PAPER

Neural Behavior-Based Approach for Neural Network Pruning

Koji KAMMA[†a)], **Yuki ISODA**[†], **Sarimu INOUE**[†], ***and*** **Toshikazu WADA**[†], ***Nonmembers***

SUMMARY This paper presents a method for reducing the redundancy in both fully connected layers and convolutional layers of trained neural network models. The proposed method consists of two steps, 1) Neuro-Coding: to encode the behavior of each neuron by a vector composed of its outputs corresponding to actual inputs and 2) Neuro-Unification: to unify the neurons having the similar behavioral vectors. Instead of just pruning one of the similar neurons, the proposed method let the remaining neuron emulate the behavior of the pruned one. Therefore, the proposed method can reduce the number of neurons with small sacrifice of accuracy without retraining. Our method can be applied for compressing convolutional layers as well. In the convolutional layers, the behavior of each channel is encoded by its output feature maps, and channels whose behaviors can be well emulated by other channels are pruned and update the remaining weights. Through several experiments, we comfirmed that the proposed method performs better than the existing methods.

key words: *neuro-coding, neuro-unification*

1. Introduction

Deep neural networks (DNNs) have been showing dominant performances in the machine learning tasks. The key is the scale of the models. In fact, all the state-of-the-art models have a great number of parameters. Although, those models are too large for most of the applications where they are desired. Therefore, it is important to make the DNN models smaller without degrading their performances.

A major approach for producing a small and accurate model is to compress a large pretrained model and retrain the compressed model to maintain the accuracy. Although, this retraining is a challenge, because it is computationally expensive, and one often needs cumbersome trials and errors for tuning hyperparameters.

In this paper, we propose a novel pruning method to compress and accelerate the convolutional neural network models while preserving the model performances, which will significantly save one's efforts on retraining the pruned models.

Our method consists of two steps: 1) Neuro-Coding: to encode the behavior of each neuron and 2) Neuro-Unification: to unify the neurons which behave similarly. Each neuron outputs a scalar value corresponding to each input data. By recording those outputs, we create a behavioral vector for each neuron. If some neurons have similar behavioral vectors, we can unify those neurons, because they are redundant. Unifying a pair of neurons is, in other words, pruning one of them and letting the remaning one emulate the behavior of the pruned one by transferring the outgoing weights from the pruned one to the remaining one. Therefore, we can unify the neurons with small impact to the model accuracy.

Neuro-Coding and Neuro-Unification can be applied to the convolutional layers as well. In the convolutional layers, we reshape the feature maps so that the siliding window operation can be described as a simple matrix multiplication, then we conduct the channel-level behavior encoding and unification.

It is worth noting that compressing fully connected layers results in saving memory consumption and that compressing convolutional layers results in reducing the computational complexity required for inference. For instance, VGG16 [13] has about 90% of the parameters in fully connected layers, and the convolutional layers account for about 99% of the floating point operations. The proposed method can contribute to reducing both the memory consumption and the computational complexity.

We conducted some experiments with VGG16s on ImageNet. The results demonstrate that the proposed method can compress both the fully connected layers and the convolutional layers with small sacrifice of accuracy compared to the existing methods.

2. Related Works

We categorize the existing researches about DNN model compression into 4 groups: 1) Pruning, 2) Low-rank approximation, 3) Sparsification, and 4) Quantization.

Pruning. The idea of pruning is to compute the saliency of each neuron or weight and to remove the least salient one. Optimal Brain Damage [8] is the pioneering pruning method which uses second derivative information of the cost function to calculate the saliency. There are also some other pruning methods for fully connected layers and/or convolutional layers [3], [4], [6], [9], [11], [19], [22]. Some pruning methods not only prune but also execute "surgery" to compensate the impact of pruning so that the model needs to be retrained less frequently. Data-free Parameter Pruning [17] evaluates similarity of the neurons based on their incoming weights and "wires the similar neurons together". Optmal Brain Surgeon [5] is also a pruning method which executes surgery based on the Hessian of the cost function.

Manuscript received June 26, 2019.
Manuscript revised December 9, 2019.
Manuscript publicized January 23, 2020.

[†]The authors are with Wakayama University, Wakayama-shi, 640–8510 Japan.
a) E-mail: kanma@vrl.sys.wakayama-u.ac.jp

DOI: 10.1587/transinf.2019EDP7177

Low rank approximation. Xue et. al. suggested a method using low-rank approximation [18]. 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 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 parameters reduces from mn to $(m+n)o$. The drawback is that the number of weight matrices doubles and the model structure becomes more complicated.

Sparcification. The idea of sparsification is to make the weight matrices sparse by fine-tuning the models with L1 regularization. Liu et. al. suggested Sparse Convolutional Neural Networks [10]. There are other methods to sparsify the weights such as [1]. Although, L1 regularization shifts the global minimum of the cost function and sacrifices accuracy. Besides, in order to take the advantage of the sparsified models, one needs the special hardwares and libraries for executing the computions on only non-zero elements.

Quantization. Quantization is a different approach from the above 5 approaches. The methods in this group reduce the redundancy of each bitwise operation, e.g. changing the floating point precision from 32-bit to 8-bit. Similarly with sparsification, the special hardwares are required to deploy the quantized models. The methods proposed in [2], [3] take this approach.

