A Layer Decomposition-Recomposition Framework for Neuron Pruning towards Accurate Lightweight Networks

Weijie Chen, Yuan Zhang, Di Xie, Shiliang Pu

Hikvision Research Institute

{chenweijie5, zhangyuan, xiedi, pushiliang}@hikvision.com

Abstract

Neuron pruning is an efficient method to compress the network into a slimmer one for reducing the computational cost and storage overhead. Most of state-of-the-art results are obtained in a layer-by-layer optimization mode. It discards the unimportant input neurons and uses the survived ones to reconstruct the output neurons approaching to the original ones in a layer-by-layer manner. However, an unnoticed problem arises that the information loss is accumulated as layer increases since the survived neurons still do not encode the entire information as before. A better alternative is to propagate the entire useful information to reconstruct the pruned layer instead of directly discarding the less important neurons. To this end, we propose a novel *Layer Decomposition-Recomposition Framework* (LDRF) for neuron pruning, by which each layer's output information is recovered in an embedding space and then propagated to reconstruct the following pruned layers with useful information preserved. We mainly conduct our experiments on ILSVRC-12 benchmark with VGG-16 and ResNet-50. What should be emphasized is that our results before end-to-end fine-tuning are significantly superior owing to the information-preserving property of our proposed framework. With end-to-end fine-tuning, we achieve state-of-the-art results of $5.13\times$ and $3\times$ speed-up with only 0.5% and 0.65% top-5 accuracy drop respectively, which outperform the existing neuron pruning methods.

Introduction

As the success of convolutional neural networks in computer vision, more and more researchers pay attention to their deployment on embedded sensors and mobile devices. Due to the enormous computational cost and storage overhead, it becomes an appealing subject on how to obtain a more efficient model without performance decline.

Recently, many compression and acceleration methods have been proposed to solve this problem. One category of the most popular methods is pruning, which can induce sparsity to weight matrices so that only the non-zero parts are involved in computation and storage. (Guo, Yao, and Chen 2016; Han et al. 2015; Liu et al. 2015; Lebedev and Lempitsky 2016; Wen et al. 2016; Molchanov et al. 2017; Li et al. 2017; Ye et al. 2018; Yu et al. 2018).
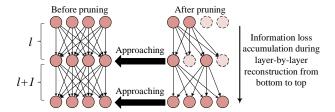
Figure 1: The information loss of the generic layer-by-layer neuron pruning method. The less important input neurons in layer l are pruned, and its output neurons are optimized to approximate the original ones. Repeatedly, only a part of important ones are propagated to reconstruct layer $l + 1$, which induces information loss during the information propagation. Less importance does not mean absolute uselessness. A subset of neurons by selection is difficult to encode the entire label-related information especially when aggressive pruning. Such information loss is accumulated layer by layer, which deeply damages the network accuracy. For instance, (He, Zhang, and Sun 2017) prune VGG-16 up to $5\times$ acceleration in such layer-by-layer pruning mode with **22%** top-5 accuracy drop before end-to-end fine-tuning.

Neuron pruning is one kind of structure pruning, which induces channel level and filter level sparsity of weight matrices so that it can directly compress the network into a slimmer one. To the best of our knowledge, most of state-of-the-art results are implemented in a layer-by-layer pruning and optimization manner (Aghasi, Nguyen, and Romberg 2017; He, Zhang, and Sun 2017; Luo, Wu, and Lin 2017). However, as shown in Fig.1, the survived neurons are inadequate to recover the original output due to the information loss, which accumulates errors as layer increases. In this way, its accuracy is deeply damaged and is difficult to restore the original performance. *In fact, this is an intrinsic drawback of the existing neuron pruning frameworks.* Their proposed greedy neuron selection methods actually attempt to ease this problem. However, without framework improved, this problem is unable to avoid, no matter how optimal the neuron selection is, especially when aggressive pruning.

Intuitively, a better solution is to squeeze the useful information of the discarded neurons into the survived ones and then propagate to reconstruct the following pruned lay-

ers. Inspired by this idea, we propose a new layer-by-layer optimization framework for neuron pruning named Layer Decomposition-Recomposition Framework (LDRF). In this framework, we elegantly generate the embedding space to each layer through cross-channel decomposition which serves as an information buffer. The information of the discarded neurons is squeezed into the survived ones in the embedding space before they are fed to reconstruct the following pruned layers. It efficiently avoids the emergence of information loss. Symmetrically, after pruning and reconstruction, each adjacent linear layers are recomposed to avoid network depth increase. Besides, via the analysis of the information propagation characteristic of Rectified Linear Units layer, i.e. ReLU (Dahl, Sainath, and Hinton 2013), we propose a method to evaluate the pruning ratio range for each layer. It simplifies our optimization process and reduces our convergence time. We first conduct experiments on CIFAR-10 (Krizhevsky and Hinton 2009) with VGG-9 (Courbariaux, Bengio, and David 2015) to compare our method with an existing layer-by-layer pruning method(He, Zhang, and Sun 2017). *It is worth noting that the performance of pruned networks before end-to-end fine-tuning are the metrics to verify the effectiveness of such pruning methods.* Sadly, the majority of related works only compare the results after fine-tuning. In this experiment, the performance of our method significantly outperforms other methods before fine-tuning. Ultimately, we evaluate the generalization of our method on the large-scale ImageNet 2012 classification dataset (Russakovsky et al. 2015) with two most representative models, say VGG-16 (Simonyan and Zisserman 2015) and ResNet-50 (He et al. 2016), and achieve state-of-the-art results by $5.13\times$ and $3\times$ speed-up with only **0.5%** and **0.65%** top-5 accuracy drop respectively.

