IEEE Open Journal of the
Solid-State Circuits Society

Received 8 August 2022; revised 12 October 2022; accepted 28 October 2022. Date of publication 2 November 2022; date of current version 13 December 2022.

Digital Object Identifier 10.1109/OJSSCS.2022.3219034

Energy-Efficient DNN Training Processors on Micro-AI Systems

DONGHYEON HAN (Graduate Student Member, IEEE), SANGHOON KANG (Member, IEEE), SANGYEOB KIM (Graduate Student Member, IEEE), JUHYOUNG LEE (Graduate Student Member, IEEE), AND HOI-JUN YOO (Fellow, IEEE)

(Invited Paper)

School of Electrical Engineering, Korea Advanced Institute of Science and Technology, Daejeon 34141, South Korea

CORRESPONDING AUTHOR: H.-J. YOO (e-mail: hjyoo@kaist.ac.kr)

This work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) Grant Funded by the Korea Government (MSIT, PIM Semiconductor Design Research Center) under Grant 2022-0-01170.

ABSTRACT Many edge/mobile devices are now able to utilize deep neural networks (DNNs) thanks to the development of mobile DNN accelerators. Mobile DNN accelerators overcame the problems of limited computing resources and battery capacity by realizing energy-efficient inference. However, its passive behavior makes it difficult for DNN to provide active customization for individual users or its service environment. The importance of on-chip training is rising more and more to provide active interaction between DNN processors and ever-changing surroundings or conditions. Despite its advantages, the DNN training has more constraints than the inference such that it was considered impractical to be realized on mobile/edge devices. Recently, there are many trials to realize mobile DNN training, and a number of prior works will be summarized. First, it arranges the new challenges of the DNN accelerator induced by training functionality and discusses new hardware features related to the challenges. Second, it explains algorithm-hardware co-optimization methods and explains why it becomes mainstream in mobile DNN training research. Third, it compares the main differences between the conventional inference accelerators and recent training processors. Finally, the conclusion is made by proposing the future directions of the DNN training processor in micro-AI systems.

INDEX TERMS Backpropagation (BP), backward unlocking (BU), bit-precision optimization, deep neural network (DNN) training, reading transposed weight, sparsity exploitation.

I. INTRODUCTION

ARTIFICIAL intelligence (AI) is already essential in the current smart machines. For example, smartphones now utilize AI-based face recognition for user authentication. Their AI voice assistant can improve convenience for users because it can remove their manual device operations. Still, AI applications used in commercial products highly depend on the cloud server. However, communication with the cloud can cause not only the unreliable latency of the communication channel but also a privacy problem because it transfers user data regardless of the user's intent. Many deep neural network (DNN) accelerators [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13] provided on-device inference to solve this problem. However, they give limited applications with lower accuracy because it just repeats the predetermined task based on the pretrained DNN. The user-specific environment cannot be covered at the pretraining level and, for this reason, the inference processor shows poor performance if it is placed in an unexpected situation. Finally, the current AI processors have weaknesses in providing user-friendly AI services.

With the need for interactive AI, the demand for on-chip training is increasing. On-chip training can give personalization functionality by tuning global knowledge to be optimized to the user-specific datasets. In addition, on-chip

training-based distributed learning can distribute the workload of the server. If we adopt the federated learning [82], it not only protects individual privacy but also makes DNN training easy to scale.

Even though the on-chip DNN training can have many advantages, the realization of training in the micro-AI systems seems challenging. Specifically, edge/mobile devices must handle training with their limited computing resources and battery capacity. Since slow training cannot quickly adapt to the ever-changing environment, training should be performed reasonably fast while maintaining high energy efficiency. Recently, there are many trials [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25], [26], [27], [28], [29], [30], [31], [32], [33], [34], [35] to realize DNN training at the edge or mobile devices but previous review papers [36], [37], [38], [39] did not cover this new research area properly.