The proposed method in this paper belongs to pruning group. However, the way it executes the surgery is different from the existing methods. The proposed method can evaluate the behavioral similarity between the neurons or the channels accurately, which enables us to maintain high level of the accuracy while compressing the models. Another advantage of the proposed method is that the pruned model can be deployed without any special hardwares or libraries.

3. Neural Behavior-Based Pruning

The steps of the proposed method are:

Step1: Encode the behavior of each neuron by its outputs (Neuro-Coding)

Step2: Unify the neurons based on their behavioral similarity (Neuro-Unification)

Step2 should be repeated while the model is not small enough. Besides, we may repeat Step1 at certain interval because the behaviors of some neurons should change after some unification happen, though it would require additional computational cost.

In addition, we can optionally conduct extra surgeries to preserve. In extra surgery, we can let more than one neuron to emulate the behavior of a pruned neuron, which results in preserving the model performances better.

We first explain the basics of Neuro-Coding and Neuro-Unification, and then explain the extra surgery. We also show how to apply our method to the convolutional layers.

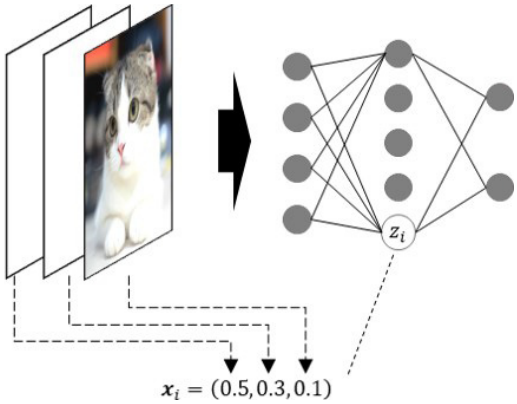

Fig. 1 The conceptual drawing of Neuro-Coding. When we input some samples to a neural network model, each neuron obtains a behavioral vector composed of their outputs.

3.1 Neuro-Coding

Neuro-Coding is a technique to capture the neuron behaviors. Figure 1 is the conceptual drawing. Let $z_1, \cdots, z_N$ denote the neurons in a certain layer, where N denotes the number of neurons. For a single input sample into the model, z_i outputs a scalar value. For D input samples, the output of z_i is obtained as a vector $\boldsymbol{x}_i \in \mathbb{R}^D$. We call it a "behavioral vector" of z_i.

3.2 Neuro-Unification

Let z'_n denote the n^{th} neuron in the next layer to the target one, and w_{in} denote the weight going from z_i to z'_n (See Fig. 2). The forward propagation is described by

$$y_n^d = x_i^d w_{in} + x_j^d w_{jn} + \sum_{k \neq i,j} x_k^d w_{kn}, \tag{1}$$

where x_i^d denotes the d^{th} element of $\boldsymbol{x}_i$ which is, in other words, the output of z_i corresponding to the d^{th} input sample, y_n^d denotes the inner activation level of z'_n.

If $\boldsymbol{x}_i = \alpha \boldsymbol{x}_j$ holds, we can unify z_i and z_j without affecting the model performance. As shown in Fig. 2 (b), we prune z_i and let z_j emulate the behavior of z_i by updating w_{jn} by

$$w_{jn} := \alpha w_{in} + w_{jn}, \tag{2}$$

where := denotes assignment. Then, we can rewrite Eq. (1) as

$$y_n^d = x_j^d \left(\alpha w_{in} + w_{jn}\right) + \sum_{k \neq i,j} x_k^d w_{kn}. \tag{3}$$

Equation (1) and Eq. (3) are equivalent because $\boldsymbol{x}_i = \alpha \boldsymbol{x}_j$ holds, which means the inner activation level of z'_n is preserved. This weight transfer is the surgery step in Neuro-Unification.

Next, we address the case of unifying the neurons with

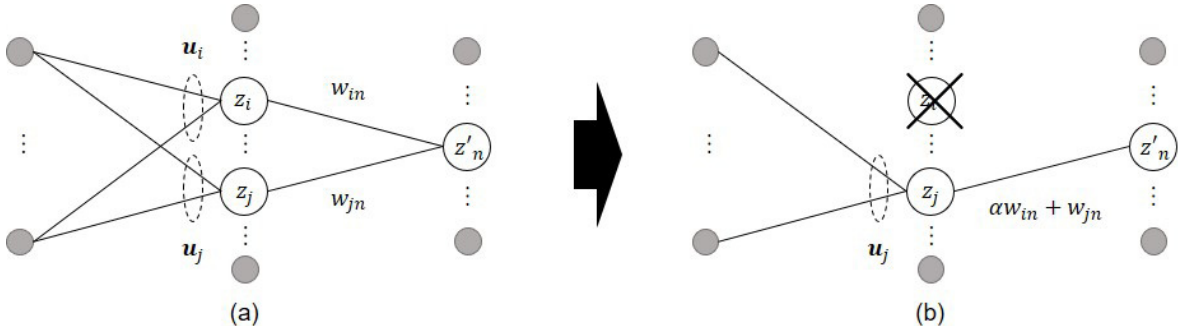

Fig. 2 Unification of neurons. (a) Initial state of the fully connected layers. (b) After merging z_i into z_j. If the behavioral vectors of z_i and z_j are similar, we can unify them with small impact to the inner activation levels of the neurons in the next layer.

linearly independent behavioral vectors. In this case, we first approximate $\boldsymbol{x}_i$ by a vector which is linearly dependent with $\boldsymbol{x}_j$:

$$\boldsymbol{x}_i \simeq \alpha \boldsymbol{x}_j. \tag{4}$$

We regard that $\alpha \boldsymbol{x}_j$ is the behavioral vector of z_i so that we can unify z_i and z_j in the same manner with Eq. (2).

