PAPER

REAP: A Method for Pruning Convolutional Neural Networks with Performance Preservation*

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

SUMMARY This paper presents a pruning method, Reconstruction Error Aware Pruning (REAP), to reduce the redundancy of convolutional neural network models for accelerating their inference. In REAP, we have the following steps: 1) Prune the channels whose outputs are redundant and can be reconstructed from the outputs of other channels in each convolutional layer; 2) Update the weights of the remaining channels by least squares method so as to compensate the error caused by pruning. This is how we compress and accelerate the models that are initially large and slow with little degradation. The ability of REAP to maintain the model performances saves us lots of time and labors for retraining the pruned models. The challenge of REAP is the computational cost for selecting the channels to be pruned. For selecting the channels, we need to solve a huge number of least squares problems. We have developed an efficient algorithm based on biorthogonal system to obtain the solutions of those least squares problems. In the experiments, we show that REAP can conduct pruning with smaller sacrifice of the model performances than several existing methods including the previously state-of-the-art one.

key words: pruning, REAP, biorthogonal system

1. Introduction

In various Computer Vision tasks, Convolutional Neural Networks (CNNs) have been showing the record-breaking performances. For example, some recent CNN models conduct object detection with high accuracy, and are expected to be used in assorted applications. Although, can we use those models as they are in the embedded systems, such as an object detection and classification system for in-vehicle camera? At this point, we need strong computational resources to use the large CNN models. For the future applications, it is important to produce fast CNN models without compromising their performances.

A major approach for producing a fast CNN model is to conduct pruning on a pretrained large model. Some early works aim for holistic network optimization based on the unified criteria across the layers, e.g. magnitude based criteria [2], loss function based one [3], and so on. However, most of these methods need a lot of mathematical approximations and/or use the heuristic criteria which do not guarantee the optimality of the pruning results. On the other hand, some recent works propose the pruning methods that focus on layer-wise optimization based on theoret-

ically sound criteria [4]–[7] and those methods have turned out to be more effective than the earlier holistic methods. The weakness of the layer-wise methods is that the proper pruning ratio (the ratio of the number of the channels to be pruned) for each layer is not trivial and has to be determined manually. Although, in [8], reinforcement learning is employed to determine the pruning ratio for each layer, which enables us to achieve holistic network optimization using the layer-wise methods. Therefore, the layer-wise methods are becoming more important recently.

In this paper, we present a novel layer-wise pruning method, Reconstruction Error Aware Pruning (REAP), by extending Neuro-Unification (NU) [9] that we proposed in our previous work. The idea of NU is to unify a pair of neurons having highly correlated outputs so that the remaining one can reconstruct the outputs of the pruned one. We extend this idea from one-to-one unification to one-to-many. In REAP, we prune a neuron and reconstruct its outputs from all the remaining neuron's outputs by solving the least squares problem.

However, our new approach requires tremendous computational cost. In REAP, we need to select the neuron to be pruned so that the output error is minimized. For this purpose, we have to repeat the following steps for each neuron: 1) prune a neuron, 2) solve the least squares problem to compute the error after reconstruction. This is computationally expensive when we have lots of neurons. Therefore, we also propose an efficient algorithm for this problem.

We conduct the experiments with several CNN models and datasets. The results show that our method can conduct pruning with smaller sacrifice of the model performances than the existing methods.

2. Related Works

The existing approaches on accelerating the CNN models can be categorized into 5 groups: 1) Factorization, 2) Sparsification, 3) Quantization, 4) Weight pruning, and 5) Channel pruning.

Factorization. The most fundamental method in this group is presented in [10]. They apply SVD 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 a $m \times n$ matrix is approximated by the product of a $m \times o$ matrix and a $o \times n$ matrix. If $o \ll m, n$, the number of the parameters reduces

Manuscript received March 9, 2020.

Manuscript revised September 11, 2020.

Manuscript publicized October 2, 2020.

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

*This paper is the updated version of [1].

a) E-mail: kanma@vrl.sys.wakayama-u.ac.jp

DOI: 10.1587/transinf.2020EDP7049

from mn to $(m+n)o$. Some other methods [11], [12] also belong here.

Sparsification. These methods make the weight tensors sparse by fine-tuning the models with L1 regularization [13]–[15]. The theoretical weak point of sparsification is that L1 regularization shifts the global minimum of the cost function and sacrifices the model performances. Besides, in order to take the advantage of the sparsified models, the special hardware and libraries are required for executing the computations on only non-zero weights.

Quantization. The methods in this group reduce the redundancy of each bit-wise operation [16]–[18], e.g. changing the floating point precision from 32-bit to 8-bit. The quantized models require the special hardware and libraries to be deployed.

Weight pruning. The idea of weight pruning is to remove the weights based on their saliencies [3], [7], [14], [17]. Similarly with sparsification methods, these methods aim to reduce the number of non-zero weights and do not aim to reduce the sizes of the weight tensors, therefore, the special hardware and libraries are required when the pruned models are deployed.

Neuron/Channel pruning. The methods in this group prune the whole weights that belong to a certain neuron/channel [4]–[6], [19], [20]. Therefore, the size of the weight tensor gets smaller, which results in saving the computational complexity for inference without any special hardware and libraries.

Our method belongs to the channel pruning group. Nonetheless, it is possible to combine our method and the ones from other groups (e.g. quantization, factorization) to achieve further acceleration.

