PCONV: The Missing but Desirable Sparsity in DNN Weight Pruning for Real-time Execution on Mobile Devices

Xiaolong Ma[†1], Fu-Ming Guo[†1], Wei Niu[2], Xue Lin[1], Jian Tang[3], Kaisheng Ma[4], Bin Ren[2], Yanzhi Wang[1]
[1]Northeastern University, [2]College of William and Mary, [3]Syracuse University, [4]Tsinghua University
E-mail: [1]{ma.xiaol, guo.fu}@husky.neu.edu, [1]{xue.lin, yanz.wang}@northeastern.edu,
[2]wniu@email.wm.edu, [2]bren@cs.wm.edu, [3]jtang02@syr.edu, [3]kaisheng@mail.tsinghua.edu.cn

Abstract

Model compression techniques on Deep Neural Network (DNN) have been widely acknowledged as an effective way to achieve acceleration on a variety of platforms, and DNN weight pruning is a straightforward and effective method. There are currently two mainstreams of pruning methods representing two extremes of pruning regularity: *non-structured*, fine-grained pruning can achieve high sparsity and accuracy, but is not hardware friendly; *structured*, coarse-grained pruning exploits hardware-efficient structures in pruning, but suffers from accuracy drop when the pruning rate is high. In this paper, we introduce *PCONV*, comprising a new sparsity dimension, – fine-grained pruning patterns inside the coarse-grained structures. *PCONV* comprises two types of sparsities, Sparse Convolution Patterns (SCP) which is generated from intra-convolution kernel pruning and connectivity sparsity generated from inter-convolution kernel pruning. Essentially, SCP enhances accuracy due to its special vision properties, and connectivity sparsity increases pruning rate while maintaining balanced workload on filter computation. To deploy *PCONV*, we develop a novel compiler-assisted DNN inference framework and execute *PCONV* models in real-time without accuracy compromise, which cannot be achieved in prior work. Our experimental results show that, *PCONV* outperforms three state-of-art end-to-end DNN frameworks, TensorFlow-Lite, TVM, and Alibaba Mobile Neural Network with speedup up to $39.2\times$, $11.4\times$, and $6.3\times$, respectively, with no accuracy loss. Mobile devices can achieve real-time inference on large-scale DNNs.

Introduction

Deep neural network (DNN) has emerged as the fundamental element and core enabler in machine learning applications due to its high accuracy, excellent scalability, and self-adaptiveness (Goodfellow et al. 2016). A well trained DNN model can be deployed as inference system for multiple objectives, such as image classification (Krizhevsky, Sutskever, and Hinton 2012), object detection (Ren et al. 2015), and natural language processing (Hinton, Deng, and Yu 2012). However, the state-of-art DNN models such as VGG-16 (Simonyan and Zisserman 2014), ResNet-50 (He et al. 2016) and MobileNet (Howard et al. 2017) involve intensive computation and high memory storage, making it very challenging to execute inference system on current mobile platforms in a real-time manner.

Recently, high-end mobile platforms are rapidly overtaking desktop and laptop as primary computing devices for broad DNN applications such as wearable devices, video streaming, unmanned vehicles, smart health devices, etc. (Philipp, Durr, and Rothermel 2011)(Lane et al. 2015)(Boticki and So 2010). Developing a real-time DNN inference system is desirable but still yield to the limited computation resources of embedded processors on a mobile platform. Multiple end-to-end mobile DNN acceleration frameworks, such as TVM (Chen et al. 2018), TensorFlow-Lite (TFLite) (Ten) and Alibaba Mobile Neural Network (MNN) (Ali), have been developed. However, the inference time of large-scale DNNs (e.g., 242ms inference time using TVM on Adreno 640 GPU with VGG-16) is still far from real-time requirement.

In order to mitigate the challenge brings by the DNN's bulky computation and achieve the goal of real-time inference, it is necessary to consider algorithm-level innovations. Various DNN model compression techniques are studied, among which *weight pruning* (Han, Mao, and Dally 2015)(Mao et al. 2017)(Dai, Yin, and Jha 2017)(Wen et al. 2016)(He, Zhang, and Sun 2017) can result in a notable reduction in the model size. Early work (Han, Mao, and Dally 2015) on *non-structured* weight pruning (fine-grained) prunes weights at arbitrary location, resulting in a sparse model to be stored in the compressed sparse column (CSC) format. It leads to an undermined processing throughput because the indices in the compressed weight representation cause stall or complex workload on highly parallel architectures (Han, Mao, and Dally 2015)(Wen et al. 2016). On the other hand, *structured* weight pruning (Wen et al. 2016) (coarse-grained) is more hardware friendly. By exploiting filter pruning and channel pruning, the pruned model is more regular in its shape, which eliminates the storage requirement in weight indices. However, it is observed that structured pruning hurts accuracy more significantly than non-structured sparsity.

It is imperative to find a new granularity level that

[†]These authors contributed equally.

can satisfy high accuracy demand as well as regularity in DNN model structure. We make the observation that non-structured and structured pruning are two extremes of the full design space. The two missing keys are: (i) Find a new, intermediate sparsity dimension that can fully leverage both the high accuracy from fine-grained model and high regularity level from coarse-grained model; (ii) Find the corresponding (algorithm-compiler-hardware) optimization framework which can seamlessly bridge the gap between hardware efficiency and the new sparsity dimension. To address the above problems, this paper proposes *PCONV*, comprising (a) a new sparsity dimension that exploits both intra-convolution and inter-convolution kernel sparsities, exhibiting both high accuracy and regularity, and revealing a previously *unknown* point in design space; and (b) a *compiler-assisted DNN inference framework* that fully leverages the new sparsity dimension and achieves real-time DNN acceleration on mobile devices.

In *PCONV*, we call our intra-convolution kernel pruning *pattern pruning* and inter-convolution kernel pruning *connectivity pruning*. For pattern pruning, a fixed number of weights are pruned in each convolution kernel. Different from non-structured weight pruning, pattern pruning produces the same sparsity ratio in each filter and a limited number of pattern shapes. Essentially, our designed patterns correspond to the computer vision concept of key convolution filters, such as Gaussian filter for smoothing, Laplacian of Gaussian filter for smoothing and sharpening. For connectivity pruning, the key insight is to *cut the connections* between certain input and output channels, which is equivalent to removal of corresponding kernels, making filter "length" shorter than original model. With connectivity pruning, we further enlarge compression rate and provide greater DNN acceleration potential, while maintaining balanced workload in filter-wise computation of DNNs. Pattern and connectivity pruning can be combined at algorithm level and accelerated under the unified compiler-assisted acceleration framework. For our advanced *compiler-assisted DNN inference framework*, we use execution code generation which converts DNN models into computational graphs and applies multiple optimizations including a high-level, fine-grained DNN layerwise information extraction, filter kernel reorder and load redundancy elimination. All design optimizations are general, and applicable to both mobile CPUs and GPUs.

We demonstrate that pattern pruning consistently improve model accuracy. When combined with connectivity pruning, the results still outperform current DNN pruning methods, both non-structured and structured weight pruning. In Section "Accuracy Analysis", we show *PCONV* is the most desirable sparsity among current prune-for-acceleration works. We also deploy *PCONV* model on our compiler-assisted mobile acceleration framework and compare with three state-of-art frameworks on mobile CPU and GPU, TensorFlow Lite, TVM, and MNN, using three widely used DNNs, VGG-16, ResNet-50, and MobileNet-v2 and two benchmark datasets, ImageNet and CIFAR-10. Evaluation results show that *PCONV* achieves up to 39.2× speedup without any accuracy drop. Using Adreno 640 embedded GPU, *PCONV* achieves an unprecedented 19.1 ms inference time of VGG-16 on ImageNet dataset. To the best of our knowledge, it is the first time to achieve real-time execution of such representative large-scale DNNs on mobile devices.

