Dynamical Channel Pruning by Conditional Accuracy Change for Deep Neural Networks

Zhiqiang Chen, Ting-Bing Xu, Changde Du, Cheng-Lin Liu, *Fellow, IEEE*, and Huiguang He, *Senior Member, IEEE*

***Abstract*— Channel pruning is an effective technique that has been widely applied to deep neural network compression. However, many existing methods prune from a pretrained model, thus resulting in repetitious pruning and fine-tuning processes. In this article, we propose a dynamical channel pruning method, which prunes unimportant channels at the early stage of training. Rather than utilizing some indirect criteria (e.g., weight norm, absolute weight sum, and reconstruction error) to guide connection or channel pruning, we design criteria directly related to the final accuracy of a network to evaluate the importance of each channel. Specifically, a channelwise gate is designed to randomly enable or disable each channel so that the conditional accuracy changes (CACs) can be estimated under the condition of each channel disabled. Practically, we construct two effective and efficient criteria to dynamically estimate CAC at each iteration of training; thus, unimportant channels can be gradually pruned during the training process. Finally, extensive experiments on multiple data sets (i.e., ImageNet, CIFAR, and MNIST) with various networks (i.e., ResNet, VGG, and MLP) demonstrate that the proposed method effectively reduces the parameters and computations of baseline network while yielding the higher or competitive accuracy. Interestingly, if we Double the initial Channels and then Prune Half (DCPH) of them to baseline's counterpart, it can enjoy a remarkable performance improvement by shaping a more desirable structure.**

***Index Terms*— Conditional accuracy change (CAC), direct criterion, dynamical channel pruning, neural network compression, structure shaping.**

Manuscript received March 20, 2019; revised July 14, 2019, October 24, 2019, and December 16, 2019; accepted March 2, 2020. Date of publication April 2, 2020; date of current version February 4, 2021. This work was supported by the National Natural Science Foundation of China (61976209, 61721004), CAS International Collaboration Key Project, and the Strategic Priority Research Program of CAS (XDB32040200). *(Corresponding author: Huiguang He.)*

Zhiqiang Chen and Changde Du are with the Research Center for Brain-Inspired Intelligence, Institute of Automation, Chinese Academy of Sciences (CASIA), Beijing 100190, China, and also with the School of Artificial Intelligence, University of Chinese Academy of Sciences (UCAS), Beijing 100049, China (e-mail: chenzhiqiang14@mails.ucas.ac.cn; duchangde@gmail.com).

Ting-Bing Xu is with the National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences (CASIA), Beijing 100190, China, and also with the School of Artificial Intelligence, University of Chinese Academy of Sciences (UCAS), Beijing 100049, China (e-mail: tingbing.xu@nlpr.ia.ac.cn).

Cheng-Lin Liu is with National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences (CASIA), Beijing 100190, China, also with the Center for Excellence in Brain Science and Intelligence Technology, Chinese Academy of Sciences, Beijing 100190, China, and also with the School of Artificial Intelligence, University of Chinese Academy of Sciences (UCAS), Beijing 100049, China (e-mail: liucl@nlpr.ia.ac.cn).

Huiguang He is with the Research Center for Brain-Inspired Intelligence, National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences (CASIA), Beijing 100190, China, also with the Center for Excellence in Brain Science and Intelligence Technology, Chinese Academy of Sciences, Beijing 100190, China, and also with the School of Artificial Intelligence, University of Chinese Academy of Sciences (UCAS), Beijing 100049, China (e-mail: huiguang.he@ia.ac.cn).

Color versions of one or more of the figures in this article are available online at https://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNNLS.2020.2979517

I. Introduction

Last 20 years have witnessed a rapid development of deep neural networks. Deep learning-based methods have achieved a great performance improvement on many computer vision tasks, such as image recognition [1]–[6], object detection [7]–[13], and semantic segmentation [14]–[17]. To pursue better performance, the network depth was increased gradually from the early LeNet architecture [1] with only five layers to the recent VGG [3] and GoogleNet [4] architectures with about 20 layers. Particularly, the newly designed ResNet [6] and DenseNet [18] have increased to the amazing hundreds or even thousands of layers.

With network architectures deeper and deeper, many networks are heavily overparameterized, which is not friendly to practical applications. To harvest a more efficient subnetwork with less redundancy, many network compression techniques have been explored, including model design [19], [20], quantization [21], low-rank decomposition [22], and network pruning [23]–[25]. Especially, the network pruning framework has been a mainstream branch in network compression and acceleration community. On the one hand, the pruning process shares many similarities to the synaptic pruning of mammalian brain development [26], [27], where some seldom-used synapses are gradually pruned to form a relatively stable adult brain. On the other hand, pruning operations can dynamically evolve the baseline network into a more desirable subnetwork structure.

So far, network pruning mainly contains two types of methods: nonstructured pruning and structured pruning. The nonstructured pruning [28]–[30] prunes connections of neural networks to result in the nonstructured matrix. Though it can yield a considerable parameter reduction for the fully connected layer, it cannot harvest obvious acceleration for modern networks with few dense fully connected layers, such as ResNet [6], [31]–[33] and DenseNet [18] due to the requirement of the dedicated software/hardware for nonstructured matrix multiplication. To achieve the higher practical speedup, many structured pruning methods, such as channel/filter pruning [24], [34], [35], have been exploited, which can be well-supported by off-the-shelf deep learning libraries.

2162-237X © 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: Lanzhou University. Downloaded on December 09,2021 at 13:29:55 UTC from IEEE Xplore. Restrictions apply.

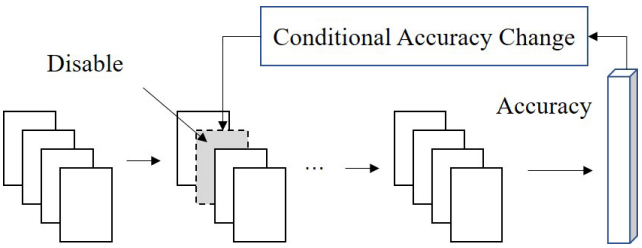

Fig. 1. Brief illustration of our method. Each channel is disabled randomly to estimate the influence to final accuracy so that the importance of each channel can be evaluated.

Although existing pruning methods have achieved considerable effectiveness, they still suffer from two key problems.

1) To evaluate the importance of each channel, a large number of common empirical criteria, such as magnitude-based criteria [28], [36], sparsity norm [37], [38], and reconstruction error [24], [34], have been explored, which are not directly related to the final performance (i.e., indirect pruning criterion). Besides, discrimination-aware channel pruning (DCP) [35] enhances the discriminate influence for the pruning criterion of reconstruction error by additional discrimination-aware losses, but it is still an indirect way.
2) Many previous works need to pretrain the network and then prune and fine-tune it repeatedly. The multistage training leads to complicated training processes and a large amount of training time. These problems motivate us to design a criterion directly related to the final accuracy and implement the unimportant channels removing during the training in an end-to-end manner.

In this article, we propose a dynamical channel pruning method with a direct criterion named conditional accuracy change (CAC). Specifically, unimportant channels are dynamically pruned to form a stable network structure at the early stage of training so that the whole training stage does not need extra separated fine-tuning processes. It is much like synaptic pruning in the mammalian brain that it prunes synapses mainly at adolescence then keeps relatively stable in the long adult period. To evaluate the importance of each channel, a CAC, as shown in Fig. 1, defined to measure the influence to final accuracy when disabling each channel by a channelwise gate. To dynamically evaluate the importance at any time of training, two practical and efficient forms of CAC are proposed, which can be obtained with few extra computations by the simple exponential moving average of accuracy at each iteration in training. To verify the effectiveness of our method, we also conduct extensive experiments on multiple data sets (i.e., ImageNet, CIFAR, and MNIST) with multiple networks (i.e., ResNet, VGG, and MLP). The results demonstrate that our method can prune 30% parameters and FLOPs with better performance and prune 50% with competitive performance. By pruning a network from FLOPs more than needed to expected, performance increases significantly. Furthermore, comprehensive evidence demonstrates that a better structure can be formed automatically and adaptively by our method.

The main contributions of our method are as follows.

1) We dynamically prune channels while training rather than pruning and fine-tuning a pretrained network.
2) We establish criteria named CAC directly related to the final performance for channel importance evaluation.
3) Our method needs no extra structure and computation except random gates and exponential moving average operation, which is efficient in computation and easy to implement.

II. Related Works

A. Architecture Design

MobileNet [19], [39] and Xception [40] replace standard convolution by depthwise separable convolution and 1×1 pointwise convolution, which leads to extremely small networks due to original dense connection for channels. ShuffleNet [20], [41] also follows the idea to reduce the dense connections of channels and proposes to use group pointwise convolution to reduce computation complexity and utilize channel shuffle to prompt information exchange for different groups. IGCV [42], [43] shares the idea of ShuffleNet but utilizing two-stage interleaved group convolution for information exchange. However, these hand-crafted networks usually require expertise design. References [44] and [45] utilize reinforcement learning to learning the architecture of networks automatically, but they need large computational costs on searching optimal architecture. Reference [46] utilizes blockwise network generation to search for optimal architecture, which reduces the space of searching, but it still needs large computation consumption due to reinforcement learning.

B. Low-Rank Decomposition

Reference [22] uses techniques, such as singular value decomposition (SVD), to approximate a big matrix by the product of two low-rank matrixes. Network decoupling [47] decomposes regular convolution into depthwise separable one in no need of time-consuming training procedure with a full training set. For networks VGG [3] and AlexNet [2] with fully connected layers, it works well, but it is not helpful for modern networks, such as ResNet and DenseNet with few fully connected layers.

C. Quantization

References [21] and [48] replace the original parameters and activations with low precision value to compresses the model. Deep compression [23] quantizes the weights and utilizes a codebook and the indices to store connections sharing the same weight. XNOR-Net [49] utilizes binary weights, activations, and gradients to compress the model. BinaryNet [50] also utilizes binary weights and activations, while it trains by expectation backpropagation (EBP), a variational Bayesian approach, to infer networks with binary weights and neurons by updating the posterior distribution over the weights. However, quantization cannot form a better structure due to the lack of structure learning ability.

Authorized licensed use limited to: Lanzhou University. Downloaded on December 09,2021 at 13:29:55 UTC from IEEE Xplore. Restrictions apply.