3. From NU to REAP

We explain NU, and show how this method can be improved by our new strategy. Then, we show the algorithms to accelerate the computation of REAP. We also explain how to apply REAP to convolutional layers.

3.1 Neuro-Unification (NU)

Let c and C denote the numbers of neurons in a layer and the next layer, and N denote the number of the input images. The forward propagation is formulated as

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

where $\mathbf{x}_i \in \mathbb{R}^N$ denotes the outputs of the i -th neuron corresponding to N input images, $\mathbf{w}_i \in \mathbb{R}^C$ denotes the weights going from the i -th neuron to the ones in the next layer, $Y \in \mathbb{R}^{N \times C}$ denotes the inner activation levels in the next layer, and $Z = \{1, \dots, c\}$ is the set of neuron indices. The goal is to reduce the number of neurons to the desired number while keeping Y as unchanged as possible.

How to unify a given pair of neurons. When we have a pair of neurons having linearly dependent outputs, we can

prune one and merge it to the other one without error. For instance, if $\mathbf{x}_j = \alpha \mathbf{x}_i$ holds, we prune the j -th neuron and update the i -th neuron's weights as

$$\mathbf{w}'_i \leftarrow \mathbf{w}_i + \alpha \mathbf{w}_j. \quad (2)$$

Then, the error of Y is given by

$$\Delta Y = \mathbf{x}_i \mathbf{w}'_i{}^\top - \mathbf{x}_i \mathbf{w}_i{}^\top - \mathbf{x}_j \mathbf{w}_j{}^\top = 0. \quad (3)$$

This shows that Y is preserved because the i -th neuron compensates the error caused by pruning the j -th neuron.

In the case of unifying the neurons having linearly independent outputs, we first approximate $\mathbf{x}_j$ by a vector which is linearly dependent with $\mathbf{x}_i$:

$$\mathbf{x}_j \simeq \alpha \mathbf{x}_i. \quad (4)$$

Then, we can prune the j -th neuron and update $\mathbf{w}_i$ by using Eq. (2). The question is how to determine α . The error of Y is given by

$$\Delta Y = \mathbf{x}_i \mathbf{w}'_i{}^\top - \mathbf{x}_i \mathbf{w}_i{}^\top - \mathbf{x}_j \mathbf{w}_j{}^\top = -(\mathbf{x}_j - \alpha \mathbf{x}_i) \mathbf{w}_j{}^\top. \quad (5)$$

Since we want to minimize the output error, we have

$$\begin{aligned} \alpha^* &= \underset{\alpha}{\operatorname{argmin}} \left\| -(\mathbf{x}_j - \alpha \mathbf{x}_i) \mathbf{w}_j{}^\top \right\|_F^2 \\ &= \underset{\alpha}{\operatorname{argmin}} \left\| \mathbf{x}_j - \alpha \mathbf{x}_i \right\|^2 \end{aligned} \quad (6)$$

Note that we omitted $\mathbf{w}_j$ in Eq. (6) as it contains only constants. Thus, we can determine α by computing orthogonal projection of $\mathbf{x}_j$ onto $\mathbf{x}_i$.

How to select the pair of neurons to be unified. We have to select the neurons to be unified when we have a lot of possible neuron pairs. As we want to minimize the error of Y , we simply select the neuron pair that minimizes the output error:

$$(i^*, j^*) = \underset{(i,j)}{\operatorname{argmin}} \left\| (\mathbf{x}_j - \alpha \mathbf{x}_i) \mathbf{w}_j{}^\top \right\|_F^2 \quad s.t. \ i \neq j. \quad (7)$$

Note that we cannot omit $\mathbf{w}_j$ here, because we have not fixed j yet. The neurons are selected one by one in a greedy fashion based on this criteria.

3.2 Reconstruction Error Aware Pruning (REAP)

We extend the idea of NU from one-to-one unification to one-to-many.

How to prune a neuron and reconstruct its outputs.

In NU, the output of the pruned neuron is reconstructed from another neuron. In REAP, we use all the remaining neurons for reconstruction, as shown in Fig. 1. This can be formulated by

$$\{v_{ji}^* | i \in Z \setminus \{j\}\} = \underset{v_{ji}}{\operatorname{argmin}} \left\| \mathbf{x}_j - \sum_{i \in Z \setminus \{j\}} v_{ji} \mathbf{x}_i \right\|^2, \quad (8)$$

Fig. 1 Concepts of NU and REAP. Left: original network. Center: network pruned with NU. Only one neuron is used for reconstruction. Right: network pruned with REAP. All other neurons are used for reconstructing the outputs of the pruned one.

where the v -s denote the coefficients for reconstructing $\mathbf{x}_j$ from the other $\mathbf{x}$ -s. Similarly with Eq. (2), the weights of the remaining neurons are updated as follows for each $i \in Z \setminus \{j\}$.

$$\mathbf{w}'_i = \mathbf{w}_i + v_{ji}^* \mathbf{w}_j. \quad (9)$$

How to select the neuron to be pruned. We should select the neuron to be pruned so as to minimize the reconstruction error of Y :

$$\begin{aligned} j^* &= \underset{j}{\operatorname{argmin}} \left\| Y - \sum_{i \in Z \setminus \{j\}} \mathbf{x}_i \mathbf{w}'_i \right\|_F^2 \\ &= \underset{j}{\operatorname{argmin}} \left\| Y - \sum_{i \in Z \setminus \{j\}} \mathbf{x}_i (\mathbf{w}_i + v_{ji}^* \mathbf{w}_j) \right\|_F^2. \end{aligned} \quad (10)$$

We prune the neurons one by one in a greedy fashion so that the error of Y is as small as possible.