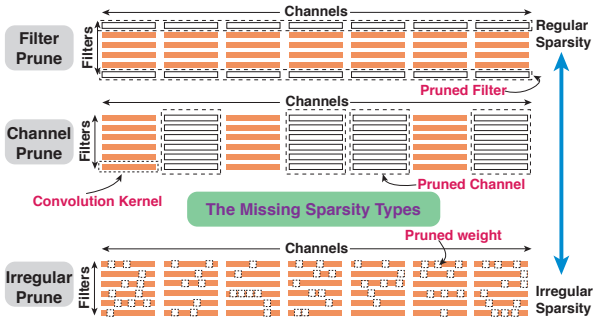

Figure 1: Overview of different weight pruning dimensions.

Background

DNN Model Compression

DNN model compression is a promising method to remove redundancy in the original model. It targets on the purpose that inference time can be reduced if fewer weights are involved in the computation graph. The weight pruning method acts as a surgeon to remove the inherently redundant neurons or synapses. As Figure 1 shows, two main approaches of weight pruning are the general, non-structured pruning and structured pruning, which produce irregular and regular compressed DNN models, respectively.

Non-structured pruning: Early work is (Han, Mao, and Dally 2015), in which an iterative, heuristic method is used with limited, non-uniform model compression rates. Flourished by (Zhang et al. 2018) and (Ren et al. 2019) with the powerful ADMM (Boyd et al. 2011) optimization framework, non-structured pruning achieves very high weight reduction rate and promising accuracy. However, for compiler and code optimization, irregular weight distribution within kernels requires heavy control-flow instructions, which degrades instruction-level parallelism. Also, kernels in different filters have divergent workloads, which burdens thread-level parallelism when filters are processed through multi-threading. Moreover, irregular memory access causes low memory performance and thereby execution overheads.

Structured pruning: This method has been proposed to address the index overhead and imbalanced workload caused by non-structured pruning. Pioneered by (Wen et al. 2016)(He, Zhang, and Sun 2017), structured weight pruning generates regular and smaller weight matrices, eliminating overhead of weight indices and achieving higher acceleration performance in CPU/GPU executions. However, it suffers from notable accuracy drop when the pruning rate increases.

Patterns in Computer Vision

Convolution operations exist in different research areas for an extended period of time, such as image processing, signal processing, probability theory, and computer vision. In this

work, we focus on the relationship between conventional image processing and state-of-art convolutional neural networks in the usage of convolutions. In image processing, the convolution operator is manually crafted with prior knowledge from the particular characteristics of diverse patterns, such as Gaussian filter. On the other hand, in convolutional neural networks, the convolution kernels are randomly initialized, then trained on large datasets using gradient-based learning algorithms for value updating.

(Mairal et al. 2014) derived a network architecture named Convolutional Kernel Networks (CKN), with lower accuracy than current DNNs, thus limited usage. (Zhang 2019) proposed to apply the blur filter to DNNs before pooling to maintain the shift-equivalence property. The limited prior work on the application of conventional vision filters to DNNs require network structure change and do not focus on weight pruning/acceleration, thus distinct from *PCONV*.

DNN Acceleration Frameworks on Mobile Platform

Recently, researchers from academia and industry have investigated DNN inference acceleration frameworks on mobile platforms, including TFLite (Ten), TVM (Chen et al. 2018), Alibaba Mobile Neural Network (MNN) (Ali), DeepCache (Xu et al. 2018) and DeepSense (Yao et al. 2017). These works do not account for model compression techniques, and the performance is far from real-time requirement. There are other researches that exploit model sparsity to accelerate DNN inference, e.g., (Liu et al. 2015), SCNN (Parashar et al. 2017), 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.

Motivations

Based on the current research progress on DNN model compression vs. acceleration, we analyze and rethink the whole design space, and are motivated by the following three points:

Achieving both high model accuracy and pruning regularity. In non-structured pruning, any weight can be pruned. This kind of pruning has the largest flexibility, thus achieves high accuracy and high prune rate. But it is not hardware-friendly. On the other hand, structured pruning produces hardware-friendly models, but the pruning method lacks flexibility and suffers from accuracy drop. Our motivation is to use the best of the above two sparsities. To achieve that, we introduce a new dimension, pattern-based sparsity, revealing a previously unknown design point with high accuracy and structural regularity simultaneously.

Image enhancement inspired sparse convolution patterns. The contemporary DNN weight pruning methods originate from the motivation that eliminating redundant information (weights) will not hurt accuracy. On the other hand, these pruning methods scarcely treat pruning as a specific kind of binary convolution operator, not to mention exploiting corresponding opportunities. Along this line, we find that sparse convolution patterns have the potential in enhancing image quality thanks to its special vision properties. Motivated by the fact that sparse convolution patterns can potentially enhance image quality, we propose our carefully designed patterns which are derived from mathematical vision theory.

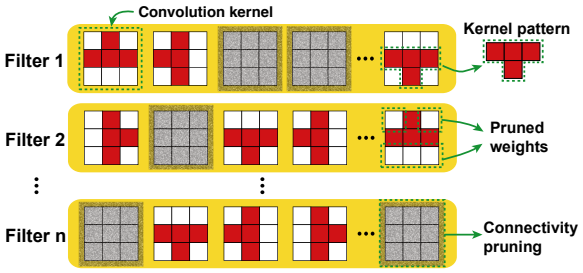

Figure 2: Illustration of pattern pruning and connectivity pruning.

Compiler-assisted DNN inference framework. With the higher accuracy enabled by fine-grained pruning patterns, the key question is how to re-gain similar (or even surpass) hardware efficiency as coarse-gained structured pruning. We take a unique approach and design an optimized, compiler-assisted DNN inference framework to close the performance gap between full structured pruning and pattern-based pruning.

Theory of Sparse Convolution Patterns (SCP)

Let an image with resolution $H \times W$ be represented by $X \in \mathbb{R}^{H \times W \times 3}$. An $L-$layer DNN can be expressed as a feature extractor $\mathcal{F}_L(\mathcal{F}_{L-1}(\dots \mathcal{F}_1(X)\dots))$, with layer index $l \in \{1, \dots, L\}$. Inside the DNN, each convolutional layer is defined as $\mathcal{F}_l(X_l) \in \mathbb{R}^{H_l \times W_l \times F_l \times C_l}$, with filter kernel shape $H_l \times W_l$, number of filters F_l and number of channels C_l.

Besides treating pruning as a redundant information removal technique, we consider it as incorporating an additional convolution kernel P to perform element-wise multiplication with the original kernel. P is termed the *Sparse Convolution Pattern* (SCP), with dimension $H_l \times W_l$ and binary-valued elements (0 and 1). Specific SCPs fit the mathematical vision theory well according to our following derivation. Based on the mathematical rigority, we propose the novel *pattern pruning* scheme, i.e., applying SCPs to convolution kernels. As illustrated in Figure 2, the white blocks denote a fixed number of pruned weights in each kernel. The remaining red blocks in each kernel have arbitrary weight values, while their locations form a specific SCP P_i. Different kernels can have different SCPs, but the total number of SCP types shall be limited.