Our major contributions are summarized as follows:

- To the best of our knowledge, we are the first one to discover the hidden drawback of the existing layer-wise neuron pruning methods, i.e. the large information loss accumulation during the information propagation layer by layer, which leads to severe accuracy damage before end-to-end fine-tuning.

- We propose a new and novel *Layer Decomposition-Recomposition Framework* for neuron pruning to solve this problem effectively which preserves the useful information during the information propagation.

- We achieve state-of-the-art results which outperform the existing methods. Additionally, before end-to-end fine-tuning, we have to emphasize that our results are significantly superior especially when aggressive pruning, which validates the impact of our proposed framework.

Related Works

Connection pruning aims to prune unimportant connections and turns the dense weight matrices into sparse ones. As early as 1990s, Lecun *et al.* (Lecun, Denker, and Solla 1990) and Hassibi *et al.* (Hassibi and Stork 1993) successively put forward optimal brain damage and optimal brain surgeon to remove useless parameters, so as to reduce model storage and computational overhead. Han *et al.* (Han et al.

2015) propose a method to iteratively remove unimportant connections through l_1 or l_2 regularization. They claim that the connections with small weight below a given threshold are useless and it is harmless to the network accuracy if one removes them. However, it is difficult to accelerate in practice since the produced filter structures are irregular.

Structure pruning is one kind of methods to prune filters into regular structure and brings convenience to practical deployment, especially on parallel computational platforms such as GPUs. Anwar *et al.* (Anwar, Hwang, and Sung 2017) adopt the particle filter to locate the unimportant region in channel level, kernel level and intra kernel dimensions. Li *et al.* (Li et al. 2017) simply use the evaluation criterion based on magnitude to prune unimportant filters. Molchanov *et al.* (Molchanov et al. 2017) compare different evaluation criteria for pruning and find that criterion based on Taylor expansion has superior performance among them. These works are tested in an end-to-end training manner. However, in a layer-by-layer pruning manner, a recent research (Mittal et al. 2018) finds that the performances of pruned networks with different neuron importance evaluation criteria are comparable. They do several experiments and show that even the performance of random pruning is comparable with the one by other pruning criteria. This work inspires us that in a layer-by-layer pruning mode the core of pruning may lie in the capacity assignment to each layer (pruning ratio) which theoretically denotes the ceiling of the accuracy of the pruned networks instead of pruning criteria.

Group Lasso (Yuan and Lin 2006) is an efficient approach to learn the valid capacity of the network adaptively. Alvarez *et al.* (Alvarez and Salzmann 2016) introduce group lasso in the training process to learn the number of neurons in each layer. Wen *et al.* (Wen et al. 2016) apply group lasso to exploit the redundancy in filters, channels, filter shapes, and network depth dimension. Besides, Liu *et al.* (Liu et al. 2017) take the scaling factor in BN layer to measure the importance of each channel and introduce a sparsity penalty on the scaling factors in the training process to learn the channel number. However, these methods are trained in an end-to-end manner and are difficult to recover the original accuracy. He *et al.* (He, Zhang, and Sun 2017), Luo *et al.* (Luo, Wu, and Lin 2017) and Aghasi *et al.* (Aghasi, Nguyen, and Romberg 2017) provide an analogous layer-wise optimization method to control the output consistency before and after pruning in each layer as shown in Fig. 1, which can be formulated as:

$$\min_{W} \eth(W) \quad \text{s.t.} \quad \| \mathbf{y} - \mathbf{y}' \|_F^2 \leq \epsilon \tag{1}$$

where $\mathbf{y}$ and $\mathbf{y}'$ correspond to the output in one layer before and after pruning respectively, ϵ is the affordable reconstruction error, and $\eth(W)$ denotes group lasso or other sparsity-induced regularization of the layer's weight matrix W.

Such layer-by-layer optimization achieves better performance compared with the end-to-end training manner, but it still has some problems on information propagation. As shown as Fig.1, there exists information loss during the reconstruction from shallow layers to high layers, since some less important but still informative neurons are discarded. This is the exact problem we want to solve in this paper.
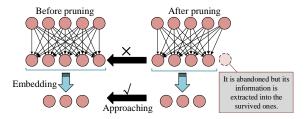
Figure 2: Due to the output dimension inconsistency, the output neurons before and after pruning are projected into a shared embedding space for information compensation.

Our Approach

Pruning for Single Layer

Consider a single layer $\mathbf{y} = \gamma(\mathbf{W}^T\mathbf{x})$, where $\mathbf{x} \in \mathbb{R}^m$ is the layer input, $\mathbf{W} \in \mathbb{R}^{m \times n}$ is a pretrained transformation matrix, $\mathbf{y} \in \mathbb{R}^n$ is the output, and $\gamma(\cdot)$ is a popular nonlinear activation function namely ReLU $(\max(\cdot, 0))$ in this paper. Layer with this pattern is the basic unit of many popular networks, such as VGG-Net (Simonyan and Zisserman 2015), GoogLeNet (Szegedy et al. 2015) and ResNet (He et al. 2016).