Here is a question. How to determine α in Eq. (4)? In order to preserve the inner activation levels of the neurons in the next layer, we simultaneously minimize the error of the inner activation levels of $z'_1, \cdots, z'_{N'}$, where N' denotes the number of the neurons in the next layer. This can be formalized as

$$\begin{aligned} \alpha^*_{ij} &= \underset{\alpha}{\operatorname{argmin}} \sum_{n=1}^{N'} \sum_{d=1}^{D} \left(\left(\alpha x_j^d - x_i^d \right) w_{in} \right)^2 . \\ &= \underset{\alpha}{\operatorname{argmin}} \sum_{n=1}^{N'} w_{in}^2 \sum_{d=1}^{D} \left(\alpha x_j^d - x_i^d \right)^2 \end{aligned} \tag{5}$$

We can omit $\sum_{n=1}^{N'} w_{in}^2$ in Eq. (5) as it is a constant when i is fixed. Moreover, we obviously have

$$\sum_{d=1}^{D} \left(\alpha x_j^d - x_i^d \right)^2 = \left\| \alpha \boldsymbol{x}_j - \boldsymbol{x}_i \right\|^2 . \tag{6}$$

Then, Eq. (5) can be rewritten as

$$\alpha^*_{ij} = \underset{\alpha}{\operatorname{argmin}} \left\| \alpha \boldsymbol{x}_j - \boldsymbol{x}_i \right\|^2 . \tag{7}$$

After all, we have to compute the orthogonal projection of $\boldsymbol{x}_i$ onto $\boldsymbol{x}_j$. Thus, we have

$$\alpha^*_{ij} = \frac{\boldsymbol{x}_i^\top \boldsymbol{x}_j}{\left\| \boldsymbol{x}_j \right\|^2}. \tag{8}$$

If the approximation of $\boldsymbol{x}_i$ is good enough, z_j can emulate the major proportion of the behavior of z_i, and the impact to the overall model performance caused by the unification is small.

3.3 Criteria for Selecting the Neurons to Be Unified

We know how to unify the neurons. Although, we have yet to know how to select the neurons to be unified when we have many possible neuron pairs.

As already discussed, we should unify the neurons while minimizing the errors of the inner activation levels in the next layer. For this purpose, we define the saliency of (z_i, z_j) by

$$r(i, j) = \sum_{n=1}^{N'} {w_{in}}^2 \left\| \alpha^*_{ij} \boldsymbol{x}_j - \boldsymbol{x}_i \right\|^2 . \tag{9}$$

In Eq. (9), $r(i, j)$ represents the error in the next layer caused by merging z_i into z_j. Note that we cannot ignore $\sum_{n=1}^{N'} {w_{in}}^2$ here as it is not a constant before we fix i.

Let Z denote the set composed of the tuples of the unified neurons' indices (e.g. $(i, j) \in Z$ means that z_i has been merged into z_j). Then, we have to solve the following combinatorial optimization problem:

$$\begin{aligned} Z^* = \underset{Z}{\operatorname{argmin}} & \left\| \sum_{(i,j) \in Z} \sum_{n=1}^{N'} w_{in} \left(\alpha^*_{ij} \boldsymbol{x}_j - \boldsymbol{x}_i \right) \right\|^2 \\ \text{subject to } & |Z| = Q, \end{aligned} \tag{10}$$

where $|Z|$ denotes the order of Z and Q denotes the desired number of the neurons to be pruned. In order to perform simplification to Eq. (10), we use the following theorem (See the appendix for the proof).

Theorem 1: Let $\boldsymbol{a}_1, \cdots, \boldsymbol{a}_p \in \mathbb{R}^P$. Then,

$$\left\| \sum_{i=1}^{p} \boldsymbol{a}_i \right\|^2 \leq p \sum_{i=1}^{p} \left\| \boldsymbol{a}_i \right\|^2 . \tag{11}$$

Using Theorem 1, we have

$$
\begin{aligned}
&\left\| \sum_{(i,j)\in Z} \sum_{n=1}^{N'} w_{in} \left(\alpha_{ij}^* \boldsymbol{x}_j - \boldsymbol{x}_i \right) \right\|^2 \\
&\leq |Z| \sum_{(i,j)\in Z} \left\| \sum_{n=1}^{N'} w_{in} \left(\alpha_{ij}^* \boldsymbol{x}_j - \boldsymbol{x}_i \right) \right\|^2 \\
&= |Z| \sum_{(i,j)\in Z} \sum_{n=1}^{N'} {w_{in}}^2 \left\| \left(\alpha_{ij}^* \boldsymbol{x}_j - \boldsymbol{x}_i \right) \right\|^2 \\
&= |Z| \sum_{(i,j)\in Z} r(i,j).
\end{aligned} \tag{12}
$$

