Acceleration-aware Fine-grained Channel Pruning for Deep Neural Networks via Residual Gating

Peer-reviewed author version

Huang, Kai; CHEN, Siang; LI, Bowen; CLAESEN, Luc; Yao, Hao; Chen, Junjian; Jiang, Xiaowen; Liu, Zhili & Xiong, Dongliang (2022) Acceleration-aware Fine-grained Channel Pruning for Deep Neural Networks via Residual Gating. In: IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, 41(6), p. 1902-1915..

DOI: 10.1109/TCAD.2021.3093835
Handle: http://hdl.handle.net/1942/34409

Acceleration-aware Fine-grained Channel Pruning for Deep Neural Networks via Residual Gating

Kai Huang, Siang Chen, Bowen Li, Luc Claesen, Hao Yao, Junjian Chen, Xiaowen Jiang, Zhili Liu, and Dongliang Xiong

***Abstract*—Deep Neural Networks have achieved remarkable advancement in various intelligence tasks. However, the massive computation and storage consumption limit applications on resource-constrained devices. While channel pruning has been widely applied to compress models, it is challenging to reach very deep compressions for such a coarse-grained pruning structure without significant performance degradation. In this article, we propose an acceleration-aware fine-grained channel pruning (AFCP) framework for accelerating neural networks, which optimizes trainable gate parameters by estimating residual errors between pruned and original channels with hardware characteristics. Our fine-grained concept consists of both algorithm and structure levels. Different from existing methods that leverage a pre-defined pruning criterion, AFCP explicitly considers both zero-out and similar criteria for each channel and adaptively selects the suitable one via residual gate parameters. For structure level, AFCP adopts a fine-grained channel pruning strategy for residual neural networks and a decomposition-based structure, which further extends the pruning optimization space. Moreover, instead of using theoretical computation costs such as FLOPs, we propose the hardware predictor that bridges the gap between realistic acceleration and pruning procedure to guide the learning of pruning, which improves the efficiency of model pruning when deployed on accelerators. Extensive evaluation results demonstrate that AFCP outperforms state-of-the-art methods, and achieves a favorable balance between model performance and computation cost.**

***Index Terms*—Deep learning system, model compression and acceleration, pruning, neural networks.**

I. Introduction

DEEP Convolutional neural networks (CNNs) have demonstrated superior performance in computer vision tasks. However, the significant improvement always comes at the expense of massive computation and storage consumption, limiting the deployment of neural networks in embedded systems. Many efforts have been made to reduce CNN model size as well as computational workload while still maintaining high performance. On the one hand, optimization approaches such as pruning [1]–[3], quantization [4], [5] and knowledge distillation [6], [7] attract continuous attention, which produce a better tradeoff between compression ratio and performance. On the other hand, various hardware accelerators such as NVDLA [8] and Eyeriss [9], have been proposed to efficiently process CNN models.

Pruning is a promising way for CNN model compression by identifying and removing redundant weights with a negligible performance drop. Recent researches on neural network pruning can be divided into either non-structured [1], [10], [11] or structured pruning [12]–[14], the former prunes weight independently, resulting in structures that are unfriendly for realistic hardware acceleration, while the latter aims at removing parameters in units of structured weights. Among the structured pruning methods, Channel pruning (a.k.a filter pruning) is a typical strategy that directly eliminates neurons, thus no specialized hardware design is required to support customized structure or extra operations. Despite effectively reducing the floating-point operations (FLOPs) of the network, which serves as the main criterion of computational burdens, channel pruning often suffers from more performance loss due to its coarse-grained pruning structure. Therefore, selecting the optimal criterion that discriminates the saliency of channels accurately is an important step for channel pruning.

In addition to the algorithm-level challenge, it is even more challenging when the pruned structure is taken into consideration. While residual learning [15], [16] makes it effective to train very deep networks, methods for pruning plain networks like VGG [17] and AlexNet [18], however, can not be applied to residual models directly. Traditional residual neural networks assume that layers connected by residual mapping must be pruned in the same pattern as others, which raises the problem of constrained channel pruning. To this end, prior works [19]–[22] adopt constrained pruning ratio on these troublesome layers by either skipping or grouping layers together, which unfortunately further limit the exploration space of channel pruning. Thus, more effective pruning strategy should be assigned for residual neural networks.

Although FLOPs is considered to be the main criterion to measure the effectiveness of channel pruning methods, reduction in FLOPs can not be translated to runtime acceleration directly. Realistic CNN model runtime depends on various resource characteristics such as dataflow, number of Processing Elements (PEs), buffer size, bandwidth, and so on. Only considering the reduction in FLOPs is not efficient for models deployed in real hardware. Therefore, it is difficult but imperative to take all these factors into consideration when applying pruning methods on neural networks.

Inspired by these insights, we propose an acceleration-

Kai Huang, Siang Chen, Bowen Li, Xiaowen Jiang and Dongliang Xiong are with the Institute of VLSI Design, Zhejiang University, Hangzhou, 310030, China (e-mail: huangk@zju.edu.cn, chensiang@zju.edu.cn, libw@zju.edu.cn, xiaowen_jiang@zju.edu.cn, xiongdl@zju.edu.cn). Dongliang Xiong is the Corresponding Author.

Luc Claesen is with the Engineering Technology-Electronics-ICT Department, University of Hasselt, 3590 Diepenbeek, Belgium (e-mail: luc.claesen@uhasselt.be).

Hao Yao and Junjian Chen are with the Digital Grid Research Institute, CSG, Guangzhou, 510670, China (e-mail: yaohao@csg.cn, chenjj4@csg.cn).

Zhili Liu is with the sec-chip, Hangzhou, 310030, China (e-mail: liuzhili@sec-chip.com).

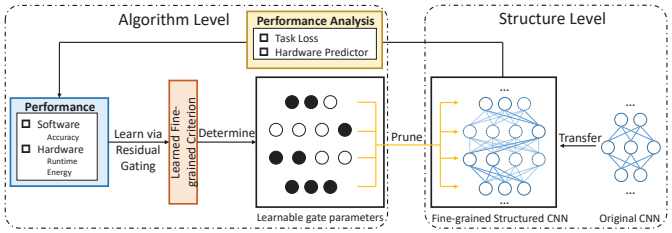


Fig. 1. Our acceleration-aware fine-grained channel pruning framework.

aware fine-grained channel pruning framework that optimizes channel pruning on both algorithm level and structure level, and take advantage of hardware characteristics to improve the efficiency of pruning, as depicted in Fig 1. Specifically, different from existing pruning works that adopt only one heuristic method, we employ trainable gate parameters guided by resource constraints and performance loss, both magnitude-based and similarity-based discrimination criteria are considered, which allows each channel to be zeroed-out or become identical with the most similar channels automatically. Instead of focusing on pruning original convolutional filters, we propose a finer-grained channel pruning structure, which includes decomposing convolutional layers into small tensors as well as allows channels between residual blocks to be pruned in a block-wise manner, thus the learnable gate parameters can be asserted into each channel between layers. Furthermore, state-of-the-art pruning methods only consider the optimization for the target FLOPs ratio, neglecting the real speedup for various platforms. Therefore we propose the acceleration-aware pruning method that given the pruned model structure and target terminal device, a hardware performance predictor is proposed to analyze the hardware acceleration, and further guides the optimization of gate parameters, which improves the pruning efficiency compared to other resource constraints such as FLOPs.

Our contributions are summarized as follows:

1) We propose a learnable fine-grained pruning criterion, which learns each channel to be zeroed-out or identical with others automatically, thereby realizing reasonable tradeoffs between compression ratio and performance.

2) We propose a fine-grained channel pruning structure for residual neural networks, which allows channels between residual blocks to be pruned in a block-wise manner. By performing such a pruning strategy, we can obtain a novel efficient residual network structure, of which connections can fully skip the residual building block.

3) We further propose to decompose convolutional layers into finer-grained tensors, which further extends the optimization of space of pruning.

4) We leverage hardware characteristics to improve the efficiency of pruning, a hardware performance predictor is proposed to bridge the gap between hardware information and pruning procedure, which helps to solve the constrained pruning problem by utilizing widely used optimizers such as stochastic gradient descent (SGD) and Adaptive Moment Estimation (Adam).

II. Related Work

A. Pruning Criterion

Extensive research efforts have been made to evaluate the importance of filters in various ways. Some leverage the static information of weights, the ℓ_1/ℓ_2-norm of weights [1], [23], the average percentage of zero activations (APoZ) [24], geometric median distance [25], and high rank of feature maps [26]. Some induce sparsity constraints into the training of models, Liu et al. [27] apply ℓ_1-norm based regularization on the scaling factors of the batch-normalization (BN) layers and prune channels of which factors are near zero, He et al. [2] propose to select channels based on LASSO regression. Similarly, group sparse regularization is imposed in order to remove zero and near-zero value channels [12], [28]. These works adopt the manually pre-defined criteria or determine saliency scores with heuristic methods, which is sub-optimal and always lead to large performance drop for deep compression ratios. Furthermore, with so many criteria to be selected, it is hard to determine the optimal one for various networks and resource constraints. Instead, the proposed AFCP introduces trainable parameters to learn the appropriate network architectures automatically, making it more convenient and optimal to obtain the pruned model. Some recent works also adopt learnable parameters to guide pruning. Taylor expansion is utilized to estimate the loss of removing filters [22], [29]. Auxiliary parameters are trained to scale single weight or channel by one or zero [30]. Ding et al. [31] introduce centripetal SGD that trains filters to become identical. However, these pruning methods only consider either zero-out or similar-based criterion, and the similar-based pruning can only take the same pruning ratio for each layer, which limits the optimization space of learnable parameters. To solve these problems, the AFCP allows selecting the proper criterion by adopting the residual gating function. He et al. [32] take different feature distributions in each layer into consideration and propose to learn a filter pruning criterion for each layer, which is the most similar to our pruning algorithm. The limitation of [32] is that they still use heuristic methods and only consider different criteria in a layer-wise manner, while the proposed AFCP learns the criterion for each channel adaptively.

