot in *Conv5-1*.

Then, why did AMC set large pruning ratios to *Conv5-1* and *Conv5-2* layers? We suppose that AMC's RL-based algorithm was simply not capable of evaluating the redundancy of the layers. As it performs pruning in all the layers simultaneously, it cannot evaluate the impact of the pruning

ratio in each layer on the accuracy separately.

Apart from the discussions above, there might be an argument that more retraining can improve the accuracy of the pruned models, canceling the advantage in accuracy preservation of the proposed method. Therefore, we performed extra training for the model pruned with AMC & CP for 10 more epochs (thus, 20 epochs in total), however, it achieved 86.8%, which is still worse than PRO.

(4) Inference time measurements

In the rightmost column of Table 1, we show the inference time (msec per image) on NVIDIA Jetson Xavier NX. The pruned models are approximately 2.5 times faster than the original models. These results support the practical effectiveness of pruning. An interesting observation is that the order of FLOPs does not necessarily reflect the order of inference time. It seems the architectural difference of the pruned models affects the efficiency of computational resource us-

Table 2 The results with the ResNet-56 model on the CIFAR-10 dataset. The top-1 accuracy is reported (the greater, the better). Just for reference, the results of AMC&CP at $\times 0.5$ FLOPs shown in [19] are -2.7% degradation before retraining and -0.9% degradation after retraining, which we could not replicate in our experiments.

	FLOPs	Acc. before rt	Acc. after rt	Time for optim.	Inf. time
Baseline	$\times 1$	92.8%		-	1.73 msec
PRO & CP	$\times 0.498$	90.0%	92.0%	3,800 sec	1.43 msec
AMC & CP	$\times 0.501$	79.0%	91.4%	6,885 sec	1.63 msec
uniform & CP	$\times 0.510$	71.4%	91.0%	550 sec	1.43 msec
PRO & REAP	$\times 0.500$	90.6%	92.1%	4,237 sec	1.60 msec
AMC & REAP	$\times 0.503$	86.1%	91.9%	7,752 sec	1.39 msec
uniform & REAP	$\times 0.510$	86.3%	91.2%	611 sec	1.44 msec

age.

4.2 ResNet-56 on CIFAR-10

We also conducted experiments with ResNet-56 model [1] and CIFAR-10 dataset [31].

4.2.1 Setups

CIFAR-10 is a major dataset for image recognition. It has 50K training images and 10K test images, and each of them is categorized in either of 10 classes. We randomly selected 4,096 images from the training dataset for performing pruning. The training images were padded by 4 pixels in each side, and randomly cropped at 32×32 . The test images were used as they were. The pruned models were retrained for 100 epochs, beginning with the learning rate 10^{-2} and dividing it by 10 at 50 epochs.

In these experiments, we set PRO's parameters as follows: $n = 27$, $t_{err} = 10^{-10}$, $t_{flops} = 1 \times 10^7$ (ResNet-56 has 1.254×10^8 FLOPs and 54 prunable convolutional layers), $P = \{0, 0.25, 0.5, 0.75\}$, and $\epsilon = 1.1$.

4.2.2 Results

The results are summarized in Table 2. Similarly with the former experiments, we could outperform AMC in the accuracy of the pruned model. Compared with the case of using REAP/CP with the uniform pruning ratios, using PRO significantly improved the accuracy of the pruned model. AMC & CP suffers 6.5% accuracy degradation while PRO & CP suffers only 2.8%. PRO & CP still outperforms AMC & CP by 0.6% after retraining. By using REAP instead of CP with PRO, the results improved even more.

Another remarkable observation is that we took only 3,800 seconds to determine the pruning ratio for each layer, while AMC needs approximately 6,900 seconds. Our greedy strategy for optimizing pruning ratios can be more efficient than the RL-based one, depending on the models and experimental setups.

We could accelerate the inference by pruning, although not very significantly. Because the input resolution of CIFAR-10 images is only 32×32 , the computation in each layer can be parallelized efficiently even without pruning. Similarly with VGG16's cases, there is inconsistency of the

order of FLOPs and that of inference time.

5. Conclusion

In this paper, we presented PRO, a method for optimizing pruning ratio in each layer of CNN models. Some of the layer-wise pruning methods are theoretically sound and better than conventional holistic pruning methods. As the CNN models usually have a lot of layers, we need to tune the pruning ratio in each layer for more efficient pruning. With PRO, we can tune pruning ratios so as to minimize the error in the final layer of the model. PRO is suited to be combined with REAP, even though other layer-wise pruning methods could be options, because REAP is the best method in terms of preserving layer-wise errors. The experimental results verify the effectiveness of PRO.

We observed that the order of FLOPs of the pruned models does not necessarily reflect the order of actual inference speed. The analysis on this observation and its countermeasures will be our future works.

Acknowledgements

This work was supported by KAKENHI 19K12020, Grant-in-Aid for Scientific Research (C).

