PCNN: Pattern-based Fine-Grained Regular Pruning towards Optimizing CNN Accelerators

Zhanhong Tan[1], Jiebo Song[1], Xiaolong Ma[2], Sia-Huat Tan[1], Hongyang Chen[3], Yuanqing Miao[1], Yifu Wu[4], Shaokai Ye[4], Yanzhi Wang[2], Dehui Li[4], Kaisheng Ma[1,4]
[1]Tsinghua University, [2]Northeastern University, [3]Xi'an Jiaotong University,
[4]Institute for Interdisciplinary Information Core Technology

***Abstract*—Weight pruning is a powerful technique to realize model compression. We propose PCNN, a fine-grained regular 1D pruning method. A novel index format called Sparsity Pattern Mask (SPM) is presented to encode the sparsity in PCNN. Leveraging SPM with limited pruning patterns and non-zero sequences with equal length, PCNN can be efficiently employed in hardware. Evaluated on VGG-16 and ResNet-18, our PCNN achieves the compression rate up to 8.4× with only 0.2% accuracy loss. We also implement a pattern-aware architecture in 55nm process, achieving up to 9.0× speedup and 28.39 TOPS/W efficiency with only 3.1% on-chip memory overhead of indices.**

I. Introduction

Convolutional Neural Networks (CNN) have been developing rapidly in many applications, such as image classification [1], object detection [2] and natural language processing [3]. When neural networks achieve higher accuracy with increasing computation and parameters [4] [5], many accelerators [6] [7] [8] [9] implemented on ASIC or FPGA with abundant parallel units are proposed to deploy neural networks on edge devices. Although these accelerators can improve throughput, it is still very challenging to deal with tremendous computation and transfer large amounts of data from DRAM to the on-chip memory. Weight pruning is an effective approach to reduce model size. However, the methods proposed in early works [10] [11] prune weights randomly (named irregular pruning). Irregular pruning needs to store weights in Compressed Sparse Column (CSC) format [12], which leads to considerable extra indices to represent weights. Besides, imbalance of workload among different computing units in the highly parallel architecture causes resource underutilization, which prevents the hardware from fully leveraging the advantages of weight pruning.

Contrary to irregular pruning, regular pruning is more hardware-friendly. Different granularities are explored in previous works, including irregular sparsity (0D), vector-level sparsity (1D), kernel-level sparsity (2D), and filter-level sparsity (3D) [13] [14] [15]. However, the accuracy drops as the increment of pruning regularity [13]. Therefore, it is imperative to find a new granularity to achieve regular pruning with high accuracy.

In this paper, we firstly propose Sparsity Pattern Mask (SPM), a novel kernel-level format to index non-zero weights of each kernel. Based on SPM, we present PCNN, a fine-grained regular 1-D pruning method with the identical number of non-zero weights in each kernel of one layer. In this case, the computation workload of different convolution windows can be balanced with a few number of kernel patterns in each layer.

To push PCNN to be more regular, we further leverage multiple knapsack framework to distillate patterns (i.e., fewer patterns). In this way, we can employ fewer bits to encode SPM.

Based on PCNN, we implement a pattern-based architecture in 55nm process. Specialized memory optimization is proposed to map the PCNN-based workload with SPM code in hardware. Leveraging the benefit of PCNN, a sparsity-aware PE array is designed to achieve highly parallel computation with a delicate pipeline. Moreover, the sparsity-aware PE array can process sparse weights and activations simultaneously.

In experiments, combined with other pruning methods like channel pruning, the PCNN algorithm can achieve up to 34.4× reduction of model size with negligible accuracy loss, which proves its orthogonality feature. With the power of PCNN, the proposed pattern-aware architecture fully leverages benefits from PCNN and shows up to 9× speedup and 28.39 TOPS/W efficiency with only 3.1% memory overhead of indices.

II. The Proposed PCNN Framework

A. Descriptions of PCNN

Generally, there are some zeros in one convolution kernel. To avoid storing zeros in the memory, we propose a kernel-level index format called Sparsity Pattern Mask (SPM) to encode sparse weights. As shown in Figure 1, the non-zero weights in a kernel are distributed in a specific pattern, which can be encoded with an SPM index. As a result, only the non-zero values and the SPM index need to be stored. Different from Compressed Sparse Column (CSC) format [12] which employs one index to each weight, we only need to apply one SPM index to each kernel. In contrast, the index overhead is much smaller.