In this article, DNN training will be discussed in terms of technical challenges. First, typical applications and requirements of the on-device training will be discussed in Section II. New design challenges caused by computational characteristics of the training will be summarized in Section III. After that, new features suggested by the recent research will be examined through Sections IV to VIII. Section IV surveys new hardware designs to support the transpose-read of weight required at the error-propagation (EP) stage. Sections V–VIII introduce: 1) sparsity-aware acceleration; 2) bit-precision optimization; 3) memory access optimization; and 4) backward unlocking (BU) methodologies. Section IX summarizes recently developed DNN training processors and introduces design examples, HNPU-V1 [29] and HNPU-V2 [80], with the design philosophies of the inference and training processor design. This article will be concluded with a discussion about future research direction and new challenges which should appear in the upcoming DNN training processors.

II. THREE MAJOR SCENARIOS OF ON-DEVICE TRAINING

Training efficiency and speed is the key enabler of on-device training but the detailed requirements can be varied according to the target application. As shown in Fig. 1, we summarize three major scenarios of on-device DNN training as follows: 1) evolution; 2) advancement; and 3) adaptation.

A. EVOLUTION: LONG-TERM DNN TRAINING

If we can collect the data with proper labels, a new DNN application can be created. However, there are too much data in the real world, and labeling all of them is challenging. In addition to labeling, it faces privacy-preserving issues during the training data gathering. Although these two problems are the main barriers to enlarge DNN application, they can be addressed using long-term DNN training such as federated learning [82]. After devicewise local DNN training, individual weights are sent to the cloud and the weights are

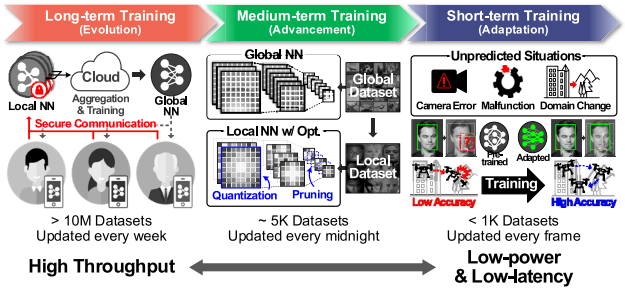


FIGURE 1. Requirements of the DNN training processor according to the on-device training scenarios.

aggregated for global DNN training. For this training process, the individual device should support on-device training for local training. It updates local DNN with enough user data collected for some period and receives global DNN after federated learning. This is an example of long-term DNN training because communication to share parameters occurs infrequently. The long-term training scenario, evolution, utilizes a large dataset such as ImageNet [81], unlike the other two training scenarios. The AI processors for evolution applications require reconfigurable logic to provide training functionality but have no strict conditions for training latency.

B. ADVANCEMENT: MEDIUM-TERM DNN TRAINING

On-device training is important when the application needs user-specific solutions. On-device training can make a small student network which shrinks the original large teacher network to focus only on user preference. It usually utilizes quantization or weight pruning for the fast and efficient acceleration of DNN on the micro-AI device. In addition to network shrinking, it can add new classes which were not considered in the original pretrained network training. It needs to fine-tune the network which utilizes the original big teacher network as a backbone but modifies its weight in the user-specific dataset. These applications can be categorized as advancement scenario which requires medium-term DNN training. In the advancement scenario, it has a relatively small dataset and the training should be finished at midnight which is generally considered as the device charging and not in use. The training time constraint is more strict than long-term DNN training cases but it still does not need a real-time or ultrafast performance like the inference applications. Instead, these training scenarios are important to offer user-friendly applications. Moreover, it can optimize networks using quantization or pruning for fast and efficient inference on edge/mobile devices.