In order to compress the layer, we aim to reduce output neurons while keeping the most information contained in the remaining neurons $\mathbf{y}' \in \mathbb{R}^{n'}$. Here $\mathbf{y}' = \gamma(\mathbf{W}'^T\mathbf{x})$ and $\mathbf{W}' \in \mathbb{R}^{m \times n'}$ is a new transformation matrix with fewer columns ($n' \le n$). As the dimension between $\mathbf{y}$ and $\mathbf{y}'$ is different, we suppose that the information of $\mathbf{y}$ and $\mathbf{y}'$ can be transformed into a shared embedding space, so that the difference can be measured which is helpful for compensating the information of discarded neurons, as shown in Fig. 2. To this end, we formulate the neuron pruning problem as:

$$\min_{0 < n' \le n} n' \quad \text{s.t.} \parallel \mathbf{y} - (\mathbf{Q}^T)^{-1}\mathbf{Q}^T\mathbf{y} \parallel_F^2 + \lambda \parallel \mathbf{Q}^T\mathbf{y} - \mathbf{Q}'^T\mathbf{y}' \parallel_F^2 \le \epsilon \tag{2}$$

where $\parallel \cdot \parallel_F$ is Frobenius norm, λ is a weighted coefficient, and ϵ is our affordable information loss which is a sufficiently small scalar value. $\mathbf{Q} \in \mathbb{R}^{n \times z}$ and $\mathbf{Q}' \in \mathbb{R}^{n' \times z}$ in Eqn.2 are the projection matrices to embed $\mathbf{y}$ and $\mathbf{y}'$ into a comparable space while not losing useful information. The first term of Eqn.2 is to project the embedded representations back to original space with minimal error which makes sure that the embedding space is adequate to accommodate the entire information. However, Eqn.2 is difficult to solve because the shape of $\mathbf{W}'$ and $\mathbf{Q}'$ is changeable during optimization. To solve this problem, we fix the shape of $\mathbf{W}'$ and $\mathbf{Q}'$ as ones of $\mathbf{W}$ and $\mathbf{Q}$, and introduce a vector $\mathbf{m}$ of size n for neuron selection. Hence, $\mathbf{y}' = \mathbf{m} \odot \gamma(\mathbf{W}'\mathbf{x})$ where $\odot$ denotes point-wise multiplication. Note that the entries of $\mathbf{m}$ are binary scalars. A 0 means that the corresponding neuron is useless which can be pruned, while a 1 indicates the expressive neuron. In this manner, the optimization problem to find the minimum of n' is reduced to the problem to find the most sparse solution of $\mathbf{m}$ (l_0 minimization problem). However, l_0 minimization problem is a NP-hard problem, so we

relax l_0 to l_1 and convert Eqn.2 into the following optimization problem:

$$\operatorname*{arg\,min}_{\mathbf{Q},\mathbf{Q}',\mathbf{W}',\mathbf{m}} \parallel \mathbf{y} - (\mathbf{Q}^T)^{-1}\mathbf{Q}^T\mathbf{y} \parallel_F^2 + \lambda_1 \parallel \mathbf{Q}^T\mathbf{y} - \mathbf{Q}'^T(\mathbf{m} \odot \gamma(\mathbf{W}'^T\mathbf{x})) \parallel_F^2 + \lambda_2 \parallel \mathbf{m} \parallel_1 \tag{3}$$

where λ_1 and λ_2 are the weighted coefficients. Naturally, the projection matrix $\mathbf{Q}$ can be obtained by Principal Component Analysis to a batch of $\mathbf{y}$. Then, fixing $\mathbf{Q}$ and initializing $\mathbf{Q}'$ and $\mathbf{W}'$ with $\mathbf{Q}$ and $\mathbf{W}$ to solve the remaining terms.

Pruning for Entire Network

In this section, the pruning method for one layer is generalized to the entire network in a layer-by-layer manner. $\mathbf{y}_l = \gamma(\mathbf{W}_l^T\mathbf{y}_{l-1})$ is the output of lth layer in a given pre-trained network. $\mathbf{y}'_l = \mathbf{m}_l \odot \gamma(\mathbf{W}'^T_l\mathbf{y}'_{l-1})$ is the compact output of lth pruned layer that attempts to produce similar information as $\mathbf{y}_l$, where $\mathbf{y}'_{l-1}$ is the generated output neurons in the l-1th layer which has been pruned. In line with pruning one layer, we aim to reduce the neurons of $\mathbf{y}'_l$ while keeping its information comparable with $\mathbf{y}_l$ in a shared embedding space. Thus our objective function is

$$\operatorname*{arg\,min}_{\mathbf{Q}_l,\mathbf{Q}'_l,\mathbf{W}'_l,\mathbf{m}_l} \parallel \mathbf{y}_l - (\mathbf{Q}_l^T)^{-1}\mathbf{Q}_l^T\mathbf{y}_l \parallel_F^2 + \lambda_1 \parallel \mathbf{Q}_l^T\mathbf{y}_l - \mathbf{Q}'^T_l(\mathbf{m}_l \odot \gamma(\mathbf{W}'^T_l\mathbf{y}'_{l-1})) \parallel_F^2 + \lambda_2 \parallel \mathbf{m}_l \parallel_1 \tag{4}$$