In order to further increase the pruning ratio and DNN inference speed, we can selectively cut the connections between particular input and output channels, which is equivalent to the removal of corresponding kernels. This is termed *connectivity pruning*. *Connectivity pruning* is illustrated in Figure 2, with gray kernels as pruned ones. The rationale of connectivity pruning stems from the desirability of locality in layerwise computations inspired by human visual sys-

tems (Yamins and DiCarlo 2016). It is a good supplement to pattern pruning. Both pruning schemes can be integrated in the same algorithm-level solution and compiler-assisted mobile acceleration framework.

The Convolution Operator

In conventional image processing, a convolution operator is formally defined by the following formula, where the output pixel value $g(x, y)$ is the weighted sum of input pixel values $f(x, y)$, and $h(k, l)$ is the weight kernel value

$$g(x, y) = \sum_{k,l} f(x + k, y + l)h(k, l) \tag{1}$$

This formula could transform to

$$g(x, y) = \sum_{k,l} f(k, l)h(x - k, y - l) \tag{2}$$

Then we derive the notation of convolution operator as:

$$g = f * h \tag{3}$$

Convolution is a linear shift-invariant (LSI) operator, satisfying the commutative property, the superposition property and the shift-invariance property. Additionally, convolution satisfies the associative property following the Fubini's theorem.

Sparse Convolution Pattern (SCP) Design

Our designed SCPs could be transformed to a series of steerable filters (Freeman and Adelson 1991), i.e., the Gaussian filter and Laplacian of Gaussian filter, which function as image smoothing, edge detection or image sharpening in mathematical vision theory.

Gaussian filter: Consider a two-dimensional Gaussian filter G:

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{4}$$

x and y are input coordinates, and σ is standard deviation of the Gaussian distribution. Typically, the Gaussian filter performs image smoothing, and further sophisticated filters can be created by first smoothing the image input with a unit area Gaussian filter, then applying other steerable filters.

Laplacian of Gaussian filter: The Laplacian operator is the 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:

$$\nabla^2 G(x, y, \sigma) = \left(\frac{x^2 + y^2}{\sigma^4} - \frac{2}{\sigma^2} \right) G(x, y, \sigma) \tag{5}$$

The LoG filter is a bandpass filter that eliminates both the high-frequency and low-frequency noises. 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 $G(x)$ are given by:

$$G(x+h) = G(x) + hG'(x) + \frac{1}{2}h^2 G''(x) + \frac{1}{3!}h^3 G'''(x) + O\left(h^4\right) \tag{6}$$

$$G(x-h) = G(x) - hG'(x) + \frac{1}{2}h^2 G''(x) - \frac{1}{3!}h^3 G'''(x) + O\left(h^4\right) \tag{7}$$

By summing (6) and (7), we have

$$G(x + h) + G(x - h) = 2G(x) + h^2 G''(x) + O\left(h^4\right) \tag{8}$$

The second derivative of Gaussian $G''(x)$ is equivalent to LoG $\nabla^2 G(x)$. Equation (8) is further transformed to

$$\frac{G(x - h) - 2G(x) + G(x + h)}{h^2} = \nabla^2 G(x) + O\left(h^2\right) \tag{9}$$

Applying central difference approximation of LoG $\nabla^2 G(x)$, 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 $\begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix}$, and get result as $\begin{bmatrix} -1 & 2 & -1 \\ 2 & -4 & 2 \\ -1 & 2 & -1 \end{bmatrix}$. According to the property of second derivative:

$$\nabla^2 G(x, y) = G_{xx}(x, y) + G_{yy}(x, y) \tag{10}$$

and Equation (9), we have

$$G_{xx}(x, y) + G_{yy}(x, y) = \left(\begin{bmatrix} 1 & -2 & 1 \end{bmatrix} + \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix} \right) * G(x, y) \tag{11}$$

Based on (11), we derive another approximation of LoG 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, and the new variance is the sum of the variances of the two original Gaussian functions. Hence, we convolve the above two approximations of LoG and then apply normalization, and get the *Enhanced Laplacian of Gaussian* (ELoG) filter as $\begin{bmatrix} 0 & 1 & 0 \\ 1 & 8 & 1 \\ 0 & 1 & 0 \end{bmatrix}$.

(Siyuan, Raef, and Mikhail 2018) have proved the convergence of the interpolation in the context of (multi-layer) DNNs, so we utilize the interpolated probability density estimation to make the further approximation. In ELoG filter where 1 appears, we mask it to 0 with the probability of $(1-p)$. Because we uniformly convolve SCPs into n convolutional layers, this random masking operation can be treated as distributed interpolation of SCPs. In continuous probability space, interpolating SCPs into convolution function is a specific Probability Density Function (PDF), so the effect of interpolating SCPs is accumulating probability expectations of interpolation into n convolutional layers. Besides, the convolution function is normalized to unity, so we separate the coefficient p in the following equation.

$$\underbrace{\begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} \cdots \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix} \cdots \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix} \cdots \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}}_{n \text{ interpolations}}$$
$$= \begin{bmatrix} 0 & p & 0 \\ p & 1 & p \\ 0 & p & 0 \end{bmatrix}^n = \begin{bmatrix} p \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1/p & 1 \\ 0 & 1 & 0 \end{bmatrix} \end{bmatrix}^n \tag{12}$$

The four SCPs are shown in colored positions in (12). In order to get the best approximation to ELoG filter, we set

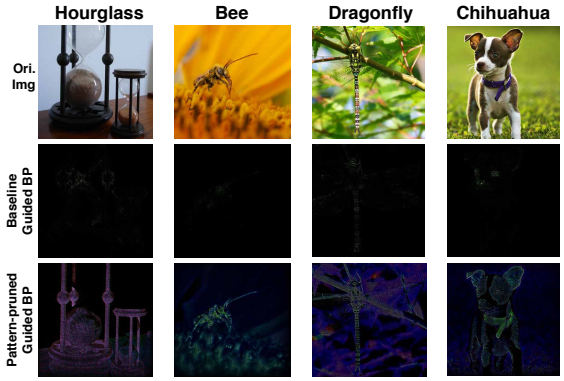

Figure 3: Visualization of intermediate results (*saliency map* of *gradient images*) in original VGG-16 model and pattern pruned VGG-16 model through *guided-backpropagation*.

$p = 0.75$ and $n = 8$, then the desired filter is equal to interpolating these four SCPs for eight times. The coefficient p has no effect after normalization.

Upper bound: According to (C.Blakemore and Campbell 1969), the optimal times for applying the *LoG* filter is six and the maximum is ten. Thus the desired number of times to interpolate the SCP in (12) is around 24 and the maximum number is around 55. This upper bound covers most of the existing effective DNNs, even for ResNet-152, which comprises 50 convolutional layers with filter kernel size of 3×3.

The four SCPs in (12) form the ELoG filter through interpolation. Hence, the designed SCPs inherit the de-noising and sharpening characteristics of LoG filters. We visualize the intermediate results of DNNs to interpret and verify the advancement of our designed SCPs in the following section.

Visualization and Interpretation

Explanations of individual DNN decision have been explored by generating informative heatmaps such as CAM and grad-CAM (Selvaraju et al. 2017), or through guided-backpropagation (BP) (Springenberg and Alexey Dosovitskiy 2015) conditioned on the final prediction. Utilizing guided-backpropagation, we can visualize what a DNN has learned. The visualization results of applying SCPs to an original DNN model (*pattern pruning*) are demonstrated in Figure 3. We sample four input images from the ImageNet dataset, as "hourglass", "bee", "dragonfly" and "chihuahua", then apply the guided-backpropagation to propagate back from each target class label and get the gradient images. Eventually, we generate the *saliency maps* of gradient images. Compared with the original VGG-16 model, the pattern pruned VGG-16 model captures more detailed information of the input image with less noise.

