PNPU: An Energy-Efficient Deep-Neural-Network Learning Processor With Stochastic Coarse–Fine Level Weight Pruning and Adaptive Input/Output/Weight Zero Skipping

Sangyeob Kim, *Graduate Student Member, IEEE*, Juhyoung Lee, *Graduate Student Member, IEEE*, Sanghoon Kang, *Graduate Student Member, IEEE*, Jinmook Lee, *Member, IEEE*, Wooyoung Jo, and Hoi-Jun Yoo, *Fellow, IEEE*

***Abstract*—Recently, deep-neural-network (DNN) learning processors for edge devices have been proposed, but they cannot reduce the complexity of over-parameterized network during training. Also, they cannot support energy-efficient zero-skipping because previous methods cannot be performed perfectly in backpropagation and weight gradient update. In this letter, energy-efficient DNN learning processor PNPU is proposed with three key features: 1) stochastic coarse–fine level pruning; 2) adaptive input, output, weight zero skipping; and 3) weight pruning unit with weight sparsity balancer. As a result, PNPU shows 3.14–278.39 TFLOPS/W energy efficiency, at 0.78 V and 50 MHz with FP8 and 0%–90% sparsity condition.**

***Index Terms*—Deep learning (DL) accelerator, DL, deep neural network (DNN), pruning, training.**

I. Introduction

Recently, pretrained deep-neural-network (DNN) with a large-scale dataset is often used for a domain-specific task in edge devices, this case, accuracy is usually degraded. Fig. 1 shows the object classification performance when the pretrained model (VGG16 for ImageNet [1]) is used for different domain (UC-Merced [2]). It shows a low accuracy of 73.8% [3]. Therefore, adaptation to the new domain is required to recover the accuracy. When fine-tuning is applied to the new dataset, the accuracy increases to 94.3%, which is a 20.5% improvement over the pretrained model [3]. Therefore, DNN training at the edge is required for high accuracy. Besides, it is even more necessary because of protection for a personal dataset [4].

Also, the model trained with the general dataset contains a lot of information that is not necessary for domain-specific applications. To solve this problem, a weight pruning (WP) that removes parameters which are not important in the current domain has been proposed [5]. In detail, fine level pruning (FLP) [5] and coarse level pruning (CLP) [6], [7] are proposed. When one-time pruning which makes zeros in weight before fine-tuning is applied to [2] as shown in Fig. 2, the accuracy decreases by 0.5% compared to fine-tuning alone, and the weight compression ratio is improved by 31.9×, as shown in Fig. 3 [3].

Iterative pruning (IP) in Fig. 2 is proposed for a higher compression ratio, and it performs fine-tuning and pruning alternately [3]. When fine-tuning is performed on the UC-Merced dataset by applying the IP, the loss is 0.2% compared to the case of fine-tuning alone, and the compression ratio of the parameter is improved by 48.8×, as shown in Fig. 3 [3]. This is an increase of 1.53× compared to the case of one-time pruning. Therefore, IP enables energy-efficient sparse DNN acceleration for both inference and training. Besides, in the case

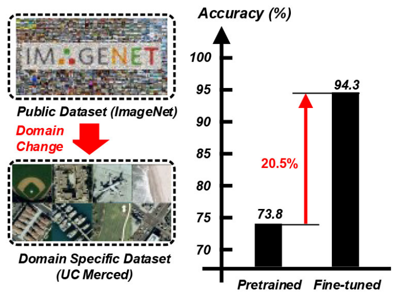


Fig. 1. Need for domain-adaption in edge devices.

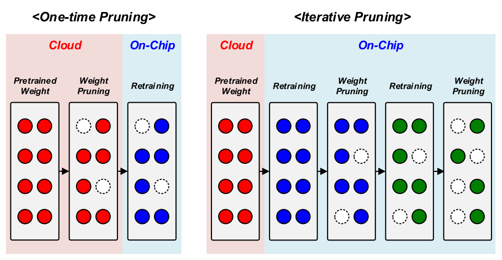


Fig. 2. One-time pruning and IP.