B. Pruning Structure

Recent efficient networks introduce complicated structures like identity mapping and dense connection [15], [16], which raises the problem of constrained channel pruning. To avoid the misalignment problem due to the existence of skip connections across layers, some avoid pruning these troublesome layers [19], [20], [33]. Obviously, skipping these channels limits the pruning exploration space. Thus the AFCP takes these channels into consideration and provides a larger search space for pruning. Liu et al. [27] prune pre-activation models by inserting an additional channel selection layer before the first convolution in each residual block. However they only consider input channels for pre-activation models while we allow both input and output channels to be pruned for any residual networks. Lemaire et al. [34] use a mixed block connectivity to avoid redundant computation, which can be

treated as a subset of our method. Recently, a group pruning strategy [21], [22], [31], [35] is proposed to assign those layers into the same group, thus filters in the same group can be pruned simultaneously. Unfortunately, pruning in a group strategy requires that all connections of one pruned filter should be removed simultaneously, of which the constraint is so strong that limits the pruning performance at especially high pruning ratios. Instead, the AFCP allows these troublesome channels to be pruned in a finer-grained block-wise manner, thus larger optimization space is provided while residual information is still retained to obtain high performance.

C. Tensor Decomposition

Tensor decomposition is another way to reduce computational cost in neural network. Specifically, the original tensor can be decomposed into a linear combination of smaller tensors, thus parameters and computations can be significantly reduced. Singular value decompositions (SVD) [36] or CP-decomposition [37] is utilized to approximate the original weight matrix. Jaderberg et al. [38] propose to decompose one filter into two sub-kernels. Astrid et al. [39] further leverage CP decomposition to compress filters into several smaller ones. Decomposition from different tensor dimensions has also been explored to enable more flexible structures [40], [41]. But the problem in these works is that how to determine the optimal rank of the decomposed tensors, which definitely impacts the model performance. Thus we combine channel pruning and decomposition to obtain the proper rank distribution in models. Recently, Hinge [42] apply the pruning method based on decomposed tensors to deal with the limitation on channel pruning for residual neural networks. However, Hinge only applies the one-step decomposition, and we realize that more flexible and fine-grained decomposition structures can be explored to further boost CNN pruning performance.

D. Hardware Acceleration

To accelerate the computation of neural networks, various hardware accelerator architectures have been proposed, including ASIC and FPGA designs. Compared to general-purpose processors (CPU/GPU), these specialized accelerators take the specific computation pattern of neural networks into consideration, and achieve remarkable performance through parallel computation and memory hierarchy optimization. Apart from hardware resources, general accelerators such as NVDLA [8], Eyeriss [9], Shi-dianano [43] mainly differ in the dataflow mapping strategy, which reflects the schedule of data operations. For example, NVDLA adopts a weight-stationary dataflow while Eyeriss employs a row-stationary style.

Recently, hardware/software co-design has been explored for its efficiency. Jiang et al. [44] propose to search architectures based on reinforcement learning which involves the reward of FPGA performance. Hao et al. [45] explore the DNN architecture based on basic blocks. Ren et al. [46] perform hardware-aware model compression by iterative binary search.These works still fail to fuse the hardware feedback into the training of networks by gradient descent, and can be time-consuming for various models and platforms. As far as we know, we are the first to consider optimizing channel pruning and hardware implementation simultaneously.

III. Motivation

A. Fine-grained Pruning Criterion

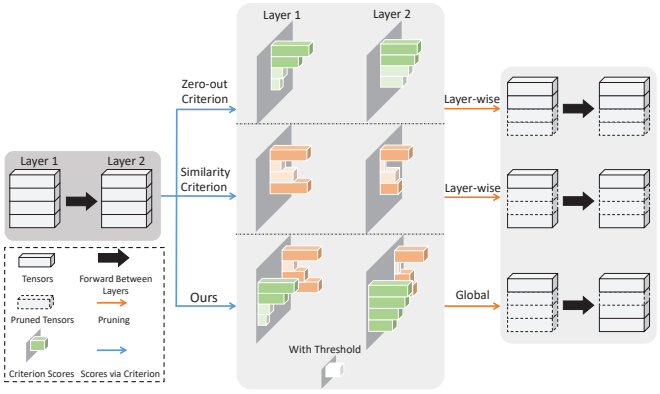


Fig. 2. Prune tensors via criterion scores in two consecutive layers, tensors with smaller scores are pruned out. Zero-out and similar-based criteria always have different scores and pruning results, which are complementary to each other. Here we use tensors rather than filters since the situation also applies to any structured patterns.

We observe that the discrimination criteria always fall into two categories: zero-out based and similarity-based. Zeroing filters out produces sparsity directly. However, while the "smaller-norm-less-important" is widely used in channel pruning, it remains a question whether this criterion is optimal [47]. Furthermore, learning a filter to be zero or not is a hard decision, it is even more difficult to recover the performance loss when the pruning ratio becomes large.

Some other prior works score channels via representative election and remove the most similar channels [31], [48], since similar or identical filters can be merged to one by simply adding up the corresponding input channel parameters of the next layer. However, while the "global pruning ratio" has been proved to be more effective than a fixed per-layer ratio, determining a layer-specific pruning ratio has rarely been investigated for such cases since it is hard to measure the similarity of filters across layers.

As depicted in Fig 2, pruning results of the two criteria can vary from each other. The remaining filter that is hard to be zeroed-out can be easily transformed into similar filters with less performance loss, and vice versa. Therefore, only consider one criterion can be sub-optimal for finding redundant channels. To this end, we propose a fine-grained criterion that incorporates these two discrimination factors and adaptively select the optimal criterion for each channel.

B. Fine-grained Channel Pruning Structure

Considering two consecutive blocks in ResNet, as shown in Fig 3(a). To solve the vanishing gradient problem that prevents neural networks from becoming deeper to demonstrate higher performance, the element-wise addition operation is proposed to connect feature maps between two residual blocks:

$$y_l = h(x_l) + F(x_l, W_l), \tag{1}$$

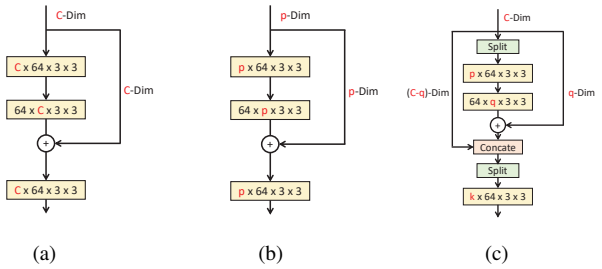


Fig. 3. An illustration of **(a)**: baseline structure. **(b)**: structure pruned by group strategy. **(c)**: structure pruned by our fine-grained strategy. The red letter denotes the channel number. For simplicity, pruning of channels inner residual blocks is not considered here.

$$x_{l+1} = f(y_l) \tag{2}$$

Here x_l denotes the input feature map of l-th Residual Unit. W_l is the weight of the Residual Unit, $F(\cdot)$ is the residual function. The function $f(\cdot)$ denotes the non-linear operation, and the function $h(\cdot)$ is the identity mapping. Therefore, traditional Residual networks assume that layers connected by residual mapping have the same pattern as others.

Unfortunately, as shown in Fig 3(b), pruning in a group technique leads to such a case that all corresponding connections of one eliminated filter should be removed simultaneously, limiting the pruning performance at especially high pruning ratios. And importance scores for these filters in the same group are accumulated together, which makes them harder to be pruned, thus always results in dense connections between residual blocks and very few connections inner residual blocks.

To this end, we allow the residual pattern to be more flexible as shown in Fig 3(c). In our fine-grained channel pruning strategy, channels between residual layers can be removed in a block-wise manner while the residual connections are still retained, which enables individual connections to fully skip the residual building block, and provides a larger search space for important filter selections.

C. Flops vs Runtime

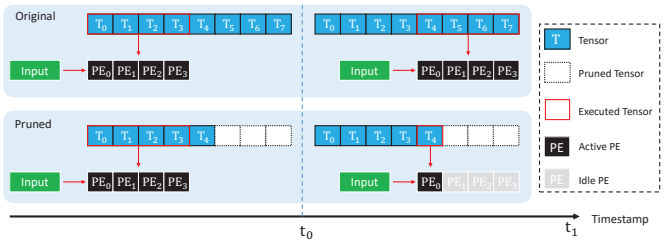


Fig. 4. The motivation behind the acceleration-aware channel pruning. Despite the reduction in FLOPs, the pruned model takes the same execution time as the original model.

Although existing channel pruning methods all focus on achieving a better tradeoff between FLOPs and performance, this theoretical computational cost elimination cannot guarantee real speedup when the pruned CNN is deployed on hardware accelerators. Fig 4 shows an example weight-stationary dataflow run on four PEs. The original model with 8 tensors needs to take t_1 times to complete all multiply-accumulate operations (MACs). For the pruned model, if the theoretical FLOPs of convolution drops from 8 tensors to 5 tensors, the real runtime, however, is still the same as the original model since PEs run in parallel, and three of four PEs are idle during t_0 - t_1 which means a low utilization of hardware resources.

In addition to the impact of dataflow mapping, number of PEs, latency of DRAM/buffer access and data read/write bandwidth all significantly affect the execution performance of DNN models. In this paper, we take all this information into account when conducting pruning, and provide a more hardware-friendly neural network architecture.

IV. Proposed Methods

A. Learn Fine-grained Pruning Criterion via Residual Gating

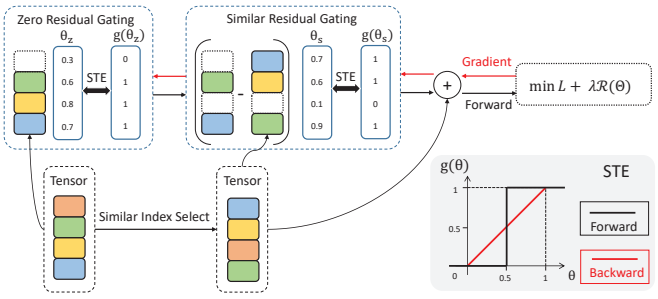


Fig. 5. The training process of the learned fine-grained pruning criterion. Rectangles with different colors and dotted lines denote different channels and pruned channels (tensors), respectively. The similar indices depend on the results of the K-means cluster, tensors after selection are the corresponding similarity centers.