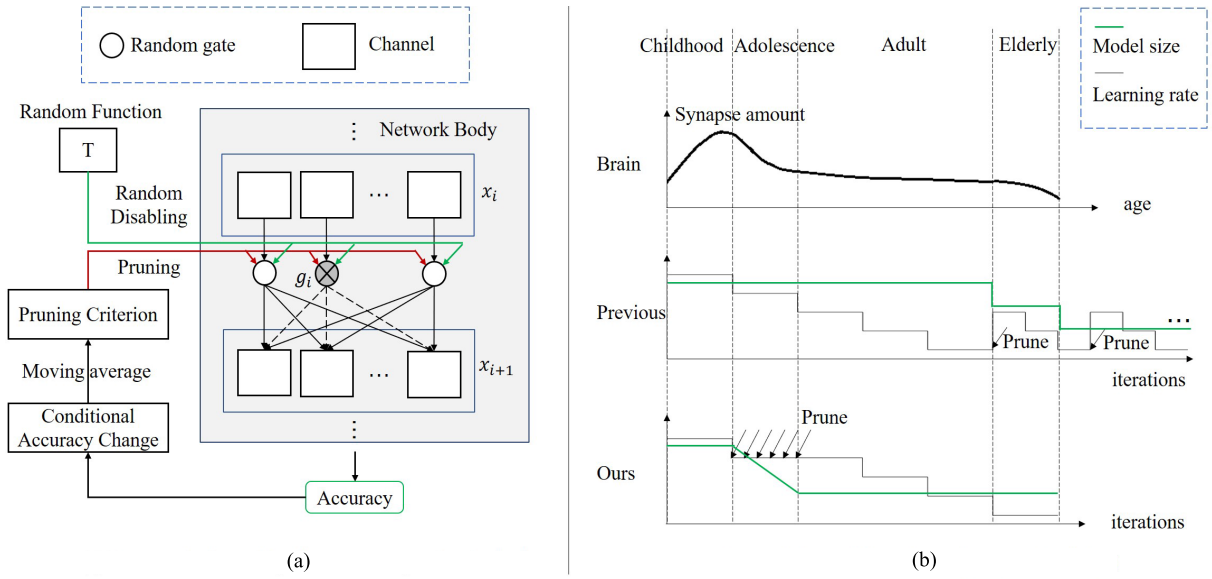

Fig. 2. Illustration of our method. (a) There are two stages in our method. In the first stage, the channel is randomly disabled by our random gate with random function T, which leads to accuracy change, so we can get the pruning criterion to evaluate the importance of each channel by the exponential moving average of accuracy change under the condition of each channel disabled. In the second stage, unimportant channels are pruned permanently by pruning criterion. (b) Each stage of the learning rate denotes the drop of learning rate divided by 10. The mammalian brain mainly prunes unimportant synapses at the early period of the whole life. Rather than previous methods prune networks after training with iterative fine-tuning and pruning scheme, we prune network at the early stage of training as the brain does. (a) Prune each channel by direct-accuracy-related criterion. (b) Prune during training (the early stage of training).

D. Pruning

Deep compression [23] prunes the redundant connections and only keeps the most informative connections. Because of connection pruning, deep compression leads to the nonstructured sparse matrix not supported by off-the-shelf libraries. Instead of nonstructure methods, structured sparsity learning [38] proposes various structured sparsities, such as filterwise, channelwise, shapewise, and depthwise sparsity to conduct structured pruning. Particularly, channel/filter pruning [24], [25], [37] that prunes the whole channel/filter have played an important role in structured pruning. To evaluate the importance of neuron, many criteria are proposed such as the magnitude of weights [25], [51], [52], the average percentage of zeros (APoZ) [53], and scaling factors of batch normalization layer [37]. All criteria mentioned earlier are not directly related to the final accuracy. To address this problem, [54] adopts the Taylor expansion to approximate the influence to loss function induced by removing each filter, but for many layers, such as the Relu layer, it is difficult to calculate the Taylor expansion. MAB [55] directly evaluates the importance of directly removing the channels of the pretrained network. However, parameters of the network are not updated during the evaluation and after the pruning, which will make the pruned network underfit and makes the subsequent evaluations less accurate. Different from producing a fixed pruned model for deployment, [56], [57] dynamically prunes network at runtime by activating different subarchitectures for different input data. However, they do not reduce the model size of the whole network and need extra hardware or software to harvest the speedup for partly activation.

Rather than evaluating the importance, sparsity [38], [58] is also a wide-used method to force some unimportant parameters close to 0 so that these parameters can be removed. To make a network more efficient after pruning, CondenseNet [59] utilizes group sparsity to force a network into several groups. However, sparsity techniques need to train the network well to force part of the parameters close to 0. As a result, sparsity techniques prune unimportant parameters at the end of training and need complicated pruning and fine-tuning processes.

III. Method

Rather than previous indirect criteria, the main idea of our criteria is that we can evaluate the importance of each channel by its direct influence on the final accuracy by removing it. As shown in Fig. 2(a), we utilize a channelwise gate before each channel to disable each channel randomly, and then, we estimate the accuracy change under the condition of each channel disabling to evaluate its importance. To avoid fine-tuning, we prune unimportant channels dynamically at the early stage of training, as shown in Fig. 2(b), and it is much like synaptic pruning in the mammalian brain, in which synapses are pruned mainly at adolescence and keep relatively stable in the long adult period. To dynamically evaluate the importance, the criterion should be both effective and efficient to evaluate at any time of training. To this end, we come up with CAC and its two practical implementations with few extra computations except the exponential moving average of accuracy.

A. Channelwise Gate

For a general layer of deep neural networks, it can be typically defined as

$$x_{i+1} = F(x_i, \{W_i\}) \tag{1}$$

Authorized licensed use limited to: Lanzhou University. Downloaded on December 09,2021 at 13:29:55 UTC from IEEE Xplore. Restrictions apply.

where x_i and x_{i+1} are input and output of layer i, $\{W_i\}$ is the parameter, and F is the operation of this layer.

To disable each channel conveniently, we add a channelwise gate to the input x_i, which denotes as

$$x_{i+1} = F(\mathbf{g} \otimes x_i, \{W_i\}) \tag{2}$$

where $\otimes$ is channel disabling function determined by the parameter $\mathbf{g} = (g_1, g_2, \ldots, g_c) \in \{0, 1\}$ with dimension c the same as the number of input channels of x. We can randomly assign a value for $\mathbf{g}$ to randomly disable each channel.

B. Naive Criterion

To evaluate the importance of each channel, we can simply disable each channel and estimate the drop of accuracy for networks. Formally, the accuracy of networks without channel disabled can be represented as

$$E(G = \mathbf{1}) = \sum_{k=1}^{K} f(X_k, Y_k, G = \mathbf{1})/K \tag{3}$$

where X_k, Y_k are images and labels of batch k, G are all gates $\{\mathbf{g}\}$ in networks, and $G = \mathbf{1}$ means that all gates are activated, $f(\cdot)$ is the accuracy between the real labels and the predicted labels of input data by the network, and $f(X_k, Y_k, G = \mathbf{1})$ is the accuracy of batch k between real labels Y_k and predicted labels of input data X_k with all gates activated. Then, $E(G = 1)$ is the evaluation of mean accuracy of all K batches. Similarly, accuracy of networks with gate g_i deactivated can be represented as

$$E(g_i = 0) = \sum_{k=1}^{K} f(X_k, Y_k, g_i = 0)/K \tag{4}$$

where $g_i = 0$ means that gate g_i is deactivated, and other gate is activated; $f(X_k, Y_k, g_i = 0)$ is the accuracy of batch k with all gates in G are activated except gate g_i. Then, $E(g_i = 0)$ is the evaluation of mean accuracy of all batches with gate g_i disabled.

Naturally, we can obtain the change of accuracy $\Delta E(g_i)$ with gate g_i disabled as follows:

$$\Delta E(g_i) = E(G = \mathbf{1}) - E(g_i = 0) \tag{5}$$

where $E(G = \mathbf{1})$ and $E(g_i = 0)$ are defined by (3) (4). $\Delta E(g_i)$ shows the influence to final accuracy by disabling g_i. The bigger the $\Delta E(g_i)$, the more important the channel corresponding to g_i. Naturally, $\Delta E(g_i)$ can be used as the criterion to prune unimportant channels.

C. Conditional Accuracy Change Criterion

For the naive criterion, it estimates channel accuracy change by disabling each channel separately and independently. For example, ResNet-110 has more than 2000 channels. To evaluate the importance, it needs running all the training data for each channel, which is inefficient in computation. Furthermore, we need to compare each channel to prune the most unimportant one. However, channels in different layers own different parameter and computation costs, which makes the comparison unfair. To address these two problems, we propose a new criterion named CAC that alleviates both two problems by disabling multiple channels at each iteration.

In detail, with random disabling denoted as T, each gate is disabled independently with a specific probability $P(g_i = 0) = p$, for example, $p = 0.05$, and then, we can get the estimation of accuracy at iteration n denoted as E_n as follows:

$$E_n = \sum_{k=1}^{K} f_n(X_k, Y_k, T(k))/K \tag{6}$$

where X_k and Y_k are images and labels of batch k, K is the number of batches, $f_n(\cdot)$ is the accuracy of input data by the network at iteration n, $T(k)$ is randomly disabling each gate in batch k by T, $f_n(X_k, Y_k, T(k))$ is the accuracy of batch k with randomly disabling $T(k)$, and E_n is the evaluation of accuracy under randomly disabling T. At the same time, we can also get estimation of accuracy $E_n(g_i = 0)$ and $E_n(g_i = 1)$ under the condition of g_i disabled and enabled, respectively, as

$$E_n(g_i = 0) = \frac{\sum_{k=1}^{K} f_n(X_k, Y_k, T(k))\mathbf{1}(g_i(k) = 0)}{\sum_{k=1}^{K} \mathbf{1}(g_i(k) = 0)} \tag{7}$$

$$E_n(g_i = 1) = \frac{\sum_{k=1}^{K} f_n(X_k, Y_k, T(k))\mathbf{1}(g_i(k) = 1)}{\sum_{k=1}^{K} \mathbf{1}(g_i(k) = 1)} \tag{8}$$

where 1(condition) is a function that outputs 1, if condition is satisfied else 0, and $1(g_i(k) = 0)$ is 1, if $g_i(k) = 0$ else 0.

Easy to prove, the following equation can be satisfied by (6), (8), and (7):

$$E_n = P(g_i = 0)E_n(g_i = 0) + P(g_i = 1)E_n(g_i = 1) \quad (P(g_i = 0) + P(g_i = 1) = 1) \tag{9}$$

where $P(\cdot)$ is probability function, and $P(g_i = 0)$ and $P(g_i = 1)$ are the probabilities of gate g_i disabled and enabled.

Then, at iteration n, $\Delta E_n(g_i)$ is defined as the change of the accuracy for channel with gate g_i

$$\Delta E_n(g_i) = E_n - E_n(g_i = 0) \tag{10}$$

where E_n and $E_n(g_i = 0)$ are defined by (6) and (7).

By (9), we can also get the equivalent form of ΔE_n as the following:

$$\Delta E_n(g_i) = \frac{P(g_i = 1)}{P(g_i = 0)}(E_n(g_i = 1) - E_n) \tag{11}$$

where E_n and $E_n(g_i = 0)$ are defined by (6) and (7).

By disabling multiple channels simultaneously at each iteration, the CAC is more efficient than the naive criterion. For each channel, the amount of disabled parameter and computation costs is expected to be almost the same under the condition that it is disabled. Therefore, the comparisons of different channels are fairer in the CAC criterion than in the naive criterion.

