An Image Enhancing Pattern-based Sparsity for Real-time Inference on Mobile Devices

Xiaolong Ma[1†], Wei Niu[2†], Tianyun Zhang[3], Sijia Liu[4], Sheng Lin[1], Hongjia Li[1], Wujie Wen[5], Xiang Chen[6], Jian Tang[7], Kaisheng Ma[8], Bin Ren[2], and Yanzhi Wang[1]

[1] Northeastern University, Boston MA 02115, USA
{ma.xiaol, yanz.wang}@northeastern.edu
[2] College of William and Mary, [3] Syracuse University, [4] IBM Research, [5] Lehigh University, [6] George Mason University, [7] DiDi AI Labs, [8] Tsinghua University
[†] Equal Contribution

Abstract. Weight pruning has been widely acknowledged as a straightforward and effective method to eliminate redundancy in Deep Neural Networks (DNN), thereby achieving acceleration on various platforms. However, most of the pruning techniques are essentially trade-offs between model accuracy and regularity which lead to impaired inference accuracy and limited on-device acceleration performance. To solve the problem, we introduce a new sparsity dimension, namely pattern-based sparsity that comprises pattern and connectivity sparsity, and becoming both highly accurate and hardware friendly. With carefully designed patterns, the proposed pruning unprecedentedly and consistently achieves accuracy enhancement and better feature extraction ability on different DNN structures and datasets, and our pattern-aware pruning framework also achieves pattern library extraction, pattern selection, pattern and connectivity pruning and weight training simultaneously. Our approach on the new pattern-based sparsity naturally fits into compiler optimization for highly efficient DNN execution on mobile platforms. To the best of our knowledge, it is the first time that mobile devices achieve real-time inference for the large-scale DNN models thanks to the unique spatial property of pattern-based sparsity and the help of the code generation capability of compilers.

1 Introduction

Weight pruning has been proven to be effective in eliminating redundancy in the original model [7,32,14,24,18,20], therefore accelerating DNN execution on target computing platforms. Non-structured pruning [10] achieves high accuracy, but is limited by its hardware unfriendliness [32,14]. Meanwhile, structured pruning [32] is hardware friendly but suffers from accuracy loss.

It is imperative to seek an approach that can offer, or even go beyond, the best of both types of sparsity. We visualize part of the normalized heat map of a pre-trained model of VGG-16 on ImageNet in Figure 1, we find that (i) the effective area (i.e. weights with higher absolute values) forms some specific shapes

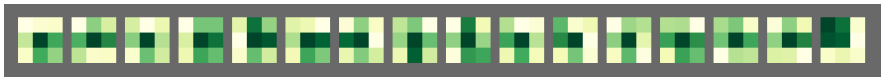

Fig. 1: Heat map of randomly selected convolution kernels in the third convolutional layer of a VGG-16 on ImageNet dataset. The weight values in each kernel are normalized and darker shade represents higher absolute value.

and repeatedly appears in the model, and (ii) some of the entire convolution kernels have very small weight values and make themselves void kernels. Motivated by the two observations, we introduce a new sparsity dimension – *pattern-based sparsity*, which exploits both intra-convolution and inter-convolution kernel sparsities, exhibiting both high accuracy and regularity, and revealing a previously *unknown* point in design space.

In pattern-based sparsity, we call our intra-convolution kernel sparsity *pattern sparsity* and inter-convolution kernel sparsity *connectivity sparsity*. To get pattern sparsity, we prune a fixed number of weights in each convolution kernel, and the remaining weights form specific "kernel patterns". Along this line, we find that some carefully designed kernel patterns have special vision properties that potentially enhance image quality, thereby enhancing feature extraction ability of DNNs. For connectivity sparsity, we cut the relatively unimportant connections between certain input and output channels, which is equivalent to removal of corresponding kernels. At the algorithm level, we design a novel pattern-aware network pruning framework that efficiently achieves pattern pruning and connectivity pruning without degrading accuracy. We begin by reforming the pruning problem into an ADMM optimization problem [4], and then solve the problem iteratively using a Primal-Proximal solution which decoupling the stochastic gradient descent process with regularization, enabling a progressive and gradual process of penalizing unimportant weight groups, meaning a more accurate selection of remaining weight patterns. Therefore, the framework can achieve pattern library extraction, pattern assignment, unimportant connectivity removal, as well as weight training simultaneously. Our proposed pattern-based sparsity is mobile hardware friendly with the help of *code generation* capability of compilers. More specifically, we design the *filter/kernel re-ordering* technique that enables compiler optimizations that maintain instruction-level and thread-level parallelism, and achieves the maximum possible hardware acceleration.

Our contributions of this paper are summarized as follows:

- We design a set of patterns, namely pattern library, and prove the image enhancement property that is related to pattern pruning. (Section 4)
- We form a novel pattern-aware network pruning framework that can extract pattern library, perform pattern and connectivity pruning and weight training at the same time. (Section 5)
- We design the corresponding (algorithm-compiler-hardware) inference framework which fully leverages the new sparsity dimension and achieves real-time DNN execution on mobile devices. (Section 6)

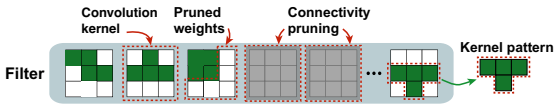

Fig. 2: Illustration of pattern-based sparsity.

Section 7 demonstrates pattern library extraction result, pattern pruning for accuracy and image enhancement results, the overall pattern-based compression results and its acceleration results on mobile devices.

2 Background

DNN model pruning techniques are studied in early work of *non-structured pruning* [10], in which an iterative, heuristic method is used with limited, non-uniform model compression rates. The irregular weight distribution causes irregular memory access and thereby execution overheads, which leads to limited acceleration performance. *Structured pruning* is pioneered by [32][14], in which regular and smaller weight matrices are generated to eliminate overhead of weight indices and achieve higher acceleration in CPU/GPU executions. However, it suffers from notable accuracy drop when the pruning rate increases. *Kernel level pruning* is studied in [5] that the sparse complimentary kernels can save half of the weights and computations, but it is different from our approach because pattern-based sparsity is theoretically and practically improving the software and hardware performance of DNN while [5] only focuses on parameter and computation reduction without discussing on platform acceleration.