The overall learning process is illustrated in Fig 5. For each convolutional layer, we first divide tensors into clusters, each cluster contains several similar tensors. We generate the clusters by K-means [49], which partitions data into clusters by minimizing the distance between each data point and the center of the cluster. Since the convolution weight parameters are 4-D tensors, we fix the target prunable dimension and flatten other three dimensions for K-means clustering.

As stated in section III-A, identical tensors can be merged into one without any performance loss. To allow tensors in the same cluster to be identical, we define the most representative tensor that has the minimum distance between the center of the cluster as the similarity center τ_{sc}, and record the center index of each tensor after clustering. Here we leverage τ to denote the target prunable tensor.

There are two possible gating ways in our method, tensors gated by either way can be removed. We define these two similar gating and zero gating functions as follows, respectively:

$$sg(\tau, \theta_s) = \tau_{sc} + (\tau - \tau_{sc}) \cdot g(\theta_s) \tag{3}$$

$$zg(\tau, \theta_z) = \tau \cdot g(\theta_z) \tag{4}$$

Where the binary gating function can be expressed as:

$$g(\theta) = \begin{cases} 0, & \theta < 0.5 \\ 1, & otherwise \end{cases} \tag{5}$$

Here $\theta \in [0,1]$ is a trainable parameter for each tensor (channel), and is initialized to 1. In (3), θ_s is used to measure the residual error between tensor and the corresponding similarity center, and θ_z in (4) measures the residual error between tensor and zero. θ moves under threshold which is 0.5 in our setting when the residual error is small enough.

Combined with (3) and (4), we have the final gated tensor after the two-step residual gating:

$$\tau_g = zg(\tau, \theta_z) + sg(zg(\tau, \theta_s), \theta_s) \tag{6}$$

Specifically, when g(θ_s) is zero, it means the residual error between the tensor and the corresponding similarity center is small enough considering the sparsity requirement and model performance, then the tensor is deactivated and the corresponding similarity center tensor will substitute for the forward propagation. Similarly, When g(θ_z) becomes zero, the tensor is replaced by zero and can be removed directly. There are three main advantages via this two-step residual gating. Firstly, instead of estimating similarity scores based on static information, our similar criterion is learned for each tensor and allows a different pruning ratio across layers. Secondly, each tensor has the chance to be removed by either zeroing-out or becoming identical to others, depending on which residual error is smaller. Finally, each pruned tensor that is estimated to be important again can be recovered back as long as the gate parameter is updated over threshold 0.5, this soft pruning can maintain the model capacity compared to methods that directly delete the pruned channels.

For a L-layer network, we aim to find the proper $\Theta = (\theta_1,...,\theta_L)$ that guide the pruning ratio and criterion for each layer and each channel, where $\theta_l = \{\ \theta_{s,l},\ \theta_{z,l}\ \}$, $\theta_{s/z,l} = [\theta_{(s/z,1)},...,\theta_{(s/z,C_l)}\ ]$ is the vector containing gate values in l-th layer. Therefore, given a dataset $D = \{X, Y\}$, where X denote the input vectors and Y are the corresponding labels, we focus on minimizing the loss function under the sparsity constraint on channels:

$$\underset{\Theta,T}{min}\mathcal{L} = \underset{\Theta,T}{min}L(\mathcal{F}(X;T,\Theta),Y) + \lambda\mathcal{R}_{prune}(\Theta) \tag{7}$$

$L(\mathcal{F}(X;T,\Theta),Y)$ denotes the standard loss function (e.g., cross-entropy loss) based on residual gating forward of each tensor as (6). $\mathcal{R}_{prune}(\Theta)$ is the resource-aware regularization function to induce sparsity. λ is a hyperparameter to control the balance between these two losses. We first set FLOPs as our desired computation loss, and further optimization for hardware-specific regularization function will be discussed in section IV-D. FLOPs can be calculated as:

$$F_{comp} = (FLOPs) = \sum_{l=1}^{L}(r_l \cdot s_l \cdot \sum_{c_l=1}^{C_l}(g(\theta_{s,c_l}) \cdot g(\theta_{z,c_l}))\cdot \sum_{c_{l-1}=1}^{C_{l-1}}(g(\theta_{s,c_{l-1}}) \cdot g(\theta_{z,c_{l-1}})) \cdot w_l \cdot h_l) \tag{8}$$

Where r_l and s_l are kernel sizes, w_l and h_l are width and height of output feature maps. For simplicity, here we only show the calculation of normal convolutions which occupy most of the computation cost in CNNs, and the cost of other types of layers can be represented by θ in the same way. Then we can formulate the regularization loss based on target compression ratio p:

$$\mathcal{R}_{prune}(\Theta) = log(|F_{comp} - F_{comporg} \cdot p| + 1) \tag{9}$$

$F_{comporg}$ denotes the total FLOPs of the original model, we select a logarithms form since we want the regularization term to decrease fast in the early stage of pruning, then slow down and always optimize near the target ratio.

So far, we have formulated the forward propagation function for training the pruned model, and one question that remains is how to learn θ via back-propagation. Since the binary gating function $g(\theta)$ is non-smooth and non-differentiable, we utilize the straight through estimator (STE) [50] that is used in network quantization to enable gradient calculation:

$$\frac{\partial\mathcal{L}}{\partial\theta} = \frac{\partial\mathcal{L}}{\partial g(\theta)} \tag{10}$$

B. Fine-grained Pruning For Residual Neural Network

Next we will discuss the proposed fine-grained structure for channel pruning. The biggest difference between our pruned residual neural network and existing methods is that we consider the problem of pruning from the perspective of gating feature maps, the gate function is induced on each channel while skip connections are always retained. One simple strategy is to prune each channel independently without any constraint among channels or layers, thus a better trade-off between compression ratio and performance should be achieved due to the larger search space of channels than strategies such as group pruning [21], [22] or skipping [19], [20]. However, too much freedom leads to irregular distributions of pruned channels between residual blocks, and during experiments we found the advantage of such strategy over group pruning or skipping is not significant as expected especially for more complex models and tasks. We believe that the irregular remained channels make it more difficult to learn the optimal network. Furthermore, input channels and output channels of each block should be selected according to the pruning indices, which increases hardware cost. Therefore, we propose the fine-grained block-wise pruning strategy to enlarge the search space of channels that allowed to be pruned while still maintaining a regular residual structure.

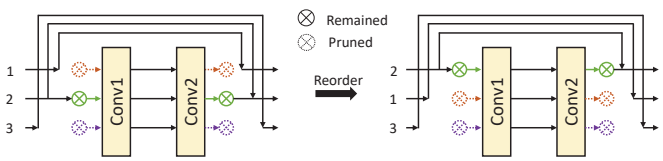


Fig. 6. The pruned and reorderd example structure intra-block. The gate functions in the same color share the same gate parameters. The dotted lines denote pruned channels.

For input channels and output channels of the residual block, the gate function is shared for channels connected by the same skip connection. As depicted in Fig 6, the gate function in the same color share the same gate parameters. Both input channels and output channels of channel 1 and channel 3

are removed while the corresponding skip connections are still retained. The remained skip connections will fully bypass through this block and flow into the next block to produce the residual information. After pruning, channels can be reordered so that the remained channels are always continuous, thus only a simple split and concate operation is needed for input and output channels, respectively.

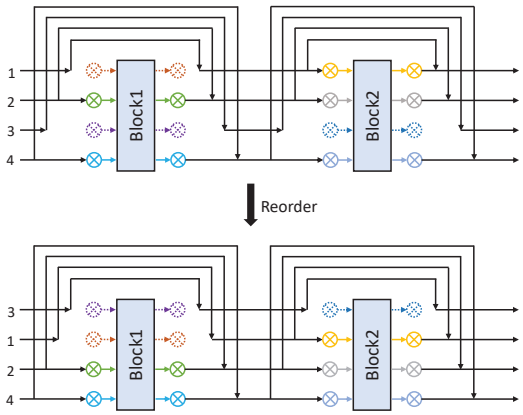


Fig. 7. The pruned and reorderd example structure inter-block.

For channels between blocks, we apply a partial synchronization strategy to guarantee that each block can be reordered to a regular structure. Specifically, the channel of current block is allowed to be pruned only when the corresponding channel has been pruned in the previous block. As shown in Fig 7, when channel 1 and channel 3 in Block 1 are pruned, only chanenl 1 and channel 3 are allowed to be pruned in Block 2. In this way, all blocks can always be reordered so that the remained channels are continuous no matter how the pruned channel distributes in each block. There is a special case when the skip connection contains a convolution instead of being a pure connection (when channel number increases), the input channels of such block are still partially synchronized with the previous block while the output channels are not limited. Note that the synchronization can also be performed in reverse order, but we observe that channels begin to be pruned from shallow layers during experiments, thus we perform the sequential synchronization strategy in this work. To produce such structure during training, the gating function of the input and output channel in b-th block is denoted as:

$$g_b = g_{b-1} + g(\theta) \cdot (1 - g_{b-1}) \tag{11}$$

C. Fine-grained Decomposition for Rank Pruning

To further produce a fine-grained structure for pruning, decomposition is applied to approximate the original filters by low-rank matrices, which includes three stages in this work. Assume that a convolution layer has c input channels and n output channels, with kernel size of $r \times s$. In the first stage, parameters of a convolutional layer W $\in \Re^{crs \times n}$ is decomposed into:

$$W \approx A \cdot B \tag{12}$$

Where $A \in \Re^{crs \times d_1}$, $B \in \Re^{d_1 \times n}$, d_1 denotes the rank size. SVD or Principal Component Analysis (PCA) can be used to solve this problem. While matrix A can be considered as a convolutional layer with d_1 filters of size $r \times s$ acting on c input channels, matrix B is regarded as a 4-D convolution parameter of size $d_1 \times 1 \times 1 \times n$.