D. Practical Implementations of Conditional Accuracy Change

When calculating the CAC, it covers enough data (K batches) in a single iteration, which needs to stop training to evaluate it. Though it is more efficient than the naive

Authorized licensed use limited to: Lanzhou University. Downloaded on December 09,2021 at 13:29:55 UTC from IEEE Xplore. Restrictions apply.

criterion, it separates the evaluation from the training. Thus, there are still potentialities to further improve its efficiency. To this end, we propose two practical implementations by incorporating the evaluation into training. Rather than covering enough data in a single iteration, the practical implementations input only a single-batch data for each iteration and combine many iterations to cover enough data. In this way, the evaluation of the CAC and the training process can be conducted simultaneously in no need of stopping training. By sharing the forward propagation, the evaluation can be well incorporated into the training process. Because the forward propagation accounts for almost all the computational costs, the evaluation is much efficient and needs few extra computational costs.

1) Practical Implementation 1: First, by exponential moving average of $\Delta E_n(g_i)$, the practical implementation of CAC denoted as $S_n(g_i)$ is defined as follows:

$$S_n(g_i) = (1-\xi)\sum_{k=2}^{n} \Delta E_k(g_i)\xi^{n-k} + \Delta E_1(g_i)\xi^{n-1} \quad (12)$$

where ξ is the parameter of exponential moving average and $\Delta E_n(g_i)$ is defined by (10).

To be more brief, the coefficient of $\Delta E_k(g_i)$ is represented as follows:

$$\lambda_k(\xi, n) = (1-\xi)\xi^{n-k}. \quad (13)$$

Second, single-batch input data instead of K batches is taken for each iteration so that we need not stop training to evaluate E_k and $E_k(g_i = 0)$. Due to the single batch, the denominator will be 0 when $g_i(k) = 1$ in $E_k(g_i = 0)$, so we remove items in $E_k(g_i = 0)$ when $g_i(k) = 1$. Because of the subset, the decay speeds of E_k and $E_k(g_i = 0)$ are different. To make the decay speeds the same, we adjust the decay parameter of $E_k(g_i = 0)$ from ξ to ξ' (see details in the Appendix). Finally, we redefine $S_n(g_i)$ as

$$S_n(g_i) = \sum_{k=1}^{n} f_k(X_k, Y_k, T(k))\lambda_k(\xi, n) - \sum_{k=1}^{m} f_{s_{ik}}\left(X_{s_{ik}}, Y_{s_{ik}}, T(s_{ik})\right)\lambda_k\left(\xi', m\right) \quad (14)$$

where $\{s_{i1}, s_{i2}, \ldots, s_{im} | g_i(s_{ij}) = 0, j \in \{1, 2, \ldots, m\}\}$ is subsequence from $\{1, 2, \ldots, n\}$. Because $g_i = 0$ appears every $(1/P(g_i = 0))$ iterations on expectation, so $\xi' = \xi^{(1/P(g_i=0))}$ (see details in the Appendix). We can represent (14) back to iterative formula so that it can be calculated efficiently in practice as follows:

$$\begin{cases} S_n(g_i) = a_n - b_{s_{im}} \\ a_n = \xi a_{n-1} + (1-\xi) f_n(X_n, Y_n, T(n)) \\ b_{s_{im}} = \xi' b_{s_{i(m-1)}} + \left(1-\xi'\right) f_{s_{im}}\left(X_{s_{im}}, Y_{s_{im}}, T(s_{im})\right) \\ a_0 = 0, \quad b_{s_{i0}} = 0, \ \xi' = \xi^{\frac{1}{P(g_i=0)}} \end{cases} \quad (15)$$

where a_n and $b_{s_i m}$ are exponential moving average of all accuracy and accuracy under condition of $g_i = 0$.

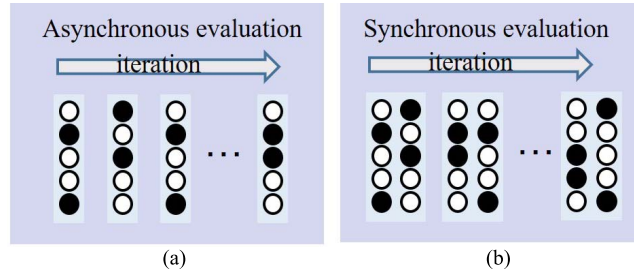

Fig. 3. By randomly disabling each gate once in each iteration in (a), practical implementation 1 calculates the exponential moving average of E_n and $E_n(g_i = 0)$ asynchronously due to the different subsets of iterations. Practical implementation 2 randomly disables each gate twice (b) to generate the synchronous enabled–disabled pairs to calculate $\Delta E_n(g_i)$. (a) Practical implementation 1. (b) Practical implementation 2.

2) Practical Implementation 2: Practical implementation 1 randomly disables each gate once in each iteration, as shown in Fig. 3(a). It results that the enabling and disabling of a gate fall into different subsets of all the iterations. Thus, E_n (or $E_n(g_i = 1)$) and $E_n(g_i = 0)$ are evaluated asynchronously, which may influence the accuracy of evaluation. To make the evaluation more accurate, practical implementation 2 randomly disables each gate twice in each iteration to generate the synchronous enabled–disabled pair, as shown in Fig. 3(b).

First, we utilize an equivalent form to calculate $\Delta E_n(g_i)$ by $E_n(g_i = 1)$ and $E_n(g_i = 0)$ as follows:

$$\begin{aligned} &E_n(g_i = 1) - E_n(g_i = 0) \\ &\quad = E_n(g_i = 1) - E_n + E_n - E_n(g_i = 0) \\ &\quad = \Delta E_n(g_i) + \frac{P(g_i = 0)}{P(g_i = 1)}\Delta E_n(g_i) \\ &\quad = \frac{1}{P(g_i = 1)}\Delta E_n(g_i) \\ &\quad \Rightarrow \Delta E_n(g_i) = P(g_i = 1)(E_n(g_i = 1) - E_n(g_i = 0)). \end{aligned} \quad (16)$$

Then, we can represent $S_n(g_i)$ by (12), (13), and (16) as

$$S_n(g_i) = P(g_i = 1)\sum_{k=1}^{n}(E_k(g_i = 1) - E_k(g_i = 0))\lambda_k(\xi, n). \quad (17)$$

To generate the synchronous enabled–disabled pairs, all gates are randomly disabled twice in each iteration. For all iterations $\{1, 2, \ldots, n\}$, the subsequence $\{t_{i1}, t_{i2}, \ldots, t_{il}\}$ denoted as $\{t_{il}\}$ are iterations, where one of random disabling is $g_i = 1$ and the other is $g_i = 0$. The probability of $k \in \{t_{il}\}$ denoted as p_t is

$$p_t = 2P(g_i = 0)P(g_i = 1). \quad (18)$$

For an iteration $k \in \{t_{il}\}$, we denote the two random disabling as $T(k)$ where $g_i = 1$ and $T'(k)$, where $g_i = 0$. Then, $S_n(g_i)$ can be calculated as follows:

$$\begin{cases} S_n(g_i) = P(g_i = 1)\sum_{k=1}^{l} D(t_{ik})\lambda_k\left(\xi^{\frac{1}{p_t}}, l\right) \\ D(t_{ik}) = f_{t_{ik}}\left(X_{t_{ik}}, Y_{t_{ik}}, T(t_{ik})\right) - f_{t_{ik}}\left(X_{t_{ik}}, Y_{t_{ik}}, T'(t_{ik})\right). \end{cases} \quad (19)$$

Authorized licensed use limited to: Lanzhou University. Downloaded on December 09,2021 at 13:29:55 UTC from IEEE Xplore. Restrictions apply.

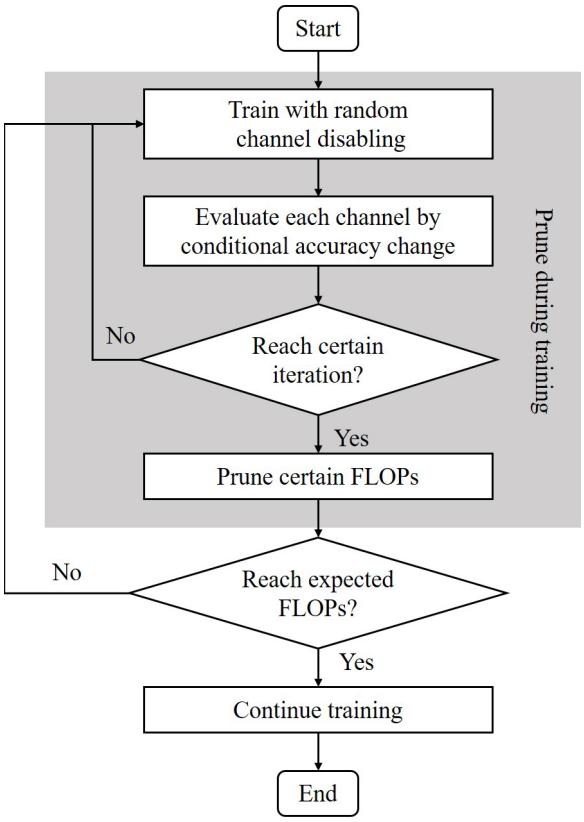

Fig. 4. Brief framework of our channel pruning algorithm. In this framework, the importance of each channel is evaluated by CAC with random channel disabling. Then, prune unimportant channels at several certain iterations to expected FLOPs. After pruning, the network will obtain a stable structure at the early stage of training. Finally, the network can continue to be trained with a stable structure to the end of the training.

Back to the iterative form

$$\begin{cases} S_n(g_i) = P(g_i = 1)c_{t_{il}} \\ c_{t_{il}} = \xi^{\frac{1}{p_t}} c_{t_{i(l-1)}} + \left(1 - \xi^{\frac{1}{p_t}}\right) D(t_{il}) \\ D(t_{il}) = f_{t_{il}}\left(X_{t_{il}}, Y_{t_{il}}, T(t_{il})\right) - f_{t_{il}}\left(X_{t_{il}}, Y_{t_{il}}, T'(t_{il})\right) \\ c_{t_{i0}} = 0,\ p_t = 2P(g_i = 0)P(g_i = 1). \end{cases} \tag{20}$$

E. Pruning Algorithm

As shown in Fig. 4, we can prune unimportant channels at the early stage of training. To evaluate the importance, each channel is disabled randomly while training so that the importance of each channel can be evaluated by CAC. At some certain iterations, some FLOPs can be pruned until the network contains expected FLOPs. With the stable pruned structure at the early stage, the network can be trained for remaining iterations to get better performance. In this framework, we propose two detailed algorithms with the proposed two practical implementations of CAC, respectively.

Algorithms 1 and 2 adopt practical implementations 1 and 2, respectively, and Algorithm 2 shares the same inputs and outputs with Algorithm 1. For the input parameters, p is the disabling probability, ξ is the decay parameter of the exponential moving average, H indicates the iterations to prune, and C indicates maximum computation costs of the network after each pruning. Both algorithms prune during training rather than after training, so it does not need fine-tuning to maintain the performance. Benefiting from the practical implementations, the importance of each channel can be evaluated at any stage of training. In the mammalian brain, synaptic pruning mainly takes place at the early stage of development childhood and adolescence, and synapses keep relatively stable in the long adult period. In our algorithms, we can also prune unimportant channels at the early stage of training so that the network can form a stable structure at the early stage of training.