We relax Eq. (10) by using this upper bound:

$$
\begin{aligned}
&Z^* = \underset{Z}{\operatorname{argmin}} \sum_{(i,j)\in Z} r(i,j) \\
&\text{subject to } |Z| = Q.
\end{aligned} \tag{13}
$$

We solve it in a greedy fashion. We select the neurons one by one based on the following cost function F, where we have $Z^* = \operatorname{argmin}_Z F(Z)$:

$$
F(Z) = \sum_{(i,j)\in Z} r(i,j). \tag{14}
$$

Here, we need a constraint to avoid a contradiction. Assume that $(i,j),(j,k) \in Z$. This means that we have merged z_i into z_j, however, z_j does not exist anymore as it has been merged into z_k. This is clearly an contradiction. Therefore, we introduce a constraint to avoid merging any neuron into z_j if $(j,k) \in Z$ for some k.

3.4 Problem Reformalization and Greedy Algorithm

For better understanding, we reformalize the problem of selecting the neurons based on graph theory, then we show the algorithm to solve the problem.

The problem of selecting the neurons to be unified is equivalent to the problem of creating a forest having minimum cost in a complete symmetric digraph. Let G denote the graph defined by

$$
G = (V, E), \tag{15}
$$

where V and E denote the the sets composed of the vertices and the edges, respectively. The vertices and the edges correspond to the neurons and the possible unifications, respectively. Since the graph is a complete symmetric digraph, we have

$$
E = V \times V \setminus \{(v_i, v_i) | v_i \in V\}. \tag{16}
$$

Let $G' = (V', E')$ be the forest which we want to create, where $V' \subseteq V$ and $E' \subseteq E$. If $(v_i, v_j) \in E'$, it means that v_i has been merged into v_j. Then, we have the following combinatorial optimization problem:

$$
\underset{E'}{\operatorname{argmin}} \sum_{(v_i,v_j)\in E'} c(v_i, v_j) \quad \text{s.t.} \quad |E'| = Q, \tag{17}
$$

Algorithm 1

Input: Complete symmetric digraph $G = (V, E)$, control parameter Q
Output: Forest $G' = (V', E')$
Define: Initial and substantive edge cost $c(\cdot,\cdot)$ and $c'(\cdot,\cdot)$, set of prohibited edges E_p, the number of elements of a set $q(\cdot)$,
Initialize: $V' = \emptyset, E' = \emptyset, E_p = \emptyset$,
for each $(v_i, v_j) \in E$ **do**
  $c'(v_i, v_j) := c(v_i, v_j)$
end for
while $|E'| < Q$ **do**
  Find $(v_i, v_j) \in E \setminus E_p$ which minimizes $c'(v_i, v_j)$
  $V' := V' \cup \{v_i, v_j\}$, $E' := E' \cup \{(v_i, v_j)\}$
  for each s fulfilling $(v_s, v_i) \in E'$ **do**
    $E' := (E' \setminus \{(v_s, v_i)\}) \cup \{(v_s, v_j)\}$
  end for
  for each s fulfilling $(v_i, v_s) \in E$ **do**
    $c'(v_i, v_s) := c(v_i, v_s)$
  end for
  for each s fulfilling $(v_s, v_i) \in E$ or $(v_i, v_s) \in E$ **do**
    $E_p := E_p \cup \{(v_i, v_s), (v_s, v_i)\}$
  end for
  for each t fulfilling $(v_j, v_t) \in E \setminus E_p$ **do**
    $c'(v_j, v_t) := c(v_j, v_t)$
    for each s fulfilling $(v_s, v_j) \in E'$ **do**
      $c'(v_j, v_t) := c'(v_j, v_t) + c(v_s, v_t) - c(v_s, v_j)$
    end for
  end for
end while

where $c(v_i, v_j)$ denotes the cost of (v_i, v_j), which is equivalent to $r(i,j)$ in Eq. (9).

Besides, we have a constraint that the height of trees composing G' must be 1. For example, assume that we have the following tree with height of 2:

$$
V' = \{v_1, v_2, v_3\}, \tag{18}
$$

$$
E' = \{(v_1, v_2), (v_2, v_3)\}. \tag{19}
$$

This means that v_1 has been merged into v_2, which has already been merged into v_3. This is an contradiction that should be avoided.

All that is left is to solve the combinatorial optimization problem. For obtaining the solution efficiently, we use a greedy algorithm shown in Algorithm 1.

Computational order. In Algorithm 1, we have the following computational steps: 1) to calculate $c(\cdot,\cdot)$ and 2) to create G' while updating V', E', E_p and $c'(\cdot,\cdot)$. Step 1) does not require heavy calculations and it needs to be done only once. Step 2) occupies most of the complexity. The order of Step 2) is $O(N^3)$, where N denotes the number of neurons.

3.5 Extra Surgery

In Neuro-Unification, the behavior of the pruned neuron is emulated by another neuron. However, it is possible to let 2 or more neurons for emulating a neuron's behavior.