Mobile DNN inference frameworks are studied, including TFLite [1], TVM [6], Alibaba MNN [2], DeepCache [33] and DeepSense [34]. These works do not account for model compression techniques, and the performance is far from real-time requirement (usually 30 frames/sec). There are other researches that exploit model sparsity to accelerate DNN inference [17] [25], but they either do not target mobile platforms (require new hardware) or trade off compression rate and accuracy, thus having different challenges than our work.

3 Overview

The pattern-based sparsity should exploit the best of both non-structured and structured pruning while hiding the disadvantages. Given that, we propose two pattern-based pruning dimensions, *pattern pruning* and *connectivity pruning*.

Pattern pruning is illustrated in Figure 2, where the white blocks denote a fixed number of pruned weights in each kernel. The remaining (four) green blocks in each kernel have arbitrary weight values, while their locations form a specific pattern. Different kernels can have different patterns, but the total number of pattern styles (i.e., the size of the pattern library) shall be limited. We focus on 3×3 kernel pattern in this work because it is widely used in various

of DNN architectures. For other kernel shape (e.g., 1×1 or 5×5), we group 1×1 kernels into 3×3 then apply patterns, or use 5×5 patterns directly (will not be discussed in this work due to space limit).

Connectivity pruning is illustrated in Figure 2, with gray kernels as pruned ones. Connectivity pruning is a good supplement to pattern pruning, as both can be integrated in the same algorithm-level solution and compiler-assisted mobile inference framework.

Compiler-assisted DNN inference framework uniquely enables optimized code generation to guarantee end-to-end inference execution efficiency supporting pattern-based sparsity. As the computation paradigm of DNN is in a manner of layerwise execution, we convert a DNN model into computational graph, which is embodied by static C++ (for CPU execution) or OpenCL and CUDA (for GPU execution) codes. The above two pruning schemes can be naturally combined, which achieves high pruning (acceleration) rate while maintaining hardware friendliness.

4 Pattern Library – Theory and Design

4.1 A Unique Perspective on Weight Pruning

Conventionally, weight pruning is considered as a redundant information removal technique. This will inevitably omit other aspects, such as the computer vision properties of pruning. In this work, we consider weight pruning as incorporating an additional convolution mask P on an original kernel. P has the same size as original kernels and binary-valued elements (0 and 1). From our perspective, pattern pruning is an element-wise multiplication of different P's and original kernels. The set of different P's is the *pattern library*.

The multi-layer DNN are formed by cascading functional layers. Applying P on every convolution kernel across layers is intrinsically an interpolation operation of P's. Different patterns can form functional steerable filters [9] (e.g., Gaussian blur filter, sharpen filter, edge detection filter, etc.) by interpolation, and this process only needs a limited number of patterns (i.e., a small pattern library). A small pattern library has two advantages, (i) at algorithm level, an appropriate number of patterns ensures the flexible search space for achieving a solution with good performance on DNN and (ii) at compiler level, fewer patterns means fewer computation paradigms after kernel reordering and grouping, which reduces thread level divergence.

4.2 Pattern Library Design

Our designed patterns could be transformed to a series of steerable filters [9], which in our case, the Gaussian filter and Laplacian of Gaussian filter by interpolating patterns through DNN layers.

Transform patterns to Gaussian filter: Consider a two-dimensional Gaussian filter $\mathcal{G}$:

$$\mathcal{G}(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{1}$$

x and y are input coordinates, and σ^2 is variance.

Binomial coefficients give a compact approximation of the Gaussian coefficients using only integers. To apply the Gaussian filters with 3×3 filter size, we utilize the following approximation. According to (1) and set $\sigma^2 = \frac{1}{2}$, in the 1-D situation, the approximation of Gaussian filter [1 2 1] is given by the convolution of two box filters [1 1]. Then we get the 2-D approximation of Gaussian filter by convolving $\begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$ and $\begin{bmatrix} 1 & 2 & 1 \end{bmatrix}^T$, and the result is $\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$.

Interpolation in multi-layer DNN is proved to be convergent [30]. We can make further approximation by interpolating patterns into convolutional layers (i.e. uniformly map patterns to each kernel). In continuous probability space, interpolating patterns into convolution function is a specific Probability Density Function (PDF), so the effect of interpolating patterns is accumulating probability expectations of interpolation into n convolutional layers.

$$\underbrace{\begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \cdots \begin{bmatrix} 0 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} \cdots \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix} \cdots \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}}_{n \text{ interpolations}} = \begin{bmatrix} p & 2p & p \\ 2p & 4p & 2p \\ p & 2p & p \end{bmatrix}^n = \begin{bmatrix} p \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \end{bmatrix}^n \tag{2}$$

The four pattern masks P shown in colored positions in (2) form the Gaussian filter through interpolation. The coefficient p has no effect after normalization.

Transform patterns to Laplacian of Gaussian filter: The Laplacian operator is a second derivative operator. According to the associative property, smoothing an image with Gaussian filter and then applying Laplacian operator is equivalent to convolve the image with the Laplacian of Gaussian (LoG) filter:

$$\nabla^2 \mathcal{G}(x, y, \sigma) = \left(\frac{x^2 + y^2}{\sigma^4} - \frac{2}{\sigma^2} \right) \mathcal{G}(x, y, \sigma) \tag{3}$$

LoG has elegant mathematical properties, and is valid for a variety of applications including image enhancement, edge detection, and stereo matching.