There are plenty of DNN visualization techniques. In Supplemental Materials, we demonstrate two more sets of visualization results using integrated gradients and inverted representation methods. And both sets show our pattern pruned model collects more information in an image than original model. We conclude that by applying our designed SCPs,

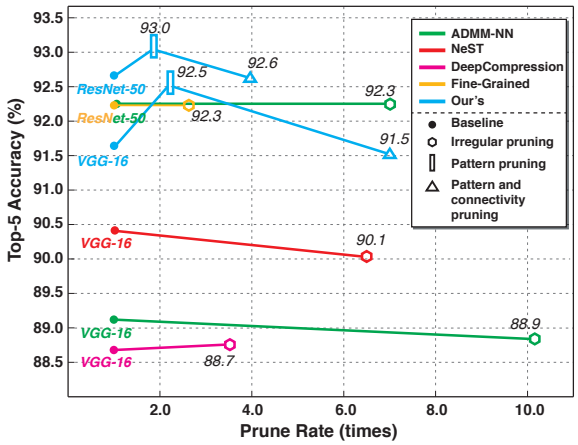

Figure 4: Comparison results of our pattern and connectivity pruning of VGG-16 and ResNet-50 on ImageNet dataset with: ADMM-NN (Ren et al. 2019), NeST (Dai, Yin, and Jha 2017), Deep Compression (Han, Mao, and Dally 2015) and Fine-grained pruning (Mao et al. 2017).

pattern pruning enhances DNNs' image processing ability.

Accuracy Analysis (Supplemental Materials)

In our previous derivation, we have determined the (four) SCPs as our pattern set. Our algorithm-level solution starts from a pre-trained DNN model, or can train from scratch. To generate *PCONV* model, we need to assign SCPs to each kernel (pattern pruning) or prune specific kernels (connectivity pruning), and train the active (unpruned) weights. To achieve this goal, we extend the ADMM-NN framework in (Ren et al. 2019) to produce pattern and connectivity-pruned models. Due to space limit, the algorithm details in *PCONV* model generation are described in Supplemental Materials.

Accuracy results are illustrated in Figure 4. Starting from the baseline accuracy results that are in many cases higher than prior work, we have the first conclusion that *the accuracy will improve when applying our designed SCPs on each convolution kernel*. For ImageNet dataset, *pattern pruning* improves the top-5 accuracy of VGG-16 from 91.7% to 92.5%, and ResNet-50 from 92.7% to 93.0% with SCPs applied to each convolution kernel. The accuracy improvement is attributed to the enhanced image processing ability of our designed SCPs.

Pruning vs. accuracy for non-structured pruning, structured pruning and *PCONV*. Combined with *connectivity pruning*, *PCONV* achieves higher compression rate without accuracy compromise. Comparing with other pruning methods, i.e., non-structured pruning and structured pruning, we conclude that: (i) *PCONV* achieves higher accuracy and higher compression rate compared with prior non-structured pruning, and close to the results in ADMM-NN; (ii) compared with structured pruning, under the same compression rate, *PCONV* achieves higher accuracy, and can structurally prune more weights without hurting accuracy. Detailed comparison is shown in Supplemental Materials.

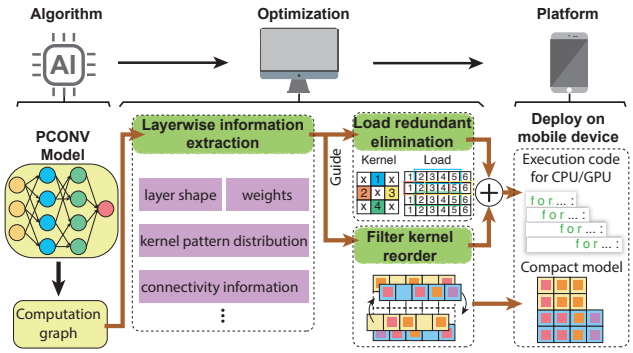

Figure 5: Overview of *PCONV* acceleration framework. From algorithm-level design to platform-level implementation.

Compiler-assisted DNN Inference Framework

In this section, we propose our novel compiler-assisted DNN inference acceleration framework for mobile devices. Motivated by the two merits – flexibility and regularity of the *PCONV* model, our compiler-assisted platform uniquely enables *optimized code generation* to guarantee end-to-end execution efficiency. As DNN's computation paradigm is in a manner of layerwise execution, we can convert a DNN model into computational graph, which is embodied by static C++ (for CPU execution) or OpenCL (for GPU execution) code. The code generation process includes three steps: (i) layerwise information extraction; (ii) filter kernel reorder; (iii) load redundancy elimination.

Layerwise information extraction is a model analysis procedure. In particular, it analyzes detailed kernel pattern and connectivity-related information. Key information such as pattern distribution, pattern order and connection between input/output channel through kernels are utilized by the compiler to perform optimizations in steps (ii) and (iii).

Filter kernel reorder is designed to achieve the best of instruction-level and thread-level parallelism. When a *PCONV* model is trained, patterns and connections of all kernels are already known, i.e., the computation pattern is already fixed before deploying the model for inference. All these information of patterns are collected from layerwise information extraction, and is leveraged by filter kernel reorder to (i) organize the filters with similar kernels together to improve *inter-thread* parallelism, and (ii) order the same kernels in a filter together to improve *intra-thread* parallelism. Figure 6 illustrates the two key steps of filter kernel reorder: (i) organizes similar filters next to each other; (ii) groups kernels with identical patterns in each filter together. As a result, the generated execution code eliminates much of 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.

Load redundancy elimination addresses the issue of irregular memory access that causes memory overhead. In DNN execution, the data access pattern of input/output is decided by the (none-zero elements) patterns of kernels. Therefore, we can generate data access code with this information for each kernel pattern and call them dynamically during DNN execution. Because the data access code consists of all information at kernel-level computation, it is possible to directly access valid input data that is associated with the non-zero elements in a pattern-based kernel. After steps (i) and (ii), patterns are distributed in a structured manner, which reduces the calling frequency of data access code and as a result, reduces the memory overhead.

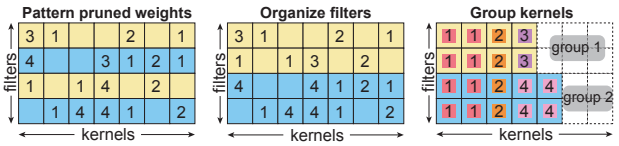

Figure 6: Steps of filter kernel reorder: each square represents a convolution kernel; the number represents the specific pattern type of this kernel.

Experimental Results

In this section, we evaluate the execution performance of our compiler-assisted framework with our *PCONV* model deployed. All of our evaluation models are generated by ADMM pruning algorithm which is described in Supplemental Materials, and are trained on an eight NVIDIA RTX-2080Ti GPUs server using PyTorch.

Methodology

In order to show acceleration of *PCONV* on mobile devices, we compare it with three state-of-art DNN inference acceleration frameworks, TFLite (Ten), TVM (Chen et al. 2018), and MNN (Ali). 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.

In our experiment, our generated *PCONV* models are based on three widely used network structures, VGG-16 (Simonyan and Zisserman 2014), ResNet-50 (He et al. 2016) and MobileNet-v2 (Howard et al. 2017). Since convolution operation is most time-consuming (more than 95% of the total inference time) in DNN computation, our evaluation on the above network structures focus on convolutional layers performance. In order to provide a very clear illustration on how *PCONV* enhances mobile performance, the whole device-level evaluation is shown in three aspects: (i) execution time, (ii) on-device GFLOPS performance and (iii) how pattern counts affect performance.