Algorithm 1 Pruning by Practical Implementation 1

Input:
ξ: the decay parameter in (15)
p: the probability $P(g_i = 0)$ that each gate is disabled
H: a set of iterations when to prune
C: a set of maximum computation capabilities of **N**et after every pruning

Output:
Net$'$: Networks after training with our method of brain development inspired pruning

Ensure:

```
Initialize network Net, gate G[c], maximum iteration s,
  a = 0, b[c] = 0 and S[c] = 0 where c is the total number
  of channels
for k = 1 to s do
  Random disable each gate in G[c] by probability p
  Train and update Net, and get accuracy A of current batch
  a ← ξa + (1 − ξ)A
  for i = 1 to length of G do
    if G[i] = 0 then
      b[i] ← ξ^(1/p) b[i] + (1 − ξ^(1/p))A
      S[i] ← a − b[i]
    end if
  end for
  if k in H and equals to H[j] then
    while the computation capability of Net > C[j] do
      Permanently deactivate g_i and remove G[i] from G
        that satisfies S[i] = min(S)
      Remove S[i], b[i] from S and b
    end while
  end if
end for
Net' ←Net
return Net'
```

IV. Experiments

A. Data Sets

ImageNet[1] Large Scale Visual Recognition Challenge (ILSVRC) is an important data set in large-scale data set for object recognition and detection. It contains 1.28 million various size training images of 1000 classes, 50 000 validation

[1] http://www.image-net.org/challenges/LSVRC/

Authorized licensed use limited to: Lanzhou University. Downloaded on December 09,2021 at 13:29:55 UTC from IEEE Xplore. Restrictions apply.

Algorithm 2 Pruning by Practical Implementation 2

Ensure:

Initialize network **N**et, gate $G[c]$ and $G'[c]$, maximum iteration s, and $S[c] = 0$ where c is the total number of channels
for $k = 1$ to s **do**
  Random disable each gate in $G[c]$ and $G'[c]$ by probability p
  Train and update **N**et with gate G, and get accuracy a and a' of current batch with gate G and G', respectively.
  for $i = 1$ to length of G **do**
    if $G[i] \neq G'[i]$ **then**
      $S[i] \leftarrow \xi^{1/2p(1-p)} S[i] + \left(1 - \xi^{1/2p(1-p)}\right)(G[i] - G'[i])(a - a')$
    end if
  end for
  if k in **H** and equals to **H**$[j]$ **then**
    while the computation capability of **N**et > **C**$[j]$ **do**
      Permanently disable g_i and remove $G[i]$ and $G'[i]$ from G and G' that satisfies $S[i] = \min(S)$
      Remove $S[i]$ from S
    end while
  end if
end for
Net$'$ ←**N**et
return **N**et$'$

images, and 100 000 testing images. We test our model on the validation images.

CIFAR[2] data set is a popular benchmark in computer vision for object recognition, which is efficient to train due to its small scale. CIFAR data set consists of CIFAR10 composed of ten coarse classes and CIFAR100 composed of 20 coarse classes and 100 fine classes. Both CIFAR10 and CIFAR100 have 60 000 32×32 color images, 50 000 and 10 000 of which are for training and testing, respectively.

MNIST[3] is tiny data sets of handwritten digits composed of 50 000 28×28 training images and 10 000 testing images for people to learn techniques and methods in pattern recognition while spending few efforts.

B. Data Preprocessing

For all data sets, simple data augmentations are applied. Specifically, we do pixelwise mean subtraction for images in each data set, then resize them to 256×256 for ImageNet or resize them to a larger size (i.e., 40×40 for CIFAR and 32×32 for MNIST) with zero padding, and, finally, crop them from resized images or their horizontal flip to smaller size (i.e., 224×224 for ImageNet, 32×32 for CIFAR, and 28×28 for MNIST). For testing, we do the same data preprocessing except random flipping and cropping in place of the single-center crop.

[2]http://www.cs.toronto.edu/ kriz/cifar.html
[3]http://yann.lecun.com/exdb/mnist/

C. Network Architecture and Training Details

For CIFAR and ImageNet, option A ResNet with zero-padding shortcuts in [6] is adopted as baseline networks. In each residual block, a random gate is added before each channel of the input of the residual block and outputs of two convolutional layers; meanwhile, the shortcut branch remains intact without the random gate. In order to fully demonstrate the effectiveness of our method, we test our method on various ResNet on CIFAR (i.e., ResNet-32/44/56/110). Because of time-consuming on ImageNet, only ResNet-18 is utilized to show the effectiveness of large scale data set. To demonstrate the effectiveness on various types of networks, we also conduct experiments on traditional convolutional neural networks VGGNet without shortcut on CIFAR [37] and fully connected network [38] on MNIST. Before each channel on VGGNet and each neuron on the fully connected network, the random gate is added.

All networks are trained by SGD with a Nesterov momentum of 0.9. To alleviate overfitting, a weight decay of 10^{-4} is adopted. After each convolutional layer or fully connected layers of all networks, batch normalization and Relu are utilized. Weights are initialized as in [60] for all networks. Batch sizes of CIFAR, ImageNet, and MNIST are 64, 192, and 64, respectively. The learning rate of all baseline networks is initialized with 0.1, divided by 10 at 96k, 192k, and 256k iterations for CIFAR and MNIST, at 96k, 160k, 224k, and 288k iterations for ImageNet, and terminated at 320k iterations.

D. Dynamical Channel Pruning Details

To evaluate the importance of each channel, each channel needs enough disabling probability to be fully evaluated, so disabling probability p cannot be set too small. At the same time, p cannot be set too big because too many channels disabled simultaneously will increase the mutual interference among different channels and will bring drastic change for a network, which may lead to difficulty in convergence for training. In our experiments, it is set as $p = 0.05$. The total number of channels disabled obeys binomial distribution $B(c, p)$, where c is the total number of channels. The expectation of the total number of channels disabled is cp with variance $cp(1 - p)$, so there will be about 5% channels disabled at each iteration, which is neither too big nor too small. To describe more easily, the number of times how many pruning operations are triggered is denoted as N_p. Networks are pruned after the first time learning rate drops and are pruned every r iterations. Thus, iterations when to prune H are

$$H[i] = h_{\text{start}} + ir, \quad i \in \left\{0, 1, 2 \ldots, N_p - 1\right\} \tag{21}$$

where h_{start} is the first iteration to prune and r is iterations between two pruning. The initial computation capability of network is denoted as c_{ini}, and the target pruning rate over c_{ini} is denoted as u. At each pruning, the same rate of FLOPs over current FLOPs is pruned, and then, the maximum FLOPs after each pruning C are

$$C[i] = (1 - u)^{\frac{i+1}{N_p}} c_{\text{ini}}, \quad i \in \left(0, 1, 2 \ldots, N_p - 1\right). \tag{22}$$

Authorized licensed use limited to: Lanzhou University. Downloaded on December 09,2021 at 13:29:55 UTC from IEEE Xplore. Restrictions apply.

In our experiments, we set $h_{\text{start}} = 96\,000$, $r = 30\,000$, $N_p = 2$ for pruning 10% of FLOPs, $N_p = 6$ for pruning 30%, and $N_p = 14$ for pruning more than 50%, in which about 5% channels are pruned each time. It is corresponding to the random disabling probability $p = 0.05$ because the importance of each channel is evaluated under the condition that 5% channels are disabled. We set $\xi = 0.9997$ ($\xi^r = 0.4$), so the contribution of accuracy change more than r iterations before current iteration will decay to lower than 0.4. To evaluate how our method improves the performance of networks, we Double the initial Channels and then Prune Half (DCPH) of them to baseline networks' counterparts, so we can compare the performance of DCPH and baseline networks with the same FLOPs.

V. Results

In this section, we carry out experiments to show the efficientness of our method on deep neural network compression. To show the superiority of our method, we compare our method with other methods. With the proposed dynamical early pruning framework, we also compare the proposed criterion with some indirect criteria, such as the magnitude and BN scale factor of a channel. Furthermore, we also carry out experiments to improve the performance by DCPH, which shows that our method can obtain a better structure to get better performance with the same FLOPs.

A. Parameter and Computation Reductions

Table I shows parameter and computation reductions by the proposed method compared with baseline on CIFAR10, CIFAR100, and ImageNet data sets. Baseline denotes results by normal training. In column-1, Network-PI1 and network-PI2 (e.g., VGG-PI1) denote pruning by practical implementation 1 and 2 of our proposed method, respectively, and 30% and 50% pruned denotes trained by our pruning method with more than 30% and 50% FLOPs pruned. FLOPs are the amount of float multiply operation in [6]. M is 10^6, and B is 10^9. The pruned ratios of parameters and FLOPs are shown in columns 4 and 6. The results show that all models we test on CIFAR can be reduced more than 30% FLOPs and parameters while 10% on ImageNet without accuracy loss by our method. Even with FLOP and parameter reduction of about 50% on CIFAR and 25% on ImageNet, networks can still maintain competitive or better performance than baseline networks.

In Table II, we compare our method with other methods on ImageNet, CIFAR10, and MNIST data sets with similar parameters and FLOPs pruned. ↓ denotes error reduction compared with the baseline. Params denote parameters. #Neurons denotes the number of neurons. For our methods, practical implementation 2 is adopted to compare with other methods. On ImageNet, error reductions are based on each method's own baseline because test errors of different methods vary a lot with different data augmentation and training details. Our baseline adopts the same data augmentation with ThiNet [24], and our baseline's top1/top5 errors are 27.07%/8.83%, which is similar to 27.12%/8.86% achieved in ThiNet. With similar parameters and FLOPs reduction, our method only results in 0.79% and 0.51% degradation in top-1 and top-5 errors, respectively, while other methods result in more than 1% and 0.6% degradation. On CIFAR10 and MNIST, error reductions of different methods are based on the same baseline. To get a similar baseline compared with other works on CIFAR, a weight decay of 0.0001 for scale factors of batch normalization is adopted. We get a baseline test error of 6.16%, which is almost the same as the DCP's baseline of 6.20%. The results show that our method can achieve a lower error than the baseline on ResNet-56 (CIFAR) with 2× parameters and FLOPs reduction, while other methods get higher errors. On MNIST, our method achieves a lower error with more than 6× parameter reduction, while others get higher errors.