In the second stage, We further operate decomposition on matrix A along spatial dimension.

$$A \approx A_1 \cdot A_2 \tag{13}$$

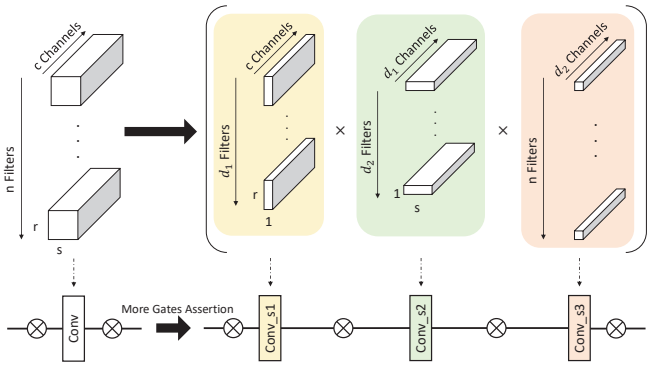


Fig. 8. The original convolutional layer is decomposed into three sequential ones, which produces more dimensions for pruning.

Where $A_1 \in \Re^{cr \times d_2}$, $A_2 \in \Re^{d_2 \times sd_1}$. Similarly, matrix A_1 is considered as a convolutional layer with d_2 filters of size $r \times 1$ acting on c input channels, matrix A_2 is regarded as a 4-D convolution parameter of size $d_2 \times 1 \times s \times d_1$. Therefore, as shown in Fig 8, the original convolutional layer is decomposed into three sequential ones: a horizontal basis layer followed by a vertical basis layer and a linear combination layer.

Take one convolutional layer as an example, the compression rate of parameters after two stages is:

$$\begin{aligned} ratio_{comp} &= \frac{d_2 \cdot c \cdot r + d_1 \cdot d_2 \cdot s + n \cdot d_1}{n \cdot c \cdot r \cdot s} \\ &= \frac{d_2}{n \cdot s} + \frac{d_2 \cdot d_1}{n \cdot c \cdot r} + \frac{d_2}{c \cdot r \cdot s} \end{aligned} \tag{14}$$

A problem that now emerges is how to decide the size of ranks. Given a compression ratio p, traditional decomposition methods determine the rank configuration manually. Obviously, the significance of each decomposed matrix is different, and share the same rank compression ratio is not optimal. Therefore, model size and computation cost are not reduced during our fine-grained decomposition, rank configuration of each layer is determined by maintaining the original convolution size in our setting:

$$d_1 = round(\frac{n \cdot c \cdot r \cdot s}{c \cdot r \cdot s + n}) \tag{15}$$

$$d_2 = round(\frac{d_1 \cdot c \cdot r \cdot s}{c \cdot r + s \cdot d_1}) \tag{16}$$

Although the decomposed matrices are attained by minimizing the reconstruction error of the original filters, this optimization is data-independent and can not guarantee the most optimized parameters for model performance. In the last stage of our fine-grained decomposition, we aim at optimizing model performance via fine-tuning:

$$\min_{\tau} \mathcal{L} = \min_{\tau} L(\mathcal{F}(X; T_D), Y) \tag{17}$$

Where T_D denotes parameters of tensors after decomposition. The model after fine-tuning is employed as the pre-trained model for pruning, which provides two additional dimensions (d_1 and d_2) that allowed to be pruned for each convolutional layer compared to the model before decomposition.

Note that tensor A_1 or A_2 can always be further decomposed with one of the tensors to be size of $d_i \times 1 \times 1 \times d_o$, where d_i and d_o are the induced approximated ranks, and this kind of decomposition can be performed forever. However, the error between the decomposed tensors and original weights will become larger as the decomposition process continues. Thus, there is a trade-off between the number of decompositions and the pruning performance. Comparison evaluations are conducted in our experiments to prove that the proposed two-step decomposition achieves a best trade-off between accuracy and compression ratio.

D. Acceleration-aware Channel Pruning

One of the advantages of our pruning method is that the criterion can be learned end-to-end and updated efficiently by gradient descent, which requires the target resource constraint function in (7) being differentiable to gate parameters. A problem that arises is how to formulate a differentiable function with respect to network architectures and accelerator performance. Fortunately, many DNN analysis frameworks have been studied to model the performance of different accelerators [51], [52]. However, it is still challenging to fuse such complex frameworks into the training process of neural networks. Therefore, we propose a hardware performance predictor to bridge the gap between architecture and hardware performance, and feed it to the pruning process to select the best fit.

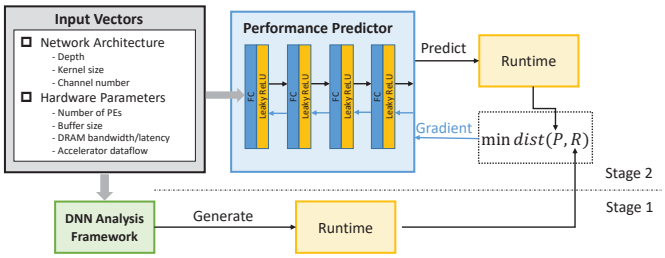


Fig. 9. The training process of hardware performance predictor.

As shown in Fig 9, the training of the predictor includes two stages. In the first stage, we leverage DNN analysis frameworks to generate dataset $D_p = \{X_p, Y_p\}$, where X_p denotes the input vector of a convolutional layer, which contains both network architecture and hardware parameters, the former includes network depth, kernel size, stride and channel number of each convolution, as well as feature maps size. Since the execution time of CNNs is calculated by adding up the time of each layer, compared to covering the possible pruned structures for all layers in the entire model, the cost of generating possible structures of one convolutional layer is much smaller. The hardware parameters are composed of the number of PEs, buffer size, DRAM bandwidth/latency and dataflow mapping. Y_p is the corresponding output which can be hardware-specific runtime, energy, throughput, and so on, we mainly focus on optimizing runtime in this work.

In the second stage, given the dataset generated in stage 1, we construct $f(\cdot)$ as a continuous and differentiable neural network parameterized by W_p to fit the function of the DNN analysis framework, the objective can be formalized as:

$$\arg\min_{w_p}(\mathcal{L}(f(X_p; W_p), Y_p)) \tag{18}$$

Where the loss function $\mathcal{L}(\cdot)$ can be a typical regression loss such as L1/L2 or MSE. The goal is to find an accurate mapping from network architecture and hardware parameters to hardware performance.

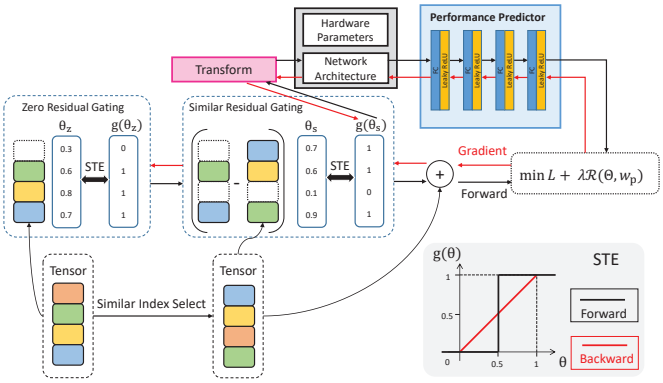


Fig. 10. The overall training process of acceleration-aware pruning. Parameters of the hardware performance predictor are fixed during training, gradients for θ from computation cost loss propagate through the predictor.

After the training process, parameters of the predictor are fixed, and the regularization function in (7) is substituted by the output value of the predictor as shown in Fig 10:

$$F_{comp} = f(M(\Theta); W_p) \tag{19}$$

$M(\Theta)$ denotes the input vector of predictor, which can be transformed by the neural network architecture with trainable parameters Θ. Specifically, for a L-layer neural network, $M(\Theta)$ consists of L sequential architecture mappings by concatenation:

$$M(\Theta) = (m(\theta_1), ..., m(\theta_L)) \tag{20}$$

Take a convolutional layer as an example, $m(\theta_l)$ can be expressed as:

$$\begin{aligned} m(\theta_l) = [&\sum_{c_{l-1}=1}^{C_{l-1}} (g(\theta_{s,c_{l-1}}) \cdot g(\theta_{z,c_{l-1}})), \\ &\sum_{c_l=1}^{C_l} (g(\theta_{s,c_l}) \cdot g(\theta_{z,c_l})), r_l, s_l, w_l, h_l, d_l] \end{aligned} \tag{21}$$

Where C_l is the channel number of l-th layer, r_l and s_l are the kernel size, w_l and h_l are feature map size, d_l is the stride step. The first two vectors denote the remained input channels and output channels respectively. It is worth noting that the order of vectors depends on the definition of the predictor. By feeding input vectors of each layer into the predictor, the total

execution time can be approximated by adding up the output vectors. Here we only list the formulation of the network architecture vector, and hardware parameters can be either concatenated with $M(\Theta)$ in one-hot encoding or as different settings for selecting a hardware-specific predictor.

E. Implementation Details

Algorithm 1 Learning process of the proposed pruning method.

Input: training data X, validation Y; pre-trained model M of layer L with weight parameters T; trainable gate parameters Θ; hardware performance predictor model P;
Output: Compact model;

```
//Step1-Decomposition
for l = 1 to L do
    Apply decomposition as described in IV-C with parameters T_D
end for
for epoch = 1 to epoch_D do
    for each (x, y) in (X, Y) do
        forward and update (T, T_D)
    end for
end for
//Step2-Learning criterion
for epoch = 1 to epoch_L do
    for each (x, y) in (X, Y) do
        forward based on Equation 6 with regularization loss
        defined in Equation (9) or (19), update (T, T_D, Θ)
    end for
end for
//Step3-Pruning and Fine-tuning
for l = 1 to L do
    for c = 1 to C_l do
        if (θ_s<0.5) or (θ_z<0.5) then
            remove channel c
        end if
    end for
end for
for epoch = 1 to epoch_F do
    for each (x, y) in (X, Y) do
        forward and and update (T, T_D)
    end for
end for
Return final pruned model
```

In this section, we describe the steps involved in the proposed pruning method, as presented in Algorithm 1. Given a pre-trained model, we first apply decomposition on each convolution except the layer with kernel size 1x1, since this kind of kernel is compact enough and further decomposition will only result in a linear combination of convolutions with the same structure, which on the contrary introduces more redundancies. Fine-tuning is needed to recover the performance loss after decomposition. Secondly, we train the model and gate parameters based on (6) to learn the pruning criterion. The predictor will be used in the regularization loss if the compression target is real hardware performance. Due to the induced sparsity constraint in the training objective, the computation cost of the pruned model will be always near the target ratio, and models with the best validation performance will be selected. Lastly, those channels of which gate parameters are below threshold 0.5 will be removed following the pruning strategy in section IV-B, and fine-tuning is again employed to get the final compact model.