Performance Evaluation

In this part, we demonstrate our evaluation results on mobile device from the three aspects we discussed above. In order to illustrate *PCONV* has the best acceleration performance on mobile devices, our comparison baselines, i.e., TFLite, TVM and MNN use the fully optimized configurations (e.g., Winograd optimization is turned on).

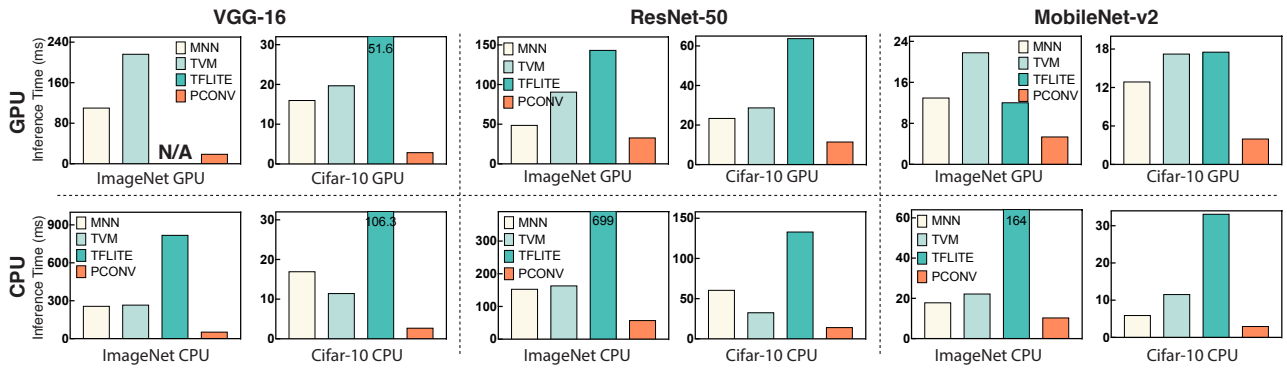

Figure 7: Mobile CPU/GPU inference time (ms) on different network structures inferring Cifar-10 and ImageNet images.

Execution time. Figure 7 shows mobile CPU/GPU performance of *PCONV* model executing on our compiler-assisted DNN inference framework. On CPU, *PCONV* achieves $9.4\times$ to $39.2\times$ speedup over TFLite, $2.2\times$ to $5.1\times$ speedup over TVM and $1.7\times$ to $6.3\times$ speedup over MNN. On GPU, *PCONV* achieves $2.2\times$ to $18.0\times$ speedup over TFLite, $2.5\times$ to $11.4\times$ speedup over TVM and $1.5\times$ to $5.8\times$ speedup over MNN. For the largest DNN (VGG-16) and largest data set (ImageNet), our framework completes computations on a single input image within $19.1ms$ (i.e., 52.4 frames/sec) on GPU, which meets the real-time requirement (usually 30 frames/sec, i.e., 33 ms/frame).

On-device GFLOPS performance. From the previous comparison results we see that MNN has the higher performance than TVM and TFLite. To show that *PCONV* has better throughput on mobile devices, we compare *PCONV* with MNN by measuring their run-time GFLOPS on both CPU and GPU. Figure 8 demonstrates layerwise GFLOPS performance comparison between *PCONV* and MNN. The 9 layers we pick from VGG-16's 13 convolutional layers are representing 9 unique layers with 9 unique layer sizes. The other 4 layers are omitted in Figure 8 because they have repeated layer sizes which product repeated GFLOPS results. From the results we can see that for both CPU and GPU throughputs, *PCONV* outperforms MNN.

Pattern counts vs. performance. In order to determine how pattern counts affects execution performance, we design some random patterns with 4 non-zero elements in one kernel alongside with our designed SCPs. Table 1 and Table 2 show accuracy and execution time under different pattern counts using VGG-16 on Cifar-10 and ImageNet datasets. The results show that the accuracy losses are not necessarily related to the increase of pattern counts, but the execution performance drops quickly, especially on ImageNet dataset. The pattern counts vs. performance results prove that our designed SCPs result in ideal performance with a negligible accuracy loss.

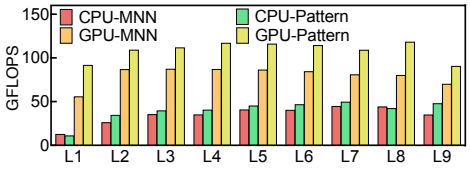

Figure 8: On-device GFLOPS performance evaluation of MNN and *PCONV*.

Table 1: Pattern counts vs. performance. Evaluation uses model with pattern ($2.25\times$) and connectivity ($8.8\times$) sparsity on VGG-16 Cifar-10 dataset. Top-1 accuracy displayed.

Dataset	Pattern#	Acc. (%)	Acc. loss (%)	Device	Speed (ms)
Cifar-10	**4**	93.8	-0.3	CPU	2.7
				GPU	2.9
	8	93.7	-0.2	CPU	2.9
				GPU	3.0
	12	93.8	-0.3	CPU	3.1
				GPU	3.3

Table 2: Pattern counts vs. performance. Evaluation uses model with pattern ($2.25\times$) and connectivity ($3.1\times$) sparsity on VGG-16 ImageNet dataset. Top-5 accuracy displayed.

Dataset	Pattern#	Acc. (%)	Acc. loss (%)	Device	Speed (ms)
ImageNet	**4**	91.5	0.2	CPU	52.7
				GPU	19.1
	8	91.6	0.1	CPU	58.9
				GPU	22.0
	12	91.6	0.1	CPU	105.2
				GPU	32.1

Conclusion

This paper presents *PCONV*, a desirable sparsity type in DNN weight pruning that elicits mobile devices acceleration, leading to real-time mobile inference. *PCONV* inherits the high flexibility in non-structured pruning which helps achieving high accuracy and compression rate, and maintains highly structured weight composition like structured pruning which leads to hardware friendlinesses such as optimized memory access, balanced workload and computation parallelism etc. To show *PCONV*'s real-time performance on mobile devices, we design a compiler-assisted DNN inference framework, which can fully leverage *PCONV*'s structural characteristics and achieve very high inference speed on representative large-scale DNNs.

References

https://github.com/alibaba/MNN.

Aravindh, M., and Andrea, V. 2015. Understanding deep image representations by inverting them. In *Computer Vision and Pattern Recognition, 2015. CVPR 2015. IEEE Conference on*.

Boticki, I., and So, H.-J. 2010. Quiet captures: A tool for capturing the evidence of seamless learning with mobile devices. In *International Conference of the Learning Sciences-Volume 1*.

Boyd, S.; Parikh, N.; Chu, E.; Peleato, B.; and Eckstein, J. 2011. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning* 3(1):1–122.

C.Blakemore, and Campbell, F. W. 1969. On the existence of neurones in the human visual system selectively sensitive to the orientation and size of retinal images. In *The Journal of Physiology*. The Physiological Society.

Chen, T.; Moreau, T.; Jiang, Z.; Zheng, L.; Yan, E.; Shen, H.; Cowan, M.; Wang, L.; Hu, Y.; Ceze, L.; et al. 2018. TVM: An automated end-to-end optimizing compiler for deep learning. In *OSDI*.

Dai, X.; Yin, H.; and Jha, N. K. 2017. Nest: a neural network synthesis tool based on a grow-and-prune paradigm. *arXiv preprint arXiv:1711.02017*.

Freeman, W., and Adelson, E. 1991. The design and use of steerable filters. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 13, 891–906. IEEE.