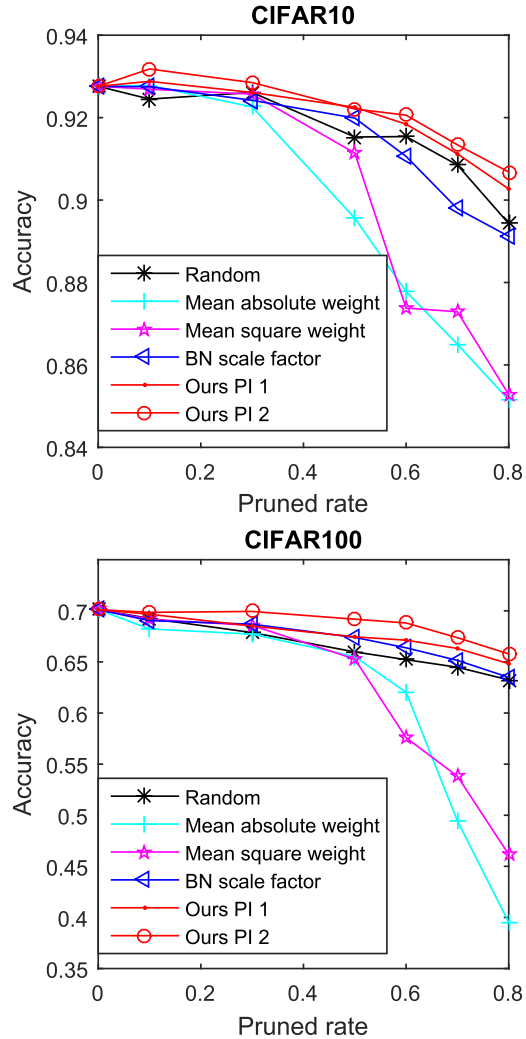

Fig. 5. Comparison with other criteria on our pruning framework. Results under all criteria are in the same training set. Pruned rate is the rate of pruned FLOPs.

With the proposed dynamical early pruning framework, we also compare the proposed criterion with some indirect criteria, including mean absolute weight, mean square weight, and BN scale factor [37] of a channel. To demonstrate the benefits of our pruning framework as well as provide a baseline for other criteria, we also conduct experiments on the random criterion, which prunes channel randomly with no intelligence. Under the same pruning framework, the settings of all the experiments on different criteria keep the same (including the total pruning times N_p and the iteration interval r between the two prunings). Fig. 5 shows the results on CIFAR10 and CIFAR100 for ResNet-32. Random criterion prunes each

Authorized licensed use limited to: Lanzhou University. Downloaded on December 09,2021 at 13:29:55 UTC from IEEE Xplore. Restrictions apply.

TABLE I

RESULTS ON (a) CIFAR10, (b) CIFAR100, AND (c) IMAGENET

(a)

Model	Error	Parameters	Pruned	FLOPs	Pruned
VGG (Baseline)	7.53%	20.02M	-	398M	-
VGG-PI1 (50% pruned)	7.10%	7.65M	61.78%	195M	51.02%
VGG-PI2 (50% pruned)	**6.81%**	5.50M	72.54%	195M	51.01%
ResNet-32 (Baseline)	7.64%	0.46M	-	68.9M	-
ResNet-32-PI1 (30% Pruned)	7.39%	0.31M	31.94%	48.2M	29.94%
ResNet-32-PI2 (30% Pruned)	**7.15%**	0.34M	27.06%	48.2M	30.01%
ResNet-32-PI1 (50% Pruned)	7.75%	0.23M	48.44%	33.8M	50.94%
ResNet-32-PI2 (50% Pruned)	7.79%	0.24M	47.96%	34.38M	50.07%
ResNet-44 (Baseline)	7.55%	0.65M	-	97.2M	-
ResNet-44-PI1 (30% Pruned)	7.24%	0.46M	29.42%	67.9M	30.08%
ResNet-44-PI2 (30% Pruned)	**6.74%**	0.45M	31.52%	68.01M	30.02%
ResNet-44-PI1 (50% Pruned)	7.44%	0.33M	49.66%	47.5M	51.14%
ResNet-44-PI2 (50% Pruned)	7.58%	0.34M	47.44%	48.56M	50.03%
ResNet-56 (Baseline)	7.12%	0.85M	-	125.5M	-
ResNet-56-PI1 (30% Pruned)	**6.52%**	0.60M	29.43%	87.6M	30.16%
ResNet-56-PI2 (30% Pruned)	6.53%	0.59M	30.42%	87.84M	30.00%
ResNet-56-PI1 (50% Pruned)	7.04%	0.41M	50.95%	61.4M	51.08%
ResNet-56-PI2 (50% Pruned)	6.90%	0.43M	49.04%	62.7M	50.03%
ResNet-110 (Baseline)	6.62%	1.72M	-	252.9M	-
ResNet-110-PI1 (30% Pruned)	6.32%	1.23M	28.56%	176.5M	30.20%
ResNet-110-PI2 (30% Pruned)	**5.89%**	1.22M	29.11%	176.8M	30.07%
ResNet-110-PI1 (50% Pruned)	6.46%	0.82M	52.31%	123.5M	51.15%
ResNet-110-PI2 (50% Pruned)	6.14%	0.88M	48.91%	126.4M	50.00%

(b)

Model	Error	Parameters	Pruned	FLOPs	Pruned
VGG (baseline)	29.83%	20.07M	-	398M	-
VGG-PI1 (40% pruned)	28.94%	11.35M	43.46%	251M	37.04%
VGG-PI2 (40% pruned)	**27.72%**	8.92M	55.57%	251M	37.01%
ResNet-32 (Baseline)	31.52%	0.46M	-	68.9M	-
ResNet-32-PI1 (30% Pruned)	31.49%	0.35M	23.66%	48.1M	30.22%
ResNet-32-PI2 (30% Pruned)	**30.49%**	0.36M	22.22%	48.2M	30.08%
ResNet-32-PI1 (50% pruned)	32.55%	0.26M	43.11%	33.8M	50.98%
ResNet-32-PI2 (50% pruned)	31.89%	0.27M	40.49%	34.36M	50.1%
ResNet-44 (Baseline)	30.13%	0.65M	-	97.2M	-
ResNet-44-PI1 (30% Pruned)	29.91%	0.49M	25.10%	67.7M	30.34%
ResNet-44-PI2 (30% Pruned)	**29.15%**	0.48M	26.10%	68.0M	30.06%
ResNet-44-PI1 (50% Pruned)	30.90%	0.36M	44.78%	47.5M	51.12%
ResNet-44-PI2 (50% Pruned)	30.18%	0.36M	45.20%	48.5M	50.10%
ResNet-56 (Baseline)	29.10%	0.85M	-	125.5M	-
ResNet-56-PI1 (30% Pruned)	29.08%	0.62M	26.50%	87.6M	30.23%
ResNet-56-PI2 (30% Pruned)	**28.69%**	0.63M	26.13%	87.8M	30.02%
ResNet-56-PI1 (50% Pruned)	30.22%	0.46M	45.51%	61.4M	51.11%
ResNet-56-PI2 (50% Pruned)	29.90%	0.46M	45.69%	62.7M	50.02%
ResNet-110 (Baseline)	27.47%	1.72M	-	252.9M	-
ResNet-110-PI1 (30% Pruned)	27.36%	1.24M	28.13%	176.5M	30.20%
ResNet-110-PI2 (30% Pruned)	**27.21%**	1.25M	27.15%	177.0M	30.01%
ResNet-110-PI1 (50% Pruned)	28.19%	0.93M	45.77%	123.5M	51.18%
ResNet-110-PI2 (50% Pruned)	27.84%	0.93M	45.65%	126.4M	50.02%

(c)

Model	Error(Top1/Top5)	Parameters	Pruned	FLOPs	Pruned
ResNet-18 (Baseline)	31.54%/11.36%	12.02M	-	1.8B	-
ResNet-18-PI1 (10% Pruned)	31.55%/11.56%	10.67M	11.22%	1.6B	10.01%
ResNet-18-PI2 (10% Pruned)	**31.38%/11.27%**	9.71M	11.67%	1.6B	10.01%
ResNet-18-PI1 (25% pruned)	31.97%/11.73%	8.65M	28.03%	1.3B	25.00%
ResNet-18-PI2 (25% pruned)	31.83%/11.65%	7.70M	29.97%	1.3B	25.01%

channel randomly with no intelligence. Mean absolute weight and mean square weight are the mean magnitudes of absolute weight and square weight in each channel's filter, respectively. BN scale factor is the absolute value of the scale factor in the batch-norm layer for each channel. our PI 1 and 2 are the two practical implementations of the proposed CAC criterion. The results show that both our criteria perform better than other criteria. Benefiting from the synchronous evaluation, practical implementation 2 achieves better performance than practical implementation 1. In our pruning framework, random criterion

Authorized licensed use limited to: Lanzhou University. Downloaded on December 09,2021 at 13:29:55 UTC from IEEE Xplore. Restrictions apply.

TABLE II

COMPARISON WITH PREVIOUS METHODS. (a) IMAGENET (RESNET-50). (b) CIFAR10 (RESNET-56). (c) MNIST

(a)

Method	Top-1 Err.(%)	Top-5 Err.(%)	#Param.↓	#FLOPs↓
ThiNet [24]	+1.87	+1.12	2.06×	2.25×
CP [34]	-	+1.40	-	2×
DCP [35]	+1.06	+0.61	2.06×	2.25×
Ours	**+0.79**	**+0.51**	2.07×	2.23×

(b)

Method	Error(%)	#Param.↓	#FLOPs↓
Baseline	6.16	-	-
ThiNet [24]	+0.86	1.97×	1.99×
CP [34]	+1.04	-	2.00×
DCP [35]	+0.35	1.97×	1.99×
Ours	**-0.06**	2.00×	2.00×

(c)

Method	Error(%)	#Param.↓	#Neurons
Baseline	1.43	-	784-500-300-10
SSL [38]	+0.10	6.06×	434-174-78-10
NS [37]	+0.06	6.45×	784-100-60-10
Ours	**-0.05**	6.45×	275-229-92-10

TABLE III

PERFORMANCE IMPROVEMENT BY DCPH. (a) CIFAR10. (b) CIFAR100

(a)

Model	Baseline	Baseline(with dropout)	DCPH-PI1	DCPH-PI2
ResNet-32	7.64%	7.24%	6.82%	**6.49%**
ResNet-44	7.55%	6.83%	6.71%	**6.24%**
ResNet-56	7.12%	6.28%	6.10%	**6.08%**
ResNet-110	6.62%	6.18%	5.71%	**5.59%**

(b)

Model	Baseline	Baseline(with dropout)	DCPH-PI1	DCPH-PI2
ResNet-32	31.52%	29.65%	28.60%	**28.05%**
ResNet-44	30.13%	28.97%	28.16%	**27.03%**
ResNet-56	29.10%	28.37%	27.14%	**26.96%**
ResNet-110	27.47%	27.05%	26.42%	**26.16%**

can still perform well due to early pruning and continue training after pruning. BN scale factor performs similar or better than random criterion. Mean absolute weight and mean square weight perform much worse than other criteria even including random criterion.