References

- [1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp.770–778, 2016.
- [2] Y. LeCun, J.S. Denker, and S.A. Solla, "Optimal brain damage," Proc. Conference on Neural Information Processing Systems (NIPS), pp.598–605, 1990.
- [3] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," Proc. IEEE International Conference on Computer Vision (ICCV), pp.1389–1397, 2017.
- [4] J.-H. Luo, J. Wu, and W. Lin, "ThiNet: A filter level pruning method for deep neural network compression," Proc. IEEE International Conference on Computer Vision (ICCV), pp.5068–5076, 2017.
- [5] Z. Zhuang, M. Tan, B. Zhuang, J. Liu, Y. Guo, Q. Wu, J. Huang, and J. Zhu, "Discrimination-aware channel pruning for deep neural networks," Proc. Conference on Neural Information Processing Systems (NIPS), pp.881–892, 2018.
- [6] H. Wang, Q. Zhang, Y. Wang, and H. Hu, "Structured probabilistic pruning for convolutional neural network acceleration," Proc. British Machine Vision Conference (BMVC), 149, 2018.
- [7] X. Dong, S. Chen, and S. Pan, "Learning to prune deep neural

- networks via layer-wise optimal brain surgeon,” Proc. Conference on Neural Information Processing Systems (NIPS), pp.4857–4867, 2017.
- [8] A. Aghasi, A. Abdi, N. Nguyen, and J. Romberg, “Net-trim: Convex pruning of deep neural networks with performance guarantee,” Proc. Conference on Neural Information Processing Systems (NIPS), pp.3177–3186, 2017.
- [9] Y. Guo, A. Yao, and Y. Chen, “Dynamic network surgery for efficient DNNs,” Advances in Neural Information Processing Systems (NIPS), 2016.
- [10] B. Hassibi, D.G. Stork, and G.J. Wolff, “Optimal brain surgeon and general network pruning,” IEEE International Conference on Neural Networks, pp.293–299, 1993.
- [11] S. Han, H. Mao, and W.J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding,” Proc. International Conference on Learning Representations (ICLR), 10 2016.
- [12] T. He, Y. Fan, Y. Qian, T. Tan, and K. Yu, “Reshaping deep neural network for fast decoding by node-pruning,” Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp.245–249, 2014.
- [13] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, “Learning efficient convolutional networks through network slimming,” Proc. IEEE International Conference on Computer Vision (ICCV), pp.2755–2763, 2017.
- [14] K. Kamma and T. Wada, “Reconstruction error aware pruning for accelerating neural networks,” Proc. International Symposium on Visual Computing (ISVC), pp.59–72, 2019.
- [15] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, “Pruning convolutional neural networks for resource efficient transfer learning,” Proc. International Conference on Learning Representations (ICLR), 2015.
- [16] P. Molchanov, A. Mallya, S. Tyree, I. Frosio, and J. Kautz, “Importance estimation for neural network pruning,” Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp.11256–11264, June 2019.
- [17] L. Theis, I. Korshunova, A. Tejani, and F. Huszár, “Faster gaze prediction with dense networks and Fisher pruning,” arXiv:1801.05787, 2018.
- [18] K. Kamma, Y. Isoda, S. Inoue, and T. Wada, “Behavior-based compression for convolutional neural networks,” Proc. International Conference on Image Analysis and Recognition (ICIAR), pp.427–439, 2019.
- [19] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, “AMC: AutoML for model compression and acceleration on mobile devices,” Proc. European Conference on Computer Vision (ECCV), pp.815–832, 2018.
- [20] J. Xue, J. Li, and Y. Gong, “Restructuring of deep neural network acoustic models with singular value decomposition,” Proc. Annual Conference of the International Speech Communication Association, INTERSPEECH, pp.2365–2369, 2013.
- [21] J. Ye, L. Wang, G. Li, D. Chen, S. Zhe, X. Chu, and Z. Xu, “Learning compact recurrent neural networks with block-term tensor decomposition,” Proc. Conference on Computer Vision and Pattern Recognition (CVPR), pp.9378–9387, 2018.
- [22] X. Yu, T. Liu, X. Wang, and D. Tao, “On compressing deep models by low rank and sparse decomposition,” Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp.67–76, 2017.
- [23] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Pensky, “Sparse convolutional neural networks,” Proc. Conference on Computer Vision and Pattern Recognition (CVPR), pp.806–814, 2015.
- [24] G. Xie, J. Wang, T. Zhang, J. Lai, R. Hong, and G.-J. Qi, “Interleaved structured sparse convolutional neural networks,” Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp.8847–8856, 2018.
- [25] M. Ren, A. Pokrovsky, B. Yang, and R. Urtasun, “SBNet: Sparse blocks network for fast inference,” Proc. IEEE Conference on Com-

- puter Vision and Pattern Recognition (CVPR), pp.8711–8720, 2018.
- [26] E. Kim, C. Ahn, and S. Oh, “NestedNet: Learning nested sparse structures in deep neural networks,” Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp.8669–8678, 2018.
- [27] M. Courbariaux, Y. Bengio, and J. David, “BinaryConnect: Training deep neural networks with binary weights during propagations,” Proc. Conference on Neural Information Processing Systems (NIPS), pp.3123–3131, 2015.
- [28] A. Zhou, A. Yao, K. Wang, and Y. Chen, “Explicit loss-error-aware quantization for low-bit deep neural networks,” Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp.9426–9435, 2018.
- [29] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” Proc. International Conference on Learning Representations (ICLR), 2015.
- [30] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “ImageNet: A large-scale hierarchical image database,” Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp.248–255, 2009.
- [31] A. Krizhevsky, V. Nair, and G. Hinton, “CIFAR-10 (Canadian Institute for Advanced Research),” <https://academictorrents.com> (accessed on 08/01/2021).