3.3 Algorithm for Efficient Computation

The problem of REAP is its computational cost. In order to solve Eq. (10), we need to solve Eq. (8) for each j , which is computationally expensive. Therefore, we have developed some efficient algorithms.

3.3.1 How to Solve Eq. (8) for Each j in One Shot.

We solve Eq. (8) for each j in *one shot* by using biorthogonal system.

Let $\mathbf{r}_j$ represent the residual of $\mathbf{x}_j$ reconstructed from the other $\mathbf{x}$ -s:

$$\mathbf{r}_j = \mathbf{x}_j - \sum_{i \in Z \setminus \{j\}} v_{ji}^* \mathbf{x}_i. \quad (11)$$

As $\mathbf{r}_j$ is the residual of $\mathbf{x}_j$, $\mathbf{r}_j$ is orthogonal to all other $\mathbf{x}$ -s. In other words, $\mathbf{r}_j$ is orthogonal to the subspace spanned by $\{\mathbf{x}_i | i \in Z \setminus \{j\}\}$, as shown in Fig. 2.

We compute $\mathbf{r}_j$ by using biorthogonal system. Let $\{\bar{\mathbf{x}}_i | i \in Z\}$ denote the dual bases of $\{\mathbf{x}_i | i \in Z\}$. The biorthogonal system is defined by

$$\langle \mathbf{x}_j, \bar{\mathbf{x}}_i \rangle = \begin{cases} 1 & (i = j) \\ 0 & (\text{otherwise}) \end{cases} \quad (12)$$

Fig. 2 Illustration of the projection of $\mathbf{x}_j$ onto a subspace $U(Z \setminus \{j\})$ spanned by $\{\mathbf{x}_i | i \in Z \setminus \{j\}\}$. Computing the orthogonal projection of $\mathbf{x}_j$ onto $U(Z \setminus \{j\})$ is equivalent to solving the problem of reconstructing $\mathbf{x}_j$ from $\{\mathbf{x}_i | i \in Z \setminus \{j\}\}$ by the least squares method. The residual $\mathbf{r}_j$ is linearly dependent on $\bar{\mathbf{x}}_j$, the dual basis for $\mathbf{x}_j$.

where $\langle \cdot, \cdot \rangle$ denotes inner product. The biorthogonal expansion for $\mathbf{r}_j$ is given by

$$\mathbf{r}_j = \sum_{i \in Z} \langle \mathbf{r}_j, \mathbf{x}_i \rangle \bar{\mathbf{x}}_i. \quad (13)$$

Obviously, $\langle \mathbf{r}_j, \mathbf{x}_i \rangle = 0$ holds for each $i \in Z \setminus \{j\}$, because $\mathbf{r}_j$ is orthogonal to the subspace spanned by $\{\mathbf{x}_i | i \in Z \setminus \{j\}\}$. Thus, Eq. (13) can be rewritten as

$$\mathbf{r}_j = \langle \mathbf{r}_j, \mathbf{x}_j \rangle \bar{\mathbf{x}}_j + \sum_{i \in Z \setminus \{j\}} \langle \mathbf{r}_j, \mathbf{x}_i \rangle \bar{\mathbf{x}}_i = \langle \mathbf{r}_j, \mathbf{x}_j \rangle \bar{\mathbf{x}}_j. \quad (14)$$

Equation (14) means that $\mathbf{r}_j$ is linearly dependent on $\bar{\mathbf{x}}_j$. Therefore, we can obtain $\mathbf{r}_j$ by computing the orthogonal projection of $\mathbf{x}_j$ onto $\bar{\mathbf{x}}_j$, as shown in Fig. 2. Thus, the following must hold:

$$\mathbf{r}_j = \frac{\langle \mathbf{x}_j, \bar{\mathbf{x}}_j \rangle}{\|\bar{\mathbf{x}}_j\|^2} \bar{\mathbf{x}}_j = \frac{\bar{\mathbf{x}}_j}{\|\bar{\mathbf{x}}_j\|^2}. \quad (15)$$

By using Eq. (15), we can compute $\mathbf{r}_j$ for each $j \in Z$ in one-shot. Let $X = [\mathbf{x}_1 \cdots \mathbf{x}_c]$ and $\bar{X} = [\bar{\mathbf{x}}_1 \cdots \bar{\mathbf{x}}_c]$. By definition of dual bases, $\bar{X}$ can be computed as

$$\bar{X} = (X^g)^\top, \quad (16)$$

where X^g denotes the generalized inverse of X . Let D denote the diagonal matrix whose (i, i) entry is given by $1/\|\bar{\mathbf{x}}_i\|^2$, and $R = [\mathbf{r}_1 \cdots \mathbf{r}_c]$. Since we have Eq. (15), we also have

$$R = \bar{X} D. \quad (17)$$

Let V^* denote a matrix, and its (j, i) entry is v_{ji}^* . Because we

have Eq. (11), the following must hold:

$$X = R + XV^{*\top}. \quad (18)$$

Then, we have

$$V^{*\top} = I - X^g R \quad (19)$$

This is equivalent to the solution of Eq. (8) for each j .

Substitute approach. We have developed another algorithm for computing the reconstruction error of each neuron by using Gram-Schmidt process. The detail of this substitute approach is provided in Appendix A.