B. Performance Improvement

To further study how our method influence the final performance, we DCPH of them to baseline's counterpart. DCPH has the same or less computation capability than its baseline but different structures due to the pruning. Table III shows test errors of baseline, baseline (with dropout), and DCPH. The baseline is trained by the standard training process. Baseline (with dropout) is trained with random disabling but no pruning. DCPH-PI1 and DCPH-PI2 are errors of DCPH by practical implementation 1 and 2, respectively. Both DCPH-PI1 and DCPH-PI2 perform significantly better than baseline (1.03%–3.47% error reduction) and baseline with dropout (0.2%–1.94% error reduction) with same or less computation capability. It shows that the network can form a better structure

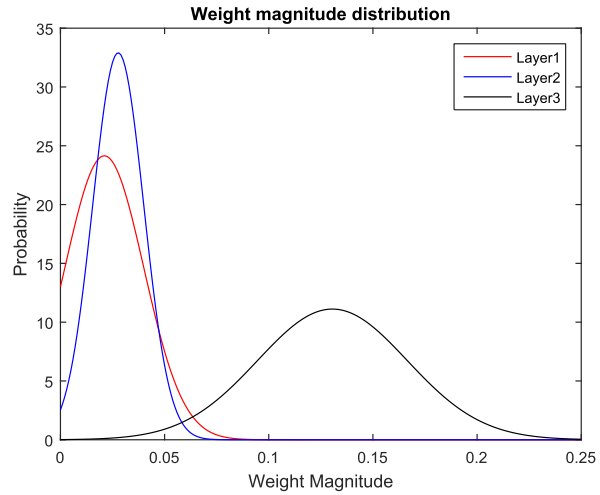

Fig. 6. Distributions of weight magnitude for each layer on MNIST. Weight magnitude is the mean of absolute weights corresponding to the same neuron for each layer on MNIST. Each line is the distribution of the weight magnitudes of a layer fit by the Gaussian distribution.

with the same or fewer FLOPs after the proposed channel pruning method by CAC.

VI. ANALYSIS

The results on multiple data sets (i.e., CIFAR, ImageNet, and MNIST) for networks with various types of architecture (i.e., ResNet, VGGNet, and MLP) demonstrate the effectiveness of our method for channel pruning. Without accuracy loss compared with the baseline network, more than 30% and 40% FLOPs for multiple ResNet and VGGNet on CIFAR, respectively, more than 10% FLOPs for ResNet-18 on ImageNet, and 80% FLOPs for MLP on MNIST can be pruned by the proposed channel pruning method. With the same or fewer FLOPs, DCPH can significantly improve the performance of networks. It demonstrates that our method can not only compress the network effectively but also form a more reasonable structure to improve the performance of the network. Benefiting from the early pruning, the network can keep a stable structure for continuous training so that even random pruning can perform well. By dynamically pruning unimportant channels during training, repetitious pruning and fine-tuning scheme can be avoided. More importantly, our method can form a better structure to achieve better performance, while some weight-magnitude-based criteria perform terribly due to the ill structure. We will analyze it in detail in the following.

A. Analysis of Different Criteria

By random pruning, each layer is expected to be pruned uniformly. As a result, pruning with random criterion will keep a similar but slim structure with the initial structure. Benefiting from our early pruning strategy, the network can adjust itself with enough continuous training iterations to adapt the pruned network. Thus, even random criterion can achieve a relatively good performance. Different from pruning without intelligence, other criteria can reshape the structure for the network. However, an improper criterion will shape an improper structure. Fig. 6 shows the distributions fit by

Authorized licensed use limited to: Lanzhou University. Downloaded on December 09,2021 at 13:29:55 UTC from IEEE Xplore. Restrictions apply.

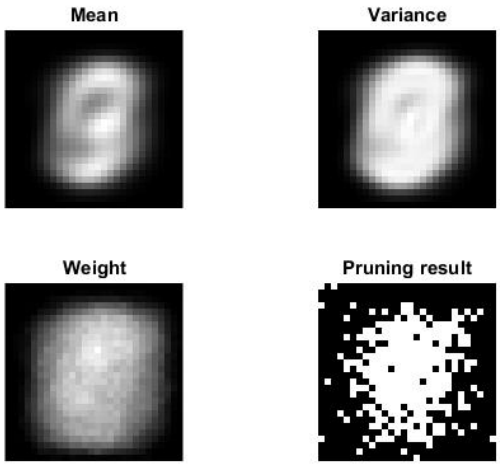

Fig. 7. Visualization of pruning. On MNIST, a three-layer fully connected network is adopted with a gate after each node. By transforming each input image into a vector of pixels, each pixel is regarded as a node; thus, it will be evaluated and pruned during training. Mean and variance are the pixelwise mean and variance of input images on MNIST, respectively. Weight is the mean weight magnitude corresponding to each pixel. Pruning result is the pruning result of each pixel.

the Gaussian distribution of weight magnitude for each layer of three-layer MLP on MNIST. Distributions of three layers differ a lot. It demonstrates that the magnitude-based criterion is improper to be utilized as an overall criterion, or it will lead to an ill structure. That is why the two magnitude-based criteria in Fig. 5 perform terribly.

A good criterion should possess the ability to measure the importance of each channel. Besides the proposed criterion is directly related to the final accuracy in theory and perform excellently in practice, we also visualize it in Fig. 7 to show its abilities for evaluating channel importance. On MNIST, a three-layer fully connected network is adopted with a gate after each node. By transforming each input image into a vector of pixels, each pixel is regarded as a node; thus, the overall importances of all nodes including pixels are evaluated as training. In Fig. 7, mean and variance are the pixelwise mean and variance of each pixel in input images. Weight is the mean absolute weight corresponding to each pixel after standard training. The pruning result is the result of each pixel pruned by the proposed criterion in Table II. There is not a ground truth of the importance of each pixel, so we show the mean and variance of each pixel, which is related to the importance of each pixel. The pixel with low mean and variance would provide less information, so it might be unimportant in high probability. Our pruning result has a strong correlation with mean and variance map, which demonstrates that the proposed criterion does possess the ability to evaluate the importance. It should be pointed out that the pruning result is under an overall pruning rather than a layerwise pruning. Though the magnitude of weight also has a strong correlation with mean and variance map, it is improper to be utilized as an overall criterion due to different distributions of different layers illustrated in Fig. 6.

B. Analysis of the Structure After Pruning

On CIFAR, about 30% FLOPs can be pruned without accuracy loss for various scales of ResNet. As the scale of the network varies from ResNet-32 to ResNet-110 on CIFAR, a similar rate of FLOPs can be pruned. It seems that pruning optimizes the structure of a network rather than remove the redundancy only, so different scale networks with similar structures can be pruned similar rate of FLOPs. It is also reasonable that DCPH performs significantly better than baseline networks with the same FLOPs but different structures. To visual the structure of networks intuitively, we divide layers of ResNet on CIFAR into low level, middle level, and high level (i.e., low level with the feature map size of 32×32, mid-level with 16×16, and high level with 8×8). To show the differences of structures after pruning better, the structure of networks with the same FLOPs pruning rate but on different data sets (i.e., CIFAR10 and CIFAR100) are placed in the same figure to compare the different change of structure. As shown in Fig. 8, blue lines are on CIFAR10, and red lines are on CIFAR100. The computation rate represents the rate of FLOPs after pruning versus baseline on each level. Notably, we can see that networks always retain more FLOPs on high-level layers after pruning on CIFAR100 than on CIFAR10. It is consistent with human experience that CIFAR100 consists of more categories than CIFAR10, so it needs more high-level channels that are near to the final classifying layer. Under all pruning ratios and all scales of ResNet, both proposed criteria can retain more high-level FLOPs on CIFAR100 steadily. It demonstrates that our method can adaptively and automatically form a better structure for different data sets or tasks.

TABLE IV
PARAMETER ANALYSES

p	Err.	ξ^r	Err.	r	Err.	h_{start}	Err.
0.01	6.44	0.2	6.47	10000	6.62	48000 ($lr = 0.1$)	6.63
0.03	6.17	0.3	6.11	20000	6.34	96000 ($lr = 0.01$)	**6.09**
0.05	**6.09**	0.4	**6.09**	30000	6.09	192000 ($lr = 0.001$)	6.49
0.1	6.28	0.5	6.54	40000	**6.07**	256000 ($lr = 0.0001$)	6.44
0.15	6.52	0.6	6.79	50000	6.11	320000 (pre-train)	6.32

C. Analysis of Regularization Effects by Random Disabling

In Table III, baseline (with dropout) is trained with random channel disabling but no pruning. Benefiting from the dropout effect, baseline (with dropout) performs better than baseline on all networks in our experiments on CIFAR. With the same or fewer FLOPs, DCPH performs better than not only baseline but also baseline (with dropout) on all networks we test. It shows that dropout effects do improve the performance, but the proposed pruning method can further improve performance based on dropout effects.

D. Ablation Study

To evaluate the influence of each parameter, a series of parameter analyses are conducted, including p, ξ, r, and h_{start}, which is demonstrated in Table IV. For random disabling probability p and ξ, it will suffer a performance reduction if we set them either too big or too small. The results show that $p = 0.05$ and $\xi^r = 0.4$ are proper, which are adopted

Authorized licensed use limited to: Lanzhou University. Downloaded on December 09,2021 at 13:29:55 UTC from IEEE Xplore. Restrictions apply.

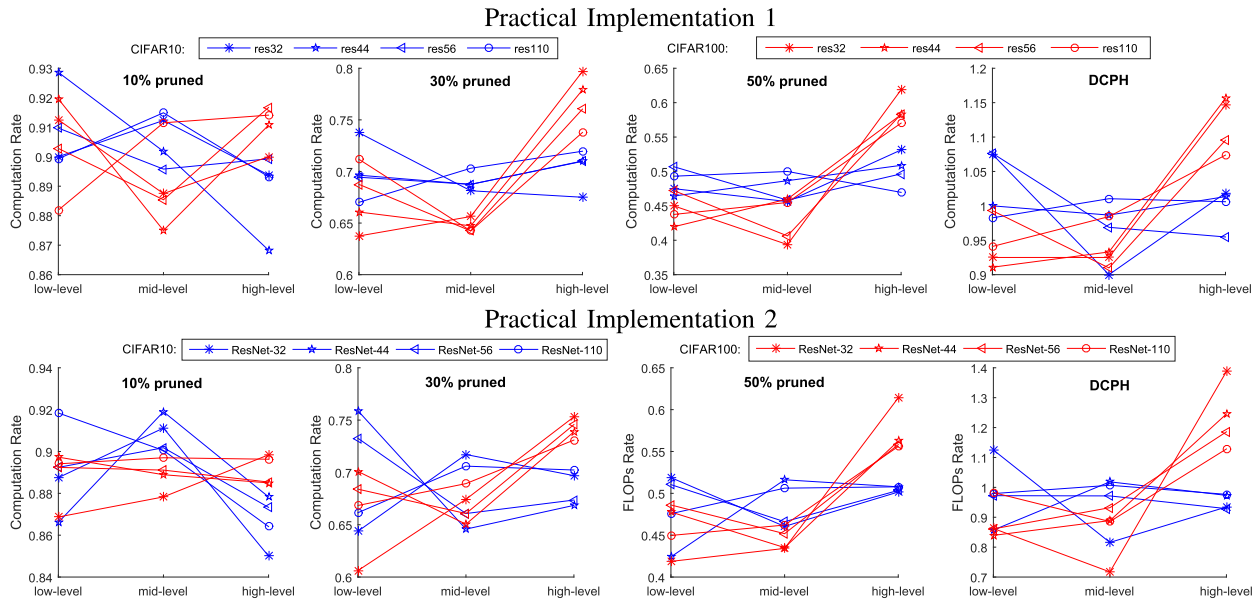

Fig. 8. Computation distributions in CIFAR10 and CIFAR100. Computation rate is FLOPs rate versus baseline in different level (i.e., low level with a feature map size of 32×32, mid-level of 16×16, and high level of 8×8). Blue lines are on CIFAR10, and red lines are on CIFAR100.