When merging z_i into z_j, $\boldsymbol{x}_i$ is approximated by $\alpha_{ij}^* \boldsymbol{x}_j$. The residual of this approximation, given by $\boldsymbol{r}_i = \boldsymbol{x}_i - \alpha_{ij}^* \boldsymbol{x}_j$, causes the errors in the next layer. In order to make this residual even smaller, we let $z_k (k \neq i, j)$ to emulate the residual of the behavior of z_i. This can be described by

$$\beta^*_{ik} = \underset{\beta}{\operatorname{argmin}} \|\beta \boldsymbol{x}_k - \boldsymbol{r}_i\|^2, \tag{20}$$

$$w_{kn} := \beta^*_{ik} w_{in} + w_{kn}. \tag{21}$$

In the same manner, we can pick yet another neuron to emulate the residual of the behavior of z_i. By repeating this procedure, the residual of $\boldsymbol{x}_i$ gets even smaller, which results in reducing the errors of the inner activation levels in the next layer.

Another good part of the extra surgery is that the required computational cost is very small compared to the step of selecting the neurons to be unified. Therefore, one can conduct lots of extra surgeries as desired without strong computational resorces.

3.6 Applying Neuro-Unification to Convolutional Layers

The Neuro-Unification 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 [20], we can describe the sliding window operations in the convolutional layers by the sum of the matrix multiplications, as shown in Fig. 3. 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

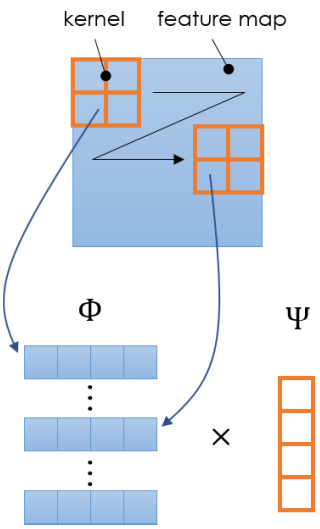

Fig. 3 The illustration of im2col method implemented in cuDNN. The kernel (corresponding to the each single input channel) is reshaped into a vertical vector, and the sub-matrices of the feature maps are reshaped into a horizontal vectors. After these reshaping, we can describe the sliding window operation of the convolutional layer by a simple matrix multiplication.

$$Y = \sum_{i\in B} \Phi_i \Psi_i^\top = \sum_{i\in B} \sum_{m\in T} \boldsymbol{\phi}_{i(m)} \boldsymbol{\psi}_{i(m)}^\top, \tag{22}$$

where $B = \{1, \cdots, a\}$ 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, $T = \{1, \cdots, s^2\}$ denotes the set of the column indices of Φs and Ψs, $\boldsymbol{\phi}_{i(m)}$ and $\boldsymbol{\psi}_{i(m)}$ 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.

4. Experiments

We evaluate our method on VGG16 [13] trained with ImageNet [15] and ResNet-56 [14] on cifar-10 [16], and compare the performances with some existing methods.

4.1 Existing Methods

We explain the existing methods to compare with ours and have the theoretical discussion.

(1) Data-free Parameter Pruning (DPP)

DPP [17] is a method for compressing the fully connected layers. DPP unifies the neurons with similar incoming weights. See Fig. 2 (a). $\boldsymbol{u}_i$ denotes the incoming weights of z_i including the bias term. If $\boldsymbol{u}_i \simeq \alpha \boldsymbol{u}_j$, they remove z_i and update the outgoing weights of z_j in the same manner with Eq. (2).

We expect that our proposed method will perform better than DPP, because DPP does not evaluate the influence of activation functions while ours does. The assumption behind DPP is that if the incoming weights of some neurons are similar, their outputs should also be similar. However, the similarity of the incoming weights of the neurons does not guarantee that their outputs are also similar due to the non-linearity of the activation functions such as ReLU.

Besides, DPP can use only one neuron to emulate a removed neuron, while the proposed method can use as many neurons as we want for emulating the removed one. This is obviously one of the reasons why our method is better.

(2) Oracle Pruning (OP)

OP [11] conducts channel-wise pruning for convolutional layers. OP computes the saliency of each channel based on the derivative information of the cost function, and prunes the least salient ones.

OP only prunes the neurons and do not conduct surgery. Therefore, iterative pruning with OP may result in a rapid degradation of the model accuracy.

Besides, differently from our method, OP's channel selection criteria is designed by heuristics, therefore, this criteria is not promising for preserving the model performances.