3.3.2 How to Solve Eq. (8) when We Prune Another Neuron

Assume that we have pruned the j -th neuron. When we prune another neuron, we just need to repeat the same procedures mentioned in Sect. 3.3 with the remaining neurons. We have to solve the following problem for each k .

$$\{u_{ki}^* | i \in Z \setminus \{j, k\}\} = \underset{u_{ki}}{\operatorname{argmin}} \left\| \mathbf{x}_k - \sum_{i \in Z \setminus \{j, k\}} u_{ki} \mathbf{x}_i \right\|_F^2, \quad (20)$$

Then, we solve the following problem for selecting the next neuron to be pruned:

$$k^* = \underset{k}{\operatorname{argmin}} \left\| Y - \sum_{i \in Z \setminus \{j, k\}} \mathbf{x}_i (\mathbf{w}'_i + u_{ki}^* \mathbf{w}'_k)^\top \right\|_F^2. \quad (21)$$

As we already have the $\mathbf{w}'$ -s, we only need to compute the u^* -s for solving Eq. (21). We may use biorthogonal system based algorithm again for solving Eq. (20). Although, by using the solution of Eq. (8), Eq. (20) can be solved even faster. We already have

$$\mathbf{x}_j = \mathbf{r}_j + v_{jk}^* \mathbf{x}_k + \sum_{i \in Z \setminus \{j, k\}} v_{ji}^* \mathbf{x}_i, \quad (22)$$

$$\mathbf{x}_k = \mathbf{r}_k + v_{kj}^* \mathbf{x}_j + \sum_{i \in Z \setminus \{j, k\}} v_{ki}^* \mathbf{x}_i. \quad (23)$$

After pruning both the j -th and the k -th neurons, we cannot use $\mathbf{x}_j$ for reconstructing $\mathbf{x}_k$. Thus, we substitute Eq. (22) to Eq. (23) and get

$$\mathbf{x}_k = \frac{\mathbf{r}_k + v_{kj}^* \mathbf{r}_j}{1 - v_{jk}^* v_{kj}^*} + \sum_{i \in Z \setminus \{j, k\}} \frac{v_{ki}^* + v_{kj}^* v_{ji}^*}{1 - v_{jk}^* v_{kj}^*} \mathbf{x}_i. \quad (24)$$

Obviously, we have $\langle \mathbf{x}_i, \mathbf{r}_j \rangle = 0$ and $\langle \mathbf{x}_i, \mathbf{r}_k \rangle = 0$ for each $i \in Z \setminus \{j, k\}$. Therefore, $\langle \mathbf{x}_i, \mathbf{r}_k + v_{kj}^* \mathbf{r}_j \rangle = 0$ holds for each $i \in Z \setminus \{j, k\}$, which means the first term of the RHS of Eq. (24) denotes the residual of $\mathbf{x}_k$ reconstructed from $\{\mathbf{x}_i | i \in Z \setminus \{j, k\}\}$ and the second term denotes the projection of $\mathbf{x}_k$ onto the subspace spanned by $\{\mathbf{x}_i | i \in Z \setminus \{j, k\}\}$. Thus, the coefficients of the $\mathbf{x}$ -s in the second term is equivalent to the solution of Eq. (20):

$$u_{ki}^* = \frac{v_{ki}^* + v_{kj}^* v_{ji}^*}{1 - v_{jk}^* v_{kj}^*}. \quad (25)$$

As above, we can compute u^* -s without directly solving least squares problems. After computing u^* -s, we can solve Eq. (21) easily.

3.3.3 Algorithm

To sum up, our algorithm can be described as below.

1. Initialize $Z' \leftarrow Z$, compute v_{ji}^* and for each (i, j) .
2. Solve Eq. (10) to select j and update $Z' \leftarrow Z' \setminus \{j\}$, $\mathbf{w}'_i \leftarrow \mathbf{w}'_i + v_{ji}^* \mathbf{w}'_j$ for each $i \in Z'$, $v_{ki}^* \leftarrow (v_{ki}^* + v_{kj}^* v_{ji}^*) / (1 - v_{jk}^* v_{kj}^*)$ for each $i, k \in Z', i \neq k$.
3. If $|Z'| > c'$ (c' is the desired number of the neurons), go to step 2.

3.4 Applying REAP to Convolutional Layers

REAP can be applied to the convolutional layers with a minor modification. By expanding the feature maps and the kernels in the convolutional layer into matrices, we can deal with the convolutional layers in the same manner with the fully connected layers.

Same with “im2col” method implemented in cuDNN [21], we can describe the sliding window operations in the convolutional layers by the sum of the matrix multiplications. Let a and A denote the numbers of the input channels and the output channels, h denotes the width and the height of the feature maps, s denote the width and the height of the weight tensor. The sliding window operations with a $\mathbb{R}^{N \times a \times h \times h}$ tensor, which denotes the feature maps corresponding to the N input images, and a $A \times a \times s \times s$ tensor, which denotes the weights, can be alternatively written as

$$Y = \sum_{i \in B} \Phi_i \Psi_i^\top = \sum_{i \in B} \sum_{m \in T} \phi_{i(m)} \psi_{i(m)}^\top, \quad (26)$$

where $B = \{1, \dots, a\}$ denotes the set of the input channel indices, $\Phi_i \in \mathbb{R}^{Nh^2 \times s^2}$ denotes the i^{th} channel of the reshaped input feature maps, $\Psi_i \in \mathbb{R}^{A \times s^2}$ denotes the reshaped weight tensor, $T = \{1, \dots, s^2\}$ denotes the set of the column indices of Φ s and Ψ s, $\phi_{i(m)}$ and $\psi_{i(m)}$ denote the m^{th} columns of Φ_i and Ψ_i .