Different from pruning single layer, $\mathbf{Q}'_l$ and $\mathbf{W}'_l$ cannot be initialized with $\mathbf{Q}_l$ and $\mathbf{W}_l$ because the numerical representations of $\mathbf{y}_{l-1}$ and $\mathbf{y}'_{l-1}$ are not identical although they contain similar information[1]. A better solution is to decompose $\mathbf{Q}_{l-1}$ from $\mathbf{W}_l$ ($\mathbf{W}_l = \mathbf{Q}_{l-1}\mathbf{R}_l$). In this way, we can directly feed the embedding representation $\mathbf{Q}'^T_{l-1}\mathbf{y}'_{l-1}$ to the next layer with transformation matrix $\mathbf{R}'_l$, where $\mathbf{R}'_l$ and the following embedding matrix $\mathbf{Q}'_l$ can be initialized with $\mathbf{R}_l$ and $\mathbf{Q}_l$. Moreover, in the situation of multi-layers, evaluating the expressiveness of $\mathbf{Q}^T\mathbf{y}$ is equivalent to evaluate whether it can flow to the next layer without information loss. So the first term of Eqn.4 is better to replace by $\parallel \mathbf{W}_{l+1}^T\mathbf{y}_l - \mathbf{R}_{l+1}^T\mathbf{Q}_l^T\mathbf{y}_l \parallel_F^2$, and we can reformulate the objective function as follows:

$$\operatorname*{arg\,min}_{\mathbf{Q}_l,\mathbf{R}_{l+1},\mathbf{Q}'_l,\mathbf{R}'_l,\mathbf{m}_l} \parallel \mathbf{W}_{l+1}^T\mathbf{y}_l - \mathbf{R}_{l+1}^T\mathbf{Q}_l^T\mathbf{y}_l \parallel_F^2 + \lambda_1 \parallel \mathbf{Q}_l^T\mathbf{y}_l - \mathbf{Q}'^T_l(\mathbf{m}_l \odot \gamma(\mathbf{R}'^T_l(\mathbf{Q}'^T_{l-1}\mathbf{y}'_{l-1}))) \parallel_F^2 + \lambda_2 \parallel \mathbf{m}_l \parallel_1 \tag{5}$$

$\mathbf{R}_{l+1}$ and $\mathbf{Q}_l$ in the first term can be determined by applying Singular Value Decomposition (SVD) to $\mathbf{W}_{l+1}$. After that, given that $\mathbf{R}_l$ and $\mathbf{Q}'_{l-1}$ have been determined in the l-1th layer, we initialize $\mathbf{Q}'_l$ and $\mathbf{R}'_l$ with $\mathbf{Q}_l$ and $\mathbf{R}_l$, and then solve $\mathbf{Q}'_l$, $\mathbf{R}'_l$ and $\mathbf{m}_l$. Repeating this process, the entire network can be pruned in a layer-by-layer manner. According to the mask vector $\mathbf{m}_l$ in each layer, we remove useless neurons and finally get a slimmer network. In order to avoid

---

[1] In single layer case, both input information is $\mathbf{x}$, so they can inherit from $\mathbf{Q}$ and $\mathbf{W}$ safely.
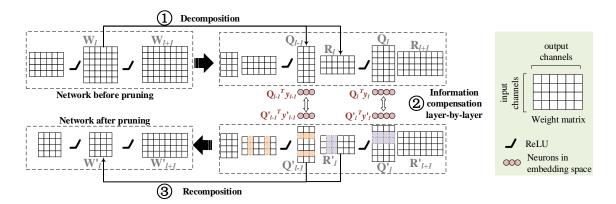
Figure 3: A Layer Decomposition-Recomposition Framework for neuron pruning. Layer decomposition is first applied to each layer to generate its embedding space. Then the redundant neurons are pruned, which is equivalent to prune the output channels of $\mathbf{R}_l'$ and the corresponding input channels of $\mathbf{Q}_l'$, while compensating the information loss of discarded neurons in each embedding space. It means one should optimize the pruned $\mathbf{R}_l'$ and $\mathbf{Q}_l'$ to force $\mathbf{Q'}_l^T \mathbf{y}_l'$ to approximate $\mathbf{Q}_l^T \mathbf{y}_l$. Ultimately, a slimmer network without depth increasing is returned through layer recomposition. (Best viewed in color)

network depth increasing, we can get the new transformation matrix $\mathbf{W}_l'$ for each layer by recomposing $\mathbf{Q}_{l-1}'$ and $\mathbf{R}_l'$ ($\mathbf{W}' = \mathbf{Q}_{l-1}'\mathbf{R}_l'$). We summarize the entire procedures as a Layer Decomposition-Recomposition Framework for neuron pruning as shown in Fig. 3. From this figure, it is easy to see that the information of discarded neurons is recovered in the embedding space and is propagated to reconstruct the following pruned layers with information preserved.

A More Efficient Solution

The optimization of Eqn.5 is divided into layer decomposition and information compensation for neuron pruning. However, in practice, solving the latter one is extremely time-consuming. Additionally, λ_2 in the third term of Eqn.5 is not only difficult to determine, but also empirical to some extent[2]. To deal with both problems, we propose a method to evaluate each layer's pruning ratio range under the guarantee of information preserved propagation.