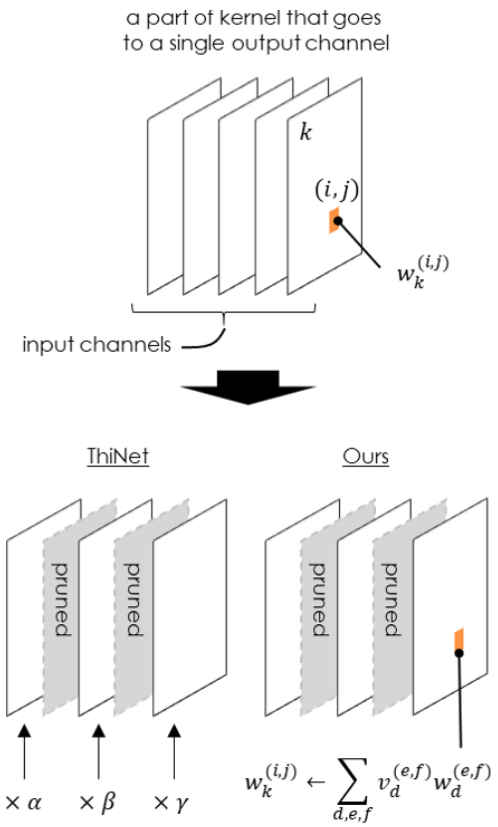

Fig. 4 Theoretical comparison of our method and ThiNet. It illustrates a part of the weight tensor that goes from all the input channels to a single output channel. In the reconstruction step, ThiNet multiplies the whole weights in each channel by a common coefficient. On the other hand, our method can tune weights independently from other weights that belong to the same channel.

(3) ThiNet

ThiNet [22] is a pruning method for convolutional layers. ThiNet prunes the channels in a greedy fashion so that the outputs of the convolutional layer (in other words, the inner activation levels in the next layer) is preserved, then reconstructs the outputs by the least squares method.

Figure 4 illustrates a part of the weight tensor that goes from all the input channels to a single output channel. In the reconstruction step, ThiNet multiplies the whole weights in each channel by a common coefficient such as $W_i \leftarrow \alpha W_i$. On the other hand, our method updates each weight independently such as $w_k^{(i,j)} \leftarrow \alpha w_1^{(1,1)} + \beta w_1^{(1,2)} + \cdots + \omega w_c^{(s,s)}$, where $w_k^{(i,j)}$ denotes the (i, j) entry of the k^{th} channel.

Another difference between our method superior and ThiNet is that we take the consistent strategies for channel (neuron) selection and surgery steps. We select channels to be pruned based on the error in the next layer after surgery, while ThiNet's channel selection criteria is based on the error before surgery. Therefore, we expect that our method can prevent the degradation of the model better than ThiNet.

Table 1 Results of VGG16 on ImageNet, fully connected layers. In this Table, "NU" of "NU(k)" stands for "Neuro-Unification" and k denotes the number of times of extra surgeries. The baseline top-5 accuracy is 0.895.

Params#	NU(0)	NU(1)	NU(10)	DPP
×1/2	0.854	0.874	**0.879**	0.856
×1/3	0.781	0.849	**0.864**	0.763

4.2 Experiments with VGG16 on ImageNet

We perform pruning on pretrained VGG16. For pruning, we use randomly selected 5000 training images. All the images are resized so that the shorter side is 256. We apply random horizontal flip and 224×224 random crop to the training images, and apply 224×224 center crop to the test images. Note that we do not conduct retraining the models after/during pruning. As one of our motivations is to save time for retraining, we want to see how well we can maintain the accuracy even without retraining.

4.2.1 Pruning Fully Connected Layers

We prune "fc1" and "fc2". The pruning ratios (the ratio of the number of neurons/channels to be pruned in a certain layer) for both layers are set evenly, and the pruning ratio is tuned so that the model size (the number of the parameters) becomes 1/2 and 1/3.

Table 1 shows the results. We only see a marginal defference between our method without extra surgeries and DPP at ×2 compression ratio. With extra surgeries, our method easily outperforms DPP. At ×3 compression ratio, our method is obviously better even without extra surgeries. After 10 extra surgeries, we only suffer 0.031 accuracy drop, while DPP suffers 0.132.

We also briefly report the computational time of Neuro-Unification. In Neuro-Unification, the computation of Algorithm 1 accounts for most of the whole computation. When $N = 4{,}096$, it took about 177 seconds computing with single thread of Intel(R) Core(TM) i7-2600K CPU @ 3.40GHz. We believe this is fast and practical enough.

4.2.2 Pruning Convolutional Layers

We set the pruning ratios in Conv1, Conv2, Conv3 and Conv4 to 6.5 : 6 : 6 : 5.5. We do not prune the layers in Conv5, because we found out that the layers in Conv5 are not redundant, and pruning those layers results in steep accuracy drops.

For pruning, we use the same subdataset that is used in Sect. 4.2.1. In the convolutional layers, the behaviors of the channels are denoted by $\{\Phi_i \in \mathbb{R}^{Nh^2 \times s^2} | i \in B\}$. Since the Φ's are too large matrices, we take randomly selected 50000 rows and deleted the other rows.

Table 2 shows the result. Our method, especially with 10 times of extra surgeries, easily outperforms the other methods. At ×2 speed-up, we only suffer 5.0% accuracy drop even though the model has not been retrained. ThiNet

Table 2 Results of VGG16 on ImageNet, convolutional layers. The baseline top-5 accuracy is 0.895.

FLOPs	NU(0)	NU(1)	NU(10)	OP	ThiNet
×1/2	0.375	0.802	**0.845**	0.024	0.245
×1/3	0.097	0.616	**0.729**	0.006	0.022

Table 3 Results of ablation study, VGG16 on ImageNet, fully connected layers. Performances of Neuro-Unification with different number of images for encoding. The baseline top-5 accuracy is 0.895.