We can regard that this layer is equivalent to the fully connected layer where as^2 neurons exists, the behaviors of those neurons are given by the ϕ -s, and the outgoing weights are given by the ψ -s. Pruning the i^{th} channel in the convolutional layer is equivalent to pruning s^2 corresponding neurons in this converted form.

3.5 Relation to CP

CP [4] is also a layer-wise pruning method that conducts reconstruction with least squares method. Although, its strategy for neuron selection is different from ours. They select

Table 1 VGG16 on ImageNet. The changes of top-5 accuracy from the baseline (89.5%) are reported (The greater, the better.). In this table, “rt” stands for “retraining”. *our implementation.

Speed-up ratio	Method	Acc. before rt	Acc. after rt	Retrain epoch#
×2	REAP	-2.0%	+0.2%	10
	NU [9]	-5.0%	-	-
	CP [4]	-2.7%	0.0%	10
	*ThiNet [5]	-65.0%	-1.0%	10
	SPP [19]	-	0.0%	-
×5	REAP	-9.4%	-1.3%	10
	CP [4]	-22.0%	-1.7%	10
	*ThiNet [5]	-88.8%	-3.4%	10
	SPP [19]	-	-2.0%	-

the neurons to be pruned by solving the following Lasso Regression problem:

$$\begin{aligned} \beta^* = \underset{\beta}{\operatorname{argmin}} \left\| Y - \sum_{i \in Z} \beta_i \mathbf{x}_i \mathbf{w}_i^\top \right\|_F^2 + \lambda \|\beta\|_1 \\ \text{subject to } \|\beta\|_0 \leq c', \end{aligned} \quad (27)$$

where c' denotes the desired number of neurons and $\beta = (\beta_1, \dots, \beta_c)^\top$ 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 weak point 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. On the other hand, REAP selects the neurons to be pruned based on the reconstruction error. Because of this difference, REAP performs better than CP, as we will show in experiments.

4. Experiments

We conducted the experiments with several CNN models and datasets (VGG16 [22] on ImageNet [23], ResNet-56 [24] on cifar-10 [25], and DenseNet-121 [26] on Stanford Dogs [27]) to verify REAP.

Even though REAP is an extended method of NU, we mainly compare REAP with CP, because REAP is the most similar with CP in theory, and CP is one of the state-of-the-art methods recently.

4.1 VGG16 on ImageNet

We conducted experiments with VGG16 on ImageNet. We pruned the convolutional layers until the theoretical speed-up ratio (the ratio of the floating point multiplications before and after the pruning) became ×2 and ×5. For pruning, we used 5000 images randomly selected from ImageNet training dataset. The images were resized so that the shorter side became 256, then 224×224 random crop was applied to the training images and 224×224 center crop was applied to the test images. The pruned models were fine-tuned for 10 epochs with the learning rate 10^{-5} . The momentum was set to 0.9, the minibatch size was set to 128, and the dropout rate in the fully connected layers was set to 0.5. The rest of the setups, including the pruning ratio for each layer, were set to the same values with [4].

The results are shown in Table 1. REAP performs consistently better than the existing methods. In the accuracy after retraining, we marginally outperform the other methods at ×2 speed-up. At ×5 speed-up, the existing methods suffer even larger accuracy drop than we do.

An important observation is that we only suffer 9.4% accuracy drop at ×5 speed-up before retraining. On the other hand, CP suffers 22.0% drop and ThiNet spoiled the model performance. This is because we use consistent strategy for channel selection and reconstruction to preserve the performances of the pruned models. As we show better performances before retraining, we can achieve higher accuracy after retraining as well. To put this observation differently, REAP enables us to achieve a certain accuracy with fewer epochs of retraining, which means that we can save time and labors for retraining by pruning with REAP.

It is also worth noting that the model pruned by REAP, at ×2 speed-up, after retraining, is better than original VGG16. This is most likely because we removed the redundant weights, the remaining weights had smaller chance of being trapped in the local minima during the retraining.

- (1) Close analysis on performance difference between CP and REAP.

As already mentioned, REAP is the most similar with CP in theory. The only difference of them is the channel selection criteria. As REAP selects the channels to be pruned in a greedy fashion, which probably raises the following questions:

1. Do the channels selected in REAP actually cause smaller reconstruction error than the channels selected by Lasso Regression in CP?
2. Although REAP takes an efficient algorithm for channel selection, does the computation finishes within reasonable time?

In order to answer these questions, we conducted additional experiments with VGG16 [22]. We pruned *conv1-1*, *conv2-1*, *conv3-1*, *conv4-1*[†] with several pruning ratios, and observe the layer-wise reconstruction errors ($\|\Delta Y\|_F^2$) and measure the computational time spent on channel selection. For tuning the hyper-parameter in CP (the coefficient for Lasso regression), we use binary search algorithm as we found

[†]VGG16 has 4 blocks that have several convolutional layers, and *convX-Y* represents the Y -th layer of the X -th block.

Fig. 3 Layer-wise analysis for VGG16 on ImageNet. The channels selected by REAP cause consistently smaller reconstruction error than the channels selected by CP.