Taylor series expansion is utilized to determine the approximate values of the LoG filter with 3×3 filter size. First, we consider the 1-D situation. The Taylor series expansions of 1-D Gaussian filter $\mathcal{G}(x)$ are given by:

$$\mathcal{G}(x+\delta) = \mathcal{G}(x) + \delta \mathcal{G}'(x) + \frac{1}{2}\delta^2 \mathcal{G}''(x) + \frac{1}{3!}\delta^3 \mathcal{G}'''(x) + \mathcal{O}\left(\delta^4\right) \tag{4}$$

$$\mathcal{G}(x-\delta) = \mathcal{G}(x) - \delta \mathcal{G}'(x) + \frac{1}{2}\delta^2 \mathcal{G}''(x) - \frac{1}{3!}\delta^3 \mathcal{G}'''(x) + \mathcal{O}\left(\delta^4\right) \tag{5}$$

By summing (4) and (5), we have

$$[\mathcal{G}(x-\delta) - 2\mathcal{G}(x) + \mathcal{G}(x+\delta)]/\delta^2 = \nabla^2 \mathcal{G}(x) + \mathcal{O}\left(\delta^2\right) \tag{6}$$

Applying central difference approximation of LoG $\nabla^2 \mathcal{G}(x)$, we derive the 1-D approximation of LoG filter as $\begin{bmatrix} 1 & -2 & 1 \end{bmatrix}$. Then we procure the 2-D approximation of LoG filter by convolving $\begin{bmatrix} 1 & -2 & 1 \end{bmatrix}$ and $\begin{bmatrix} 1 & -2 & 1 \end{bmatrix}^T$, and get $\begin{bmatrix} -1 & 2 & -1 \\ 2 & -4 & 2 \\ -1 & 2 & -1 \end{bmatrix}$ as the *1st approximation*. According to (6), we have

$$\nabla^2 \mathcal{G}(x, y) = \left(\begin{bmatrix} 1 & -2 & 1 \end{bmatrix} + \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix} \right) * \mathcal{G}(x, y) \tag{7}$$

Based on (7), we derive the *2nd approximation* as $\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$.

According to the central limit theorem, the convolution of two Gaussian functions is still a Gaussian function. Hence, we convolve the above two approximations of LoG and then apply normalization, and get the *Enhanced Laplacian of Gaussian* (ELoG) filter as $\begin{bmatrix} 0 & 1 & 0 \\ 1 & 8 & 1 \\ 0 & 1 & 0 \end{bmatrix}$.

Similarly, we make the further approximation by interpolating patterns into convolutional layers.

$$\underbrace{\begin{bmatrix} 0 & \mathbf{1} & 0 \\ \mathbf{1} & \mathbf{1} & \mathbf{1} \\ 0 & 0 & 0 \end{bmatrix} \cdots \begin{bmatrix} 0 & \mathbf{1} & 0 \\ \mathbf{1} & \mathbf{1} & 0 \\ 0 & \mathbf{1} & 0 \end{bmatrix} \cdots \begin{bmatrix} 0 & 0 & 0 \\ \mathbf{1} & \mathbf{1} & \mathbf{1} \\ 0 & \mathbf{1} & 0 \end{bmatrix} \cdots \begin{bmatrix} 0 & \mathbf{1} & 0 \\ 0 & \mathbf{1} & \mathbf{1} \\ 0 & \mathbf{1} & 0 \end{bmatrix}}_{n \text{ interpolations}} = \begin{bmatrix} 0 & p & 0 \\ p & 1 & p \\ 0 & p & 0 \end{bmatrix}^n = \left[p \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1/p & 1 \\ 0 & 1 & 0 \end{bmatrix} \right]^n \tag{8}$$

The four pattern masks P shown in colored positions in (8) form the ELoG filter through interpolation. In order to get the best approximation to ELoG filter, we set $p = 0.75$ and $n = 8$, then the desired filter is equal to interpolating these four patterns for eight times. The coefficient p has no effect after normalization.

5 Pattern-Aware Network Pruning Framework for Pattern Library Extraction

In Section 4, we have determined the (eight) patterns as our pattern library through theoretical derivation. However, are these theoretically derived patterns also the most desirable at algorithm level? How to select the appropriate pattern for each kernel and train corresponding (remaining) weights? To answer these questions, we propose a novel *pattern-aware network pruning* framework, simultaneously achieving pattern library extraction (with predefined number of patterns in library), pattern assignment, and weight training.

In pattern library extraction, we start from a large library comprising all possible candidate patterns. By extending ADMM [4] and incorporating Primal-Proximal solution technique, we make convolution kernels dynamically "select" the best suited patterns within the library and train the unpruned weights. Then we delete the least selected patterns in the library, thereby updating the library. The previous step is iterated on the updated library, with a single step as shown below.

5.1 Pattern Library Extraction – A Single Step

For an N-layer DNN of interest, let $\mathbf{W}$ denote the collection of weights for all 3×3 kernels, i.e., $\mathbf{W} = \{\mathbf{W}_i\}_{i=1}^N$. The pattern of each kernel $\mathbf{W}_i$ is restricted to a finite pattern library $\Omega = \{\mathbf{M}_1, \ldots, \mathbf{M}_j, \ldots, \mathbf{M}_K\}$, where $\mathbf{M}_j$ denotes a binary mask, and K denotes the total number of possible patterns. We choose to reserve 4 non-zero entries in a kernel to match the SIMD (single-instruction multiple-data) architecture of embedded CPU/GPU processors, thereby maximizing throughput. As a result, the initial $K = \binom{9}{4} = 126$, and K will decrease in each step.

The purpose of each step is to select a pattern from the current library for each kernel, and train the non-zero weights. Let $f(\mathbf{W};\mathcal{D})$ denote the training loss ($\mathcal{D}$ denotes training data), we pose the following optimization problem

$$\begin{aligned} &\underset{\mathbf{W},\mathbf{z}}{\text{minimize}} \;\; f(\{\mathbf{W}_i \circ (\textstyle\sum_{j=1}^{K} z_j \mathbf{M}_j)\}_{i=1}^{N};\mathcal{D}) \\ &\text{subject to } z_j \in \{0,1\}, \forall j, \quad \textstyle\sum_{j=1}^{K} z_j = 1, \end{aligned} \tag{9}$$

where z_j denotes the Boolean selection variable to indicate which pattern in Ω is chosen for $\mathbf{W}_i$. The constraint $\sum_{j=1}^{K} z_j = 1$ indicates that only one pattern is selected, and thus $\mathbf{W}_i \circ (\sum_{j=1}^{K} z_j \mathbf{M}_j)$ denotes the pattern-pruned kernel using one of pruning patterns. Here $\circ$ denotes element-wise product. In (9), we have *two* types of optimization variables: (i) 3×3 kernel weights $\mathbf{W}$, (ii) pattern Boolean selection variables $\mathbf{z} \in [0,1]^K$. The pattern selection scheme is co-optimized with non-zero weight training.