However, there are $\sum_{i=0}^{9}\binom{9}{i} = 512$ in total patterns in 3×3 kernels, indicating that the bitwidth of SPM index is 9. Therefore, in order to reduce the bitwidth overhead of SPM encoding and simultaneously maintain the balanced workload for parallel computation, we propose PCNN which keeps identical sparsity in each kernel of one layer (i.e., the numbers of zeros in different kernels are the same). In this way, the number of patterns is reduced to $\max\{\binom{9}{0}, \binom{9}{1}, ..., \binom{9}{9}\} = 126$. In Figure 2, there are even some redundant patterns when we

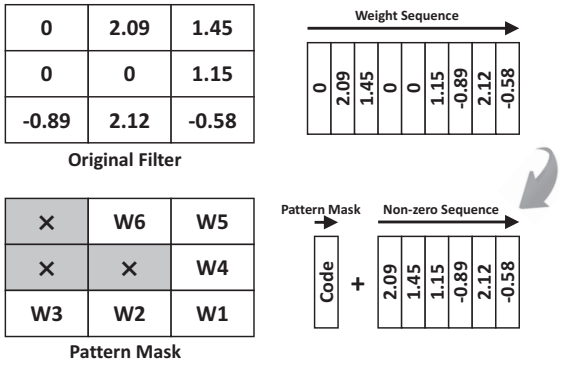


Fig. 1: Top: the original representation for a kernel. Bottom: pattern-based representation using an SPM index and the non-zero sequence.

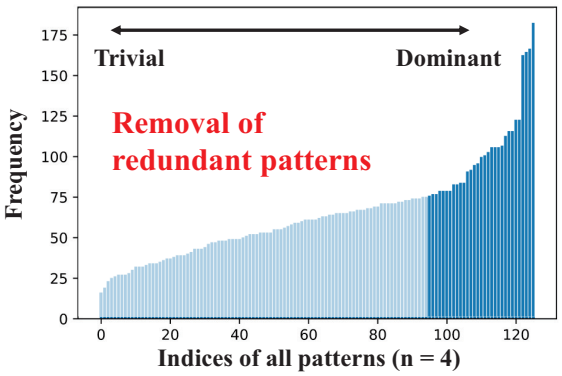


Fig. 2: Pattern distribution in CONV4 of VGG-16, where the number of non-zero is 4. Thus there are 126 patterns in total.

apply PCNN, which means that the number of patterns can be ulteriorly reduced to some extent.

Based on the above considerations, we expect to find the optimal pruning manner to deploy PCNN with appropriate sparsity and the number of patterns. Consequently, we establish a framework to describe PCNN with the following terminologies. The set $S = \{s_1, s_2, ..., s_l\}$ is the kernel sparsity of each convolution layer, determined by n_l/K_l, where n_l denotes the number of non-zero values while K_l is the kernel size for each layer.

Next, the set $\boldsymbol{F} = \{F_1, F_2, ..., F_l\}$ is a full collection of pattern sets for each layer and the extracted one $\boldsymbol{P} = \{P_1, P_2, ..., P_l\}$ is extracted from $\boldsymbol{F}$ without redundant patterns as shown in Figure 2. Thus, in the PCNN learning framework, we intend to explore the appropriate s_l and P_l for each layer, aiming to achieve a smaller model with fewer patterns.

B. KP-based Pattern Distillation

Problem modeling. As mentioned above, we intend to further reduce the number of patterns in our PCNN framework. Thus, we employ pattern distillation to choose the dominant ones. Pattern distillation means we have to select finite valuable patterns from a candidate set, which perfectly corresponds with the core idea of the knapsack problem.