The information propagation characteristic of ReLU is the prerequisite to inspire us to propose our method. It is a non-linear activation layer which collapses negative signals to zero and only allows positive signals through the layer. It may bring about information loss and hurt the expression capability of current layer. For irredundant signals through ReLU layer, it is inevitable to lose the flow-in information unless all input signals are positive. In this way ReLU degenerates to an identity transformation and loses its nonlinearity. To preserve the flow-in information and maintain the nonlinearity of ReLU, the Lemma 2 of the appendix in MobileNetV2 (Sandler et al. 2018) demonstrates that the irredundant input signals should be expanded to a higher dimensional representation for introducing artificial redundancies before they are fed into the ReLU layer so as to maintain the invertibility of ReLU. This analysis directly provides us an

idea to estimate the range of valid neurons for each layer, and guides us to assign pruning ratio to each layer.

To achieve this, we pick out the first term of Eqn.5 and solve it for each layer at first before solving the remaining terms. In addition, we add a constrain to the first term to calculate the lowest dimensional space to accommodate flow-in information in each layer. So the first term becomes:

$$\min \ z_l \quad \text{s.t.} \ \| \ \mathbf{W}_l^T\mathbf{y}_{l-1} - \mathbf{R}_l^T(\mathbf{Q}_{l-1}^T\mathbf{y}_{l-1}) \ \|_F^2 \leq \epsilon \quad (6)$$

where ϵ is our affordable information loss, and $\mathbf{Q}_{l-1}^T\mathbf{y}_{l-1}$ is an irredundant representation before it is fed into ReLU layer and its dimension is z_l. We can get singular values of $\mathbf{W}_l$ by SVD decomposition and keep the cumulative sum of the largest ones as small as possible for rank reduction. Once the minimum of z_l is determined, we can learn that $\| \ \mathbf{m}_l \ \|_0 > z_l$ is a necessary condition for the guarantee of lossless information propagation through the following ReLU layer, because only if $\mathbf{R}_l$ acts as an expansion function can $\mathbf{Q}_{l-1}^T\mathbf{y}_{l-1}$ possibly passes through ReLU layer without information loss. This is the same motivation from which Sandler et al. (Sandler et al. 2018) design inverted residual and linear bottlenecks for MobileNetV2, but from different views. Therefore, the number of preserving output neurons should at least range in (z_l, n_l).

Moreover, inspired by the experiments in (Mittal et al. 2018) that even random pruning can provide a comparable result, it is the remaining capacity of each layer instead of a specific neuron selection criterion that matters to the final recovered network performance, on the premise of layer-by-layer training mode. In other words, we just concern about the amount of loss rather than which part of the information is selected to lose. Specific to our framework, as shown in Fig.4, our framework is insensitive to neuron selection method since the intermediate pruned information is easy to restore in the output embedding space owing to the complete input information. So we simply use Top-k to set $\mathbf{m}_l = [1_1, 1_2, \ldots, 1_{k_l}, 0_{k_l+1}, \ldots, 0_{n_l}]$ where k_l is the re-

---

[2]For instance, if we want to speed the network up to $2\times$, we have no idea how to set an exact scalar value to λ_2 unless we run many trial-and-error experiments.
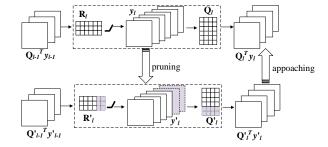
Figure 4: Our framework propagates the entire information generated from the preceding layers to reconstruct current pruned layer WITHOUT input neuron ($\mathbf{Q}'^T_{l-1}\mathbf{y}'_{l-1}$) selection. During the reconstruction, the information in the output embedding space ($\mathbf{Q}'^T_l\mathbf{y}'_l$) is approaching to the original one, which indirectly squeeze the useful information of the intermediate pruned neurons into the survived ones ($\mathbf{y}'_l$).

maining capacity of each layer ranging in $(z_l, n_l]$. Therefore, Eqn.5 can be simplified as:

$$\underset{\mathbf{Q}'_l, \mathbf{R}'_l}{\arg\min} \parallel \mathbf{Q}^T_l\mathbf{y}_l - \mathbf{Q}'^T_l(\mathbf{m}_l \odot \gamma(\mathbf{R}'^T_l(\mathbf{Q}'^T_{l-1}\mathbf{y}'_{l-1}))) \parallel^2_F$$

(7)

Eqn.7 is more efficient to solve by gradient descent optimizers once $\mathbf{m}_l$ is specified in advance.

Implementation Details

Different from other methods, as shown in Fig.3, our method requires layer decomposition before pruning to obtain each layer's embedding space. In the following experiments, we first apply SVD cross-channel decomposition to each weight matrix. We decompose $k \times k$ kernels into $k \times k$ and 1×1, which serve as an embedding matrix and transformation matrix, respectively.

To further estimate the valid capacity of each layer, we lower the rank of each layer as much as possible during SVD decomposition under the guarantee of barely accuracy drop (its another attached benefit is the reduction of optimization parameter space of the pruned layers). According to the information propagation characteristic of ReLU, the valid number of output neurons is at least greater than this rank.

Evaluation and Results

Benchmark Datasets

In this section, we mainly conduct our experiments on CIFAR-10 (Krizhevsky and Hinton 2009) and ILSVRC-2012 (Russakovsky et al. 2015) to demonstrate the less information loss property of our proposed method.

- **CIFAR-10** is a dataset for 10-categories image classification, consisting of 50k color images for training and 10k color images for testing with resolution of 32×32. For this experiment, we adopt VGG-9 (Courbariaux, Bengio, and David 2015) to compare our results with another state-of-the-art neuron pruning method, and analyze the advantages of our method in detail.

