Fast CNN Pruning via Redundancy-Aware Training

Xiao Dong[1,2], Lei Liu[1(✉)], Guangli Li[1,2], Peng Zhao[1,2], and Xiaobing Feng[1]

[1] State Key Laboratory of Computer Architecture,
Institute of Computing Technology, Chinese Academy of Sciences,
Beijing 100190, China
[2] University of Chinese Academy of Sciences, Beijing 100049, China
{dongxiao,liulei,liguangli,zhaopeng,fxb}@ict.ac.cn

Abstract. The heavy storage and computational overheads have become a hindrance to the deployment of modern Convolutional Neural Networks (CNNs). To overcome this drawback, many works have been proposed to exploit redundancy within CNNs. However, most of them work as post-training processes. They start from pre-trained dense models and apply compression and extra fine-tuning. The overall process is time-consuming. In this paper, we introduce redundancy-aware training, an approach to learn sparse CNNs from scratch with no need for any post-training compression procedure. In addition to minimizing training loss, redundancy-aware training prunes unimportant weights for sparse structures in the training phase. To ensure stability, a stage-wise pruning procedure is adopted, which is based on carefully designed model partition strategies. Experiment results show redundancy-aware training can compress LeNet-5, ResNet-56 and AlexNet by a factor of 43.8×, 7.9× and 6.4×, respectively. Compared to state-of-the-art approaches, our method achieves similar or higher sparsity while consuming significantly less time, e.g., 2.3×–18× more efficient in terms of time.

Keywords: In-training pruning · Model compression
Convolutional neural networks · Deep learning

1 Introduction

In recent years, convolutional neural networks (CNNs) have been playing an important role in the remarkable improvements achieved in a wide range of challenging computer vision tasks such as large-scale image classification [11], object detection [3], and segmentation [6]. Deploying CNN models in real-world applications has attracted increasing interests.

However, the state-of-the-art accuracy delivered by these CNNs comes at the cost of significant storage and computational overheads. For instance, AlexNet [11] has 61 million parameters, takes up more than 243 MB of storage and requires 1.4 billion floating point operations to classify a 224×224 image.

V. Kůrková et al. (Eds.): ICANN 2018, LNCS 11139, pp. 3–13, 2018.
https://doi.org/10.1007/978-3-030-01418-6_1

As a result, deploying CNNs on devices with limited resources, such as mobile phones and wearable devices, could be infeasible.

Since large CNNs are highly over-parameterized [2], many methods have been proposed to compress them. Pruning methods have attracted much attention due to its simplicity and effectiveness. However, most of these methods work as post-training processes. Based on dense pre-trained models, unimportant connections and neurons are pruned to reduce the model size and the computational complexity. The following fine-tuning step is responsible for compensating the accuracy loss. The pruning and fine-tuning steps may be repeated several times for a good balance between accuracy and sparsity (the ratio of pruned weights). Some methods introduce sparsity-inducing regularizers to learn sparse structures from a pre-trained dense model. The overall process consumes significant time to get sparse models, resulting in poor time efficiency as summarized in Table 1.

In this paper, we propose *redundancy-aware training*, which can exploit redundancy efficiently by *learning both sparse neural network structures and weight values from scratch*. Besides minimizing training loss, it prunes unimportant connections for sparse structures. Varying structure may bring difficulty in achieving good accuracy. Redundancy-aware training solves this problem by adopting a stage-wise pruning procedure. It leverages novel partition strategies to divide the network into layer classes. The pruning starts from one class in the first stage and extends to the left classes in following stages. Our training method yields sparse and accurate models when it finishes. Evaluations on several datasets, including MNIST, CIFAR10 and ImageNet, demonstrate our redundancy-aware training can achieve state-of-the-art compression results. Meanwhile, our method is much more efficient in terms of time as it requires neither extending normal training iterations nor any post-training compression procedure.