To solve the above problem analytically, we introduce auxiliary variables $\mathbf{u}$ together with constraints $\mathbf{z} = \mathbf{u}$. Based on that, we reformulate problem (9) as

$$\begin{aligned} &\underset{\mathbf{W},\mathbf{u}}{\text{minimize}} \;\; f(\{\mathbf{W}_i \circ (\textstyle\sum_{j=1}^{K} z_j \mathbf{M}_j)\}_{i=1}^{N};\mathcal{D}) + \mathcal{I}(\mathbf{u}) \\ &\text{subject to } \mathbf{z} = \mathbf{u} \end{aligned} \tag{10}$$

where $\mathcal{I}(\mathbf{u})$ is the indicator function

$$\mathcal{I}(\mathbf{u}) = \begin{cases} 0 & \text{if } u_j \in [0,1], \forall j, \quad \sum_{j=1}^{K} u_j = 1 \\ \infty & \text{otherwise.} \end{cases} \tag{11}$$

Here we relax the binary selection variable $z_i \in \{0,1\}$ to the (continuous) probabilistic selection variable $u_i \in [0,1]$.

The augmented Lagrangian function of problem (10) is given by

$$\begin{aligned} \mathcal{L}(\mathbf{W},\mathbf{z},\mathbf{u},\boldsymbol{\mu}) = f\big(\{\mathbf{W}_i \circ (\sum\nolimits_{j=1}^{K} z_j \mathbf{M}_j)\}_{i=1}^{N};\mathcal{D}\big) \\ + \mathcal{I}(\mathbf{u}) + \boldsymbol{\mu}^T(\mathbf{z}-\mathbf{u}) + \frac{\rho}{2}\|\mathbf{z}-\mathbf{u}\|_2^2 \end{aligned} \tag{12}$$

where $\boldsymbol{\mu}$ is Lagrangian multipliers, and $\|\cdot\|_2$ denotes the Frobenius norm. $\rho > 0$ is a given augmented penalty value, and for ease of notation we view matrices as *vectors* in optimization.

ADMM is then given by the following alternating optimization process. At iteration t, ADMM yields

$$\mathbf{W}^{(t)}, \mathbf{z}^{(t)} = \underset{\mathbf{W},\mathbf{z}}{\arg\min}\, \mathcal{L}(\mathbf{W},\mathbf{z},\mathbf{u}^{(t-1)},\boldsymbol{\mu}^{(t-1)}) \tag{Primal}$$

$$\mathbf{u}^{(t)} = \underset{\mathbf{u}}{\arg\min}\, \mathcal{L}(\mathbf{W}^{(t)},\mathbf{z}^{(t)},\mathbf{u},\boldsymbol{\mu}^{(t-1)}) \tag{Proximal}$$

$$\boldsymbol{\mu}^{(t)} = \boldsymbol{\mu}^{(t-1)} + \rho(\mathbf{z}^{(t)} - \mathbf{u}^{(t)}), \tag{13}$$

where the initial values $\mathbf{u}^{(0)}$ and $\boldsymbol{\mu}^{(0)}$ are given.

Problem (Primal) can be simplified to

$$\underset{\mathbf{W},\mathbf{z}}{\text{minimize}}\ f(\{\mathbf{W}_i \circ (\textstyle\sum_{j=1}^{K} z_j \mathbf{M}_j)\}_{i=1}^{N}; \mathcal{D}) + \frac{\rho}{2}\|\mathbf{z} - \mathbf{a}\|_2^2 \tag{14}$$

where $\mathbf{a} := (\mathbf{u}^{(t-1)} - (1/\rho)\boldsymbol{\mu}^{(t-1)})$. In problem (14), the objective function is differentiable, and can thus be solved by standard DNN solvers in SGD.

Problem (Proximal) can be equivalently decomposed over $\mathbf{u}$. This leads to problem

$$\begin{aligned} &\underset{\mathbf{u}}{\text{minimize}}\ \ \frac{\rho}{2}\|\mathbf{u} - \mathbf{d}\|_2^2 \\ &\text{subject to } u_j \in [0,1], \forall j, \quad \textstyle\sum_{j=1}^{K} u_j = 1, \end{aligned} \tag{15}$$

where $\mathbf{d} := \mathbf{z}^{(t)} + (1/\rho)\boldsymbol{\mu}^{(t-1)}$.

Based on [26], the analytical solution to problem (15) is

$$\mathbf{u}^{(t)} = [\mathbf{d} - \nu\mathbf{1}]_+, \tag{16}$$

where $[x]_+ = x$ if $x \geq 0$ and 0 otherwise, ν is the root of the equation

$$\mathbf{1}^T [\mathbf{d} - \nu\mathbf{1}]_+ = 1. \tag{17}$$

Once $\mathbf{W}$ and $\mathbf{z}$ are solved, $\mathbf{z}$ is a continuous variable rather than a binary variable. We need an intermediate step to project continuous $\mathbf{z}_{\text{admm}}$ to integer $\mathbf{z}_{\text{binary}}$, yielding

$$\begin{aligned} &\underset{\mathbf{z}_{\text{binary}}}{\text{minimize}}\ \ \|\mathbf{z}_{\text{binary}} - \mathbf{z}_{\text{admm}}\|_2^2 \\ &\text{subject to } \mathbf{1}^T\mathbf{z} = 1, z_i \in \{0,1\}, \forall i. \end{aligned} \tag{18}$$

The solution is given by $[\mathbf{z}_{\text{binary}}]_i = 1$ if $i = \text{argmax}_j [\mathbf{z}_{\text{admm}}]_j$, and 0 otherwise. At this point, we have simultaneously selected pattern for each kernel and trained the non-zero weights.

5.2 Pattern Library Extraction – Overall

The overall pattern library extraction starts from $K = 126$ and decreases K in each step, with algorithm brief shown in Algorithm 1. In actual implementation we set the new K to be 12 in the first step as most of the patterns occur in very few times. We set the target K to be either 12, 8, or 4. When the type of patterns is within this range, the overhead in code generation at compiler level can be kept small and parallelism can be maximized.