V. Experiments

A. Experimental Setup

Evaluation Platforms. The model training and pruning method are implemented by the deep learning framework PyTorch [53]. For evaluating hardware performance, we leverage a cycle-accurate DNN accelerator simulator MAESTRO [52], which can analyze the performance for various existing accelerator designs such as NVDLA [8], Eyeriss [9], Shidiannao [43] and so on. We follow the default resource parameter settings in MAESTRO of which the results are highly consistent with original designs. And with the help of mapping optimizers like GAMMA [54], more flexible and optimal designs can be explored.

Benchmark Datasets. We evaluate the proposed method on three standard image classification benchmarks: CIFAR-10, CIFAR-100 [55] and ImageNet (ILSVRC-2012) [56]. CIFAR-10 contains 50000 training images and 10000 testing images of size 32×32, which are categorized into 10 different classes. CIFAR-100 is similar to CIFAR-10 but has 100 classes. ImageNet contains 1.28 million training images and 50k validation images of 1000 classes.

Neural Network Models. While ResNet models show state-of-the-art performance with efficient architectures, previous works claim that it is more difficult to compress such structures. Therefore, besides plain neural networks like VGG, we mainly focus on pruning the challenging ResNet. Both shallow (ResNet-18,20) and deep (ResNet-50,56,110,164) versions are evaluated on different datasets.

When real hardware performance is the target resource constraint, we adopt a 4-layer fully-connected network as the performance predictor, with hidden layer of width 200, 100, 80 and each followed by a Leaky ReLU activation function. It is worth noticing that the predictor will only be used when training and pruning, and no additional computation cost is required for model inference.

Optimization Settings. For training baseline models on CIFAR-10 and CIFAR-100, we follow the parameter settings in [42]. Models are trained for 160 epochs using SGD optimization with 0.9 momentum and 10^{-4} weight decay, batch size is 64, initial learning rate is set to 0.1, and is multiplied by 0.1 at 50% and 75% epochs. For ImageNet, baselines are downloaded from the pretrained PyTorch model [57].

When performing criterion learning, trainable gate parameters are updated via Adam optimization with initial learning rate set to 10^{-3} while weight parameters are still optimized by SGD with initial learning rate 0.01. The learning rate scheduler of both gate parameters and weight parameters are multiplied by 0.1 at 50% and 75% epochs, training epochs are set to 80 and 50 for CIFAR-10/100 and ImageNet respectively. Given

TABLE I
COMPARISON OF PRUNING RESULTS ON CIFAR-10. THE "PARAMS" REPRESENTS THE NUMBER OF PARAMETERS. THE "↓" DENOTES THE DROP BETWEEN BASELINE AND THE PRUNED MODEL. A NEGATIVE VALUE IN "ACC ↓" INDICATES AN IMPROVED MODEL ACCURACY OVER THE BASELINE MODEL. THE "-" DENOTES THAT RESULTS ARE NOT REPORTED IN ORIGINAL PAPERS. THE AFCP-40 DENOTES THAT THE TARGET FLOPS COMPRESSION RATIO IS SET TO 0.4. OTHER TABLES AND FIGURES FOLLOW THE SAME CONVENTION.

Model	Method	Baseline Acc (%)	Pruned Acc (%)	Acc ↓ (%)	Params ↓ (%)	FLOPs ↓ (%)
ResNet-20	SFP [20]	92.20	90.83	1.37	-	42.20
	FPGM [25]	92.20	90.44	1.76	-	54.00
	GBN [22]	92.07	91.39	0.68	36.08	44.53
	Hinge [42]	92.54	91.84	0.70	55.45	54.50
	AFCP-40	92.44	92.41	**0.03**	**49.55**	**60.02**
	AFCP-30	92.44	92.19	**0.25**	**59.75**	**70.05**
ResNet-56	SFP [20]	93.59	92.26	1.33	-	52.60
	FPGM [25]	93.59	93.49	0.10	-	52.60
	LFPC [32]	93.59	93.24	0.35	-	52.90
	GBN [22]	93.10	93.07	0.03	66.70	70.30
	HRank [26]	93.26	90.72	2.44	68.10	74.10
	Hinge [42]	92.95	92.65	0.30	79.20	76.00
	AFCP-30	93.56	93.96	**-0.40**	59.24	**70.01**
	AFCP-20	93.56	93.83	**-0.27**	76.88	**79.89**
ResNet-110	SFP [20]	93.68	93.38	0.30	-	40.80
	FPGM [25]	93.68	93.74	-0.16	-	52.30
	LFPC [32]	93.68	93.07	0.61	-	60.30
	C-SGD [31]	94.38	94.41	-0.03	-	60.89
	HRank [26]	93.50	92.65	0.95	68.70	68.60
	AFCP-30	93.88	94.12	**-0.24**	61.9	**70.00**
	AFCP-20	93.88	94.06	**-0.18**	**74.00**	**80.01**
VGG-16	Hinge [42]	94.02	93.59	0.43	80.05	39.07
	HRank [26]	93.96	92.34	1.62	82.10	65.30
	AFCP-15	93.89	94.07	**-0.18**	**93.16**	**84.99**
	AFCP-05	93.89	92.30	**1.59**	**97.04**	**95.01**

the compression ratio p, the K-means cluster ratio for each tensor is set to $(p + 0.1)$. λ in (6) is set to 4. And all other parameters are the same as training baseline models.

Fine-tuning for models after decomposition and pruning follows the same setting as criterion learning, except that gate parameters are frozen during fine-tuning.

For each hardware platform, the predictor is trained for 50 epochs with a batch size of 256 updated via the L1 loss function, parameters are updated using the Adam optimizer with initial learning rate set to 10^{-2} and multiplied by 0.1 at 15, 30 and 45 respectively. For generating datasets that are used to train the predictor, we traverse all possible convolution structures (input vector) in the pruned model and feed these configurations to MAESTRO to get the corresponding accelerator runtime performance (output vector). It takes about 4 hours to obtain the training dataset for a specific platform. To guarantee fast convergence, data vectors are linearly normalized to [0,1] before training.

B. Evaluation

In this section, we first compare the performance of the pruned model with state-of-the-art methods with respect to FLOPs constraint. Then each step in the proposed method is studied to show the improvement. Pruning performance with the hardware performance predictor is further conducted to demonstrate more efficiency than the FLOPs constraint. Lastly, different settings of hyperparameters are analyzed.

1) Overall Comparison: We compare our proposed AFCP with state-of-the-art channel pruning methods, including SFP [20], FPGM [25], C-SGD [31], GBN [22], HRank [26], Hinge [42] and LFPC [32]. For fairness, we mainly focus on comparing the top-1 accuracy drop between the baseline and the pruned model, since model performance is slightly different due to different settings of hyper-parameters such as weight decay and batch size. All the reference results are directly cited from papers or produced by official public codes. To compare with other pruning methods with various compression ratios, we set different FLOPs budget in (9) to achieve the target FLOPs.

Results on CIFAR-10 dataset are shown in Table I. We first observe that the pruned model can accurately achieve the target compression ratio, which shows that the proposed AFCP is a budge-aware pruning method. Secondly, the AFCP always outperforms other methods on various models. Take ResNet-56 as an example, SFP only optimizes the algorithm level with a zero-out criterion, thus it shows the worst performance. FPGM and HRank employ a more effective pre-defined criterion, and the pruning results are better than SFP, but the improvement is just incremental. LFPC learns the adaptive criterion of SFP and FPGM in a layer-wise manner, thus can achieve higher performance than SFP and FPGM, which indicates that the learned criteria is better than a pre-defined one. GBN learns the zero-out criterion and prunes residual networks in a group-wise strategy, thus outperforms other methods that leverage the skip strategy for residual networks, which shows the effectiveness of the learned criterion and optimization of residual structure. Hinge is the only method that adopts decomposition for enlarging pruning space, and achieves better performance than most of the existing works, which demonstrates the advantage

TABLE II
COMPARISON OF PRUNING RESULTS ON CIFAR-100.

Model	Method	Baseline Acc (%)	Pruned Acc (%)	Acc ↓ (%)	Params ↓ (%)	FLOPs ↓ (%)
ResNet-20	Hinge [42]	68.83	66.34	2.49	66.36	67.06
	AFCP-30	68.24	68.39	**-0.15**	54.15	**70.02**
	AFCP-20	68.24	67.40	**0.84**	64.98	**80.09**
ResNet-56	SFP [20]	71.41	68.79	2.61	-	39.30
	LFPC [32]	71.41	70.83	0.58	-	51.60
	FPGM [25]	71.41	69.66	1.75	-	52.60
	AFCP-45	71.90	73.01	**-1.11**	**22.30**	**54.99**
	AFCP-40	71.90	72.80	**-0.90**	**51.70**	**70.01**
ResNet-164	Hinge [42]	76.78	76.88	-0.10	23.43	44.68
	AFCP-45	77.15	77.97	**-0.82**	**36.18**	**54.98**

TABLE III
COMPARISON OF PRUNING RESULTS ON IMAGENET.

Model	Method	Baseline Acc (%)	Pruned Acc (%)	Acc ↓ (%)	Params ↓ (%)	FLOPs ↓ (%)
ResNet-18	SFP [20]	70.28	67.10	3.18	-	41.80
	FPGM [25]	70.28	68.34	1.94	-	41.80
	AFCP-40	69.76	70.05	**-0.29**	**63.81**	**60.35**
	AFCP-30	69.76	69.99	**-0.23**	**69.99**	**71.92**
ResNet-50	SFP [20]	76.15	74.61	1.54	-	41.80
	HRank [26]	76.15	74.98	1.17	36.66	43.76
	Hinge [42]	76.15	74.70	2.49	-	53.45
	FPGM [25]	76.15	74.83	1.32	-	53.50
	GBN [22]	75.88	75.18	0.67	53.40	55.06
	C-SGD [31]	75.33	74.54	0.79	-	55.76
	LFPC [32]	76.15	74.46	1.69	-	60.80
	AFCP-40	76.13	76.69	**-0.56**	52.45	**60.01**
	AFCP-25	76.13	75.85	**0.28**	**66.56**	**75.01**

of pruning with decomposition. However, the proposed AFCP considers both zero-out and similar criteria adaptively for each channel, and leverages a finer-grained decomposition step as well as a more flexible block-wise pruning manner for residual networks. Therefore, AFCP shows a significant improvement over all other pruning methods.