With a given Convolutional Neural Network containing N convolution layers, W_l denotes the weights in the l-th convolution layer and the set $\boldsymbol{W} = \{W_1, ..., W_l, ..., W_N\}$ is a weight collection of N layers. n denotes the number of non-zero weights in each kernel. We formulate pattern distillation as:

$$\begin{aligned} &\underset{x_{li}}{min} \sum_l \sum_j \left\| w_{lj} - \Pi_{P_l}^{w_{lj}} \right\|_2, \\ &s.t. \quad \sum x_{li} \leqslant V_l, x_{li} \in \{0,1\}, i = 1, ..., t_n, \\ &P_l = \bigcup p_{li} x_{li}, \ p_{li} \in F_n, \\ &l = 1, ..., N, \ j = 1, ..., N_l, \end{aligned} \tag{1}$$

where w_{lj} is the j-th kernel in W_l and N_l is the number of kernels in W_l. F_n denotes the full set or candidate set of the patterns that have uniform sparsity and t_n denotes the total number of patters in F_n (i.e., $t_n = |F_n|$). P_l is the selected patterns in the l-th layer that is derived form F_n. Hyper-parameter V_l is the desired number of the selected patterns ($V_l = |P_l|$). The i-th pattern in F_n to P_l is selected when $x_{lj} = 1$, and vice versa. $\Pi_{P_l}^{w_{lj}}$ is a projection function that matches w_{lj} to the nearest pattern in P_l by keeping top n absolute values.

The pattern distillation problem is similar to the knapsack problem (KP). If we regard each w_{lj} as a single knapsack, then the capacity of each knapsack is 1. In other words, we can only choose one pattern from the candidate set for w_{lj}. Representing $\boldsymbol{W}$ with the selected patterns, the problem can be regarded as the multiple knapsack problem (MKP). Particularly, since all capacities of w_{lj} are 1, KP-based pattern selection is a multiple knapsack problem with identical capacities (MKP-1).

Solution with greedy algorithm. In the case of our special problem, we propose an efficient greedy algorithm to solve KB-based pattern optimization. For each layer, we first select the most valuable pattern for each w_{lj} and then collect the number of patterns that have been chosen. Finally, we keep the patterns that have the highest frequency (top V_l patterns). Details are shown in Algorithm 1. As a result, the set P_l will contain a fewer number of patterns.

Algorithm 1: Pattern distillation algorithm

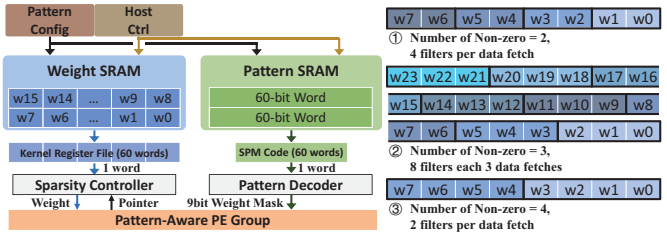
Input: F_n, V_l, $l = 1, ..., N$;
Initialization: $P_l = \emptyset$, $l = 1, ..., N$;
for $l \leftarrow 1$ **to** N **do**
  $N_i = 1, i = 1, ..., t_n$;
  for $j \leftarrow 1$ **to** N_l **do**
    for $i \leftarrow 1$ **to** t_n **do**
      if w_{lj} *is nearest to* p_i **then**
        $N_i = N_i + 1$;
      end
    end
  end
  $\{idx_1, ..., idx_{V_l}\} = max\_idx\_of\_sorted(N_1, ..., N_{t_n})$;
  for $i \leftarrow 1$ **to** V_l **do**
    $P_l = P_l \bigcup p_{idx_i}$;
  end
end
Return: P_l, $l = 1, ..., N$.

III. The Architecture for PCNN-Based Computing

A. Mapping PCNN in Memory with SPM

Figure 3a is the overall architecture for PCNN-based computation. Pattern Config (PaC) provides information of kernel sparsity and employs SPM mapping table for the decoder. In the pattern pruning framework, a kernel is denoted by a corresponding SPM code and a non-zero sequence, stored separately in Weight SRAM and Pattern SRAM. For a 3×3 kernel, the length of the non-zero sequence ranges from 1 to 9. Therefore, the sizes of kernel and SPM registers are 60-word which can integrally store kernels that contain 1 to 6 non-zero weights. For other sparsities, we pad zeros to align the memory. A kernel with non-zero weights is fetched into the register and the corresponding SPM code is simultaneously decoded into a 9-bit weight mask. After generating weight pointers based on the mask, the pattern-aware PE group will properly fetch weights from the kernel register with sparsity pointers which will be presented in the next subsection.