Table 1. Time breakdown of some pruning methods. For post-training methods, we show epochs spent in the training phase (*Training*) and the post-training phase (*Post-Training*). For in-training pruning methods (denoted by $*$), we report the epochs taken by the method (*Training*) and the normal training epochs (*Normal*).

Method	CNN	Dataset	Training	Post-training	Normal
DC [5]	AlexNet	ImageNet	90	>960	
DNS [4]	LeNet-5	MNIST	11	17	
NISP [18]	GoogLeNet	ImageNet	60	60	
LSN* [14]	LeNet-5	MNIST	200		11
NSN* [10]	ResNet-56	CIFAR10	205		164

2 Related Work

According to whether pre-trained models are required, we divide existing pruning methods into two categories: post-training methods and in-training methods.

Post-training Pruning. Deep compression [5] prunes trained CNNs through a magnitude-based weight pruning method, showing a significant reduction in model size. DNS [4] improves deep compression [5] by allowing the recovery of pruned weights. NISP [18] prunes unimportant neurons based on its neuron importance estimation. SSL [17] makes use of group lasso regularization to remove groups of weights, e.g., channels, filters, and layers, in CNNs. Compression-aware training [1] takes post-training compression into account in the training phase. A regularizer is added to encourage the weights to have lower rank. These methods often suffer from poor time efficiency. Table 1 lists time taken by some pruning methods. We can see the post-training compression procedure takes considerable time. Redundancy-aware training adopts in-training pruning, thus improving the time efficiency significantly.

In-training Pruning. AL [15] introduces binary parameters to prune neurons and layers. A binarizing regularizer is used to attract them to 0 or 1. Similar approach as [15] is adopted to prune weights in [16]. The above two methods only evaluate the *in-training compression ability* on small datasets. Method attempting to use L_0 regularization to directly learn sparse structures is proposed in [14]. To enable gradient-based optimizations, approximation of the non-differentiable L_0 norm is added to the loss. But more training iterations are required (See Table 1). Redundancy-aware training adopts pruning approach to remove redundant weights. By incorporating stage-wise pruning within training process, our method outperforms other in-training pruning works in terms of both compression results and time efficiency.

Fig. 1. Pruning *(b)* with $u = 0.2$ and $l = 0.1$. Weights marked *red* are pruned. The pruning states of the last iteration and this iteration are shown in *(a)* and *(c)* respectively. (Color figure online)

Algorithm 1. Redundancy-Aware Training

Input: CNN to train *network*
the maximum number of training iterations *max_iterations*
the interval of extending pruning to the next class *extending_interval*
Output: network trained by redundancy-aware training

divide *network* into layer classes based on the partition strategies: $classes \leftarrow \{c_1, c_2, ..., c_m\}$
$i \leftarrow 0$
$pruning\_classes \leftarrow \{\}$
initialize *network*
while $i < max\_iterations$ **do**
  if $mod(i, extending\_interval) = 0$ **then**
    $c \leftarrow classes.pop()$
    append c to *pruning_classes*
  end if
  forward and backward through *network*
  update weights in *network*
  for each class c in *pruning_classes* **do**
    for each layer l in c **do**
      pruning layer l
    end for
  end for
  $i \leftarrow i + 1$
end while

3 Redundancy-Aware Training

In this section, we introduce our redundancy-aware training method. The overview of the proposed method is displayed in Algorithm 1. For a given CNN, redundancy-aware training first divides it into layer classes based on the partition strategies. In each training iteration, it prunes layers in *pruning_classes* after the update of weights. More classes will be appended into the *pruning_classes* as training proceeds. We first introduce how to prune unimportant weights during training. Then, we present the model partition strategies.