- **ImageNet** is a large scale image classification dataset, which comprises 1.28 million images from 1000 categories for training and 50k images for validation. For this experiment, we adopt two of the most representative networks, i.e., VGG-16 (Simonyan and Zisserman 2015) and ResNet-50 (He et al. 2016) to demonstrate our method's generalization to large scale networks.

VGG-9 on CIFAR-10

VGG-9 (Courbariaux, Bengio, and David 2015) is derived from VGGNet (Simonyan and Zisserman 2015) and its architecture can be denoted as "$(2 \times 64C3) - MP2 - (2 \times 128C3) - MP2 - (2 \times 256C3) - MP2 - (2 \times 512FC) - 10/100FC - Softmax$". In order to improve its convergence, a batch normalization layer is appended after each convolutional layer. We train this network with SGD optimization method with mini-batch size 100, weight decay 1e-4 and momentum 0.9. Training is started by a learning rate 0.1 with linear decaying policy, and is stopped after 100k iterations. For data augmentation, only horizontal flipping and random cropping are adopted. 92% accuracy is achieved as our baseline after training. After layer-by-layer pruning and reconstruction, the slimmer network is fine-tuned with 10k iterations by a learning rate 0.01 with linear decaying policy. In the following experiments, "speed-up" means theoretical speed-up, namely FLOPs reduction in this paper.

Since almost all computational cost lies in convolutional layers instead of fully-connected layers, we only prune the preceding six convolutional layers of VGG-9 in this experiment in order to reach largest speed-up. Before that, we apply SVD decomposition to these layers and analyze the low ranks of them. We find that when we lower the preserving cumulative sum of largest singular values in each layer to 0.55, the accuracy of the decomposed VGG-9 can be exactly recovered through short-term fine-tuning and the ranks are shown in Tab. 1. Based on the discussion about the valid number of neurons in the above section, we can assign a proper pruning ratio range to each layer. As shown in Tab.1, we design four pruning configurations for four different speed-up demands. Considering the reconstruction error in shallow layers can be compensated by deeper layers and not vice versa, we assign lower sparsity to deeper layers.

According to these configurations, we apply neuron pruning to VGG-9 by our proposed LDRF. Before that, we do comparison experiments with several different neuron selection methods (Hu et al. 2016; Molchanov et al. 2017) to demonstrate our framework's insensitivity to neuron selection methods as discussed in above session. In Tab.2, each result is comparable. So we simply use Top-k to initialize $\mathbf{m}$ to solve Eqn.7 in the following experiments. The accuracy is recovered under $2\times$ speed-up, and only suffers from 1.4% drop under $5\times$ speed-up. To further verify the efficiency of our method, we implement the channel pruning method of He et al. (He, Zhang, and Sun 2017) with the same pruning configurations, which adopts a layer-by-layer optimization mode similar to Fig.1. From Fig.5, before end-to-end fine-tuning, our method suffers from significantly less accuracy drop owing to the less information loss during propagation. Our superiority is more apparent as the acceleration rate in-

Table 1: Rank analysis of VGG-9 on CIFAR-10 and four different speed-up settings. (Num. denotes the number of output feature maps, and Spar. denotes the sparsity of weight matrices. Both the filter level and channel level sparsity are considered.)

Layers	Num.	Low rank	2×speed-up		3×speed-up		4×speed-up		5×speed-up	
			Num.	Spar.	Num.	Spar.	Num.	Spar.	Num.	Spar.
CONV1_1	64	6	12	81.3%	6	90.6%	6	90.6%	6	90.6%
CONV1_2	64	18	36	89.5%	18	97.4%	18	97.4%	18	97.4%
CONV2_1	128	37	74	67.5%	65	85.7%	37	91.9%	37	91.9%
CONV2_2	128	49	98	55.7%	98	61.1%	69	84.4%	49	88.9%
CONV3_1	256	89	236	29.4%	178	46.8%	178	62.5%	152	77.3%
CONV3_2	256	103	256	7.8%	206	44.0%	206	44.0%	206	52.2%

Table 2: Performance comparison of pruned VGG-9 (5×) on CIFAR-10 with different heuristic neuron selection methods before end-to-end fine-tuning.

Method	Top-k	Random	APoZ	Activation	Weight
Error↑	2.61%	2.81%	2.72%	2.57%	2.76%

(a) before end-to-end fine-tuning
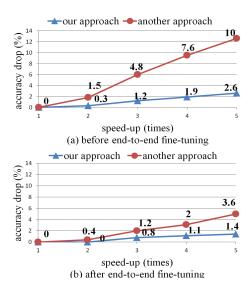

(b) after end-to-end fine-tuning

Figure 5: Performance comparison of pruned VGG-9 on CIFAR-10 between our method and another state-of-the-art neuron pruning method (He, Zhang, and Sun 2017) with four different speed-up configurations.
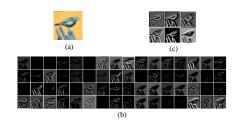

Figure 6: Feature maps visualization. (a) is an image from CIFAR-10. (b) are the redundant output feature maps from the first layer of VGG-9 before pruning. (c) are the informative output feature maps from the first layer reconstructed by our proposed method. (All feature maps are non-linearly enhanced for more clear visualization)

creases. We can achieve better performance after fine-tuning since we provides a better initiation point for fine-tuning.