Figure 3 is the memory layout for PCNN. In different sparsity cases, weights are similarly stored in order. The host controller can delicately access memory, fetching data to fill in the registers according to the kernel sparsity configured in PaC.



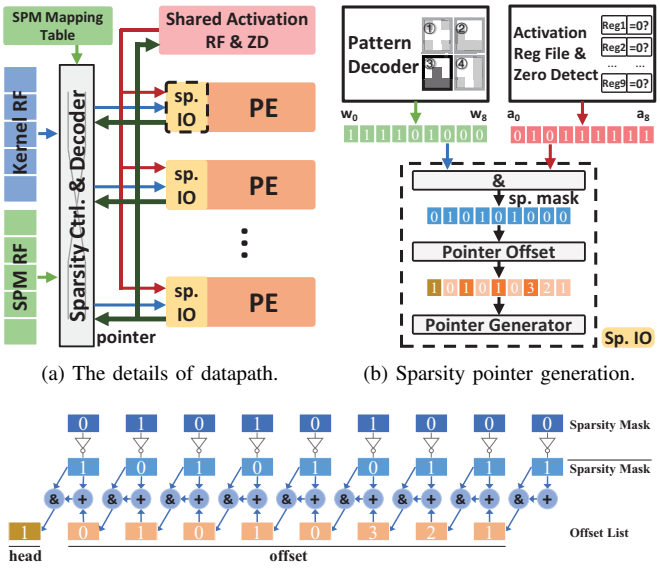
(a) Overall architecture. (b) Storing format for weights.

Fig. 3: Memory design for weights and patterns in the pattern-aware architecture.

B. Sparsity-Aware Processing Element Group

A detailed overview of the pattern-aware PE group is shown in Figure 4, where the sparsity IO can generate pointers based on the weight mask from the SPM decoder to fetch weights properly. In our design, we implement 64 PEs with 4 MAC units in each one. Consequently, our architecture can perform at most 256 MACs per cycle. Besides weight sparsity, we also leverage activation sparsity to further improve computing efficiency. Therefore, we employ shared-activation datapath to balance the workload from activations.

The sparsity pointer generation is shown in Figure 4b. Both the weight mask and the activation mask are transferred to the sparsity IO and then the sparsity mask is generated. Later, pointer offsets can be obtained with the sparsity mask (Figure 4c). There is an adder–AND chain to attain nine offsets, each of which denotes the distance to the nearest zero. The offsets can be elaborated as follows. Firstly, NOT operation is applied to each bit of sparsity mask. Secondly, we can obtain the pointer offsets by accumulating the number of zeros between every two non-zero weights. With pointer offsets,



(a) The details of datapath. (b) Sparsity pointer generation.

(c) The calculation of pointer offset.

Fig. 4: Overview of the pattern-aware PE group.

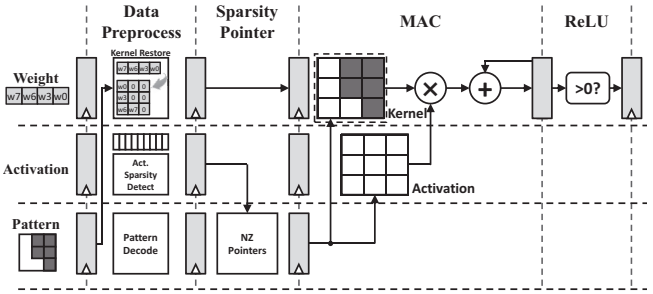


Fig. 5: Pipeline strategy for pattern-aware architecture.

we can attain the corresponding pointers to fetch the needed weights from the kernel register. With the help of PCNN and the shared-activation dataflow, the workloads of weights and activations in different PEs are identical, contributing to higher resource utilization and better parallelism.

Figure 5 shows the pipeline strategy in the proposed pattern-aware PE. In order to achieve high throughput, all the operations are pipelined. The first stage is the data pre-process stage. Weights are restored to the original kernel according to the SPM indices. Activations in a convolution window are loaded into the registers and activation sparsity masks are generated simultaneously. In the second stage, non-zero pointers are generated with the calculated offsets, which will select the effectual workload in the next stage to perform MACs. Last, when all partial sums from various input channels are added up together, ReLU is employed to attain the final result.