Appendix A: Reconstruction Error Aware Pruning (REAP)

In a fully connected layer, the forward propagation formula is given by

$$Y^k = \sum_{i \in \mathcal{Z}^k} \mathbf{x}_i^k \mathbf{w}_i^{k\top}, \quad (\text{A} \cdot 1)$$

The goal of REAP is to reduce the number of neurons to the desired number while keeping Y^k as unchanged as possible. This can be formulated as follows.

$$\mathcal{A}^{k*} = \operatorname{argmin}_{\mathcal{A}^k} \min_{\mathbf{w}_i^k} \left\| Y^k - \sum_{i \in \mathcal{Z}^k \setminus \mathcal{A}^k} \mathbf{x}_i^k \mathbf{w}_i^{k\top} \right\|_F^2. \quad (\text{A} \cdot 2)$$

where $\mathcal{A}^k \subset \mathcal{Z}^k$ denotes the set of pruned neurons’ indices and $\setminus$ denotes set difference. By solving this, we can select the neurons whose outputs can be reconstructed the best from the remaining ones’ outputs.

Because it is difficult to solve this problem analytically, REAP solve it in a greedy fashion. We select the neuron to be pruned based on the reconstruction error of Y^k :

$$j^* = \operatorname{argmin}_j \min_{\mathbf{w}_i^k} \left\| Y^k - \sum_{i \in \mathcal{Z}^k \setminus \{j\}} \mathbf{x}_i^k \mathbf{w}_i^{k\top} \right\|_F^2. \quad (\text{A} \cdot 3)$$

Then, we update the weights of the remaining neurons by least squares method so as to reconstruct the outputs.

$$\{\mathbf{w}_i^{k*}\} = \operatorname{argmin}_{\mathbf{w}_i^k} \left\| Y^k - \sum_{i \in \mathcal{Z}^k \setminus \{j\}} \mathbf{x}_i^k \mathbf{w}_i^{k\top} \right\|_F^2. \quad (\text{A} \cdot 4)$$

For solving Eq. (A. 3), we first need to solve Eq. (A. 4)

by least squares method for each j , which is computationally intensive. In [14], we present a biorthogonal system-based algorithm that can solve Eq. (A·4) for each j in one-shot.

REAP can be applied to the convolutional layers with a minor modification, which is also explained in [14].

Appendix B: Channel Pruning (CP)

CP [3] is a layer-wise pruning method that conducts reconstruction with least squares method. CP's neuron selection is different from REAP. They select the neurons to be pruned by solving the following Lasso Regression problem:

$$\begin{aligned} \beta^{k*} = \underset{\beta^k}{\operatorname{argmin}} & \left\| Y^k - \sum_{i \in \mathcal{Z}^k} \beta_i^k \mathbf{x}_i^k \mathbf{w}_i^{k\top} \right\|_F^2 + \lambda \|\beta^k\|_1 \\ & \text{subject to } \|\beta^k\|_0 \leq \text{threshold}, \end{aligned} \quad (\text{A} \cdot 5)$$

where β^k denotes a vector used for neuron selection. If $\beta_j = 0$, the j -th neuron can be pruned. Then, the reconstruction is done in the same way with ours.

The weakness of CP is that it selects the neurons to be pruned based on the error *before* reconstruction, which does not guarantee the minimal error *after* reconstruction. Because REAP selects the neurons to be pruned based on the error after reconstruction, REAP performs better than CP.

Toshikazu Wada received the D.Eng. degree in applied electronics from Tokyo Institute of Technology, in 1990. He is currently a professor in Faculty of Systems Engineering, Wakayama University. His research interests include Pattern Recognition, Computer Vision, Artificial Intelligence, and Deep Learning. He received the Marr Prize at the International Conference on Computer Vision in 1995, the Yamashita Memorial Research Award from the Information Processing Society Japan (IPSJ) in

1997, Excellent Paper Award from Institute of Electronics, Information, and Communication Engineers (IEICE), Japan in 1999, and Excellent Paper Award from The Institute of Systems Control and Information Engineers (ISCIE) in 2007. He is a member of the IPSJ, the ISCIE, and the IEEE.

Koji Kamma received the master degree in chemical engineering from the University of Tokyo, in 2014. He worked for a international heavy industries company for 3 years and a start-up web marketing company for a year. He is currently a Ph.D. student in Faculty of Systems Engineering, Wakayama University. His research interests include Computer Vision, Pattern Recognition, and particularly, Deep Learning.

Sarimu Inoue received the master degree of Systems Engineering from Wakayama University in 2020. He currently works at DENSO WAVE INCORPORATED. His research interests include Pattern Recognition, Computer Vision, Artificial Intelligence, and Deep Learning.