Goodfellow, I.; Bengio, Y.; Courville, A.; and Bengio, Y. 2016. *Deep learning*, volume 1. MIT press Cambridge.

Han, S.; Pool, J.; Tran, J.; and Dally, W. 2015. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, 1135–1143.

Han, S.; Mao, H.; and Dally, W. J. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 770–778.

He, Y.; Lin, J.; Liu, Z.; Wang, H.; Li, L.-J.; and Han, S. 2018. Amc: Automl for model compression and acceleration on mobile devices. In *European Conference on Computer Vision*, 815–832.

He, Y.; Liu, P.; Wang, Z.; Hu, Z.; and Yang, Y. 2019. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 4340–4349.

He, Y.; Zhang, X.; and Sun, J. 2017. Channel pruning for accelerating very deep neural networks. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, 1398–1406. IEEE.

Hinton, G.; Deng, L.; and Yu, D. e. a. 2012. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*.

Howard, A. G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; and Adam, H. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

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

Huang, Z., and Wang, N. 2018. Data-driven sparse structure selection for deep neural networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

Kingma, D. P., and Ba, J. 2014. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *NeurIPS*.

Lane, N. D.; Bhattacharya, S.; Georgiev, P.; Forlivesi, C.; and Kawsar, F. 2015. An early resource characterization of deep learning on wearables, smartphones and internet-of-things devices. In *International workshop on IOT towards applications*.

Li, H.; Kadav, A.; Durdanovic, I.; Samet, H.; and Graf, H. P. 2016. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*.

Liu, B.; Wang, M.; Foroosh, H.; Tappen, M.; and Pensky, M. 2015. Sparse convolutional neural networks. In *CVPR*, 806–814.

Liu, Z.; Sun, M.; Zhou, T.; Huang, G.; and Darrell, T. 2018. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*.

Luo, J.-H.; Wu, J.; and Lin, W. 2017. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE international conference on computer vision*, 5058–5066.

Mairal, J.; Koniusz, P.; Harchaoui, Z.; and Schmid, C. 2014. Convolutional kernel networks. In *NeurIPS*.

Mao, H.; Han, S.; Pool, J.; Li, W.; Liu, X.; Wang, Y.; and Dally, W. J. 2017. Exploring the regularity of sparse structure in convolutional neural networks. *arXiv preprint arXiv:1705.08922*.

Min, C.; Wang, A.; Chen, Y.; Xu, W.; and Chen, X. 2018. 2pfpce: Two-phase filter pruning based on conditional entropy. *arXiv preprint arXiv:1809.02220*.

Mukund, S.; Ankur, T.; and Qiqi, Y. 2017. Axiomatic attribution for deep networks. In *2017 International Conference on Machine Learning (ICML)*. ACM/IEEE.

Parashar, A.; Rhu, M.; Mukkara, A.; Puglielli, A.; Venkatesan, R.; Khailany, B.; Emer, J.; Keckler, S. W.; and Dally, W. J. 2017. Scnn: An accelerator for compressed-sparse convolutional neural networks. In *ISCA*.

Philipp, D.; Durr, F.; and Rothermel, K. 2011. A sensor network abstraction for flexible public sensing systems. In *2011 IEEE Eighth International Conference on Mobile Ad-Hoc and Sensor Systems*, 460–469. IEEE.

Ren, S.; He, K.; Girshick, R.; and Sun, J. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, 91–99.

Ren, A.; Zhang, T.; Ye, S.; Xu, W.; Qian, X.; Lin, X.; and Wang, Y. 2019. Admm-nn: an algorithm-hardware co-design framework of dnns using alternating direction methods of multipliers. In *ASPLOS*.

Selvaraju, R. R.; Cogswell, M.; Das, A.; Vedantam, R.; Parikh, D.; and Batra, D. 2017. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *ICCV*.

Simonyan, K., and Zisserman, A. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Siyuan, M.; Raef, B.; and Mikhail, B. 2018. The power of interpolation: Understanding the effectiveness of sgd in modern over-parametrized learning. In *2018 International Conference on Machine Learning (ICML)*. ACM/IEEE.

Springenberg, J. T., and Alexey Dosovitskiy, T. B. a. R. 2015. Striving for simplicity: The all convolutional net. In *ICLR-2015 workshop track*.

https://www.tensorflow.org/mobile/tflite/.

Wen, W.; Wu, C.; Wang, Y.; Chen, Y.; and Li, H. 2016. Learning structured sparsity in deep neural networks. In *Advances in neural information processing systems*, 2074–2082.

Xu, M.; Zhu, M.; Liu, Y.; Lin, F. X.; and Liu, X. 2018. Deepcache: Principled cache for mobile deep vision. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, 129–144. ACM.

Yamins, D. L., and DiCarlo, J. J. 2016. Using goal-driven deep learning models to understand sensory cortex. *Nature neuroscience* 19(3):356.

Yao, S.; Hu, S.; Zhao, Y.; Zhang, A.; and Abdelzaher, T. 2017. Deepsense: A unified deep learning framework for time-series mobile sensing data processing. In *Proceedings of the 26th International Conference on World Wide Web*.

Yu, R.; Li, A.; Chen, C.-F.; Lai, J.-H.; Morariu, V. I.; Han, X.; Gao, M.; Lin, C.-Y.; and Davis, L. S. 2018. Nisp: Pruning networks using neuron importance score propagation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 9194–9203.

Zhang, T.; Ye, S.; Zhang, Y.; Wang, Y.; and Fardad, M. 2018. Systematic weight pruning of dnns using alternating direction method of multipliers. *arXiv preprint arXiv:1802.05747*.

Zhang, R. 2019. Making convolutional networks shift-invariant again. In *ICML*.

Appendices

Pattern and Connectivity Pruning Algorithm

From our derivation in Section "Theory of Sparse Convolution Patterns (SCP)" of our main paper, we have determined the (four) SCPs as our desired patterns. In this section, we describes the methods to generate compressed DNN models for *PCONV*. The procedure is composed of two steps: (1) we use the four SCPs to form a pattern set; (2) assign a pattern from pattern set for each kernel (pattern pruning) or prune the whole kernel (connectivity pruning), and train the pattern-based weights for maintaining accuracy. The overall flow is shown in Figure 9. Essentially, it reflects the algorithm aspects of *PCONV*. Our method can be applied to either a pre-trained DNN or train a model from scratch.

Problem Formulation: Consider an N-layer DNN, and we focus on the most computationally intensive CONV layers. The weights and biases of layer k are respectively denoted by $\mathbf{W}_k$ and $\mathbf{b}_k$, and the loss function of DNN is denoted by $f\big(\{\mathbf{W}_k\}_{k=1}^N, \{\mathbf{b}_k\}_{k=1}^N\big)$; see (Zhang et al. 2018). In our discussion, $\{\mathbf{W}_k\}_{k=1}^N$ and $\{\mathbf{b}_k\}_{k=1}^N$ respectively characterize the collection of weights and biases from layer 1 to layer N. Then the pattern and connectivity pruning is formulated as an optimization problem:

$$\begin{aligned} \underset{\{\mathbf{W}_k\},\{\mathbf{b}_k\}}{\text{minimize}} \quad & f\big(\{\mathbf{W}_k\}_{k=1}^N, \{\mathbf{b}_k\}_{k=1}^N\big), \\ \text{subject to} \quad & \mathbf{W}_k \in \mathcal{S}_k,\ \mathbf{W}_k \in \mathcal{S}_k',\ \ k = 1, \ldots, N. \end{aligned} \tag{12}$$

The collection of weights in the k-th CONV layer forms a four-dimensional tensor, i.e., $\mathbf{W}_k \in R^{P_k \times Q_k \times C_k \times C_{k+1}}$, where P_k, Q_k, C_k, and C_{k+1} are respectively the height of kernel, the width of kernel, the number of kernels, and the number of filters, in layer k. Suppose $\mathbf{X}$ denotes the weight tensor in a specific layer, then $(\mathbf{X})_{:,:,a,b}$ denotes a specific kernel.