Table 2 Time (sec.) spent for channel selection in *Conv1-1*, *Conv2-1*, *Conv3-1* and *Conv4-1* (channels# in the parentheses), at the pruning ratios of 0.25, 0.5, 0.75.

Method	<i>Conv1-1</i> (64)			<i>Conv2-1</i> (128)			<i>Conv3-1</i> (256)			<i>Conv4-1</i> (512)		
	0.25	0.50	0.75	0.25	0.50	0.75	0.25	0.50	0.75	0.25	0.50	0.75
REAP	21	22	24	35	44	53	116	168	229	777	1307	1872
CP	90	84	83	117	130	120	209	226	220	385	434	418

Table 3 ResNet-56 on cifar-10. The changes of top-1 accuracy (baseline: 93.4%) are reported (The greater, the better.). *our implementation. **results taken from [4].

Speed-up ratio	Method	Acc. before rt	Acc. after rt	Retrain epoch#
×2	REAP	-1.9%	-0.5%	100
	NU [4]	-13.3%	-1.9%	10
	*CP [4]	-3.7% (**-2.0%)	-0.9% (**-1.0%)	100
	*ThiNet [5]	-56.9%	-1.9%	100
	DCP [6]	-	-0.3%	400

out it was the fastest. All the methods are implemented with python3.6 and tested on Intel(R) Core(TM) i7-2600K CPU @ 3.40GHz.

As shown in Fig. 3, REAP suffers smaller error than CP. Besides, the trend is that the higher the pruning ratios are, the larger the error gaps between REAP and CP are. Because REAP selects channels based on the reconstruction error, we suffer consistently smaller reconstruction error than CP, despite we use greedy algorithm for channel selection.

Table 2 shows the results of computational time measurements. In *conv4-1* that has 512 channels, REAP consumes more computational time than CP. However, REAP requires 1872 seconds at the pruning ratio of 0.75, which we think is acceptable enough, considering that REAP saves us time for retraining the pruned model and that the training typically takes much more time (e.g. 1 epoch takes over 8 hours on NVIDIA Geforce GTX 1080 Ti). When the number of channels is no greater than 256 (in *conv1-1*, *conv2-1*, and *conv3-1*), REAP is faster than or as fast as CP.

4.2 ResNet-56 on Cifar-10

We pruned the pretrained model taken from [28]. For pruning, we used 6400 randomly sampled training images. ResNet-56 has 27 residual units that have 2 sequential convolutional layers and a shortcut path. The pruning ratios in the first 9 units, the second 9 ones and the rest were set to 3 : 2 : 1. The pruned models were retrained for 100 epochs, beginning with the learning rate 10^{-2} and dividing it by 10 every 25 epochs. The rest of the training setups were aligned with [24]. Since we wanted to compare REAP with CP in the same conditions, we tried to evaluate CP on our own and put the results reported in [4] just for reference (because the pretrained model used in [4] is not available, and the experimental setups are not mentioned in [4]).

The results are shown in Table 3. Before retraining, we could easily outperform the existing methods. We suffer only 1.9% accuracy drop without retraining, while CP suffers 3.7%. After retraining, we are slightly worse than,

Table 4 DenseNet-121 on Stanford Dogs. The changes of top-1 accuracy (baseline: 84.6%) are reported (The greater, the better.). *our implementation.

Speed-up ratio	Method	Acc. before rt	Acc. after rt	Retrain epoch#
×2	REAP	-3.1%	-3.3%	20
	*CP [4]	-4.7%	-3.5%	20
	*ThiNet [5]	-63.7%	-4.9%	20

although competitive with another state-of-the-art pruning method DCP [6]. However, while DCP needs 400 epochs of retraining to achieve this result, we only need 100 epochs to achieve the competitive result. In this way, the fact that we can save efforts on retraining is a strength of REAP.

4.3 DenseNet-121 on Stanford Dogs

Finally, we conducted the experiments on DenseNet-121 fine-tuned with StanfordDogs dataset. For transfer learning, we set the learning rate to 10^{-2} for the first 30 epochs and set it to 10^{-3} for 20 more epochs. For retraining after pruning, we set the learning rate as 10^{-3} and trained the models for 10 epochs, then repeated another 10 epochs with the learning rate 10^{-4} . We set the pruning ratios in Block1, Block2, Block3 and Block4 to 5 : 5 : 4 : 3. The rest of the setups were aligned with Sect. 4.1.

The results are shown in Table 4. Similarly with other experiments, REAP could preserve the model accuracy better than the other methods did. It is also remarkable that the model pruned by REAP without retraining is as accurate as the model after retraining. REAP preserves the model performances so well that we sometimes do not even need to retrain the pruned models.

5. Conclusion

We have proposed REAP, a channel pruning method to accelerate the inference of CNNs. REAP is an extension of the currently state-of-the-art pruning method CP. REAP prunes the channels based on the reconstruction error of the outputs, then reconstructs the outputs by the least squares method. Thus, REAP can reduce the computational complexity of the CNN models while maintaining their performances, which not only makes it possible to produce a fast, compact, and accurate models but also saves time and labors required for retraining the pruned models. On the experiments, we could confirm these strengths of REAP.

6. Future Works

As the nature of a layer-wise pruning method, the performance of REAP depends on the pruning ratio that is manually determined for each layer. We plan to extend REAP to overcome this weak point. It should be possible to determine the proper pruning ratio for a layer based on the reconstruction error (e.g. imposing a threshold on the maximum reconstruction error in each layer). Then, we can conduct holistic optimization with REAP and achieve even better pruning performances.