Params	Images# for Neuro-Coding		
	128	1024	5000
×1/2	0.704	0.769	0.854
×1/3	0.388	0.537	0.781

is better than OP, however, much worse than our method even without extra surgeries. As we discuss later, this is because Neuro-Unification takes more sophisticated approach for surgery than that of ThiNet.

4.2.3 Ablation Study

For evaluating how important it is to encode the neuron behaviors, we set the number of samples used for Neuro-Coding as 128, 1024, 5000, and applied Neuro-Unification to the fully connected layers of VGG16. We do not conduct extra surgeries in this experiment.

See Table 3. The trend is that the more samples are used for Neuro-Coding, the better the performance of Neuro-Unification is. It implies that when the number of samples for Neuro-Coding is not many enough, the neuron behaviors cannot be described well by the behavioral vectors, and the performance of Neuro-Unification becomes poor. Thus, it is crucial to use many data for Neuro-Coding enough to describe the neuron behaviors well.

4.3 Experiments with ResNet-56 on Cifar-10

We conduct experiments with ResNet-56 on cifar-10 taken from [21]. In this experiment, we not only prune but also retrain the models for 10 epochs at 0.01 learning rate and another 10 epochs at 0.001 learning rate, and evaluate the performances before and after the retraining.

For pruning, we use 5120 randomly sampled images. The training images are padded by 4 pixels in each side, and randomly cropped at 32×32. The rest of the parameters are as below: the momentum is 0.9, the minibatch size is 128, the gradients was computed by SGD with cross entropy loss.

The targets are only the convolutional layers since ResNet has only one fully connected layer. ResNet-56 has 27 branched units that have 2 convolutional layers, such as the one shown in Fig. 5. We conduct pruning on the layers that are not located in the branching points and the merge points (Therefore, the targets are 27 layers). We set the pruning ratio in each layer constantly. As we think the results in Sect. 4.2.2 is enough to conclude that our method is better than OP, we use only ThiNet for comparison.

Table 4 shows the result. Consistently with the other

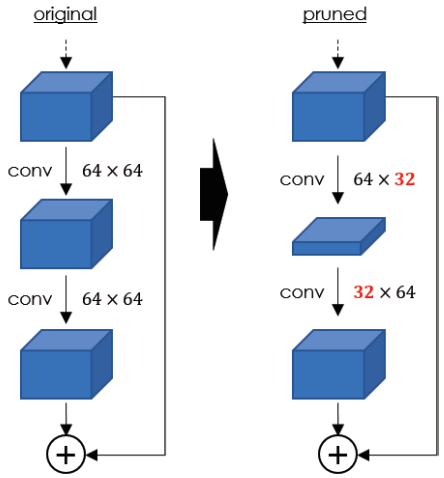

Fig. 5 The illustration of a branched structure of ResNet-56. The data tensors are forwarded through the both paths and are eventually merged. Thus, we prune only the intermediate layers so that we keep the output shape unchanged at the end of the residual unit.

Table 4 ResNet-56 on cifar-10. The baselines are 0.934 (top-1).

FLOPs	Retraining	NU(0)	NU(1)	NU(10)	ThiNet
×1/2	No	0.655	0.810	0.834	0.827
×1/2	Yes	0.927	0.924	0.925	0.923

experiments, our method outperforms the other methods when we do not conduct retraining. After fine-tuning, the accuracy of our pruned ResNet-56 is from 0.924 to 0.927. This is competitive with ResNet-32 in [14] (The accuracy is 0.925.), while our pruned model has fewer FLOPs than unpruned ResNet-32 by approximately 10%.

It should be noted that the performance difference of our method and ThiNet is not very significant after retraining. Although, the strength of our Neuro-Unification is not only preserving the accuracy but also optimizing the network architectures. We expect that we can further outperform ThiNet by extending Neuro-Unification in the future. This is mensioned in Sect. 6.

5. Conclusion

We proposed Neuro-Coding and Neuro-Unification, a method for neural network pruning and surgery. The proposed mehtod encodes the behaviors of the neurons or the channels by their outputs and unify the neurons with similar behaviors, in other words, we prune a neuron and transfer its weights to another one. Moreover, we can conduct extra surgeries, where we let more than one neurons to emulate the behavior of a pruned one. Therefore, our method enables to maintain the model performances in high level without retraining. On the experiments, the proposed methods performs much better than the existing methods, and the effectiveness of the proposed method is confirmed.

6. Future Work

We plan to extend Neuro-Unification so that we can optimize the pruning ratio in each layer. Currently, the model is pruned with manually determined pruning ratios and retrained. It requires lots of labors to optimize the pruning ratios, because the model accuracy achieved by retraining can be known only after retraining. On the other hand, our method can preserve the accuracy of the model very well even without retraining. We can judge if we should stop or keep on pruning based on the error after surgery, when we prune each layer. This is very efficient, because we can know the proper pruning ratios without conducting retraining. Note that this is only possible with our method, because we can preserve the accuracy of the model very well even without retraining and the existing methods cannot preserve the accuracy as well as ours.