To further comfirm the impact of our method, we select an image from CIFAR-10, and visualize the feature maps of CONV1_1 before pruning and our reconstructed feature maps after pruning in Fig.6. Despite the redundancy in the original feature maps, it is impossible to select a fraction of feature maps from them, which involve the entire information, to reconstruct the output of the next layer. Our method provides a direction to squeeze the entire information into a fraction of feature maps as much as possible and propagate to the following layers. As shown in Fig.6, our reconstructed feature maps are more informative than any fractions selected from the original feature maps.

VGG-16 and ResNet-50 on ImageNet-2012

To demonstrate the generalization ability of our method to large scale networks, we adopt two most representative networks, i.e., VGG-16 and ResNet-50 to compare our method with other recently proposed state-of-the-art pruning methods. We conduct our experiments on ImageNet-2012 benchmark. We scale the short-side of images to 256 and adopt 224×224 random crop as well as horizontal flip to augment the training dataset. At the validation phase, we only center crop the feeding resized images to 224×224. We present our results by top-5 validation accuracy of single-view approach. The single-view top-5 accuracy of the baseline network is 89.9% and 92.2% for VGG-16 and ResNet-50 respectively. To further evaluate the performance, we use Caffe framework to test the practical speed-ups of each pruned network on GPU (TITAN X Pascal, CUDA8 and cuDNN5) and CPU (Intel Xeon E5-2650, atlas).

VGG-16 is a redundant network with nearly 90% parameters stored in fully-connected layers. To reduce its storage overhead is an easier task already solved by other methods, like Han *et al.* (Han et al. 2015; Han, Mao, and Dally 2016). However, almost 97% computational cost comes from its convolutional layers. Pruning convolutional layers is more difficult than pruning fully-connected layers. Hence, we mainly focus on pruning convolutional layers just like other state-of-the-art neuron pruning methods (Li et al. 2017; He, Zhang, and Sun 2017; Luo, Wu, and Lin 2017).

We apply cross-channel decomposition to each convolutional layers through SVD to obtain each layer's embedding space. During this process, the preserving cumulative sum
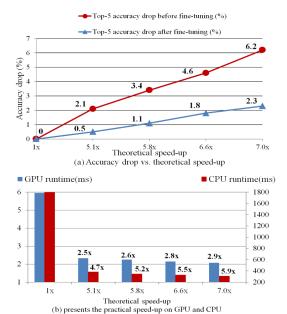
Figure 7: The results of VGG-16 on ImageNet with four different speed-up configurations, as well as the comparison between theoretical and practical speed-up.

Table 3: Performance of pruned VGG-16 on ImageNet compared with other neuron pruning methods.

Method	Top-5 (1-view)	Increased error	Speed-up
Original	89.9%	0.0%	1.00×
Molchanov *et al.*-1	87.0%	2.9%	2.69×
Molchanov *et al.*-2	84.5%	5.4%	3.87×
Luo *et al.*	89.5%	0.4%	3.22×
He *et al.*-1	88.9%	1.0%	4.00×
He *et al.*-2	88.2%	1.7%	5.00×
LDRF (Ours)	**89.4%**	**0.5%**	**5.13×**

Table 4: Performance of pruned ResNet-50 on ImageNet compared with other neuron pruning methods.

Method	Top-5 (1-view)	Increased error	Speed-up
Original	92.20%	0.00%	1.00×
He *et al.*	90.80%	1.40%	2.00×
Luo *et al.*-1	90.67%	1.53%	1.58×
Luo *et al.*-2	90.02%	2.18%	2.26×
LDRF (Ours)-1	**92.01%**	**0.19%**	**2.27×**
LDRF (Ours)-2	**91.55%**	**0.65%**	**3.00×**

Table 5: The comparison between an existing method and our method before and after end-to-end fine-tuning (FT).

Network (5×speed-up)	FT	Error ↑ (He *et al.*)	Error ↑ (Ours)
VGG9	✗	10.0% (Our impl.)	2.6%
(Cifar-10)	✓	3.6% (Our impl.)	1.4%
VGG16	✗	22.0%	2.1%
(ImageNet)	✓	1.7%	0.5%

of largest singular values can be lowered to 0.65 while keeping the accuracy unchanged. The low rank of each layer provides us a proper pruning ratio range. Accordingly, we assign 4 different pruning ratios in this range to meet four different speed-up demands ($5.1\times$, $5.8\times$, $6.6\times$, $7.0\times$). For each layer's reconstruction, it begins converging with about 0.1 epoch training. The experiment results after pruning and optimization are shown in Fig.7(a), where the error increases accordingly as the pruning ratio increases. Our method can speed the network up to $5.1\times$ with only 0.5% top-5 accuracy drop, which is significantly better than other neuron pruning methods as shown in Tab.3. In Fig.7(b), there is a gap between practical and theoretical speed-up. The incremental computation efficiency reflected on CPU is more than that on GPU, since GPU is a memory-bound computational platform. It means FLOPs reduction is not a complete metric to measure practical runtime. I/O delay or other factors should be taken into consideration which will be studied in our future works.

ResNet-50 is a very deep network which is more challenging than VGG-16 for neuron pruning. The information loss accumulation during layer-by-layer pruning and reconstruction will keep growing as layer increases. Notably, our proposed method benefits to avoid such problem owing to the lossless information propagation characteristic. In line with the procedures to VGG-16, our method can speed ResNet-50 up to about $2.27\times$ and $3\times$ with only 0.19% and 0.65% top-5 accuracy drop respectively. In Tab.4, our results outperform other state of the arts by a large margin.