IV. Evaluation

A. Methodology

Setup for evaluating the proposed PCNN. We summarize our PCNN results for CNN model compression on a series of benchmarks, including VGG-16 [5] on CIFAR10 [16] and ImageNet [1], and ResNet-18 [4] on CIFAR10. We initialize

Benchmark	Top1 acc	Top1 acc Loss	CONV FLOPs	FLOPs Pruned	CONV Parameters	Compression (weight)	Compression (weight+idx)
VGG-16, Baseline	93.54%	-	3.13×10^8	-	1.47×10^7	-	-
VGG-16, n = 4	93.79%	+0.25%	1.39×10^8	56.5%	0.65×10^7	2.3×	2.2×
VGG-16, n = 3	93.58%	+0.04%	1.04×10^8	66.7%	0.49×10^7	3.0×	2.9×
VGG-16, n = 2	93.52%	-0.02%	0.30×10^8	77.8%	0.33×10^7	4.5×	4.1×
VGG-16, n = 1	93.07%	-0.21%	0.35×10^8	88.9%	0.16×10^7	9.0×	8.4×
VGG-16, Various setting[a]	93.33%	-0.21%	0.35×10^8	88.8%	0.16×10^7	9.0×	8.4×

[a] n in various layers: 2-1-1-1-1-1-1-1-1-1-1-1-1 with 32 patterns in n = 2 layers and 8 patterns in n = 1 layers.

TABLE I: Pruning rate and accuracy of different n for VGG-16 on CIFAR10.

Benchmark	Top1 acc	Top1 acc Loss	CONV FLOPs	FLOPs Pruned	CONV Parameters	Compression (weight)	Compression (weight+idx)
ResNet-18, Baseline	96.58%	-	5.55×10^8	-	1.12×10^7	-	-
ResNet-18, n = 4	96.58%	+0.06%	2.50×10^8	54.5%	0.51×10^7	2.2×	2.1×
ResNet-18, n = 3	96.38%	-0.20%	1.89×10^8	65.5%	0.38×10^7	3.0×	2.8×
ResNet-18, n = 2	96.15%	-0.43%	1.28×10^8	76.7%	0.26×10^7	4.3×	4.0×
ResNet-18, n = 1	95.55%	-1.03%	0.67×10^8	88.0%	0.14×10^7	7.9×	7.3×
ResNet-18, Various setting[a]	95.83%	-0.75%	0.86×10^8	84.5%	0.14×10^7	7.9×	7.3×

[a] n in various layers in bottle: 2-2-2-1-1-1-1-1-1-1-1-1-1-1-1-1-1 with 32 patterns in n = 2 layers and 8 patterns in n = 1 layers.

TABLE II: Pruning rate and accuracy of different n for ResNet-18 on CIFAR10.

Benchmark	Top1 acc	Top1 acc Loss	CONV FLOPs	FLOPs Pruned	CONV Parameters	Compression (weight)	Compression (weight+idx)
VGG-16, Baseline	92.10%	-	6.82×10^9	-	1.47×10^7	-	-
VGG-16, n = 5	92.47%	+0.37%	0.85×10^9	44.4%	0.82×10^7	1.8×	1.7×
VGG-16, n = 4	92.45%	+0.35%	0.68×10^9	56.5%	0.65×10^7	2.3×	2.2×

TABLE III: Pruning rate and accuracy of different n for VGG-16 on ImageNet.

our learning framework with pre-trained models on PyTorch, following the pattern distillation. After that, an Alternating Direction Method of Multipliers (ADMM) [17] is employed to fine-tune our model. Considering that convolution layers are getting more and more dominant at present, we mainly focus on convolution layers.

Setup for evaluating our PCNN-based architecture. Based on PCNN encoded with SPM, we implement the pattern-aware architecture with RTL and evaluate VGG-16 based on PCNN with VCS to obtain the performance. The area and power of our architecture are attained with Design Compiler in UMC 55nm standard power CMOS process.

B. Evaluating the Kernel Sparsity and the Number of Patterns

In this part, we study different choices of kernel sparsity for VGG-16 and ResNet-18. In ResNet-18, we only process the layers with 3×3 filters and ignore 1×1 ones which are too accuracy-sensitive. We set n as 1, 2, 3, and 4 in all layers with at most 8, 32, 32, and 32 patterns respectively.