Total Runtime: Despite an iterative process, the total number of epochs (and training time) can be limited. This is because except for the last step, we only need to extract a number of patterns instead of finishing the final training of non-zero weights. As a result, we can finish each step with 10% to 20% of the total epochs as training of the original DNN. In the last step, we need around 9 - 12 ADMM iterations, each requiring less than 20% of the total epochs of

original DNN training. So the total number of training epochs using PyTorch [27] is around 300 - 400 for the whole process, which is even lower compared with many prior art [10,22].

Algorithm 1: Pattern library extraction process.

Initialization: $\Omega = \{\mathbf{M}_1, \mathbf{M}_2 \ldots, \mathbf{M}_K\}$ with $K = 126$;
Result: Subsets Ω' with $K = 12, 8$ or 4;
while *training neural network* **do**
  Update W by solving (Primal) ;
  for $K \leftarrow 126$ *until* $K = 12, 8$ *or* 4 **do**
    Solving (Proximal) using current Ω;
    Update μ in (13);
    Calculate pattern distribution of current Ω ;
    Removing patterns with fewest occurrences in Ω ;
  end
end

6 Connectivity Sparsity and the New Sparsity Induced Inference Framework

6.1 Connectivity Sparsity

Connectivity sparsity is achieved by connectivity pruning which can be integrated in the same algorithm-level solution in Section 5.1 and compiler-assisted mobile inference framework. Using the same notations as in Section 5.1, we define the collection of weights in i-th layer as $\mathbf{W}_i \in \mathbb{R}^{H_i \times W_i \times F_i \times C_i}$, where H and W denote the dimension of the convolution kernel. F and C denote the number of filters and channels, respectively. We further define critical connectivity score for each convolution kernel as

$$\gamma_{i,f,c}(\mathbf{W}_i) = ||[\mathbf{W}_i]_{:,:,f,c}||_2 \tag{19}$$

where f and c are filter and channel indices, respectively. The problem formulation and solution framework for achieving connectivity sparsity is similar with the ones in Section 5.1. The difference is that the constraint in the framework is related to $\gamma_{i,f,c}$. Please note that our algorithm level solution can solve the problems of pattern and connectivity pruning simultaneously or individually.

6.2 Compiler-assisted Inference Framework for Real-time Execution

After we obtain pattern and connectivity sparsity combined in a DNN model, we use a compiler-assisted inference framework to maximize the execution efficiency by utilizing multiple optimization techniques that are induced by pattern-based sparsity. The compiler optimizations showing in Figure 3 target on DNN computation graph and memory access for on-device executions.

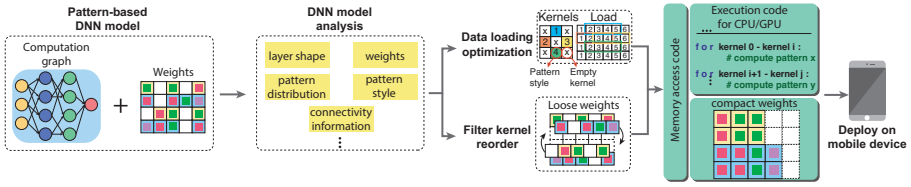

Fig. 3: Overview of the compiler level DNN inference framework.

Layerwise optimization for DNN computation graph is designed to achieve the best of instruction-level and thread-level parallelism by utilizing the unique filter/kernel re-ordering technique as Figure 3 shows. In the weight matrix illustration, the internal squares with different colors denote different pattern styles, and empty white squares denote connectivity sparsity. By filter/kernel re-ordering, we (i) organize the filters with similar kernels together to improve *inter-thread* parallelism, and (ii) group kernels with identical patterns in each filter together to improve *intra-thread* parallelism. By DNN computation graph optimization, the generated execution code eliminates all of the execution branches, implying higher instruction-level parallelism; meanwhile, similar filter groups escalate execution similarity and result in a good load balance, achieving better thread-level parallelism.

Memory access optimizations for hardware execution address the poor memory performance due to the irregular memory access. In DNN execution, the input/output data access is associated with the non-zero elements of the weights. Since in pattern-based sparse model, the non-zero pattern of each kernel is already known, we can generate data access code with this information for each kernel pattern and call them dynamically during DNN execution. With the data access code, it is possible to directly access valid input data that is associated with the non-zero elements in a pattern-based kernel. Moreover, after DNN computation graph optimization, the model weights distribution is highly compact and structured as Figure 3 shows, which reduces the calling frequency of data access code and as a result, reduces the memory overhead.

7 Experimental Results

In our experiment, our generated pattern-based sparse models are based on four widely used network structures, VGG-16 [29], ResNet-18/50 [11] and MobileNet-V2 [15], and are trained on an eight NVIDIA RTX-2080Ti GPUs server using PyTorch [27]. We show the consistency of pattern library extraction results with the theoretically designed pattern library in Section 4.2, and provide the accuracy improvement and image enhancement demonstrations. We also show the overall compression results of pattern-based pruning in different DNN models. In order to show acceleration of pattern-based sparsity on mobile devices, we compare it with three state-of-the-art DNN inference acceleration frameworks,

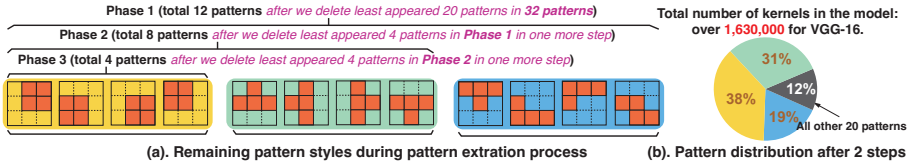

Fig. 4: The pattern library extraction result. When $K = 32$ after two steps, the pattern distribution is shown in *(b)* with different colors representing different pattern styles in *(a)*. The 20 less significant patterns only account for 12% of the total 32 patterns, and the rest 12 patterns form the *Phase 1* pattern library. If we continue the extraction step, we can get *Phase 2* and *Phase 3* pattern libraries as *(a)* shows.