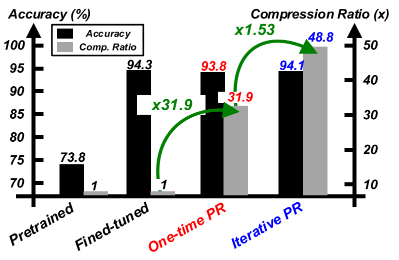


Fig. 3. Performance of one-time pruning and IP.

of scratch learning, IP is necessary, because there are no pretrained parameters to apply pruning.

The previous DNN learning processors support fine-tuning through input zero skipping [8], and input/output zero skipping [9]. However, the zero skipping supported by [8] and [9] cannot accelerate all three stages of DNN learning perfectly. Lee *et al.* [8] supported only input zero skipping, but there is no input sparsity due to batch normalization during backpropagation (BP) and weight gradient update (WG), as shown in Fig. 4. Kang *et al.* [9] supported dual zero skipping for input and output, but only output activation sparsity exists during BP and WG. Therefore, [9] enables only single zero skipping during BP and WG, reducing energy efficiency. Also, they cannot improve energy efficiency by handling the weight sparsity that can be generated with little accuracy loss. Therefore, the processor which supports WP and energy-efficient zero skipping for all stages of DNN learning is required.

However, there are three challenges to designing this processor.

Manuscript received August 15, 2020; revised November 9, 2020; accepted November 25, 2020. Date of publication December 1, 2020; date of current version January 20, 2021. This article was approved by Associate Editor Vinayak Honkote. This work was supported by the Basic Science Research Program through the National Research Foundation of Korea funded by the Ministry of Science, ICT and Future Planning under Grant NRF-2018R1A2A1A19023099. *(Corresponding author: Hoi-Jun Yoo.)*

The authors are with the School of Electrical Engineering, Korea Advanced Institute of Science and Technology, Daejeon 34141, South Korea (e-mail: hjyoo@kaist.ac.kr).

Digital Object Identifier 10.1109/LSSC.2020.3041497

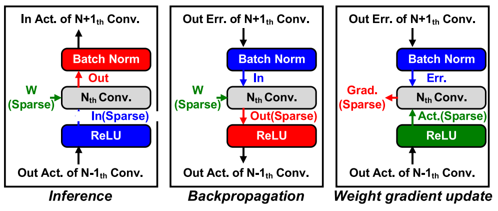


Fig. 4. Sparsity pattern of each stages of DNN learning.

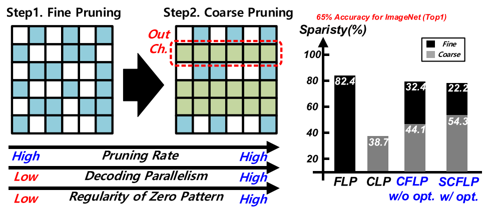


Fig. 5. CFLP and performance.

1) CLP requires a similarity comparison between different output channels. Therefore, 5.4× more computation amount is required than DNN learning (AlexNet, batch = 1, iteration = 1).
2) During DNN learning with IP, the sparsity ratio of input/output/weight is changed, so the efficiency of zero skipping is seriously degraded depending on the situation. For example, input zero skipping cannot be used in BP/WG because batch-norm removes zeros in the input error.
3) Utilization drop occurs due to weight sparsity imbalance. Lee *et al.* [8] and Kang *et al.* [9] handle the workload unbalancing by activation, but they cannot handle weight sparsity imbalance.

In this letter, we propose PNPU with the following features to solve design issues: 1) stochastic coarse–fine level pruning (SCFLP); 2) adaptive input/output/weight zero skipping (AIOWS); and 3) WP unit with weight sparsity balancer.