Table I shows the pattern pruning results of various n on the VGG-16 model on CIFAR10. PCNN leads to less than 0.5% accuracy loss even when there is one weight left in each filter. When we apply a different sparsity setting over layers, accuracy can be improved with similar compression rates and speedup.

Similar results can be achieved for ResNet-18 on CIFAR10 as shown in Table II. Within 0.5% accuracy loss, pattern pruning achieves the compression rate ranging from 1.78× to 4.31× with the unified sparsity setting (n = 4, 3, and 2). Also, when various sparsity settings are applied, we can achieve better performance than the unified counterpart with a similar compression rate of 9×. Furthermore, the results for VGG-16 on ImageNet are shown in Table III and we achieve 1.8× compression rate and 2.25× speedup with no harm to accuracy.

Note the last columns of Table I∼III containing the actual compression rate considering the overhead of indies. With PCNN, there are marginal compression rate drops due to kernel-level SPM indices. On the contrary, for irregular pruning, taking VGG-16 with n = 4 as an example, the actual compression rate is 2.0×, three times as low as ours.

Later, we further restrain the number of patterns in each layer to study how regularity impacts accuracy. As Table IV shows, we evaluate n = 4 and n = 2 with full patterns, 32, 16, 8, and 4 patterns. We use full patterns as baselines with 93.79% and 93.52% for n = 4 and n = 2 respectively and the weight compression rates are 2.3× and 4.5×. The results show that there are no obvious accuracy drops with fewer patterns, which can help us to save the overhead of index storing. While in the higher sparsity case, the accuracy is more sensitive to the decrease of patterns. Actually, in the cases with high sparsity, we do not need to focus too much on the number of patterns because the loss of compression is little with SPM in PCNN. Averagely, 16 patterns are enough to maintain accuracy with less index overhead.

C. Comparison to Other Regular Compression Methods

In this part, we compare PCNN to other pruning methods in other works for VGG-16 and ResNet-18 on CIFAR10. For various baselines used in different works, we employ

the accuracy loss relative to the respective baseline. The comparison for VGG-16 is shown in Table V. Our method can remarkably compress parameters and reduce FLOPs simultaneously with negligible accuracy loss. As for ResNet-18 shown in Table VI, PCNN also achieves better performance than other coarse-grained pruning. Especially, PCNN enjoys higher FLOPs reduction.

D. The Orthogonality to Other Compression Methods

Combined with kernel level pruning. As shown in Table VII, we perform some experiments for VGG-16 on ImageNet in the case where n = 5. We apply 2.4× and 4.1× kernel pruning with 1.8× compression rate from PCNN to achieve fused pruning. Results show that the PCNN is orthogonal to kernel pruning and the combination of them achieves promising results.

Combined with channel level pruning. Channel level pruning has been proved to be more regular with a higher compression rate but less friendly to accuracy [13]. As results shown in Table VIII, the fused compression achieves 34.4× compression rate contributed by 3.75× PCNN compression and 9× channel pruning. Therefore, PCNN is also orthogonal to channel pruning.

E. Evaluation on Pattern-Aware Architecture

The overhead evaluation. The layout of our pattern-aware architecture is shown in Figure 6 and the overhead of each component is listed in Table IX. The power consumption is measured under the circumstance of 300 MHz on-chip frequency and 1 V voltage. Contrary to irregular pruning, PCNN requires an SPM index for each kernel rather than weight. With PCNN, a 128KB weight SRAM is employed holding up to 32768 kernels of 3×3 size with 4 non-zeros with 8-bit quantization for common cases. In this case, a 4K pattern SRAM is enough for the workload with 16 patterns in each layer. As shown in analyses in section IV, 16 patterns are sufficient to maintain accuracy. Consequently, this architecture introduces only 3.1% memory overhead to store indices. The pattern SRAM only takes up 1.9% area and 1.9% power of the whole chip. according to Table IX. In comparison to the irregular pruning based architecture like EIE [12], 64KB index SRAM is needed to denote 128K weights.