TFLite [1], TVM [6], and MNN [2]. Our experiments are conducted on a Samsung Galaxy S10 cell phone with the latest Qualcomm Snapdragon 855 mobile platform that consists of a Qualcomm Kryo 485 Octa-core CPU and a Qualcomm Adreno 640 GPU.

7.1 Pattern Library Extraction Result

We use VGG-16 on ImageNet dataset to extract pattern libraries. VGG-16 has more than 1,630,000 convolution kernels. However, patterns can be concentrated to 12 styles in only a couple of steps. Figure 4 shows the pattern styles distribution results when K decreases to 32 after two steps. We can see that most of the patterns are distributed in the top 12 styles, namely Phase 1 pattern library. If we continue to decrease K to 8, the remaining 8 patterns form Phase 2 pattern library. We can notice that Phase 2 is *exactly the same* with our derived pattern library in Section 4.2. Further extraction step will give us Phase 3 pattern library, which is the top-4 pattern styles. Using other DNNs and datasets gives us the same extraction results, thereby we can conclude that the theoretically derived patterns are also the most desirable ones at algorithm level.

7.2 Visualization Demonstration and Accuracy Analysis for Pattern Pruning

After we obtain the extracted pattern libraries in three phases (i.e., containing 12, 8 or 4 patterns respectively), we need to validate the image enhancement effects and evaluate the accuracy of the pattern pruned DNN.

Visualization comparisons of applying Phase 2 pattern library to an original DNN model (*pattern pruning*) are demonstrated in Figure 5. To ensure the fairness in comparisons, we adopt three visualization methods to eliminate the impact of causal factors. They are (a) *Guided-backpropagation (BP)* [31], (b) *Integrated gradients* [23], and (c) *Inverted representation* [3]. Through different visualization techniques, we can see what a DNN has learned and how well it can preserve the photographically accurate information from an image.

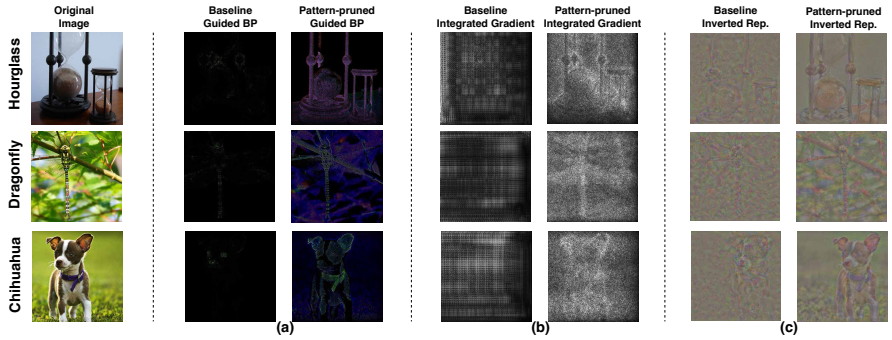

Fig. 5: Visualization comparisons of three images from ImageNet dataset on original and pattern pruned VGG-16 model using (a) guided-backpropagation (BP); (b) integrated gradients and (c) inverted representation methods.

We provide strong evidence in Figure 5 that pattern pruned VGG-16 model can effectively capture more image details and less noise compared with the original VGG-16 model. We conclude that the accuracy improvement is attributed to the enhanced image processing ability of our designed pattern library.

Accuracy evaluation is shown in Figure 6 (a). Starting from the baseline accuracy results that are in many cases higher than prior works, we have the first conclusion that *the accuracy improvements are more significant when applying the designed 8 patterns (i.e., pattern library at Phase 2) on each convolution kernel*. The accuracy improvements are consistently observed on various network structures (e.g., VGG-16, ResNet-18/50, MobileNet-V2) on CIFAR-10 and ImageNet datasets.

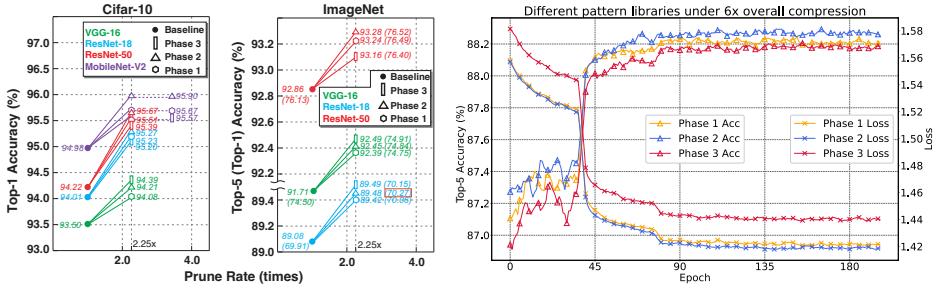

Fig. 6: (a) Accuracy improvement results from pattern pruning on different DNN models and datasets (CIFAR-10 & ImageNet). (b) Overall 6× compression for ResNet-18 on ImageNet training curves for connectivity sparsity.

Table 1: Pattern-based pruning results (%) on convolution layer for CIFAR-10 and ImageNet using VGG-16, ResNet-18 and ResNet-50. (**O**: original, **P**: prune)