As shown in Table II, we also evaluate our method on CIFAR-100. Compared to the CIFAR-10 dataset, classification on CIFAR-100 is more challenging due to the larger number of classes, which limits the performance of channel pruning. When achieving a similar FLOPs ratio, models pruned by our method show superior performance than others. For example, while Hinge achieves 67.06% FLOPs reduction with 2.49% accuracy drop on ResNet-20, AFCP shows even 0.15% higher accuracy than the baseline with slightly fewer FLOPs, and achieves 1.65% accuracy improvement over Hinge with even 13.03% FLOPs more reduction. On the pre-activation model ResNet-164, AFCP can prune 10% more FLOPs than Hinge with higher accuracy and the less accuracy drop.

Table III depicts the results on the ImageNet dataset. For ResNet-18, AFCP obviously outperforms SFP and FPGM, a higher compression ratio can be achieved with less accuracy drop. While FPGM compresses ResNet-18 by 41.8% with 1.94% accuracy drop, our method prunes more than 70% FLOPs with even higher accuracy. For ResNet-50, AFCP can also achieve higher accuracy than existing methods with similar computation cost elimination.

The above overall comparisons validate the effectiveness of the proposed pruning framework on various datasets and models. We show that our method can achieve superior performance with comparable computation cost reduction, or reduce more computation cost with similar accuracy degradation.

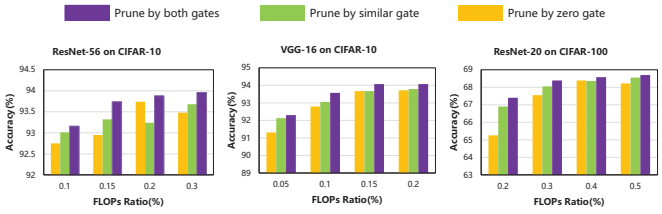


Fig. 11. Comparing the proposed criteron with two basic criteria when employed on ResNet-20, ResNet-56 and VGG-16 models with datasets CFIAR-10/100.

2) Effect of Learned Criterion: One of the reasons for our superior results is that the proposed AFCP adaptively selects a suitable criterion for each channel, Fig 11 demonstrates the effect of this learned criterion. We compare the performance of pruning by zero-out gate parameters, pruning by similar criterion gate parameters and pruning by considering both criteria, three criteria are evaluated under the same FLOPs ratio with same hyper-parameters settings. For different models, pruning ratios and datasets, the fine-grained criterion always outperforms the other two methods, showing that the two basic criteria are complementary to each other. The proposed

similar criterion shows comparable performance with the zero-out criterion, and always performs better at deep compression ratio, which indicates that it is easier and more efficient to make channels become identical than directly remove them. As pruning ratio becomes larger, the performance gap also becomes larger, which validates the effectiveness of our method especially for ultra-low computation cost.

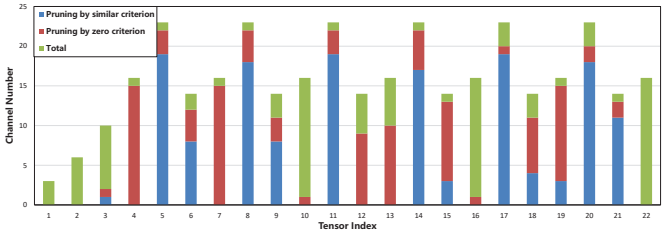


Fig. 12. Visualization of the learned criterion for ResNet-20 on CIFAR-100 with 90% FLOPs pruned. The green strip denotes the remained channels, the blue and orange color denotes channels pruned by the similar criterion and the zero criterion, respectively. When the similar and zero gate parameter values are both below the pruning threshold (0.5), the channel is defined to be pruned by the criterion with a lower value.

Visualization of the learned pruning criterion for ResNet-20 on CIFAR-100 is shown in Fig 12. Obviously, different channels and tensors adopt different criteria, which again validates the benefit of considering both pruning criteria automatically. Interestingly, we observe that deeper layers are totally pruned with only the residual connections remained, which indicates that AFCP not only finds the appropriate channel configurations, but also learns the proper neural network depth under the target budget constraint.

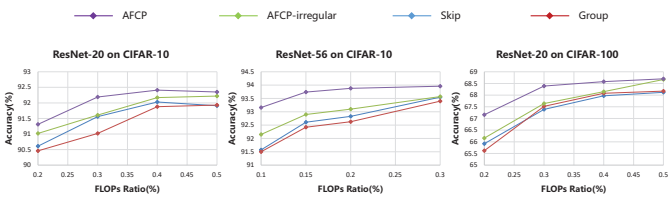


Fig. 13. Comparing the proposed fine-grained structure with group and skip strategies for ResNet-20, ResNet-56 models on datasets CIFAR-10/100.

3) Effect of Fine-grained Structure for Residual Neural Networks: Another advantage of our proposed method results from the fine-grained structure for residual neural networks. Fig 13 shows the comparisons of different channel pruning strategies for ResNet-20, ResNet-56 on CIFAR-10 and CIFAR-100. In the figure,"AFCP-irregular" denotes pruning each channel independently without any constraint, "Group" indicates the strategy that prunes channels connected by pure shortcut connections together, "Skip" denotes pruning only channels inside the residual blocks. For fairness, we employ the same pruning criterion and hyper-parameters for three structure strategies. We find that the two proposed fine-grained structures both achieve higher accuracy at the same compression ratio, which comes from the fact that we consider a larger prunable channel space. AFCP always shows the best performance among these strategies. We explain that although AFCP-irregular provides the largest search space of channels that are allowed to be pruned, the irregular channel distribution fails to fully exploit this advantage. Thus the improvements of AFCP-irregular at some compression ratio are not significant. Interestingly, the Group strategy has a lower accuracy than the Skip strategy in our pruning framework, an explanation is that the constraint that channels in the same group should be pruned simultaneously is too hard that restricts the optimal learning of gate parameters. The superior evaluation results demonstrate the effectiveness of the proposed fine-grained structure for residual neural networks.

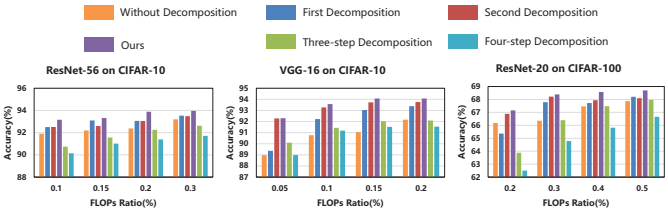


Fig. 14. Comparison of pruning with different decomposition steps.

4) Effect of Decomposition: We further evaluate the benefit of the proposed fine-grained decomposition, as depicted in Fig 14. The "First Decomposition" denotes the step in (12) in section IV-C, which is also used by [42]. And the "Second Decomposition" denotes performing the step in (13) on the original filter independently. The "Three-step Decomposition" and "Four-step Decomposition" denote further decomposing tensor A_1. The proposed fine-grained decomposition-based pruning always shows the best performance for ResNet-20, ResNet-56, VGG on CIFAR10/100. We find that the second step has higher accuracy than the first step in most situations, and these two steps are complementary to each other. For example, for pruning results on ResNet-56, the basic two steps have comparable performance, and the proposed method shows a significant improvement in comparison to others. As the decomposition step further increases, the performance begins to drop due to the larger approximation error between the decomposed tensors and original weights, which makes it hard to recover the loss. Therefore, the proposed two-step decomposition achieves the best trade-off between accuracy and compression ratio.

5) Speed-up in Hardware: In this section, we evaluate the performance of applying hardware-aware pruning instead of considering theoretical computation cost. We compare the realistic acceleration of the pruned model when deployed on various devices including NVDLA [8], Shi-diannao [43], Eyeriss [9] and GAMMA [54]. The decomposition step is not performed when deployed on NVLDA, Shi-diannao and Eyeriss, since these accelerators do not optimize the acceleration for compact filters such as 1×1 and 1×3, and runtime can not be efficiently reduced compared with the original 3×3 filters. GAMMA denotes the adaptive accelerator generated by [54], which supports the acceleration for compact filters. The inference runtime is reported with a batch size of 1.

Table IV depicts the results. While the model accuracy is hard to control, to compare with the pruning with FLOPs constraint, we set the resource budget of training with predictor a little lower than the FLOPs, and to show that better

TABLE IV
COMPARISON OF PRUNING RESNET-20 ON CIFAR-100 WHEN DEPLOYED ON ACCELERATORS. IN "HARDWARE OPT" COLUMN, "✓" AND "×" INDICATES WHETHER TO UTILIZE ACCELERATION-AWARE PRUNING OR NOT, RESPECTIVELY. ACCURACY OF THE ORIGINAL MODEL IS 68.24%.