The performance evaluation. We evaluate VGG-16 based on PCNN with sparsities of 55.6% (n = 4), 66.7% (n = 3), 77.8% (n = 2), and 88.9% (n = 1). The average activation sparsity is 0.8. The results show that we can achieve 2.3×, 3.1×, 4.5×, and 9.0× speedup compared to the dense counterpart. With 256 MAC units running in 300 MHz frequency and 1V voltage, our pattern-aware architecture achieves 3.15 TOPS/W (no sparsity) ∼ 28.39 TOPS/W (88.9% sparsity) power efficiency. It can be observed that our patter-aware architecture can fully leverage the strengths of our PCNN method.

Benchmark		Relative acc	Compression (weight+idx)
n = 4	$\|P_n\|$=126 (full) **baseline**	-	2.14×
	$\|P_n\|$ = 32	+0.32%	2.18×
	$\|P_n\|$ = 16	+0.10%	2.20×
	$\|P_n\|$ = 8	-0.05%	2.21×
	$\|P_n\|$ = 4	-0.17%	2.23×
n = 2	$\|P_n\|$=36 (full) **baseline**	-	4.08×
	$\|P_n\|$ = 32	+0.00%	4.13×
	$\|P_n\|$ = 16	-0.22%	4.19×
	$\|P_n\|$ = 8	-0.54%	4.26×
	$\|P_n\|$ = 4	-0.71%	4.32×

TABLE IV: Comparison of $|P_n|$ for VGG-16 on CIFAR10. If no index, the compression is 2.3× and 4.5× for n=4 and n=5.

Benchmark	Relative acc	FLOPs	Compression
PCNN	+0.01%	66.7%	3.0×
PCNN	-0.21%	88.8%	9.0×
Filter pruning [18]	+0.15%	33.3%	2.8×
Network slimming [19]	+0.14%	51.0%	8.7×
try-and-learn b=1 [20]	-1.10%	82.7%	2.2×
IKR [21][a]	-0.90%	84.7%	4.3×

[a] A VGG-16 inspired CNN containing 6 convoution and 2 FC layers

TABLE V: Comparison of various regular compression methods for VGG-16 on CIFAR10.

Benchmark	Relative acc	FLOPs	Compression
PCNN	-0.20%	66.0%	3.0×
PCNN	-0.78%	82.5%	7.1×
Band-limited [22]	-1.67%	-	2.0×
try-and-learn b=4 [20]	-2.90%	76.0%	4.6×

TABLE VI: Comparison of various regular compression methods for ResNet-18 on CIFAR10.

Benchmark	Relative acc	Compression
PCNN n = 5	+0.38%	1.8×
PCNN n = 5 + Kernel Pruning-A	+0.28%	4.4×
PCNN n = 5 + Kernel Pruning-B	-0.27%	7.3×

TABLE VII: Combined with kernel-level pruning for VGG-16 on ImageNet.

Benchmark	Relative acc	Compression
PCNN + Channel Pruning-A	-0.02%	34.4×
PCNN + Channel Pruning-B	-0.46%	50.3×
Structured ADMM [23]	-0.60%	50.0×
SNIP [24]	-0.45%	20.0×
Synaptic Strength [25]	+0.43%	25.0×

TABLE VIII: Combined with channel-level pruning for VGG-16 on CIFAR10.

V. Conclusion

We present PCNN, a novel fine-grained regular pruning method that uses SPM to encode the sparsity. Contrary to irregular pruning, the sparsity of every kernel is the same in each layer, which can achieve regularity and maintain fine granularity. Experiments show that 8× ∼ 9× compression rate and computing speed-up can be achieved with less than 1% accuracy loss. Additionally, pattern pruning can be easily

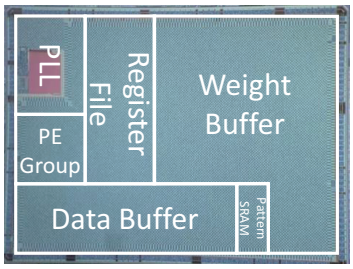


Fig. 6: The layout of pattern-aware architecture.