Training Burden Discussion

Cross-channel decomposition is the only extra burden compared with the existing layer-by-layer pruning and reconstruction methods, which is only executed once before pruning and can be viewed as a preprocessing to generate embedding space and analyse the channel redundancy for each layer (note that pruning ratio assigned to each layer denotes the remaining capacity and determines the ceiling of network accuracy). It costs us several epochs training to do such preprocessing. But it is a worthy cost to get a much better results. For a fair comparison, we compare our results before end-to-end fine-tuning with the results of an existing method after fine-tuning. As shown in Tab.5, our results before end-to-end fine-tuning are comparable with the results of the existing method after fine-tuning. Our method has a much greater potential to achieve a much better result after fine-tuning. The essence of our good results is from the less information loss property of our proposed framework.

Conclusion

In this paper we propose a novel neuron pruning method that squeezes intact information into survived neurons instead of discarding unimportant neurons abruptly to prevent from information loss. Signals before and after pruning are projected into an shared embedding space which serves as a function of information buffer. Because of less information loss accumulation compared with other neuron pruning methods, our method achieves superior results, which is more apparent before end-to-end fine-tuning.

References

Aghasi, A.; Nguyen, N.; and Romberg, J. K. 2017. Net-trim: Convex pruning of deep neural networks with performance guarantee. In *Advances in Neural Information Processing Systems*.

Alvarez, J. M., and Salzmann, M. 2016. Learning the number of neurons in deep networks. In *Advances in Neural Information Processing Systems*.

Anwar, S.; Hwang, K.; and Sung, W. 2017. Structured pruning of deep convolutional neural networks. *ACM Journal on Emerging Technologies in Computing Systems* 13(3):32.

Courbariaux, M.; Bengio, Y.; and David, J.-P. 2015. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems*.

Dahl, G. E.; Sainath, T. N.; and Hinton, G. E. 2013. Improving deep neural networks for lvcsr using rectified linear units and dropout. In *International Conference on Acoustics, Speech, and Signal Processing*.

Guo, Y.; Yao, A.; and Chen, Y. 2016. Dynamic network surgery for efficient dnns. In *Advances in Neural Information Processing Systems*.

Han, S.; Pool, J.; Tran, J.; and Dally, W. J. 2015. Learning both weights and connections for efficient neural networks. In *Advances in Neural Information Processing Systems*.

Han, S.; Mao, H.; and Dally, W. J. 2016. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *International Conference on Learning Representations*.

Hassibi, B., and Stork, D. G. 1993. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in Neural Information Processing Systems*.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

He, Y.; Zhang, X.; and Sun, J. 2017. Channel pruning for accelerating very deep neural networks. In *Proceedings of International Conference on Computer Vision*.

Hu, H.; Peng, R.; Tai, Y.; and Tang, C. 2016. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *CoRR* abs/1607.03250.

Krizhevsky, A., and Hinton, G. 2009. Learning multiple layers of features from tiny images.

Lebedev, V., and Lempitsky, V. 2016. Fast convnets using group-wise brain damage. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Lecun, Y.; Denker, J. S.; and Solla, S. A. 1990. Optimal brain damage. In *Advances in Neural Information Processing Systems*.

Li, H.; Kadav, A.; Durdanovic, I.; Samet, H.; and Graf, H. P. 2017. Pruning filters for efficient convnets. In *International Conference on Learning Representations*.

Liu, B.; Wang, M.; Foroosh, H.; and Tappen, M. 2015. Sparse convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Liu, Z.; Li, J.; Shen, Z.; Huang, G.; Yan, S.; and Zhang, C. 2017. Learning efficient convolutional networks through network slimming. In *Proceedings of International Conference on Computer Vision*.

Luo, J. H.; Wu, J.; and Lin, W. 2017. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of International Conference on Computer Vision*.

Mittal, D.; Bhardwaj, S.; Khapra, M. M.; and Ravindran, B. 2018. Recovering from random pruning: On the plasticity of deep convolutional neural networks. In *Workshop on Applications of Computer Vision*.

Molchanov, P.; Tyree, S.; Karras, T.; Aila, T.; and Kautz, J. 2017. Pruning convolutional neural networks for resource efficient inference. In *International Conference on Learning Representations*.

Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; and Bernstein, M. 2015. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision* 115(3):211–252.

Sandler, M.; Howard, A. G.; Zhu, M.; Zhmoginov, A.; and Chen, L. 2018. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Simonyan, K., and Zisserman, A. 2015. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*.

Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S. E.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; and Rabinovich, A. 2015. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Wen, W.; Wu, C.; Wang, Y.; Chen, Y.; and Li, H. 2016. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*.

Ye, J.; Lu, X.; Lin, Z.; and Wang, J. Z. 2018. Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. In *International Conference on Learning Representations*.

Yu, R.; Li, A.; Chen, C.; Lai, J.; Morariu, V. I.; Han, X.; Gao, M.; Lin, C.; and Davis, L. S. 2018. Nisp: Pruning networks using neuron importance score propagation. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Yuan, M., and Lin, Y. 2006. Model selection and estimation in regression with grouped variables. *Journal of The Royal Statistical Society Series B-statistical Methodology* 68(1):49–67.