TABLE V

COMPARISON WITH IDEAL CRITERION, CAC CRITERION, AND PRACTICAL IMPLEMENTATION OF CAC CRITERION (PRACTICAL)

Criterion	Naive	CAC	CAC (practical)
Evaluating by disabling multiple channels	No	Yes	Yes
Stopping updating model to evaluate	Yes	Yes	**No**
Error	1.61%	**1.36%**	1.38%

TABLE VI

WALL-CLOCK TIME SAVING ON THE ACTUAL DEVICE

Flops Pruned	Baseline	51% pruned	77.2% pruned
Time/iter(CPU)	1.33s	0.57s	0.28s
Time/iter(GPU)	0.058s	0.037s	0.022s
Error	7.53%	6.81%	6.93%

by our approach. r is the number of iterations between two pruning periods, and it should be large enough to provide sufficient iterations to adapt the pruned model and reevaluate the importance of each channel for the next pruning period. As the results show, it works well when $r \geq 30\,000$, and we set $r = 30\,000$ in our experiments. h_{start} controls when the network starts to be pruned. If it is too early, such as $h_{\text{start}} = 48\,000$, the network is not fully trained, and the importance of each channel cannot be well evaluated. If it is too late, such as $h_{\text{start}} \geq 1\,92\,000$, the learning rate is too small so that the plasticity of the network is poor. The results show that the second learning stage $h_{\text{start}} = 96\,000$ with $lr = 0.01$ can achieve better performance. We also carry out experiment on a pretrained model with fine-tuning the pruned model from $lr = 0.01$ to $lr = 0.0001$. It cannot perform as well as pruning at the second learning stage during training. That is because it cannot be trained in time to recover the performance after each pruning.

To evaluate the performance of different types of criteria, including naive criterion and proposed CAC, further experiments are conducted on MNIST. In Table V, the naive criterion disables and evaluates each channel separately and independently. CAC is the CAC we proposed, which stops updating the model to evaluate the importance of each channel. CAC (practical) is the practical implementation 2 of CAC, which does not stop updating the model while evaluating. Compared with the naive criterion, though the proposed CAC brings the mutual interferences among channels by disabling multiple channels simultaneously, the mutual interferences will be decreased by combining the evaluation of many iterations. By disabling the same parameter and computation costs (on expectation) each time, the comparisons of different channels suffer few effects by different parameter and computation costs for different channels. As a result, our proposed criteria perform much better than the naive criterion. CAC and CAC (practical) achieve similar performance. It demonstrates that it is not necessary to stop updating the model while evaluating.

We test the wall-clock speed of VGGNet on an ordinary PC with CPU Intel Core i7-4790 and GPU NVIDIA GeForce GTX 750 during inference time. The experiments are conducted using Caffe with batch size 64, and the results are shown in Table VI. The wall-clock time saving of this model roughly matches the FLOPs saving. The speedup on CPU is more significant than that on GPU because GPU is not as sensitive to FLOPs as CPU due to parallel computing. Note that all speedups require no special libraries/hardware.

VII. CONCLUSION

We have proposed a novel channel pruning method that prunes networks at the early stage of training so that it does

Authorized licensed use limited to: Lanzhou University. Downloaded on December 09,2021 at 13:29:55 UTC from IEEE Xplore. Restrictions apply.

not need complicated pruning and fine-tuning scheme. Rather than the indirect criterion, we proposed CAC to evaluate the importance of each channel, which is related to final accuracy directly. For training more efficiently, we proposed two practical implementations of the CAC to keep the proposed method both effective and efficient. Many experiments demonstrate that our method can compress the network efficiently and form a better structure to achieve better performance. Besides, it is easy to implement because it demands a few extra structures except the random gate and an exponential moving average of accuracy.

Appendix
Detailed Proofs and Explanations for Practical Implementation 1

First, by exponential moving average of $\Delta E_n(g_i)$, the practical implementation of CAC denoted as $S_n(g_i)$ is defined as follows:

$$S_n(g_i) = (1-\xi)\sum_{k=2}^{n}\Delta E_k(g_i)\xi^{n-k} + \Delta E_1(g_i)\xi^{n-1}$$

$$S_n(g_i) = (1-\xi)\sum_{k=2}^{n}\Delta E_k(g_i)\xi^{n-k} + (1-\xi+\xi)\Delta E_1(g_i)\xi^{n-1}$$

$$S_n(g_i) = (1-\xi)\sum_{k=1}^{n}\Delta E_k(g_i)\xi^{n-k} + \Delta E_1(g_i)\xi^{n}. \tag{23}$$

In our practical implementations, n is large, so $\Delta E_1(g_i)\xi^n$ is very small. We can omit $\Delta E_1(g_i)\xi^n$. Then

$$S_n(g_i) = (1-\xi)\sum_{k=1}^{n}\Delta E_k(g_i)\xi^{n-k}. \tag{24}$$

To be more brief, let

$$\lambda_k(\xi, n) = (1-\xi)\xi^{n-k}. \tag{25}$$

Second, single-batch input data instead of K batches is taken for each iteration so that we need not stop training to evaluate E_k and $E_k(g_i = 0)$. Then, $S_n(g_i)$ can be represented as follows by (6), (7), and (23):

$$S_n(g_i) = \sum_{k=1}^{n} f_k(X_k, Y_k, T(k))\lambda_k(\xi, n) - \sum_{k=1}^{n}\frac{f_k(X_k, Y_k, T(k))\mathbf{1}(g_i(k)=0)}{\mathbf{1}(g_i(k)=0)}\lambda_k(\xi, n) \tag{26}$$

where λ_k is defined by (25). Thus, we can obtain $S_n(g_i)$ by the exponential moving average of all accuracy and accuracy when gate g_i is disabled.

From the CAC criterion to practical implementation, there are some practical problems caused by single disabling for each iteration. Thus, we will give a redefinition. The redefinition inherits the main idea of the original definition and made some adjustments to solve practical problems. The first problem is that $E_n(g_i = 0)$ does not exist when $g_i = 1$ and $K = 1$ according to (7), so we need remove these items. The details are shown in Fig. 9. $\lambda_k(\xi, n)$ is a proportional series, and $\lim_{n\to\infty}\sum_{k=1}^{n}(1-\xi)\xi^{n-k} = 1$. If we directly

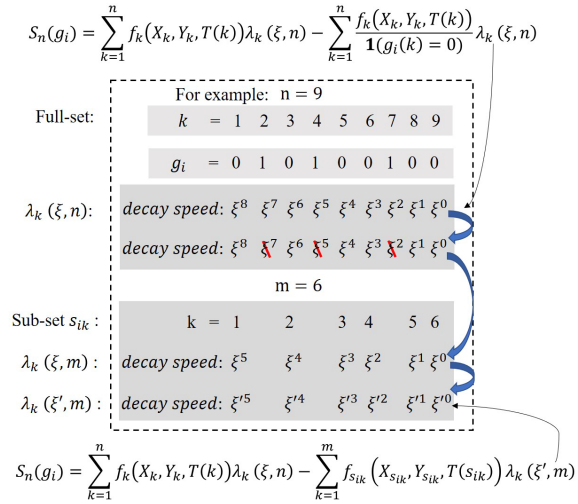

Fig. 9. Clarifications of the redefinition of S_n. For the factor of f_k in $E_k(g_i = 0)$, we remove the items when $g_i = 1$. By picking the subset when $g_i = 0$, $\lambda_k(\xi, m)$ keeps a proportional series. To make the decay speed the same, we adjust the decay parameter from ξ to ξ', where $\xi' = \xi^{1/P(g_i=0)}$.

remove the items when $g_i = 1$, it could not keep the original proportional series (see Fig. 9), whose sum is not 1 (biased). In Fig. 9, we adjust it to a proportional series $\lambda_k(\xi, m)$. Here comes a new problem, because of the subset, the decay speeds of exponential moving average are different between E_k and $E_k(g_i = 0)$ (see Fig. 9). To ensure the same decay speeds, we set a new decay parameter ξ' to balance them

$$S_n(g_i) = \sum_{k=1}^{n} f_k(X_k, Y_k, T(k))\lambda_k(\xi, n) - \sum_{k=1}^{m} f_{s_{ik}}\left(X_{s_{ik}}, Y_{s_{ik}}, T(s_{ik})\right)\lambda_k\left(\xi', m\right) \tag{27}$$

where $\{s_{i1}, s_{i2}, \ldots, s_{im} | g_i(s_{ij}) = 0, j \in \{1, 2, \ldots, m\}\}$ is subsequence from $\{1, 2, \ldots, n\}$.

We set $\xi' = \xi^{1/P(g_i=0)}$, and the proof process is in the following. For an iteration s_{ik}, $n - s_{ik}$ represents the number of all iterations from s_{ik} to n, and $m-k$ represents the number of iterations that gate g_i is disabled ($g_i = 0$) from s_{ik} to n. So

$$\frac{E(m-k)}{E(n-s_{ik})} = P(g_i = 0) \Rightarrow E(m-k) = P(g_i = 0)E(n-s_{ik}). \tag{28}$$

Note that $E()$ here is the expectation function. To make the decay speeds the same, let

$$E\left(\xi^{(n-s_{ik})}\right) = E\left(\xi'^{(m-k)}\right) \Rightarrow \xi^{E(n-s_{ik})} = \xi'^{E(m-k)} \Rightarrow \xi' = \xi^{\frac{1}{P(g_i=0)}}. \tag{29}$$

Acknowledgment

The authors would like to appreciate SITONHOLY Company for providing them the GPU station.

Authorized licensed use limited to: Lanzhou University. Downloaded on December 09,2021 at 13:29:55 UTC from IEEE Xplore. Restrictions apply.

REFERENCES

[1] Y. LeCun *et al.*, "Backpropagation applied to handwritten zip code recognition," *Neural Comput.*, vol. 1, no. 4, pp. 541–551, Dec. 1989.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[3] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*. [Online]. Available: http://arxiv.org/abs/1409.1556

[4] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 1–9.

[5] M. Lin, Q. Chen, and S. Yan, "Network in network," 2013, *arXiv:1312.4400*. [Online]. Available: https://arxiv.org/abs/1312.4400

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[7] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2014, pp. 580–587.

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 9, pp. 1904–1916, Jan. 2014.

[9] R. Girshick, "Fast R-CNN," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 1440–1448.

[10] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2015, pp. 91–99.

[11] W. Liu *et al.*, "SSD: Single shot multibox detector," in *Proc. Int. Eur. Conf. Comput. Vis.* Springer, 2016, pp. 21–37.

[12] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 779–788.

[13] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 2, pp. 386–397, Feb. 2020.

[14] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 3431–3440.

[15] K. Kamnitsas *et al.*, "Efficient multi-scale 3D CNN with fully connected CRF for accurate brain lesion segmentation," *Med. Image Anal.*, vol. 36, pp. 61–78, Feb. 2017.

[16] L. C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Semantic image segmentation with deep convolutional nets and fully connected CRFS," *Comput. Sci.*, no. 4, pp. 357–361, Dec. 2014.