In *pattern pruning*, the constraint in the k-th CONV layer is $\mathbf{W}_k \in \mathcal{S}_k := \{\mathbf{X} \mid$ each kernel in $\mathbf{X}$ needs to satisfy one specific pattern shape in the pattern set (and non-zero weight values can be arbitrary)$\}$. In *connectivity pruning*, the constraint in the k-th CONV layer is $\mathbf{W}_k \in \mathcal{S}_k' := \{\mathbf{X} \mid$ the number of nonzero kernels in $\mathbf{X}$ is less than or equal to $\alpha_k\}$ (α_k is a predetermined hyperparameter with more discussions later). Both constraints need to be simultaneously satisfied.

Extended ADMM-based Solution Framework: The constraint $\mathbf{W}_k \in \mathcal{S}_k$ in problem (12) is different from the clustering-like constraints in ADMM-NN (Ren et al. 2019), in that it is flexible to select a pattern for each kernel from the pattern set. As long as a pattern is assigned for each kernel, constraints in problem (12) become clustering-like and ADMM compatible. Similar to ADMM-NN (Ren et al. 2019), the ADMM-based solution is an iterative process, starting from a pre-trained DNN model. We assign an appropriate pattern for each kernel based on the L_2-norm metric in each iteration, to achieve higher flexibility.

By incorporating auxiliary variables $\mathbf{Z}_k$'s and $\mathbf{Y}_k$'s, and dual variables $\mathbf{U}_k$'s and $\mathbf{V}_k$'s, we decompose (12) into three

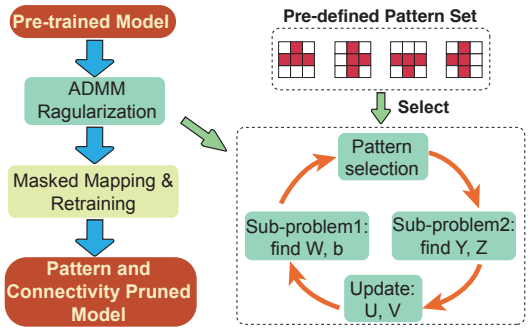

Figure 9: Algorithm-level overview of ADMM pattern regularization.

subproblems, and iteratively solve until convergence. In iteration l, after assigning patterns we solve the first subproblem

$$\begin{aligned} \underset{\{\mathbf{W}_k\},\{\mathbf{b}_k\}}{\text{minimize}}\ & f\big(\{\mathbf{W}_k\}_{k=1}^N, \{\mathbf{b}_k\}_{k=1}^N\big) + \sum_{k=1}^N \frac{\rho_k}{2} \|\mathbf{W}_k - \mathbf{Z}_k^l + \mathbf{U}_k^l\|_F^2 \\ & + \sum_{k=1}^N \frac{\rho_k}{2} \|\mathbf{W}_k - \mathbf{Y}_k^l + \mathbf{V}_k^l\|_F^2. \end{aligned} \tag{13}$$

The first term is the loss function of the DNN, while the other quadratic terms are convex. As a result, this subproblem can be solved by stochastic gradient descent (e.g., the ADAM algorithm (Kingma and Ba 2014)) similar to training the original DNN.

The solution $\{\mathbf{W}_k\}$ of subproblem 1 is denoted by $\{\mathbf{W}_k^{l+1}\}$. Then we aim to derive $\{\mathbf{Z}_k^{l+1}\}$ and $\{\mathbf{Y}_k^{l+1}\}$ in subproblems 2 and 3. These subproblems have the same form as those in ADMM-NN (Ren et al. 2019). Thanks to the characteristics in combinatorial constraints, the optimal, analytical solution of the two subproblems are Euclidean projections, and are polynomial time solvable. For example, for connectivity pruning, the projection is: keeping α_k kernels with largest L_2 norms and setting the rest of kernels to zero. For pattern pruning it is similar. Finally, we update dual variables $\mathbf{U}_k$ and $\mathbf{V}_k$ according to the ADMM rule (Boyd et al. 2011) and thereby complete the l-th iteration in the ADMM-based solution.

The hyperparameter determination process is relatively straightforward for joint pattern and connectivity pruning. There is no additional hyperparameters for pattern pruning when the pattern set has been developed. For connectivity pruning we need to determine the pruning rate α_k for each layer. In this paper, we adopt a heuristic method of uniform pruning rate for all layers except for the first layer (which is smaller, yet more sensitive to pruning).

Accuracy Evaluation for Different Pruning Methods

In this section, we provide comparison results on accuracy and compression rate of *PCONV* and several baseline works. Based on two different datasets, we category our comparison into two parts – one for ImageNet compression results

and another one for Cifar-10. In both categories, we compare *PCONV* with prior works in *non-structured* pruning and *structured* pruning. The comparison results show that (i) *PCONV* achieves higher accuracy and higher compression rate compared with prior non-structured pruning, and close to the results in ADMM-NN; (ii) compared with structured pruning, under the same compression rate, *PCONV* achieves higher accuracy, and can structurally prune more weights without hurting accuracy.

ImageNet Dataset

Table 3 and Table 4 illustrate the Top-5 accuracy comparison on joint pattern pruning and connectivity pruning, on VGG-16 and ResNet-50 using ImageNet dataset. For VGG-16, all kernels are 3×3. After applying SCPs on all kernels and $3.1\times$ uniform connectivity pruning, we achieve around $7\times$ weight reduction on convolution layers of VGG-16. For ResNet-50, a portion of kernels are 1×1 besides the majority of 3×3 kernels. We apply pattern pruning on all 3×3 ones, and apply uniform $2.7\times$ connectivity pruning on all kernels. We achieve $3.9\times$ weight reduction on convolution layers.

Table 3: *PCONV* and non-structured pruning comparison on ImageNet using VGG-16 and ResNet-50. We select widely-acknowledged works as our baseline: Deep Compression (Han, Mao, and Dally 2015), NeST (Dai, Yin, and Jha 2017), ADMM-NN (Ren et al. 2019), SSS (Huang and Wang 2018) and Fine-grained pruning (Mao et al. 2017).

	Method	Top-5 Accuracy	CONV Compression rate
VGG-16	Deep compression	88.7%	3.5×
	NeST	90.1%	6.5×
	ADMM-NN	88.9%	10.2×
	Our's (pattern + connectivity)	**91.5%**	**7.0**×
ResNet-50	Fine-grained pruning	92.3%	2.6×
	SSS-32	91.8%	1.4×
	SSS-26	90.8%	1.6×
	ADMM-NN	92.3%	7.0×
	Our's (pattern + connectivity)	**92.6%**	**3.9**×

Table 4: *PCONV* and structured pruning comparison on ImageNet using VGG-16 and ResNet-50. We select widely-acknowledged works as our baseline: ThiNet (Luo, Wu, and Lin 2017), APoZ (Hu et al. 2016) and Efficient ConvNet (Li et al. 2016)

	Method	Top-5 Accuracy	CONV Compression rate
VGG-16	ThiNet	89.5%	1.1×
	APoZ	87.6%	2.0×
	Our's (pattern + connectivity)	**91.5%**	**7.0**×
ResNet-50	ThiNet-50	90.0%	2.0×
	ThiNet-30	88.3%	3.3×
	Efficient ConvNet	91.1%	1.4×
	Our's (pattern + connectivity)	**92.6%**	**3.9**×

Cifar-10 Dataset

Table 5 and Table 6 illustrate the Top-1 accuracy comparison on joint pattern pruning and connectivity pruning, on VGG-16 and ResNet-50 using Cifar-10 dataset. For VGG-16, all kernels are 3×3. After applying SCPs on all kernels and $8.8\times$ uniform connectivity pruning, we achieve around $19.7\times$ weight reduction on convolution layers of VGG-16. For ResNet-50, a portion of kernels are 1×1 besides the majority of 3×3 kernels. We apply pattern pruning on all 3×3 ones, and apply uniform $8\times$ connectivity pruning on all kernels. We achieve $11.5\times$ weight reduction on convolution layers.