3.1 Pruning Weights During Training

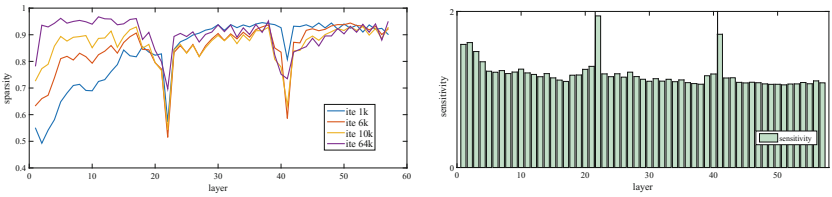
As the pruning works on each layer independently, we take pruning one layer as an example to illustrate the in-training pruning.

Let us denote the parameters of a layer by K. Redundancy-aware training adopts a magnitude-based pruning approach. Specifically, two thresholds u and l are introduced. In each iteration, weights with absolute value below l are pruned, while others with magnitude above u are kept. Weights with absolute value in the range of $[l, u]$ are skipped in this iteration and their pruning states stay unchanged. To reduce the risk of pruning important weights wrongly, we use the update scheme in [4] where pruned weights can also be updated in the

back-propagation. This scheme enables the recovery of wrongly pruned weights. Figure 1 shows an example.

To avoid tuning u and l for each layer manually, we choose to compute them based on K as shown in Eq. 1. μ and σ represent the mean and the standard variation of K, respectively. Two hype-parameters $range$ and ϵ are introduced to provide more flexibility. Increasing $range$ will make l larger, resulting in pruning more weights from network. ϵ is a small positive value and controls the difference between u and l. We analyze how μ and σ influence the compression results in Sect. 4.2.

$$\begin{aligned} u &= max(\mu + \sigma(range + \epsilon), 0) \\ l &= max(\mu + \sigma(range - \epsilon),\ 0). \end{aligned} \tag{1}$$



(a) layer sparsity (b) sensitivity

Fig. 2. Sparsity and sensitivity of layers in ResNet-56. The shapes of sparsity lines of different training time are quite similar, indicating the difference of sparsity between layers stays stable during training. Based on the sensitivity, ResNet-56 is divided into three classes as shown by the black vertical lines in *(b)*.

3.2 Model Partition

In-training pruning allows learning sparse structures during the training phase. However, pruning all layers in network simultaneously causes instability and slows down the learning process, resulting in difficulty in reaching as good accuracy as the normal training.

Redundancy-aware training adopts a stage-wise pruning procedure. The pruning scope in each stage is orchestrated by our model partition strategies. When layers within the pruning scope are being pruned, the left layers can adapt to it and alleviate the impact through updating their weight values. Formally, we call the unit of adjusting the pruning scope 'class'. A class contains several consecutive layers. Based on our model partition strategies, redundancy-aware training divides the CNN into classes. Then, the in-training pruning starts from the first class and extends to one more class at the beginning of each of the following stages. Both layer by layer pruning and pruning all layers together are special cases of our approach.

Partition Strategy. We propose two heuristic strategies for two different types of CNN. The first type is called simple CNN, which refers to networks composed of stacked convolution layers and several fully-connected layers. LeNet-5 [12] and AlexNet [11] fall into this category. For simple CNN, the partition strategy is:

Strategy1: Layers with the same type are divided into the same class.

Thus, simple CNNs will be divided into two classes. The first class contains convolution layers and fully-connected layers belong to the second class. Strategy1 is not applicable to recently designed CNNs, which tend to avoid using fully-connected layers. For example, ResNet [7] has only one fully-connected layer to produce the possibilities over given number of classes. Inspired by [13] which prunes filters based on the analysis of layer sensitivity to pruning, we propose the second strategy for these CNNs:

Algorithm 2. Partition Strategy2

Input: sensitivity difference threshold δ
layers' sensitivity to pruning $s[...]$
layers in given network $layers[...]$
Output: the partition result of network

```
c ← {layers[1]}
s_avg ← s[1]
for l ← 2 to layers.size do
    diff ← abs(s[l] − s_avg)
    if diff > δ then
        set c a new partition class
    end if
    add layers[l] to c
    update s_avg to the average sensitivity of layers in c
end for
```

Strategy2: Divide model at layers which are quite sensitive to pruning.