References

[1] A. Aghasi, A. Abdi, N. Nguyen, and J. Romberg, Convex pruning of deep neural networks with performance guarantee, Proc. Neural Information Processing Systems, pp.1–10, 2017.
[2] M. Courbariaux, Y. Bengio, and J.P. David, Binaryconnect: Training deep neural networks with binary weights during propagations, Proc. Neural Information Processing Systems, pp.1–9, 2015.
[3] 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.
[4] S. Han, J. Pool, J. Tran, and W. Dally, Learning both weights and connections for efficient neural networks, Proc. Neural Information Processing Systems, pp.1–9, 2015.
[5] B. Hassibi, D.G. Stork, and G.J. Wolff, "Optimal brain surgeon and general network pruning," Proc. International Conference on Neural Networks, pp.293–299, 1993.
[6] T. He, Y. Fan, Y. Qian, T. Tan, and K. Yu, "Reshaping deep neural network for fast decoding by vertex-pruning," Proc. International Conference on Acoustics, Speech and Signal Processing, pp.245–249, 2014.
[7] J. Yim, D. Joo, J. Bae, and J. Kim, "A gift from knowledge distillation: Fast optimization, network minimization and transfer learningk," Proc. Computer Vision and Pattern Recognition, pp.4133–4141, 2017.
[8] Y. LeCun, J.S. Denker, and S.A. Solla, Optimal brain damage, Proc. Advances in Neural Information Processing Systems, pp.598–605, 1989.
[9] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H.P. Graf, Pruning filters for efficient convnets, Proc. International Conference on Learning Representations, pp.1–13, 2017.
[10] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Penksy, "Sparse convolutional neural networks," Proc. Computer Vision and Pattern Recognition, pp.806–814, 2015.
[11] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, Pruning convolutional neural networks for resource efficient inference, Proc. International Conference on Learning Representations, pp.1–17, 2017.
[12] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, Automatic differentiation in pytorch, NIPS-W, 2017.
[13] K. Simonyan and A. Zisserman, Very deep convoolutional networks for large-scale image recognition, Proc. International Conference on Learning Representations, pp.1–14, 2015.
[14] 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.
[15] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," CVPR, pp.248–255, 2009.
[16] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton, Cifar-10 (canadian institute for advanced research).
[17] S. Srinivas and R.V. Babu, "Data-free parameter pruning for deep neural networks," Proc. British Machine Vision Conference, pp.31.1–31.12, 2015.
[18] J. Xue, J. Li, and Y. Gong, Restructuring of deep neural network acoustic models with singular value decomposition, Proc. INTERSPEECH, pp.2365–2369, 2013.
[19] R. Yu, A. Li, C. Chen, J. Lai, V.I. Morariu, X. Han, M. Gao, C. Lin, and L.S. Davis, Pruning filters for efficient convnets, Proc. Computer Vision and Pattern Recognition, pp.9194–9203, 2018.
[20] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, cuDNN: Efficient Primitives for Deep Learning, Technical report, 2011.
[21] akamaster, Proper implementation of resnet-s for cifar10/100 in pytorch that matches description of the original paper, 2019.
[22] 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, 2017.

Appendix:

Proof of Theorem 1: Let $a_{i(j)}$ denote the j^{th} entry of $\boldsymbol{a}_i$. Then, we need to prove

$$\sum_{j=1}^{P}\left(\sum_{i=1}^{p} a_{i(j)}\right)^2 \leq p\sum_{j=1}^{P}\sum_{i=1}^{p} a_{i(j)}^2. \tag{A·1}$$

Based on Cauchy-Schwarz inequality, we have

$$\begin{aligned}\left(\sum_{i=1}^{p} a_{i(j)}\right)^2 &= \left(\sum_{i=1}^{p} 1\cdot a_{i(j)}\right)^2 \\ &\leq \left(\sum_{i=1}^{p} 1^2\right)\left(\sum_{i=1}^{p} a_{i(j)}^2\right) = p\sum_{i=1}^{p} a_{i(j)}^2.\end{aligned} \tag{A·2}$$

Therefore, we have

$$\sum_{j=1}^{P}\left(\sum_{i=1}^{p} a_{i(j)}\right)^2 \leq \sum_{j=1}^{P} p\sum_{i=1}^{p} a_{i(j)}^2 = p\sum_{j=1}^{P}\sum_{i=1}^{p} a_{i(j)}^2. \tag{A·3}$$

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 Pattern Recognition and Deep Learning.

Yuki Isoda received the bachelor degree in Faculty of Systems Engineering, Wakayama University in 2019. He is currently a graduate student in the Faculty of Systems Engineering, Wakayama University. His research interests include Pattern Recognition, Computer Vision, Artificial Intelligence, and Deep Lerning.

Sarimu Inoue received the bachelor degree in Faculty of Systems Engineering, Wakayama University in 2018. He is currently a graduate student in the Faculty of Systems Engineering, Wakayama University. His research interests include Pattern Recognition, Computer Vision, Artificial Intelligence, and Deep Lerning.

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 Lerning. He received the Marr Prize at the International Conference on Computer Vision in1995, 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) in2007. He is a member of the IPSJ, the ISCIE, and the IEEE.