[17] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Proc. Int. Conf. Med. Image Comput. Comput.-Assist. Intervent.*, 2015, pp. 234–241.

[18] G. Huang, Z. Liu, L. V. D. Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, vol. 1, no. 2, p. 3.

[19] A. G. Howard *et al.*, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*. [Online]. Available: http://arxiv.org/abs/1704.04861

[20] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 6848–6856.

[21] W. Chen, S. Tyree, S. Tyree, K. Q. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick," in *Proc. Int. Conf. Int. Conf. Mach. Learn.*, Jun. 2015, pp. 2285–2294.

[22] E. Denton, W. Zaremba, J. Bruna, Y. Lecun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2014, pp. 1269–1277.

[23] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," *Fiber*, vol. 56, no. 4, pp. 3–7, 2016.

[24] J.-H. Luo, H. Zhang, H.-Y. Zhou, C.-W. Xie, J. Wu, and W. Lin, "ThiNet: Pruning CNN filters for a thinner net," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 41, no. 10, pp. 2525–2538, Oct. 2019.

[25] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient ConvNets," 2016, *arXiv:1608.08710*. [Online]. Available: http://arxiv.org/abs/1608.08710

[26] J. Rauschecker, "Neuronal mechanisms of developmental plasticity in the cat's visual system," *Human Neurobiol.*, vol. 3, no. 2, pp. 109–114, 1984.

[27] M. Alavash, S.-J. Lim, C. Thiel, B. Sehm, L. Deserno, and J. Obleser, "Dopaminergic modulation of hemodynamic signal variability and the functional connectome during cognitive performance," *NeuroImage*, vol. 172, pp. 341–356, May 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S105381191830048X

[28] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 1135–1143.

[29] B. Hassibi, D. G. Stork, and G. J. Wolff, "Optimal brain surgeon and general network pruning," in *Proc. IEEE Int. Conf. Neural Netw.*, Mar. 1993, pp. 293–299.

[30] Y. L. Cun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 1989, pp. 598–605.

[31] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *Proc. Int. Eur. Conf. Comput. Vis.* Springer, 2016, pp. 630–645.

[32] S. Zagoruyko and N. Komodakis, "Wide residual networks," 2016, *arXiv:1605.07146*. [Online]. Available: https://arxiv.org/abs/1605.07146

[33] S. Xie, R. Girshick, P. Dollar, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 5987–5995.

[34] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017. vol. 2, no. 6, pp. 1389–1397.

[35] Z. Zhuang *et al.*, "Discrimination-aware channel pruning for deep neural networks," in *Proc. Adv. Neural Inf. Process. Syst.* New York, NY, USA: Curran Associates, 2018, pp. 875–886.

[36] X. Dai, H. Yin, and N. K. Jha, "NeST: A neural network synthesis tool based on a Grow-and-Prune paradigm," *IEEE Trans. Comput.*, vol. 68, no. 10, pp. 1487–1497, Oct. 2019.

[37] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 2755–2763.

[38] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 2074–2082.

[39] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 4510–4520.

[40] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," 2016, *arXiv:1610.02357*. [Online]. Available: http://arxiv.org/abs/1610.02357

[41] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "ShuffleNet v2: Practical guidelines for efficient CNN architecture design," 2018, *arXiv:1807.11164*. [Online]. Available: http://arxiv.org/abs/1807.11164

[42] T. Zhang, G.-J. Qi, B. Xiao, and J. Wang, "Interleaved group convolutions," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 4373–4382.

[43] G. Xie, J. Wang, T. Zhang, J. Lai, R. Hong, and G.-J. Qi, "IGCV2: Interleaved structured sparse convolutional neural networks," 2018, *arXiv:1804.06202*. [Online]. Available: http://arxiv.org/abs/1804.06202

[44] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," 2016, *arXiv:1611.01578*. [Online]. Available: http://arxiv.org/abs/1611.01578

[45] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," 2016, *arXiv:1611.02167*. [Online]. Available: http://arxiv.org/abs/1611.02167

[46] Z. Zhong, J. Yan, W. Wu, J. Shao, and C.-L. Liu, "Practical block-wise neural network architecture generation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 2423–2432.

[47] J. Guo, Y. Li, W. Lin, Y. Chen, and J. Li, "Network decoupling: From regular to depthwise separable convolutions," 2018, *arXiv:1808.05517*. [Online]. Available: http://arxiv.org/abs/1808.05517

[48] Y. Xu, Y. Wang, A. Zhou, W. Lin, and H. Xiong, "Deep neural network compression with single and multiple level quantization," in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2018, pp. 4335–4342.

[49] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 525–542.

[50] M. Courbariaux and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to $+1$ or -1," Feb. 2016, *arXiv:1602.02830*. [Online]. Available: https://arxiv.org/abs/1602.02830

Authorized licensed use limited to: Lanzhou University. Downloaded on December 09,2021 at 13:29:55 UTC from IEEE Xplore. Restrictions apply.

[51] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *Proc. Int. Conf. Learn. Represent.*, 2019, pp. 1–42. [Online]. Available: https://openreview.net/forum?id=rJl-b3RcF7

[52] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," in *Proc. Int. Conf. Learn. Represent.*, 2019, pp. 1–21.

[53] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang, "Network trimming: A data-driven neuron pruning approach towards efficient deep architectures," 2016, *arXiv:1607.03250*. [Online]. Available: http://arxiv.org/abs/1607.03250

[54] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," 2016, *arXiv:1611.06440*. [Online]. Available: http://arxiv.org/abs/1611.06440

[55] S. Ameen and S. Vadera, "Pruning neural networks using multi-armed bandits," *Comput. J.*, Sep. 2019, bxz078, doi: 10.1093/comjnl/bxz078.

[56] J. Lin, Y. Rao, J. Lu, and J. Zhou, "Runtime neural pruning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 2181–2191.

[57] X. Gao, Y. Zhao, Ł. Dudziak, R. Mullins, and C.-Z. Xu, "Dynamic channel pruning: Feature boosting and suppression," 2018, *arXiv:1810.05331*. [Online]. Available: http://arxiv.org/abs/1810.05331

[58] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Penksy, "Sparse convolutional neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 806–814.

[59] G. Huang, S. Liu, L. V. D. Maaten, and K. Q. Weinberger, "CondenseNet: An efficient DenseNet using learned group convolutions," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 2752–2761.

[60] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 1026–1034.

Zhiqiang Chen received the B.S. degree from the College of Automation, University of Science and Technology Beijing, Beijing, China, in 2014, and the M.S. degree from the Research Center for Brain-Inspired Intelligence, Institute of Automation, Chinese Academy of Sciences, Beijing, in 2017, where he is currently pursuing the Ph.D. degree.

He is also with the School of Artificial Intelligence, University of Chinese Academy of Sciences, Beijing. His current research interests include machine learning, brain-inspired intelligence, medical image, and computer vision.

Ting-Bing Xu received the B.S. degree in automation from the China University of Petroleum, Qingdao, China, in 2014. He is currently pursuing the Ph.D. degree with the National Laboratory of Pattern Recognition (NLPR), Institute of Automation, Chinese Academy of Sciences, Beijing, China.

He is also with the School of Artificial Intelligence, University of Chinese Academy of Sciences, Beijing. His research interests include deep learning, machine learning, pattern recognition, handwriting recognition, and computer vision.

Changde Du received the B.S. degree in automatic control from the Beijing Information Science and Technology University, Beijing, China, in 2013, and the M.S. degree in data mining from the University of Chinese Academy of Sciences, Beijing, in 2016. He is currently pursuing the Ph.D. degree with the Research Center for Brain-Inspired Intelligence, Institute of Automation, Chinese Academy of Sciences, Beijing.

He is also with the School of Artificial Intelligence, University of Chinese Academy of Sciences, Beijing. His current research interests include machine learning, the Bayesian statistics, brain-inspired intelligence, computational neuroscience, and applications in computer vision.

Mr. Du received the AMD Science & Technology Award in 2016 and the Third Prize of Final Contest of National Collegiate Contest & International Invitational Tournament for Brain-inspired Computing and Application in 2017.

Cheng-Lin Liu (Fellow, IEEE) received the B.S. degree in electronic engineering from Wuhan University, Wuhan, China, in 1989, the M.E. degree in electronic engineering from Beijing Polytechnic University, Beijing, China, in 1992, and the Ph.D. degree in pattern recognition and intelligent control from the Chinese Academy of Science, Beijing, in 1995.

He was a Post-Doctoral Fellow with the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, and later with the Tokyo University of Agriculture and Technology, Fuchu, Japan, from March 1996 to March 1999. From 1999 to 2004, he was a Research Staff Member and later a Senior Researcher with the Central Research Laboratory, Hitachi, Ltd., Tokyo, Japan. Since 2005, he has been a Professor with the National Laboratory of Pattern Recognition (NLPR), Institute of Automation, Chinese Academy of Sciences, where he is currently the Director. He has published over 200 technical articles at prestigious international journals and conferences. His research interests include pattern recognition, image processing, neural networks, machine learning, and, especially, the applications to character recognition and document analysis.

Dr. Liu is also a fellow of the IAPR. He is also an Associate Editor-in-Chief of *Pattern Recognition* and an Associate Editor of *Image and Vision Computing*, the *International Journal on Document Analysis and Recognition*, and *Cognitive Computation*.

Huiguang He (Senior Member, IEEE) received the B.S. and M.S. degrees from Dalian Maritime University (DMU), Dalian, China, in 1994 and 1997, respectively, and the Ph.D. degree (Hons.) in pattern recognition and intelligent systems from the Institute of Automation, Chinese Academy of Sciences (CASIA), Beijing, China, in 2002.

From 1997 to 1999, he was an Associate Lecturer with DMU. From 2003 to 2004, he was a Post-Doctoral Researcher with the University of Rochester, Rochester, NY, USA. From 2014 to 2015, he was a Visiting Professor with the University of North Carolina at Chapel Hill, Chapel Hill, NC, USA. He is currently a Full Professor with the Research for Brain-Inspired Intelligence, National Laboratory of Pattern Recognition, Institute of Automation Chinese Academy of Sciences, Beijing. He is also with the School of Artificial Intelligence, University of Chinese Academy of Sciences, Beijing, and the Center for Excellence in Brain Science and Intelligence Technology, CAS. His research has been supported by several research grants from the National Science Foundation of China. He has authored or coauthored more than 150 peer-reviewed articles. His current research interests include pattern recognition, medical image processing, and brain–computer interfaces (BCIs).

Dr. He was an Excellent Member of the Youth Innovation Promotion Association, CAS, in 2016. He was a recipient of the Excellent Ph.D. dissertation of CAS in 2004, the National Science and Technology Award in 2003 and 2004, the Beijing Science and Technology Award in 2002 and 2003, the K.C. Wong Education Prizes in 2007 and 2009, and the Jia-Xi Lu Young Talent Prize in 2009.

Authorized licensed use limited to: Lanzhou University. Downloaded on December 09,2021 at 13:29:55 UTC from IEEE Xplore. Restrictions apply.