Device	Baseline Time(cycle)	Hardware Opt	FLOPs↓ (%)	Pruned Time(cycle)	Speedup	Pruned Acc(%)	Acc↓ (%)
NVDLA [8]	86069	×	69.86	51469	1.67×	66.36	1.88
		✓	60.23	**33925**	**2.54**×	**67.94**	**0.30**
		×	79.72	28229	3.05×	66.18	2.06
		✓	73.52	**25646**	**3.36**×	**67.43**	**0.81**
Shi-diannao [43]	42045	×	69.86	19375	2.17×	66.36	1.88
		✓	65.93	**19285**	**2.18**×	**67.62**	**0.62**
		×	79.72	10771	3.90×	66.18	2.06
		✓	74.22	**9222**	**4.56**×	**67.25**	**0.99**
Eyeriss [9]	120171	×	69.86	47385	2.54×	66.36	1.88
		✓	61.32	**44117**	**2.72**×	**67.15**	**1.09**
		×	79.72	30907	3.89×	66.18	2.06
		✓	77.28	**30207**	**3.98**×	**66.55**	**1.69**
GAMMA [54]	30439	×	80.04	10300	2.96×	67.16	1.08
		✓	76.87	**10031**	**3.03**×	**67.40**	**0.84**
		×	90.00	6212	4.90×	63.99	4.25
		✓	87.61	**5961**	**5.11**×	**64.75**	**3.49**

pruning performance should be obtained with the similar or even less execution time, all other hyper-parameters are set to the same values. There are three observations. First, models pruned with the hardware predictor show superior performance in comparison to models pruned by FLOPs constraint with slightly higher acceleration ratio. For example, pruning 20.28% FLOPs without hardware predictor accelerates ResNet-20 by 3.05× with 2.06% accuracy drop, but pruning with predictor achieves 3.36× speedup ratio with only 0.81% accuracy drop. This validates the motivation that even if an optimal pruning strategy is learned for a given FLOPs, it may not be optimal for the corresponding execution time. Therefore, by utilizing the hardware predictor, a better training performance should be obtained under the same execution time constraint, or the hardware cost should be lower with nearly the same training accuracy. Second, as discussed in the above section, there are gaps between the theoretical (FLOPs) and realistic speedup ratios, 20% FLOPs reduction can not translate into 5× speedup, however, the hardware predictor can make pruning more efficient and achieve a similar speedup ratio with fewer FLOPs elimination. Finally, our method demonstrates improvement on different devices and shows effectiveness for pruning both original filters and decomposed tensors. These results demonstrate that with the help of the hardware predictor, our method can produce a more efficient compressed model.

We further perform the hardware-aware pruning on CPU (Intel Xeon E5-2650) and GPU (NVIDIA TITAN Xp) to show the effect on various platforms. The training datasets for the predictor are built by collecting the realistic inference time of convolutional layers with all possible architectures covered in PyTorch. All evaluation results are averaged over 10 measurements. As depicted in Table V, similar to the results on accelerators, pruning with hardware feedback achieves less execution time with higher accuracy on both CPU and GPU. Specifically, pruning with the predictor can reach nearly 20% more speedup over the 30% FLOPs model on CPU, which again validates the benefit of pruning guided by the predictor.

TABLE V
COMPARISON OF PRUNING RESNET-20 ON CIFAR-100 WHEN DEPLOYED ON CPU/GPU. THE NUMBER OF THREADS IN CPU IS SET TO THE DEFAULT VALUE 24. THE INFERENCE RUNTIME IS REPORTED WITH A BATCH SIZE OF 64. ACCURACY OF THE ORIGINAL MODEL IS 68.24%.

Device	Hardware Opt	FLOPs↓ (%)	Pruned Time(ms)	Speedup	Acc↓ (%)
CPU (32.01ms)	×	69.86	22.48	1.42×	1.88
	✓	69.30	**20.62**	**1.55**×	**0.64**
	×	79.72	17.58	1.82×	2.06
	✓	72.16	**11.42**	**2.80**×	**1.07**
GPU (4.03ms)	×	69.86	3.38	1.19×	1.88
	✓	64.20	**2.98**	**1.35**×	**0.98**
	×	79.72	2.58	1.56×	2.06
	✓	76.38	**2.16**	**1.87**×	**1.57**

We also observe that the speedup of GPU is smaller than CPU under the same FLOPs reduction, the reason is that the parallelism level of GPU is higher and the problem of under-utilization is more severe for the pruned model.

6) Ablation Study: There are two hyper-parameters in our pruning method, including K-means cluster ratio for each layer and λ to control the balance between performance loss and computation cost loss, ablation studies are conducted on these two hyper-parameters.

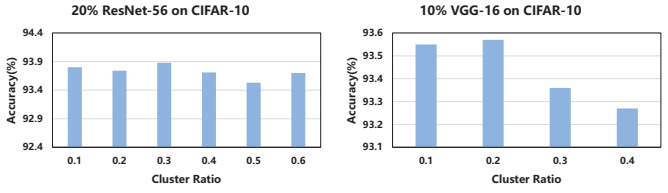


Fig. 15. Sensitivity analysis about K-means cluster ratio for each layer on CIFAR-10.

Effect of different K-means cluster ratio. Fig 15 shows the model accuracy of different K-means cluster ratios for ResNet-56 and VGG-16. Although the final pruned ratio for

each layer is determined automatically by trainable gate parameters, the number of initial clusters is essential for learning. Given a global pruning ratio p for each layer, the remaining computation cost ratio can be simply estimated as $(1-p)^2$, and we find model accuracy reaches the highest with cluster ratio around p for each layer. Therefore, we set the K-means cluster ratio to $(p+0.1)$ in this work.

TABLE VI
SENSITIVITY ANALYSIS ABOUT BALANCING TERM λ FOR THE SPARSITY LOSS ON CIFAR-10.

Model	FLOPs↓ (%)	λ	Pruned Acc (%)
VGG-16	90	2	93.57
		4	93.57
		6	93.59
		8	93.36
ResNet-56	80	2	93.88
		4	93.88
		6	93.81
		8	93.74

Selection of λ. We compare the accuracy of the pruned model with different $\lambda \in \{2, 4, 6, 8\}$, as shown in Table VI. There is nearly no difference in pruned accuracy for different selections under the same target computation cost constraint, which indicates that our pruning algorithm is stable and robust to this hyper-parameter. And we set λ to 4 in this work.

VI. CONCLUSION

In this article, we have proposed AFCP, a novel acceleration-aware fine-grained channel pruning framework for accelerating DNN models on accelerators, which optimizes models from the perspective of both pruning algorithm and structure. Different from existing methods, AFCP explicitly considers both zero-out and similar criteria for each tensor and adaptively selects the suitable one via residual gate parameters. A Fine-grained channel pruning strategy for residual neural networks and decomposition-based structure are proposed to achieve ultra-deep compression. To further improve the efficiency of model pruning when deployed on accelerators, AFCP adopts a hardware predictor to guide pruning learning. Experiments have shown that AFCP outperforms state-of-the-art pruning methods, and demonstrate a favorable balance between classification accuracy and computation cost.

VII. ACKNOWLEDGEMENT

This work is supported by the National Key R&D Program of China (2020YFB0906000, 2020YFB0906001).

REFERENCES

[1] H. Song, P. Jeff, T. John, and D. William, "Learning both weights and connections for efficient neural networks," in *Advances in Neural Information Processing Systems, NeurIPS*, 2015, pp. 1135–1143.

[2] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *IEEE International Conference on Computer Vision, ICCV*, 2017, pp. 1398–1406.

[3] S. Guo, Y. Wang, Q. Li, and J. Yan, "DMCP: differentiable markov channel pruning for neural networks," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR*, 2020, pp. 1536–1544.

[4] S. Zhou, Z. Ni, X. Zhou, H. Wen, Y. Wu, and Y. Zou, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," 2016. [Online]. Available: arXiv:1606.06160

[5] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless cnns with low-precision weights," in *International Conference on Learning Representations, ICLR*, 2017.

[6] J. Ba and R. Caruana, "Do deep nets really need to be deep?" in *Advances in Neural Information Processing Systems, NeurIPS*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds., 2014, pp. 2654–2662.

[7] G. E. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015. [Online]. Available: arXiv:1503.02531

[8] "Nvdla deep learning accelerator," 2017, http://nvdla.org.

[9] Y. Chen, J. S. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *ACM/IEEE Annual International Symposium on Computer Architecture, ISCA*, 2016, pp. 367–379.

[10] A. Ardakani, C. Condo, and W. J. Gross, "Sparsely-connected neural networks: Towards efficient VLSI implementation of deep neural networks," in *International Conference on Learning Representations, ICLR*, 2017.

[11] Y. Lin, S. Han, H. Mao, Y. Wang, and B. Dally, "Deep gradient compression: Reducing the communication bandwidth for distributed training," in *International Conference on Learning Representations, ICLR*, 2018.

[12] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Advances in Neural Information Processing Systems, NeurIPS*, 2016, pp. 2074–2082.

[13] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," in *International Conference on Learning Representations, ICLR*, 2017.

[14] W. Niu, X. Ma, S. Lin, S. Wang, X. Qian, X. Lin, Y. Wang, and B. Ren, "Patdnn: Achieving real-time DNN execution on mobile devices with pattern-based weight pruning," in *Architectural Support for Programming Languages and Operating Systems, ASPLOS*, 2020, pp. 907–922.

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR*, 2016, pp. 770–778.

[16] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR*, 2017, pp. 2261–2269.

[17] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations, ICLR*, 2015.

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[19] J. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression," in *IEEE International Conference on Computer Vision, ICCV*, 2017, pp. 5068–5076.

[20] Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang, "Soft filter pruning for accelerating deep convolutional neural networks," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI*, 2018, pp. 2234–2240.

[21] S. Gao, X. Liu, L. Chien, W. Zhang, and J. M. Alvarez, "VACL: variance-aware cross-layer regularization for pruning deep residual networks," in *IEEE/CVF International Conference on Computer Vision Workshops, ICCV Workshops*, 2019, pp. 2980–2988.

[22] Z. You, K. Yan, J. Ye, M. Ma, and P. Wang, "Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks," in *Advances in Neural Information Processing Systems NeurIPS*, 2019, pp. 2130–2141.

[23] J. Guo, W. Ouyang, and D. Xu, "Multi-dimensional pruning: A unified framework for model compression," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR*, 2020, pp. 1505–1514.

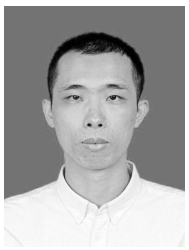
[24] H. Hu, R. Peng, Y. Tai, and C. Tang, "Network trimming: A data-driven neuron pruning approach towards efficient deep architectures," 2016. [Online]. Available: arXiv:1607.03250

[25] Y. He, P. Liu, Z. Wang, Z. Hu, and Y. Yang, "Filter pruning via geometric median for deep convolutional neural networks acceleration," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR*, 2019, pp. 4340–4349.