Algorithm 2 illustrates how this strategy works. The sensitivity to pruning is determined through our proposed 'probe' phase which is described in the next section. We also analyze the impact of δ in Sect. 4.2.

Determine Layer's Sensitivity Efficiently. The in-training pruning zeros out unimportant weights. Layers with relatively low sparsity should be important and sensitive to pruning. *Thus, we define layer's sensitivity as the reciprocal of its sparsity achieved by the in-training pruning.* A naive but inefficient approach to determine the sensitivity works as follows. We train the CNN with all layers under in-training pruning and use the layer's sparsity after training to compute the sensitivity. Based on a key observation, we propose a more efficient approach. Figure 2a shows the sparsity of ResNet-56 at different time of training. The relative sparsity between layers is actually quite stable in training. As the partition result only depends on the difference of sparsity between layers, we can use the sparsity at early training time to obtain the partition result.

More precisely, we introduce a probe phase where the CNN is trained with all layers under the in-training pruning. When the probe phase finishes, we compute layer's sensitivity based on its sparsity, which is then used by the strategy2. In our experiments, we find tenth of the training time is sufficient for the probe phase. Figure 2b shows the sensitivity of layers in ResNet-56. It's noticeable that layers of residual blocks where the number of output channels changes are sensitive to pruning. This discovery is consistent with the results reported in [13].

Table 2. Comparison to other compression works. Results of our method are denoted by *RA-range-*ϵ. The result of *DC* for ResNet-56 is provided in [10]. The result of *PF* is based on our implementation. The *scratch-train* models show notable accuracy drops, demonstrating the difficulty of training a sparse network from scratch.

Network	In-training methods	Baseline accuracy	Accuracy change	Sparsity	Post-training methods	Baseline accuracy	Accuracy change	Sparsity
LeNet-5	LNA [15]	99.3%	−0.23%	90.5%	SSL [17]	99.1%	−0.1%	75.1%
	LSN [14]	99.1%	0	90.7%	DC [5]	99.2%	+0.03%	92%
	TSNN [16]	99.2%	−0.01%	95.8%	DNS [4]	99.1%	0	99.91%
	RA-2-0.1	99.1%	0	**97.7%**	Scratch-train	99.1%	−1.5%	97.7%
ResNet-56	NCP [10]	93.4%	−0.5%	50%	CP [8]	92.8%	−1.0%	50%
	NWP [10]	93.4%	−0.6%	66.7%	PF [13]	92.4%	−1.04%	62%
	RA-1.8-0.1	92.4%	−0.1%	**87.4%**	DC [5]	93.4%	−0.8%	66.7%
	RA-3.0-0.1	92.4%	−1.0%	**92.1%**	Scratch-train	92.4%	−2.8%	87.4%

4 Evaluation

In this section, we evaluate redundancy-aware training on MNIST, CIFAR10, and ImageNet with LeNet-5, ResNet-56, and AlexNet, respectively. First, we compare the compression result and the time efficiency with state-of-the-art compression methods. The compression result includes achieved sparsity and accuracy loss. Sparsity is defined as the percentage of the zeroed out weights. We compare the time efficiency based on the number of iterations or epochs required to obtain sparse models. Then, we analyze the effectiveness of the model partition and the effect of hyper-parameters in Sect. 4.2. We implement our method in Caffe [9].

4.1 Compression Result and Time Efficiency

The comparison to other methods on LeNet-5 and ResNet-56 is summarized in Table 2. We also train models with the same sparsity as the models trained through redundancy-aware training from scratch (the *scratch-train*).