II. Stochastic Coarse–Fine Pruning

Recently, FLP [5] and CLP [6], [7] have been proposed to generate zeros in weight. Work from Han *et al.* [5] generates zeros of a random pattern by converting small values to zero. Ayinde and Zurada [6] calculated the similarity between different output channels and pruned all weights of output channel which is similar to others. Molchanov *et al.* [7] calculated L1-norm of each output channel, and converting all weights of output channel which has small L1-norm value to zeros. FLP generates high sparsity with little loss of accuracy because it simply converts small values to zero. When FLP and CLP are applied to the ResNet trained for CIFAR-10, FLP shows 93.30% weight sparsity (simulated by method from [5]) under 93.56% accuracy condition, and CLP shows 34.2% coarse weight sparsity [6] under 93.3% accuracy condition. However, CLP is much simpler to handle zero skipping in hardware, since it generates continuous zeros.

Fig. 5 shows coarse–fine level pruning (CFLP) to increase coarse level sparsity (AlexNet, ImageNet). First, FLP is performed to generate a lot of random pattern zeros, increasing the similarity between different output channels, and reducing L1-norm of each output channel. Then, CLP is applied to weights in which FLP is applied, and coarse level sparsity is increased by 5.4% compared to only CLP. Therefore, CFLP changes the weight in a form that includes a lot of consecutive zeros while maintaining high weight sparsity.

Besides, the structured pruning method is applied to increase the ratio of consecutive zeros as shown in Fig. 6. Previous CLP makes all parameters corresponding to a specific output channel to zeros. However, it is difficult to generate a large number of consecutive zeros because of the large accuracy degradation. Also, if the workload is divided by input channel division (ICD) for each core, communication between cores is required for pruning. We adopt structured pruning to eliminate communication and increase coarse level sparsity. Weights

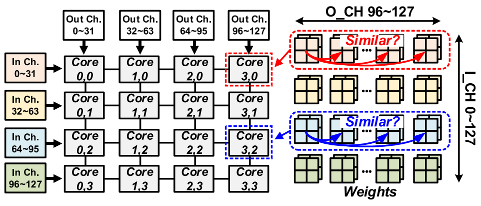


Fig. 6. Structured pruning for high coarse level sparsity.

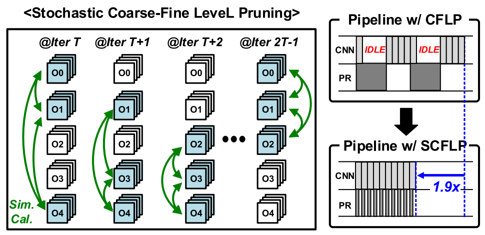


Fig. 7. SCFLP.

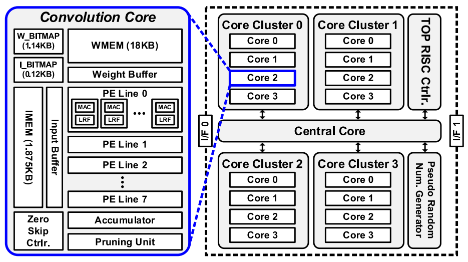


Fig. 8. Overall architecture of PNPU.

are divided by the number of cores in which ICD is applied, and similarity is compared only within core. Coarse sparsity increased to 55.8%, and communication is also removed because similarity and L1-norm are compared for a narrower range than conventional CLP.

However, CLP requires a large amount of computation because of similarity calculation between all output channel. When training AlexNet for ImageNet classification, CLP requires a 5.4× more computation amount than network training (Network = AlexNet, batch size = 1, iteration = 1). Therefore, the bottleneck occurs due to pruning, and the utilization of DNN core drops. In the case of DNN learning with IP, the throughput of the DNN core decreases by 0.53×.

Therefore, SCFLP is proposed to reduce the computational overhead of CLP, as shown in Fig. 7. One group was created by sampling among all output channels, and the similarity is compared only within the group, and more frequent pruning is performed to make comparisons between all output channel probabilistically. By reducing the number of samples for similarity comparison, 99.7% of the computation amount is reduced. Coarse sparsity decreases only 1.5% at the same accuracy by repeatedly performing stochastic grouping and pruning. As a result, throughput is improved by 1.9× compared to CFLP due to SCFLP.