[26] M. Lin, R. Ji, Y. Wang, Y. Zhang, B. Zhang, Y. Tian, and L. Shao, "Hrank: Filter pruning using high-rank feature map," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR*, 2020, pp. 1526–1535.

[27] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *IEEE International Conference on Computer Vision, ICCV*, 2017, pp. 2755–2763.
[28] J. M. Alvarez and M. Salzmann, "Learning the number of neurons in deep networks," in *Advances in Neural Information Processing Systems, NeurIPS*, 2016, pp. 2262–2270.
[29] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," in *International Conference on Learning Representations, ICLR*, 2017.
[30] X. Xiao, Z. Wang, and S. Rajasekaran, "Autoprune: Automatic network pruning by regularizing auxiliary parameters," in *Advances in Neural Information Processing Systems, NeurIPS*, 2019, pp. 13 681–13 691.
[31] X. Ding, G. Ding, Y. Guo, and J. Han, "Centripetal SGD for pruning very deep convolutional networks with complicated structure," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR*, 2019, pp. 4943–4953.
[32] Y. He, Y. Ding, P. Liu, L. Zhu, H. Zhang, and Y. Yang, "Learning filter pruning criteria for deep convolutional neural networks acceleration," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR*, 2020, pp. 2006–2015.
[33] Z. Chen, Y. Li, S. Bengio, and S. Si, "You look twice: Gaternet for dynamic filter selection in cnns," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR*, 2019, pp. 9172–9180.
[34] C. Lemaire, A. Achkar, and P. Jodoin, "Structured pruning of neural networks with budget-aware regularization," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR*, 2019, pp. 9108–9116.
[35] J. Luo and J. Wu, "Neural network pruning with residual-connections and limited-data," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR*, 2020, pp. 1455–1464.
[36] V. Klema and A. Laub, "The singular value decomposition: Its computation and some applications," *IEEE Transactions on Automatic Control*, vol. 25, no. 2, pp. 164–176, 1980.
[37] V. Lebedev, Y. Ganin, M. Rakhuba, I. V. Oseledets, and V. S. Lempitsky, "Speeding-up convolutional neural networks using fine-tuned cp-decomposition," in *International Conference on Learning Representations, ICLR*, 2015.
[38] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," in *British Machine Vision Conference, BMVC*, 2014.
[39] M. Astrid and S. Lee, "Cp-decomposition with tensor power method for convolutional neural networks compression," in *IEEE International Conference on Big Data and Smart Computing, BigComp*, 2017, pp. 115–118.
[40] X. Zhang, J. Zou, K. He, and J. Sun, "Accelerating very deep convolutional networks for classification and detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 38, no. 10, pp. 1943–1955, 2016.
[41] Y. Li, S. Gu, L. V. Gool, and R. Timofte, "Learning filter basis for convolutional neural network compression," in *IEEE International Conference on Computer Vision, ICCV*, 2019, pp. 5622–5631.
[42] Y. Li, S. Gu, C. Mayer, L. V. Gool, and R. Timofte, "Group sparsity: The hinge between filter pruning and decomposition for network compression," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR*, 2020, pp. 8015–8024.
[43] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "Shidiannao: shifting vision processing closer to the sensor," in *Annual International Symposium on Computer Architecture, ISCA*. ACM, 2015, pp. 92–104.
[44] W. Jiang, X. Zhang, E. H. Sha, L. Yang, Q. Zhuge, Y. Shi, and J. Hu, "Accuracy vs. efficiency: Achieving both through fpga-implementation aware neural architecture search," in *Annual Design Automation Conference, DAC*. ACM, 2019, p. 5.
[45] C. Hao, X. Zhang, Y. Li, S. Huang, J. Xiong, K. Rupnow, W. Hwu, and D. Chen, "FPGA/DNN co-design: An efficient design methodology for iot intelligence on the edge," in *Annual Design Automation Conference, DAC*. ACM, 2019, p. 206.
[46] A. Ren, T. Zhang, S. Ye, J. Li, W. Xu, X. Qian, X. Lin, and Y. Wang, "ADMM-NN: an algorithm-hardware co-design framework of dnns using alternating direction methods of multipliers," in *International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS*. ACM, 2019, pp. 925–938.
[47] J. Ye, X. Lu, Z. Lin, and J. Z. Wang, "Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers," in *International Conference on Learning Representations, ICLR*. OpenReview.net, 2018.
[48] Y. Wang, C. Xu, C. Xu, and D. Tao, "Beyond filters: Compact feature map for portable deep model," in *International Conference on Machine Learning, ICML*, ser. Proceedings of Machine Learning Research, vol. 70. PMLR, 2017, pp. 3703–3711.
[49] J. A. H. A. Wong, "Algorithm as 136: A k-means clustering algorithm," *Journal of the Royal Statistical Society*, vol. 28, no. 1, pp. 100–108, 1979.
[50] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Advances in Neural Information Processing Systems, NeurIPS*, 2016, pp. 4107–4115.
[51] A. Samajdar, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "Scale-sim: Systolic cnn accelerator simulator," 2018. [Online]. Available: arXiv:1811.02883
[52] H. Kwon, P. Chatarasi, M. Pellauer, A. Parashar, V. Sarkar, and T. Krishna, "Understanding reuse, performance, and hardware cost of DNN dataflow: A data-centric approach," in *Annual IEEE/ACM International Symposium on Microarchitecture, MICRO*. ACM, 2019, pp. 754–768.
[53] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems, NeurIPS*, 2019, pp. 8024–8035.
[54] S. Kao and T. Krishna, "GAMMA: automating the HW mapping of DNN models on accelerators via genetic algorithm," in *IEEE/ACM International Conference On Computer Aided Design, ICCAD*. IEEE, 2020, pp. 1–9.
[55] A. Krizhevsky, "Learning multiple layers of features from tiny images," in *Technical report*, 2009.
[56] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and F. Li, "Imagenet: A large-scale hierarchical image database," in *IEEE/CVF Computer Society Conference on Computer Vision and Pattern Recognition CVPR*. IEEE Computer Society, 2009, pp. 248–255.
[57] "Pretrained model zoo," 2020, https://github.com/pytorch/vision/blob/master/torchvision/models.

Kai Huang received the B.S.E.E. degree from Nanchang University, Nanchang, China, in 2002, and the Ph.D. degree in engineering circuit and system from Zhejiang University, Hangzhou, China, in 2008. In 2006, he was a short-term Visitor with the TIMA Laboratory, Grenoble, France. From 2009 to 2011, he was a post-doctoral research assistant with the Institute of VLSI Design, Zhejiang University. In 2010, he also worked as a collaborative expert in VERIMAG Laboratory, Grenoble, France. From 2012 to 2020, he was an associate professor with college of information science & electronic engineering, Zhejiang University. He is currently a full professor and director of the Institute of VLSI design, Zhejiang University. He is also director of Zhejiang University & APEX Joint Laboratory. He has successfully designed over 20 SoC products for different application domains with totally 600 million chips shipped. He holds 12 patents and has published more than 60 papers in international conferences and journals. His current research interests include embedded processors and SoC design methodology.

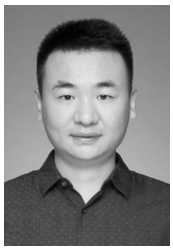


Siang Chen received the B.S degrees from Zhejiang University, Hangzhou, China, in 2016. He is currently a Ph.D student in the institute of VLSI Design, Zhejiang University. His research interests include model compression and hardware acceleration for neural networks and computer vision.

Bowen Li received the B.S degrees from Zhejiang University, Hangzhou, China, in 2016. He is currently a Ph.D student in the institute of VLSI Design, Zhejiang University. His research interests include model compression and hardware acceleration for neural networks.

Luc Claesen obtained his Electronics Engineering degree (1979) and his Ph.D. degree (1984) from KULeuven, Belgium. In 1984 he joined the newly founded IMEC research center in Leuven Belgium with employee number 007. In the period 1984-2007 he was active in IMEC as scientific group leader in the area of VLSI and System-on-Chip design methods. From 1989-2007 he was professor at KULeuven (ESAT) where he has been teaching in VLSI CAD methods and in the Master of Artificial Intelligence Program. He studied the Chinese language at the KULeuven. From 2003 to 2005 he was a professor at National Chiao Tung University (now: National Yang Ming Chiao Tung University) in Hsinchu Taiwan, where he has been teaching on subjects of System-on-Chip design and advanced computer architectures, using the Chinese language. Since 2007 he is at Hasselt University Belgium where he is a professor doing research in SoC architectures for computational multi-camera systems, camera based Deep Learning architectures and applications as well as teaching in these subjects in the joint UHasselt-KULeuven Engineering Department. He is a guest professor at National Chiao Tung University, Taiwan. He published more than 200 scientific papers. In 1991 he obtained the "Laureate of the Royal Academy of Belgium Award in the Class of Sciences", which is an honor awarded to only one scientist a year in Belgium. The R&D results under his guidance was awarded the "Grand European IT Prize" (200.000 Euro) by the European Union; this is until now the only Belgian innovation that was granted with this award.

Hao Yao received the BE dgree from Tianjin University, in 2014.His research interests include smart grid and Electric power chip development.



Junjian Chen received the bachelor's degree in information engineering from University of Electronic Science and Technology of China ,in 2012.Since2012, he has been mainly engaged in the research and development of Power system SoC, embedded system development, SoC product planning, and SoC industrialization applications.

Xiaowen Jiang received the Ph.D. degree from College of Information Science and Electronic Engineering, Zhejiang University, Hangzhou, China, in 2018. He is currently a research assistantat in the Institute of VLSI Design, Zhejiang University, Hangzhou, China. His current research interests include Multi-processor system-on-chip, edge intelligence, safety and reliability design.

Zhili Liu received the Master of Engineering degree in electronics and Communications Engineering, Hangzhou Dianzi University, China in 2017. His current research interests include the application of embedded CPU and high performance low power software exploration.

Dongliang Xiong received the B.S. and Ph.D. degrees from Zhejiang University, Hangzhou, China, in 2012 and 2017, respectively. He is currently a post-docter in the institute of electrical machines and control, Zhejiang University. His research interests include model compression and hardware acceleration for neural networks, heterogeneous system-on-chip design, and computer vision.