Component	Area (mm^2)	Share	Power(mW)	Share
Overall	8.00	100%	48.7	100%
Data SRAM	3.25	40.6%	13.7	28.2%
Weight SRAM	2.48	31.0%	15.6	32.1%
Pattern SRAM	0.19	2.4%	0.9	1.9%
Register File	1.58	19.8%	13.6	27.4%
PE group	0.50	6.2%	4.9	10.0%

TABLE IX: Area and power characteristics of the chip (not including PLL and IO).

combined with coarse-grained pruning methods, achieving 34.4× compression ratio with negligible accuracy loss. With SPM, we can deploy indices at the kernel level rather than weight level, which helps to save a great amount of memory overhead. For computation, with only 3.1% memory overhead for indices, the proposed architecture can achieve full parallelism and obtain up to 9× speedup and 28.39 TOPS/W efficiency based on the PCNN.

REFERENCES

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, May 2017.

[2] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.

[3] Deng Li Yu Dong Dahl George Mohamed Abdel-rahman & Jaitly Navdeep Hinton, Geoffrey. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, Nov 2012.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Sun Jian. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[5] Yoshua Bengio and Yann LeCun, editors. *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[6] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '14, pages 269–284, New York, NY, USA, 2014. ACM.

[7] Yu Hsin Chen, Tushar Krishna, Joel S. Emer, and Vivienne Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits*, 52(1):127–138, Jan 2017.

[8] Jinmook Lee, Changhyeon Kim, Sanghoon Kang, Dongjoo Shin, Sangyeob Kim, and Hoi-Jun Yoo. UNPU: an energy-efficient deep neural network accelerator with fully variable weight bit precision. *J. Solid-State Circuits*, 54(1):173–185, 2019.

[9] Caiwen Ding, Shuo Wang, Ning Liu, Kaidi Xu, Yanzhi Wang, and Yun Liang. REQ-YOLO: A resource-aware, efficient quantization framework for object detection on fpgas. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA 2019, Seaside, CA, USA, February 24-26, 2019*, pages 33–42, 2019.

[10] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[11] Suraj Srinivas and R. Venkatesh Babu. Data-free parameter pruning for deep neural networks. In *Proceedings of the British Machine Vision Conference 2015, BMVC 2015, Swansea, UK, September 7-10, 2015*, pages 31.1–31.12, 2015.

[12] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally. EIE: efficient inference engine on compressed deep neural network. In *43rd ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2016, Seoul, South Korea, June 18-22, 2016*, pages 243–254, 2016.

[13] Huizi Mao, Song Han, Jeff Pool, Wenshuo Li, Xingyu Liu, Yu Wang, and William J. Dally. Exploring the granularity of sparsity in convolutional neural networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 1927–1934, 2017.

[14] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. Structured pruning of deep convolutional neural networks. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 13(3):32:1–32:18, 2017.

[15] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 2074–2082, 2016.

[16] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[17] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, Jonathan Eckstein, et al. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, 3(1):1–122, 2011.

[18] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

[19] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 2755–2763, 2017.

[20] Qiangui Huang, Shaohua Kevin Zhou, Suya You, and Ulrich Neumann. Learning to prune filters in convolutional neural networks. In *2018 IEEE Winter Conference on Applications of Computer Vision, WACV 2018, Lake Tahoe, NV, USA, March 12-15, 2018*, pages 709–718, 2018.

[21] Maurice Yang, Mahmoud Faraj, Assem Hussein, and Vincent C. Gaudet. Efficient hardware realization of convolutional neural networks using intra-kernel regular pruning. In *48th IEEE International Symposium on Multiple-Valued Logic, ISMVL 2018, Linz, Austria, May 16-18, 2018*, pages 180–185, 2018.

[22] Adam Dziedzic, John Paparrizos, Sanjay Krishnan, Aaron J. Elmore, and Michael J. Franklin. Band-limited training and inference for convolutional neural networks. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, pages 1745–1754, 2019.

[23] Yanzhi Wang, Shaokai Ye, Zhezhi He, Xiaolong Ma, Linfeng Zhang, Sheng Lin, Geng Yuan, Sia Huat Tan, Zhengang Li, Deliang Fan, Xuehai Qian, Xue Lin, and Kaisheng Ma. Non-structured DNN weight pruning considered harmful. *CoRR*, abs/1907.02124, 2019.

[24] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip H. S. Torr. Snip: single-shot network pruning based on connection sensitivity. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

[25] Chen Lin, Zhao Zhong, Wu Wei, and Junjie Yan. Synaptic strength for convolutional neural network. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.*, pages 10170–10179, 2018.