III. Detailed Building Blocks

Fig. 8 shows the overall architecture of PNPU, which includes 16 AIOWS DNN cores, the aggregation core and batch normalization unit are integrated to support DNN learning. The batch size can be adjusted by instruction, and 1–16 batch sizes are supported by PNPU. Also, PRNG is integrated to allocate random output channels to each core. Each core consists of an SCFLP unit, an 8 × 8 processing element (PE) array, and an accumulator. Each PE consists of the AIOWS controller, single-FP16 and double-FP8 MAC and buffers. Each PE line is allocated for different input channel, and loads nonzero input features and nonzero weights corresponding to the inputs into the buffer. Each PE requests data from the buffer

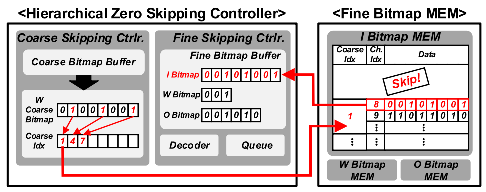


Fig. 9. HZS controller.

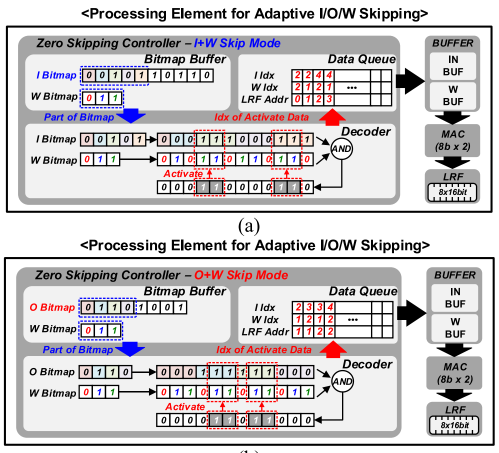


Fig. 10. Adaptive input/output/weight fine zero skipping controller.

and performs convolutional reuse. The partial sums from different PE line are accumulated to generate complete output by the accumulator.

Kang *et al.* [9] supported dual fine level zero skipping. Each PE requests very large memory access at high sparsity, and PE utilization decreases due to limited memory bandwidth. Therefore, PNPU supports hierarchical zero skipping (HZS) for efficient dual zero skipping.

Fig. 9 shows the HZS controller, the coarse skipping controller (CSC) requires the coarse bitmap and stores the location of 1 (noncoarse pruned output channel). By using this information, fine bitmaps for noncoarse pruned output channels are transferred to the fine skipping controller (FSC) in order. For example, FSC requires fine bitmaps for the 8th channel where the coarse index corresponds to 1, as shown in Fig. 9. Then, FSC requires fine bitmaps in the order of the 9th and 10th. After fetching all bitmap corresponding to specific coarse index (1st), FSC requires fine bitmaps for channels corresponding to the next coarse index (4th). Coarse skipping does not cause a memory bottleneck unlike fine-zero skipping, because a lot of operations are skipped at once simply. As a result, under 90% input/weight sparsity condition, HZS increases the throughput by 2.3× compared to fine-zero skipping only.

Fig. 10(a) shows the details about FSC for input/weight zero skipping. First, the fine bitmaps are fetched from the bitmap buffer. During the convolution, the weight is swept and multiplied with the input. FSC expands bitmaps according to the operation pattern. Thereafter, AND operation is performed between the extended bitmaps, and the activated operations are obtained. The index of the input and weight for activated operation and the local register file address to store are calculated, and they are stored in the queue. When the queue is completed, the input and weight in the buffer are fetched in the order of the index stored in the queue for convolution operation. Fig. 10(b) shows the details about FSC for output/weight zero skipping. First, the output bitmap and weight bitmap are fetched from the bitmap buffer. The bitmaps are expanded according to the operation pattern. In the same way as the input/weight zero skipping method, the queue is generated, and zero skipping convolution is performed.

For SCFLP, it is necessary to group output channels randomly. Fig. 11 shows random channel allocator (RCA) for SCFLP. The random sequence is generated from PRNG, and RCA assigned an output

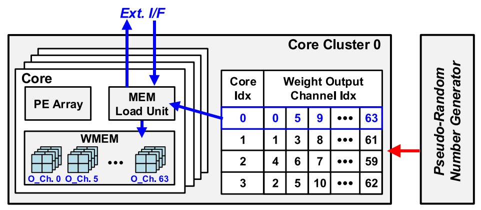


Fig. 11. RCA.