LeNet-5. Redundancy-aware training reduces the model size of LeNet-5 by 43.8× without accuracy loss and outperforms all in-training methods by a notable margin, validating its ability to reduce redundancy in the training phase. Compared to post-training methods, redundancy-aware training achieves higher or similar sparsity. Our method prunes more weights in every layer than [14] and [5]. As for time efficiency, our method only takes 11 epochs which is equal to the normal training time and is about 18× more efficient than the in-training method in [14] and 2.5× more efficient than the method in [4].

ResNet-56. Based on the strategy2 in Sect. 3.2, ResNet-56 is divided into three classes. We extend the in-training pruning at 10k and 20k iterations. Redundancy-aware training achieves a 7.9× reduction with only 0.1% top-1 accuracy drop. Importantly, our method achieves this without any post-training procedures. By using a larger *range*, we can achieve a 12.6× compression at the cost of 1.13% accuracy loss, which can be reduced to 1% after a fine-tuning of 20k iterations. As far as we know, our method achieves state-of-the-art compression result for ResNet-56. In terms of time-efficiency, our method takes 70k iterations (64k for training and 6k for the probe phase), which is about 2.3× more efficient than NWP in [10] and PF in [13].

Table 3. Layer-by-layer comparison to deep compression on AlexNet.

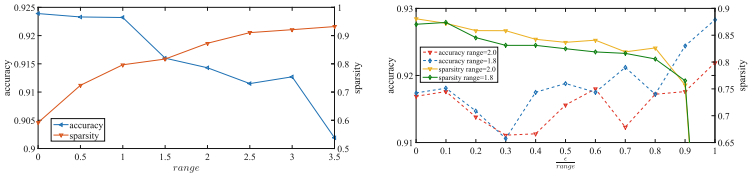
Method/layer	conv1	conv2	conv3	conv4	conv5	fc1	fc2	fc3	Total
DC	16%	62%	65%	63%	63%	91%	91%	75%	89%
Ours	31%	65%	69%	63%	61%	88%	81%	80%	84%

AlexNet. Finally, we experiment with AlexNet on ImageNet. We train the bvlc_alexnet in Caffe and get 78.65% top-5 accuracy on validation dataset with single-view testing. Redundancy-aware training reduces the model size by 6.4× with 0.36% accuracy loss. We further fine-tune it for 45k iterations and obtain a model with 78.54% accuracy. We display sparsity achieved by our method and DC [5] in Table 3. Our method takes 99 epochs in total, which is 9.69× more efficient in terms of time.

4.2 Ablation Study

Hyper-parameter Sensitivity. We make use of ResNet-56 to measure the impact of varying *range* and ϵ. The result is shown in Fig. 3.

Increasing *range* leads to larger l and more weights will be pruned in training. Thus we can make trade-offs between the sparsity and the accuracy through adjusting *range*. Note the accuracy does not drop dramatically (2.2% drop) when *range* increases from 0 to 3.5. Since increasing ϵ makes l smaller, weights



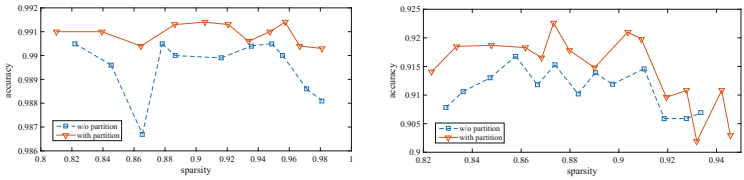
(a) *range* (b) ϵ

Fig. 3. Impact of hyper-parameters *range* and ϵ. The model is divided into three partition classes.

are less likely to be pruned and the sparsity decreases. We can observe that the accuracy does not change drastically for a wide range of ϵ.

Table 4. Accuracy with varying δ.

δ	$+\infty$	0.5 * s_avg	$0.4 * s\_avg$	0.3 * s_avg
# partition classes	1	2	3	5
Accuracy	91.3%	91.7%	92.3%	90.2%



(a) LeNet-5 (b) ResNet-56

Fig. 4. Effect of partition with varying *range*s.