Acknowledgments

This work was supported by JSPS KAKENHI Grant Number 19K12020 and the Environment Research and Technology Development Fund (3-1905) of the Environmental Restoration and Conservation Agency of Japan.

References

- [1] K. Kamma and T. Wada, "Reconstruction error aware pruning for accelerating neural networks," *Advances in Visual Computing, Lecture Notes in Computer Science*, vol.11844, pp.59–72, 10 2019.
- [2] T. He, Y. Fan, Y. Qian, T. Tan, and K. Yu, "Reshaping deep neural network for fast decoding by node-pruning," *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp.245–249, 2014.
- [3] Y. LeCun, J.S. Denker, and S.A. Solla, "Optimal brain damage," in *Advances in Neural Information Processing Systems 2*, ed. D.S. Touretzky, pp.598–605, 1990.
- [4] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," *Proc. International Conference on Computer Vision*, pp.1398–1406, 2017.
- [5] J.-H. Luo, J. Wu, , and W. Lin, "ThiNet: A filter level pruning method for deep neural network compression," *Proc. International Conference on Computer Vision*, pp.5068–5076, 2017.
- [6] 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. Advances in Neural Information Processing Systems*, pp.875–886, 2018.
- [7] X. Dong, S. Chen, and S. Pan, "Learning to prune deep neural networks via layer-wise optimal brain surgeon," in *Advances in Neural Information Processing Systems 30*, pp.4857–4867, Curran Associates, Inc., 2017.
- [8] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, "Amc: Autotml for model compression and acceleration on mobile devices," *The European Conference on Computer Vision (ECCV)*, vol.11211, pp.815–832, Sept. 2018.
- [9] K. Kamma, Y. Isoda, S. Inoue, and T. Wada, "Behavior-based compression for convolutional neural networks," *Image Analysis and Recognition - 16th International Conference, ICIAR 2019, Waterloo, ON, Canada, Aug. 27-29, 2019, Proceedings, Part I*, ed. F. Karay, A. Campilho, and A.C.H. Yu, *Lecture Notes in Computer Science*, vol.11662, pp.427–439, Springer, 2019.
- [10] J. Xue, J. Li, and Y. Gong, "Restructuring of deep neural network acoustic models with singular value decomposition," *INTER-SPEECH*, pp.2365–2369, 2013.
- [11] J. Ye, L. Wang, G. Li1, D. Chen, S. Zhe, X. Chu, and Z. Xu, "Learning compact recurrent neural networks with block-term tensor decomposition," *Proc. Computer Vision and Pattern Recognition*, pp.9378–9387, 2018.
- [12] X. Yu, T. Liu, X. Wang, and D. Tao, "On compressing deep models by low rank and sparse decomposition," *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp.67–76, July 2017.
- [13] B. Liu, M. Wang, H. Foroosh, M.F. Tappen, and M. Pensky, "Sparse convolutional neural networks," *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp.806–814, 2015.

- [14] A. Aghasi, A. Abdi, N. Nguyen, and J. Romberg, “Net-trim: Convex pruning of deep neural networks with performance guarantee,” *Advances in Neural Information Processing Systems* 30, pp.3177–3186, Curran Associates, Inc., 2017.
- [15] G. Xie, J. Wang, T. Zhang, J. Lai, R. Hong, and G.-J. Qi, “Interleaved structured sparse convolutional neural networks,” *Proc. Computer Vision and Pattern Recognition*, pp.8847–8856, 2018.
- [16] M. Courbariaux, Y. Bengio, and J.P. David, “Binaryconnect: Training deep neural networks with binary weights during propagations,” *Advances in Neural Information Processing Systems* 28, pp.3123–3131, Curran Associates, Inc., 2015.
- [17] 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*, pp.1–14, 2016.
- [18] A. Zhou, A. Yao, K. Wang, and Y. Chen, “Explicit loss-error-aware quantization for low-bit deep neural networks,” *Proc. Computer Vision and Pattern Recognition*, pp.9426–9435, 2018.
- [19] H. Wang, Q. Zhang, Y. Wang, and H. Hu, “Structured probabilistic pruning for convolutional neural network acceleration,” *Proc. British Machine Vision Conference*, 2018.
- [20] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, “Learning efficient convolutional networks through network slimming,” *Proc. International Conference on Computer Vision*, pp.2755–2763, 2017.
- [21] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, “cudnn: Efficient primitives for deep learning,” *CoRR*, vol.abs/1410.0759, 2014.
- [22] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *Proc. International Conference on Learning Representations*, pp.1–14, 2015.
- [23] J. Deng, W. Dong, R. Socher, L. jia Li, K. Li, and L. Fei-fei, “Imagenet: A large-scale hierarchical image database,” *In CVPR*, pp.248–255, 2009.
- [24] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp.770–778, 2016.
- [25] A. Krizhevsky, V. Nair, and G. Hinton, “Cifar-10 (canadian institute for advanced research).”
- [26] G. Huang, Z. Liu, L. van der Maaten, and K.Q. Weinberger, “Densely connected convolutional networks,” *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp.2261–2269, 2017.
- [27] A. Khosla, N. Jayadevaprakash, B. Yao, and L. Fei-Fei, “Novel dataset for fine-grained image categorization,” *First Workshop on Fine-Grained Visual Categorization*, IEEE Conference on Computer Vision and Pattern Recognition, Colorado Springs, CO, June 2011.
- [28] akamaster, “Proper implementation of resnet-s for cifar10/100 in pytorch that matches description of the original paper,” 2019.