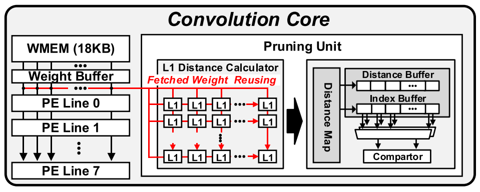


Fig. 12. WP unit.

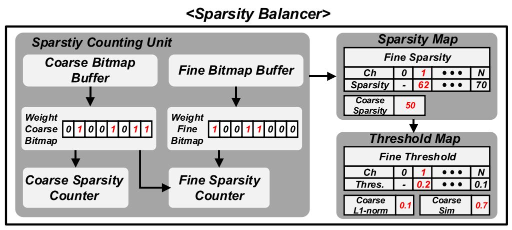


Fig. 13. Weight sparsity balancer.

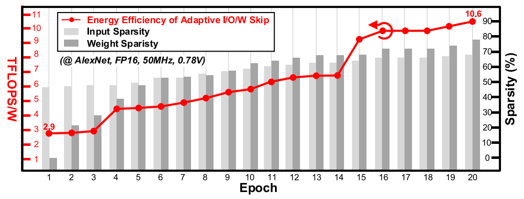


Fig. 14. Measurement result for target network.

channel to each core by using it. When the workload is divided by ICD between cores, the same output channel must be allocated to aggregate partial sums from each core. Therefore, the RCA allocates the same output channels to each core by sharing a sequence.

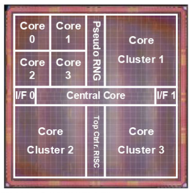
Besides, additional weight memory access is required for SCFLP. To eliminate this, weight data sharing is proposed, as shown in Fig. 12. The weight values which are transferred from the weight memory to the PE array are shared with the pruning unit by broadcasting. Then, the pruning unit calculates the L1 distance between output channels and the L1-norm value. When the calculation is finished, channels with high similarity or small L1-norm are converted to coarse-zero.

Lee *et al.* [8] and Kang *et al.* [9] handled the utilization drop caused by activation sparsity imbalance. However, they cannot solve the additional workload unbalancing caused by weight sparsity. Fig. 13 shows the weight sparsity balancer of PNPU. After the workload is allocated to each core, it calculates weight sparsity before pruning iteration. The core with high sparsity raises the similarity threshold and lowers the L1-norm threshold to generate fewer coarse zeros by pruning. Conversely, the threshold of the core with low coarse sparsity is adjusted to generate more coarse-zeros. For example, the threshold value for coarse pruning is defined as TH, and the threshold value to be compared with the L1-norm value of the weight is defined as [(TH/total iteration) × (current iteration)]. In the first iteration, each output channel is allocated to a different core of the PNPU, and pruning is performed simultaneously with the convolution operation. At this time, the coarse sparsity of each core is calculated, and average coarse sparsity is computed and stored. In the next iteration, the coarse sparsity of each core is computed by using coarse bitmap