Table 5: *PCONV* and non-structured pruning comparison on Cifar-10 using VGG-16. We select widely-acknowledged methods – Iterative pruning in (Han et al. 2015) and get the reproduced results from (Liu et al. 2018), and One Shot Pruning results in (Liu et al. 2018).

	Method	Top-1 Accuracy	CONV Compression rate
VGG-16	Iterative Pruning	93.3%	3.5×
	One Shot Pruning	93.7%	5.0×
	Our's (pattern + connectivity)	**93.8%**	**19.7**×
ResNet-50	One Shot Pruning	93.6%	2.5×
	One Shot Pruning	92.7%	3.3×
	Our's (pattern + connectivity)	**95.6%**	**11.5**×

Table 6: *PCONV* and structured pruning comparison on Cifar-10 using VGG-16 and ResNet-50. We select widely-acknowledged works as our baseline: Efficient ConvNet (Li et al. 2016), 2PFPCE (Min et al. 2018), AMC (He et al. 2018), NISP (Yu et al. 2018), FPGM (He et al. 2019)

	Method	Top-1 Accuracy	CONV Compression rate
VGG-16	2PFPCE	92.8%	4.0×
	Efficient ConvNet	93.4%	2.7×
	FPGM	93.2%	1.6×
	Our's (pattern + connectivity)	**93.8%**	**19.7**×
ResNet-50	AMC	93.5%	1.7×
	NISP	93.2%	1.7×
	Our's (pattern + connectivity)	**95.6%**	**11.5**×

Visualization Interpretation for Verification

In our main paper, by adopting *guided-backpropagation* technique, we generate one set of visualization results of four images from ImageNet dataset to demonstrate the image enhancement ability of *PCONV*. In this part, we extend the visualization results by adopting two more widely used visualization methods, *integrated gradients* (Mukund, Ankur, and Qiqi 2017) and *inverted representation* (Aravindh and Andrea 2015). By using three visualization techniques, we provide strong evidence that *PCONV* can effectively capture more image details.

Integrated gradients attribute a complex DNN's prediction to its input features. Integrated gradients differentiate between artifacts that stem from perturbing the data, a misbehaving model and a misbehaving attribution method. This is distinguished from the previous visualization methods, which are characterized by intuitive design and empirical evaluations (Mukund, Ankur, and Qiqi 2017). Hence, applying integrated gradients is a desired visualization methodology to verify the advancement of *PCONV*.

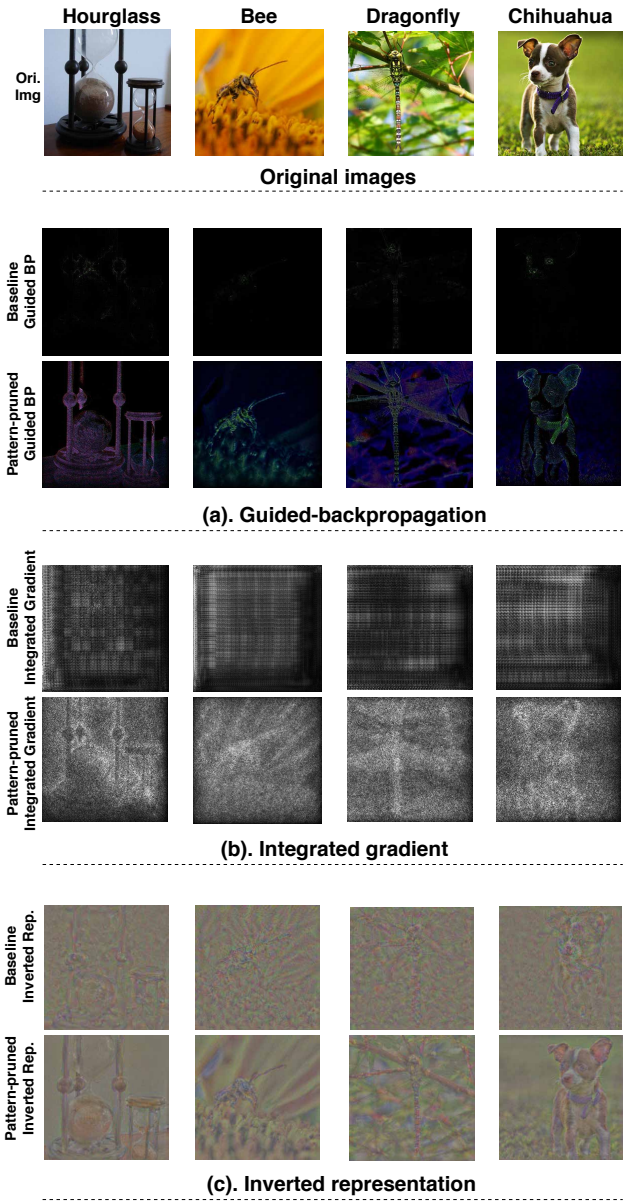

Figure 10: Visualization of intermediate results in the original VGG-16 baseline model and pattern pruned VGG-16 model through three different methods: (a). Visualization of the *saliency map* of *gradient images*; (b). Visualization of *integrated gradients*; (c). Visualzation of *inverted representation*.

The integral of integrated gradients is efficiently approximated via a summation. We sum the gradients at points occurring at sufficiently small intervals along the straight-line path from the baseline's x' to the input x:

$$\text{IntegratedGrads}_{i}^{\text{approx}}(x) ::= (x_i - x_i') \times \sum_{k=1}^{m} \frac{\partial F\left(x' + \frac{k}{m} \times (x - x')\right)}{\partial x_i} \times \frac{1}{m} \quad (14)$$

Here m is the number of steps in the Riemann approximation of the integral. In practice, the recommended value of m is $m \in [20, 300]$ (Mukund, Ankur, and Qiqi 2017). We set $m = 100$ steps and compute the integrated gradients.

Figure 10 (b) visualizes the integrated gradients from the original VGG-16 model and the pattern pruned VGG-16 model. By contrast, the pattern pruned VGG-16 model learns more comprehensive information, according to the visualization of integrated gradients.

Inverted representation originates from the following question: given an encoding of an image, to which extent is it possible to reconstruct the image itself through the DNN? (Aravindh and Andrea 2015) has shown that several layers in DNN retrain photographically accurate information about the image, with different degrees of geometric and photometric invariance. Hence, we utilize inverted representation to interpret the difference between original VGG-16 model and pattern pruned VGG-16 model.

The visualization results of inverted representation are demonstrated in Figure 10 (c). We can clearly see that the pattern pruned VGG-16 model retains more photographically accurate information.

After visualizing the original DNN models and pattern pruned DNN models through different visualization methods, we conclude that by applying our designed SCPs, pattern pruning enhances DNNs' image processing ability.