Appendix A: Substitute approach for computing reconstruction error of each neuron

We try to accelerate the computation for (8) by parallel computing. However, it is not smart to implement (8) as is, because the naive solution of the least squares problem is not memory-efficient, and the degree of parallelism would be limited because of the heavy memory requirement. Therefore, we have developed a memory efficient algorithm that uses Gram-Schmidt process.

As already mentioned, solving (8) is equivalent to computing the orthogonal projection of $\mathbf{x}_j$ onto the subspace $U_{(Z \setminus \{j\})}$ spanned by $\{\mathbf{x}_i | i \in Z \setminus \{j\}\}$, as shown in Fig. 2. Then, we have

Algorithm 1

Input: $\mathbf{x}_i | i \in \{1, \dots, c\}$ and iterations# d
Output: $\mathbf{r}_i | i \in Z$ and $v_{ji} | i, j \in Z$
Initialize: $\mathbf{r}_j \leftarrow \mathbf{x}_j$ and $v_{ji} \leftarrow 0$ for each $i, j \in Z$
while the number of iterations $\leq d$ **do**
 for $(j, i) \in Z \times Z$ **do**
 if $i \neq j$ **then**
 $v_{ji} \leftarrow v_{ji} + f(\mathbf{r}_j, \mathbf{x}_i)$
 $\mathbf{r}_j \leftarrow \mathbf{r}_j - f(\mathbf{r}_j, \mathbf{x}_i)\mathbf{x}_i$
 end if
 end for
end while

$$\begin{aligned} \mathbf{r}_j &= \mathbf{x}_j - \sum_{i \in Z \setminus \{j\}} v_{ji}^* \mathbf{x}_i \\ &= \mathbf{x}_j - M_{(Z \setminus \{j\})} M_{(Z \setminus \{j\})}^T \mathbf{x}_j, \end{aligned} \quad (\text{A} \cdot 1)$$

where $M_{(Z \setminus \{j\})}$ represents the orthogonal basis of $U_{(Z \setminus \{j\})}$. However, $M_{(Z \setminus \{j\})}$ is typically a large matrix, and computing $M_{(Z \setminus \{j\})}$ for every possible j is not memory-efficient. We need some more tricks to accelerate the computation.

We obtain the approximate solution of the $\mathbf{r}$ -s and the v -s by applying Gram-Schmidt process iteratively. Let $\leftarrow$ denote assignment and f denote the function to compute the orthogonal projection coefficient:

$$f(\mathbf{r}, \mathbf{x}) = \frac{\langle \mathbf{r}, \mathbf{x} \rangle}{\|\mathbf{x}\|^2}. \quad (\text{A} \cdot 2)$$

Then, the idea of this algorithm is illustrated in Fig. A·1 and can be described as follows.

- We first initialize: $\mathbf{r}_j \leftarrow \mathbf{x}_j$, $v_{ji} \leftarrow 0$ for each $i \in Z \setminus \{j\}$.
- We apply Gram-Schmidt process to make $\mathbf{r}_j$ orthogonal to $\mathbf{x}_i$ for each $i \in Z \setminus \{j\}$ alternately, such that $\mathbf{r}_j \leftarrow \mathbf{r}_j - f(\mathbf{r}_j, \mathbf{x}_i)\mathbf{x}_i$. At the same time, we also update the coefficient v_{ji} such that $v_{ji} \leftarrow v_{ji} + f(\mathbf{r}_j, \mathbf{x}_i)$.
- After repeating this sequential orthogonalizations many times, $\mathbf{r}_j$ becomes a vector which is approximately orthogonal to all the vectors in $\{\mathbf{x}_i | i \in Z \setminus \{j\}\}$, in other words, $\mathbf{r}_j^n$ becomes approximately orthogonal to $U_{(Z \setminus \{j\})}$. Then, $\{v_{ji} | i \in Z \setminus \{j\}\}$ is the approximate solution of Eq. (8).

The detailed procedure is summarized in Algorithm 1.

The theoretical computational order of this algorithm is $O(c^3)$. However, it substantially reduces to $O(c^2)$ by parallel computing. In this algorithm, we do not need any large tensors except for the $\mathbf{x}$ -s, the $\mathbf{r}$ -s, and the v -s. In CUDA, they can be stored in the shared memory and passed to each thread by reference. Therefore, each thread suffers little memory consumption, and $\mathbf{r}_j$ and v_{ji} for $i, j \in Z$ can be computed parallelly.

Fig. A.1 An example of the Gram-Schmidt process based algorithm. We want to obtain r_j , the residual of the projection of x_j onto the subspace spanned by x_k and x_l . a) r_j is initialized to x_j . b) r_j is projected onto x_l and is replaced by the residual. c) r_j is projected onto x_k and is replaced by the residual. Note that r_j becomes NOT orthogonal to x_l again, as x_k and x_l are not orthogonal. d) After repeating b) and c) lots of times, r_j converges into the vector that is orthogonal to both of x_k and x_l .

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 1 year. He is currently a Ph.D. student in Faculty of Systems Engineering, Wakayama University. His research interests include Pattern Recognition and Deep Learning.

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.

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.