TABLE I
COMPARISON TABLE

		[10]	[11]	[8]	[9]	This Work
Pruning		X	X	X	X	O
Sparsity Support	INF	X	X	IN Act.	IN/Out Act.	Adaptive Triple Sparsity (IN/OUT/W)
	EP	X	X	X	Out Act.	
	GG	X	X	IN Act.	Out Act.	
Process [nm]		65	65	65	65	65
Die Area [mm^2]		5.76	16	16	32.4	16
Supply Voltage [V]		0.78-1.1	0.67-1.1	0.78-1.1	0.70-1.1	0.78-1.1
Max Frequency [MHz]		200	200	200	200	200
Precision	Weight	FXP16	FXP 4/8/16	FP8, FP16	FP8, FP16	FP8,FP16
	Activation	FXP13	BFloat16	FP8, FP16	FP8, FP16	FP8,FP16
Peak Performance [(TOPS or TFLOPS)]		0.13	0.2 (FXP 16)	0.3* (FP16), 0.6* (FP8)	0.54* (FP16), 1.08* (FP8)	0.31*-9.98** (FP16), 0.61*-18.01**(FP8)
Energy Efficiency*** [(TOPS or TFLOPS)/W]		0.77	2.16 (50MHz, 0.73V)	1.74*-15.6** (50MHz, 0.78V)	1.81*-75.68** (50MHz, 0.75V)	1.57*-82.04** (50MHz, 0.78V)
Power Consumption [mW]		168	2.4 (10MHz, 0.67V)	43.1 (50MHz, 0.78V)	58 (25MHz, 0.70V)	49 (50MHz, 0.78V)
			196 (200MHz, 1.1V)	367 (200MHz, 1.1V)	647 (200MHz, 1.1V)	425 (200MHz, 1.1V)

*0% Sparsity (Input & Weight) ** 90% Sparsity (Input & Coarse Weight) *** Highest Precision Condition



		Specifications
Technology		65nm 1P8M CMOS
Die Area		4mm x 4mm (16mm^2)
SRAM		338 KB
Supply Voltage		0.78V ~ 1.1V
Frequency		~200MHz
Data Type		FP8, FP16
Peak Performance [TFLOPS]		0.31*-9.98** (FP16)
		0.61*-18.01**(FP8)
Power Consumption [mW]		49mW @ 0.78V, 50MHz
		425mW @ 1.1V, 200MHz
Energy Efficiency [TFLOPS/W]	200MHz, @1.1V	0.72$^{1)}$ - 41.01$^{2)}$ - 76.69$^{3)}$ (FP16) 1.44$^{1)}$ - 73.87$^{2)}$ - 142.09$^{3)}$ (FP8)
	50MHz @0.78V	1.57$^{1)}$ - 82.04$^{2)}$ - 151.78$^{3)}$ (FP16) 3.14$^{1)}$ - 146.52$^{2)}$ - 278.39$^{3)}$ (FP8)

1) Input & Weight Sparsity = 0%
2) Input & Coarse Weight Sparsity = 90%
3) Input & Coarse Weight & Fine Weight Sparsity = 90%

Fig. 15. Chip photograph and performance.

before the operation starts. And the current core's threshold is redefined as [(TH/total iteration) × (current iteration) × (previous average coarse sparsity/current core's coarse sparsity)]. As shown in Fig. 14, when fine-tuning for AlexNet (for ImageNet classification) learned by PNPU, the sparsity balancer increases the average energy efficiency of PNPU by 1.24×. The threshold of fine pruning is adjusted according to the sparsity in the same way. Through this, the workload unbalancing of intracore and intercore is solved.

IV. MEASUREMENT RESULT

Fig. 15 shows the chip photograph and performance of PNPU. PNPU is fabricated in a 65-nm CMOS process and supports SCFLP and AIOWS. PNPU has an area of 16 mm^2, operates from 50 to 200 MHz, under 0.78 to 1.1 V. PNPU consumes 425 mW at 200 MHz and 1.1 V, and 49 mW at 50 MHz and 0.78 V with no sparsity. PNPU achieves an energy efficiency of 0.72 TFLOPS/W at FP16 and 1.44 TFLOPS/W at FP8 when there is no sparsity at 200 MHz and 1.1 V. Peak Performance is 306 GFLOPS (FP16) and 612 GFLOPS (FP8) with no sparsity and 9.98 TFLOPS (FP16) and 18.01 TFLOPS (FP8) with 90% sparsity. Peak energy efficiency is 151.78 TFLOPS/W (FP16) and 278.39 TFLOPS/W (FP8) at 50 MHz and 0.78-V supply.