Effectiveness of Partition Strategies. We first analyze the impact on accuracy with different number of partition classes. To this end, we fix $range = 1.8$ and $\epsilon = 0.1$ and vary δ to change the partition result. Results are shown in Table 4. When δ is set to $+\infty$, all layers belong to the same class and the network is pruned all through the training phase, which shows a 1.1% accuracy drop. Dividing ResNet-56 into two or three classes improves accuracy. The model with five classes has inferior accuracy, implicating too many classes result in insufficient training iterations in each stage.

We also verify the effectiveness of model partition with varying *range*s. Results are shown in Fig. 4. The model partition helps to improve accuracy over a wide scope of *range*s, confirming the benefit of our model partition approach in stabilizing training and helping in good convergence.

5 Conclusion

In this paper, we propose an in-training compression method, redundancy-aware training. Our method can learn both sparse connections and weight values from scratch. We highlight our redundancy-aware training achieves state-of-the-art compression results without any post-training compression procedures and consumes significantly less time when compared to other methods.

Acknowledgments. This work is supported by National Key R&D Program of China under Grant No. 2017YFB0202002, Science Fund for Creative Research Groups of the National Natural Science Foundation of China under Grant No. 61521092 and the Key Program of National Natural Science Foundation of China under Grant Nos. 61432018, 61332009, U1736208.

References

1. Alvarez, J.M., Salzmann, M.: Compression-aware training of deep networks. In: Advances in Neural Information Processing Systems, pp. 856–867 (2017)
2. Denil, M., Shakibi, B., Dinh, L., de Freitas, N., et al.: Predicting parameters in deep learning. In: Advances in Neural Information Processing Systems, pp. 2148–2156 (2013)
3. Girshick, R.B.: Fast R-CNN. In: 2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, 7–13 December 2015, pp. 1440–1448 (2015)
4. Guo, Y., Yao, A., Chen, Y.: Dynamic network surgery for efficient DNNs. In: Advances in Neural Information Processing Systems, pp. 1379–1387 (2016)
5. Han, S., Mao, H., Dally, W.J.: Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. In: Proceedings of the International Conference on Learning Representations, ICLR (2016)
6. He, K., Gkioxari, G., Dollár, P., Girshick, R.B.: Mask R-CNN. In: IEEE International Conference on Computer Vision, pp. 2980–2988 (2017)
7. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)
8. He, Y., Zhang, X., Sun, J.: Channel pruning for accelerating very deep neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1389–1397 (2017)
9. Jia, Y., et al.: Caffe: convolutional architecture for fast feature embedding. In: Proceedings of the 22nd ACM International Conference on Multimedia, pp. 675–678. ACM (2014)
10. Kim, E., Ahn, C., Oh, S.: Learning nested sparse structures in deep neural networks. arXiv preprint arXiv:1712.03781 (2017)
11. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems, pp. 1097–1105 (2012)
12. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998)
13. Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H.P.: Pruning filters for efficient ConvNets. In: Proceedings of the International Conference on Learning Representations, ICLR (2017)

14. Louizos, C., Welling, M., Kingma, D.P.: Learning sparse neural networks through l_0 regularization. In: Proceedings of the International Conference on Learning Representations, ICLR (2018)
15. Srinivas, S., Babu, R.V.: Learning neural network architectures using backpropagation. In: Proceedings of the British Machine Vision Conference. BMVA Press (2016)
16. Srinivas, S., Subramanya, A., Babu, R.V.: Training sparse neural networks. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops, pp. 455–462 (2017)
17. Wen, W., Wu, C., Wang, Y., Chen, Y., Li, H.: Learning structured sparsity in deep neural networks. In: Advances in Neural Information Processing Systems, pp. 2074–2082 (2016)
18. Yu, R., et al.: NISP: pruning networks using neuron importance score propagation. arXiv preprint arXiv:1711.05908 (2017)