		CIFAR-10				**ImageNet**					
	Pruning Framework	**Top-1**		**Comp. Rate**	**Sparse Type**	**Top-1**		**Top-5**		**Comp. Rate**	**Sparse Type**
		O	**P**			**O**	**P**	**O**	**P**		
ResNet-18†	AMC [13]	90.5	90.2	2.0×	Struct.	-	-	-	-	-	-
	Tiny [21]	94.1	93.2	15.1×	Struct.	*N/A*	*N/A*	89.1	88.4	3.3×	Struct.
	TAS [8]	92.8	92.8	1.8×	Struct.	70.6	69.1	89.8	89.2	1.5×	Struct.
	FPGM [12]	92.2	91.9	2.5×	Struct.	70.2	68.3	89.6	88.5	3.3×	Struct.
	Ours	94.0	94.7	8.0×	Phase 2	69.9	69.6	89.1	89.2	4.0×	Phase 2
	Ours	94.0	94.6	12.0×	Phase 3	69.9	68.2	89.1	88.3	6.0×	Phase 2
	Ours	94.0	94.2	16.0×	Phase 2	69.9	67.1	89.1	87.7	8.0×	Phase 2
ResNet-50*	One Shot [19]	93.8	93.6	2.5×	Irreg.	-	-	-	-	-	-
	ADMM-NN [28]	-	-	-	-	*N/A*	*N/A*	*N/A*	92.3	7.0×	Irreg.
	TAS [8]	94.5	93.7	2.0×	Struct.	77.5	76.2	93.5	93.1	1.7×	Struct.
	GAL [16]	93.3	90.4	2.9×	Struct.	76.4	69.3	92.8	89.1	2.5×	Struct.
	FPGM [12]	93.6	93.5	2.5×	Struct.	76.2	75.6	92.8	92.6	3.3×	Struct.
	GBN [35]	-	-	-	-	75.8	75.2	92.7	92.4	2.2×	Struct.
	Ours	94.2	95.2	8.0×	Phase 3	76.1	75.9	92.9	92.7	3.9×	Phase 2
	Ours	94.2	94.9	12.0×	Phase 3	76.1	75.8	92.9	92.8	4.9×	Phase 3
	Ours	94.2	94.5	16.0×	Phase 3	76.1	75.6	92.9	92.6	5.8×	Phase 2
VGG-16	NeST [7]	-	-	-	-	71.6	69.3	90.4	89.4	6.5×	Irreg.
	ADMM-NN [28]	-	-	-	-	69.0	68.7	89.1	88.9	10.2×	Irreg.
	DecorReg [36]	93.5	93.3	8.5×	Struct.	73.1	73.2	*N/A*	*N/A*	3.9×	Struct.
	GAL [16]	93.9	90.8	5.6×	Struct.	-	-	-	-	-	-
	Ours	93.5	93.4	8.0×	Phase 2	74.5	74.4	91.7	91.5	8.0×	Phase 2
	Ours	93.5	93.3	11.6×	Phase 2	74.5	74.1	91.7	91.3	10.0×	Phase 2
	Ours	93.5	93.2	19.7×	Phase 1	74.5	73.6	91.7	91.0	12.0×	Phase 2

† TAS, FPGM use ResNet-20 network structure on CIFAR-10 dataset.
* TAS, GAL, FPGM use ResNet-56 network structure on CIFAR-10 dataset.

7.3 Connectivity Pruning and Overall Model Compression Results

Combining connectivity sparsity with pattern sparsity has different DNN performances with different pattern libraries. Figure 6 (b) illustrates testing accuracies of training connectivity sparsity combined with existing pattern sparsity. From diagram, we can clearly notice that by using designed pattern library (Phase 2), we can achieve better training performance, thereby higher DNN accuracy. Similar paradigm can be observed with different compression rates and on different networks/datasets. Please note that pattern sparsity already reserves 2.25× compression rate, and we add different connectivity compression rates upon it to achieve the different overall compression rates. Table 1 records the best final DNN accuracies and compression rates regarding their pattern styles, and are compared with several pruning methods with their sparsity types.

7.4 Performance Evaluation on Mobile Platform

In this part, we demonstrate our evaluation results on mobile devices. To guarantee fairness, all frameworks are using the same pattern-based sparse model, and we also enable the fully optimized configurations of TFLite, TVM and MNN (e.g., Winograd optimization is turned on).

Execution time. Figure 7 shows mobile CPU/GPU execution time of pattern-based model on different platforms. Since Phase 2 pattern library has best performance on pruning, our testing model are using Phase 2 patterns and 8× overall compression rate for ResNet-18, 5.8× for ResNet-50 and 12× for VGG-16. The inference is using images from ImageNet dataset. We can see our approach achieves significant acceleration on mobile device compared with other frameworks. Real-time execution usually requires 30 frames/sec (i.e., $33ms$/frame). From our results, all of our DNN models on ImageNet meet or far exceed this requirement, and some of them can even accomplish real-time inference on mobile CPU.

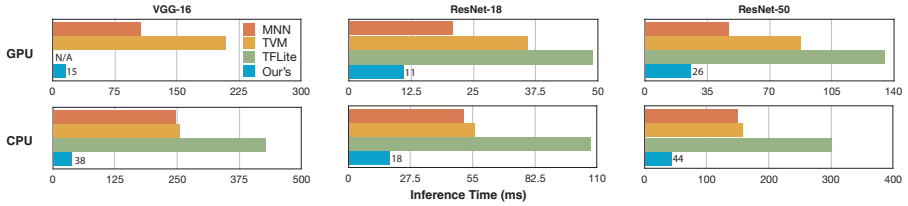

Fig. 7: Inference time (ms) comparisons for different mobile inference frameworks using image from ImageNet dataset.

8 Conclusion

This paper proposes pattern-based sparsity, along with the highly efficient algorithm level pruning framework and the novel compiler level inference framework. Pattern-based sparsity inherits the flexibility from non-structured sparsity and regularity from structured sparsity, achieving both highly accurate/compressed model and hardware friendliness. Particularly, with carefully designed pattern library, pattern pruning achieves image enhancement and accuracy improvement. The pattern-based sparsity elicits compiler optimization, achieving real-time inference on mobile devices on various representative large-scale DNNs.

9 Acknowledgment

This work is supported by the National Science Foundation CCF-1919117, CCF-1937500, CNS-1909172, CNS-2011260, and is sponsored by DiDi GAIA Research Collaboration Initiative. We thank all anonymous reviewers for their feedback.