Fig. 14 shows the performance of PNPU when ImageNet is retrained using AlexNet, 78.2% weight sparsity was generated and 54.1% of the total weight is converted to coarse level zero within 5% accuracy loss. Besides, 50%–70% activation sparsity and 0%–80% weight sparsity are generated during DNN learning. As a result, the proposed processor achieves an energy efficiency of 2.9–10.6 TOPS/W when training AlexNet. In the case of retraining ResNet on CIFAR-10 using PNPU, it generates 54.3% coarse-level weight sparsity and 86% total weight sparsity (fine and coarse) with only 1% accuracy degradation.

Table I shows the performance comparison with the previous DNN learning hardware [8]–[11]. PNPU shows the highest performance under same activation and weight sparsity (90%, 90%) conditions. In the process of learning a real DNN, activation and weight sparsity (90%, 90%) condition assumed in [8] and [9] are impossible. References [8], [9] do not support the method as pruning, which increases the sparsity of the weight while increasing the activation sparsity. Therefore, Kang *et al.* [9] achieved only 1TFLOPS/W energy efficiency when operating the target network (GAN), increasing the only 1.76× compared to no sparsity condition. However, PNPU increases weight and input sparsity through pruning, and dual-zero skipping is possible in all three stages of learning. Therefore, as shown in Fig. 14, throughput increases by 1.84×–6.75× compared to the baseline when learning the target network.

V. CONCLUSION

This letter proposed an energy-efficient DNN accelerator with SCFLP and AIOWS. SCFLP eliminates bottleneck caused by coarse pruning, which has a high amount of computation. Besides, PNPU generates high weight sparsity by using on-chip WP differently from [8]–[11] through the pruning unit. AIOWS DNN core increases energy efficiency by utilizing the weight sparsity and activation sparsity. Also, dual zero skipping is possible in all three stages of DNN learning through AIOWS. As a result, PNPU is a fully trainable DNN processor, it achieves 151.78 TFLOPS/W (FP16) and 278.39 TFLOPS/W (FP8) energy efficiency at 50-MHz frequency and 0.78-V supply.

REFERENCES

[1] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Miami, FL, USA, 2009, pp. 248–255.
[2] Y. Yang and S. Newsam, "Bag-of-visual-words and spatial extensions for land-use classification," in *Proc. ACM SIGSPATIAL Int. Conf. Adv. Geogr. Inf. Syst.*, 2010, pp. 270–279.
[3] F. Tung, S. Muralidharan, and G. Mori, "Fine-pruning: Joint fine-tuning and compression of a convolutional network with Bayesian optimization," in *Proc. Brit. Mach. Vis. Conf. (BMVC)*, 2017, p. 12.
[4] H.-J. Yoo, "1.2 Intelligence on silicon: From deep-neural-network accelerators to brain mimicking AI-SoCs," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, San Francisco, CA, USA, 2019, pp. 20–26.
[5] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," 2015. [Online]. Available: arXiv:1510.00149.
[6] B. O. Ayinde and J. M. Zurada, "Building efficient convnets using redundant feature pruning," 2018. [Online]. Available: arXiv:1802.07653.
[7] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient transfer learning," 2016. [Online]. Available: arXiv:1611.06440.
[8] J. Lee, J. Lee, D. Han, J. Lee, G. Park, and H.-J. Yoo, "7.7 LNPU: A 25.3TFLOPS/W sparse deep-neural-network learning processor with fine-grained mixed precision of FP8-FP16," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, San Francisco, CA, USA, 2019, pp. 142–144.
[9] S. Kang *et al.*, "7.4 GANPU: A 135TFLOPS/W multi-DNN training processor for GANs with speculative dual-sparsity exploitation," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, San Francisco, CA, USA, 2020, pp. 140–142.
[10] D. Han, J. Lee, J. Lee, and H.-J. Yoo, "A 1.32 TOPS/W energy efficient deep neural network learning processor with direct feedback alignment based heterogeneous core architecture," in *Proc. Symp. VLSI Circuits*, Kyoto, Japan, 2019, pp. C304–C305.
[11] C. Kim, S. Kang, D. Shin, S. Choi, Y. Kim, and H.-J. Yoo, "A 2.1TFLOPS/W mobile deep RL accelerator with transposable PE array and experience compression," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, San Francisco, CA, USA, 2019, pp. 136–138.