References

1. https://www.tensorflow.org/mobile/tflite/
2. https://github.com/alibaba/MNN
3. Aravindh, M., Andrea, V.: Understanding deep image representations by inverting them. In: Computer Vision and Pattern Recognition, 2015. CVPR 2015. IEEE Conference on (2015)
4. Boyd, S., Parikh, N., Chu, E., Peleato, B., Eckstein, J.: Distributed optimization and statistical learning via the alternating direction method of multipliers. Foundations and Trends® in Machine Learning **3**(1), 1–122 (2011)
5. Chen, C.F., Oh, J., Fan, Q., Pistoia, M.: Sc-conv: Sparse-complementary convolution for efficient model utilization on cnns. In: 2018 IEEE International Symposium on Multimedia (ISM). pp. 97–100. IEEE (2018)
6. Chen, T., Moreau, T., Jiang, Z., Zheng, L., Yan, E., Shen, H., Cowan, M., Wang, L., Hu, Y., Ceze, L., et al.: TVM: An automated end-to-end optimizing compiler for deep learning. In: OSDI (2018)
7. Dai, X., Yin, H., Jha, N.K.: Nest: A neural network synthesis tool based on a grow-and-prune paradigm. IEEE Transactions on Computers **68**(10), 1487–1497 (2019)
8. Dong, X., Yang, Y.: Network pruning via transformable architecture search. In: Advances in Neural Information Processing Systems. pp. 759–770 (2019)
9. Freeman, W., Adelson, E.: The design and use of steerable filters. In: IEEE Transactions on Pattern Analysis and Machine Intelligence. vol. 13, pp. 891–906. IEEE (1991)
10. Han, S., Mao, H., Dally, W.J.: Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In: International Conference on Learning Representations (ICLR) (2016)
11. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 770–778 (2016)
12. He, Y., Liu, P., Wang, Z., Hu, Z., Yang, Y.: Filter pruning via geometric median for deep convolutional neural networks acceleration. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 4340–4349 (2019)
13. He, Y., Lin, J., Liu, Z., Wang, H., Li, L.J., Han, S.: Amc: Automl for model compression and acceleration on mobile devices. In: European Conference on Computer Vision. pp. 815–832 (2018)
14. He, Y., Zhang, X., Sun, J.: Channel pruning for accelerating very deep neural networks. In: Computer Vision (ICCV), 2017 IEEE International Conference on. pp. 1398–1406. IEEE (2017)
15. Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861 (2017)
16. Lin, S., Ji, R., Yan, C., Zhang, B., Cao, L., Ye, Q., Huang, F., Doermann, D.: Towards optimal structured cnn pruning via generative adversarial learning. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2790–2799 (2019)
17. Liu, B., Wang, M., Foroosh, H., Tappen, M., Pensky, M.: Sparse convolutional neural networks. In: CVPR. pp. 806–814 (2015)
18. Liu, N., Ma, X., Xu, Z., Wang, Y., Tang, J., Ye, J.: Autocompress: An automatic dnn structured pruning framework for ultra-high compression rates. In: AAAI. pp. 4876–4883 (2020)

19. Liu, Z., Sun, M., Zhou, T., Huang, G., Darrell, T.: Rethinking the value of network pruning. In: International Conference on Learning Representations (2019)
20. Ma, X., Guo, F.M., Niu, W., Lin, X., Tang, J., Ma, K., Ren, B., Wang, Y.: Pconv: The missing but desirable sparsity in dnn weight pruning for real-time execution on mobile devices. In: AAAI. pp. 5117–5124 (2020)
21. Ma, X., Yuan, G., Lin, S., Ding, C., Yu, F., Liu, T., Wen, W., Chen, X., Wang, Y.: Tiny but accurate: A pruned, quantized and optimized memristor crossbar framework for ultra efficient dnn implementation. ASP-DAC (2020)
22. Molchanov, P., Tyree, S., Karras, T., Aila, T., Kautz, J.: Pruning convolutional neural networks for resource efficient inference. arXiv preprint arXiv:1611.06440 (2016)
23. Mukund, S., Ankur, T., Qiqi, Y.: Axiomatic attribution for deep networks. In: 2017 International Conference on Machine Learning (ICML). ACM/IEEE (2017)
24. Niu, W., Ma, X., Lin, S., Wang, S., Qian, X., Lin, X., Wang, Y., Ren, B.: Patdnn: Achieving real-time dnn execution on mobile devices with pattern-based weight pruning. In: Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems. pp. 907–922 (2020)
25. Parashar, A., Rhu, M., Mukkara, A., Puglielli, A., Venkatesan, R., Khailany, B., Emer, J., Keckler, S.W., Dally, W.J.: Scnn: An accelerator for compressed-sparse convolutional neural networks. In: ISCA (2017)
26. Parikh, N., Boyd, S.: Proximal algorithms. Foundations and Trends® in Optimization **1**(3), 127–239 (2014)
27. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al.: Pytorch: An imperative style, high-performance deep learning library. In: NeurIPS (2019)
28. Ren, A., Zhang, T., Ye, S., Li, J., Xu, W., Qian, X., Lin, X., Wang, Y.: Admm-nn: An algorithm-hardware co-design framework of dnns using alternating direction methods of multipliers. In: ASPLOS. pp. 925–938 (2019)
29. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
30. Siyuan, M., Raef, B., Mikhail, B.: The power of interpolation: Understanding the effectiveness of sgd in modern over-parametrized learning. In: 2018 International Conference on Machine Learning (ICML). ACM/IEEE (2018)
31. Springenberg, J.T., Alexey Dosovitskiy, T.B.a.R.: Striving for simplicity: The all convolutional net. In: ICLR-2015 workshop track (2015)
32. Wen, W., Wu, C., Wang, Y., Chen, Y., Li, H.: Learning structured sparsity in deep neural networks. In: Advances in neural information processing systems. pp. 2074–2082 (2016)
33. Xu, M., Zhu, M., Liu, Y., Lin, F.X., Liu, X.: Deepcache: Principled cache for mobile deep vision. In: Proceedings of the 24th Annual International Conference on Mobile Computing and Networking. pp. 129–144. ACM (2018)
34. Yao, S., Hu, S., Zhao, Y., Zhang, A., Abdelzaher, T.: Deepsense: A unified deep learning framework for time-series mobile sensing data processing. In: Proceedings of the 26th International Conference on World Wide Web (2017)
35. You, Z., Yan, K., Ye, J., Ma, M., Wang, P.: Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks. In: Advances in Neural Information Processing Systems. pp. 2130–2141 (2019)
36. Zhu, X., Zhou, W., Li, H.: Improving deep neural network sparsity through decorrelation regularization. In: IJCAI (2018)