C. ADAPTATION: SHORT-TERM DNN TRAINING

Both long-term and medium-term training are important to preserve users' privacy while expanding DNN applications to broad areas. However, their timing constraints are not that strict compared with the DNN inference scenario. However, real-time or ultrafast training is required when DNN needs

environmental adaptation. There are two typical examples. The first example is object tracking. Han et al. [34], [43] utilized a light object detection network but supports object tracking functionality by adopting online DNN training. Since the shape of the target object can be deformed due to its individual movement, it learns new shapes through online DNN training. Furthermore, it adapts new illuminations or occlusions to be robust to environmental changes. Generally, object tracking targets real-time operation, the online training process should be accelerated within 30 ms not to degrade the performance of the conventional inference process. Adaptation after an unexpected situation, such as a camera malfunction or abrupt domain change is also important to prevent fatal operational errors. Han et al. [80] showed that online DNN tuning performed right after an unpredictable accident is one of the solutions to recovering its original performance

As shown in both two examples, on-device adaptation seems promising but it must be accompanied by an energy-efficient and low-latency DNN training processor. Long latency due to the training rather disturbs the DNN inference and can cause other problems due to slow response. Although it can collect only a small amount of dataset during the runtime, it should realize the fastest training speed compared with the other two training scenarios. Unlike both the evolution and advancement scenarios, the adaptation scenario requires ultrafast DNN training but it can lose the generality of the original network by using only a small amount of data collected every frame.

The design direction of the on-chip DNN training processor can be varied according to the target DNN training scenarios. In this article, we will introduce various DNN training processors including these three target application scenarios.

III. BACKGROUND: BACKPROPAGATION

Even though there are many DNN training methodologies, backpropagation (BP, [40]) is the mostly used DNN training method. Thus, distinct computing characteristics of the BP compared with the inference will be briefly explained. We will also introduce the main challenges for realizing fast and energy-efficient training on micro-AI systems.

A. BP OF FULLY CONNECTED LAYER AND CONVOLUTION LAYER

The BP is a loss minimization method based on gradient descent by using the chain rule. It stores all intermediate activations and errors generated in every layer and loads them to calculate the new gradients. Unlike numerical gradient descent, the BP-based training is one of the analytic gradient descent methods, which can calculate accurate gradients of the multiple parameters with just one inference. Therefore, the BP can significantly reduce the computational requirement, instead, it needs large storage to remember entire activations and errors to update the weights.

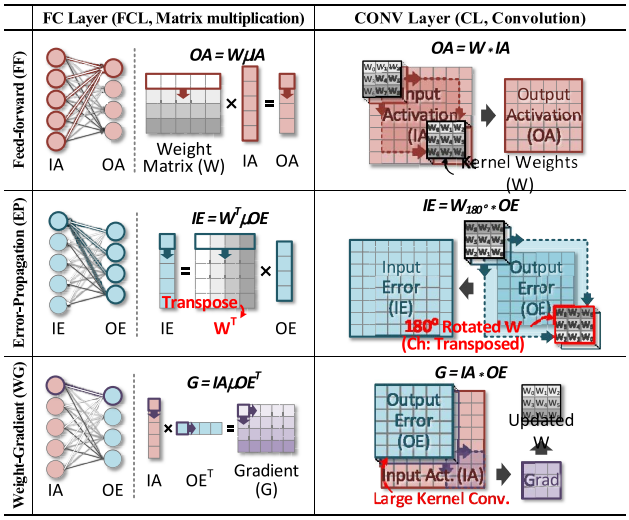


FIGURE 2. Three training stages during BP: FF, EP, and weight-gradient-update (WG).

As shown in Fig. 2, the BP generally consists of three training stages: 1) feedforward (FF); 2) EP; and 3) weight-gradient-update computing (WG). In the FF stage, the DNN gets inference results with the predefined weights. After that, the loss function is calculated to generate the error. In the EP stage, this error is now propagated from the last layer to the prior layers step by step. Finally, the gradient of the weight is calculated by utilizing both activations and errors generated in FF and EP stages. This training process is general for any type of DNN, but the required computing methods can be varied according to the target layer type. In this section, we will explain the detailed computation of the three training stages in two commonly used layer types" 1) the fully connected layer (FCL) and 2) convolution layer (CL).

1) BP OF FCL

When input and output neurons are fully connected in a layer, we call it FCL and it is the most general layer type of the DNN. For example, multilayer perceptrons and recurrent neural networks use the stacks of FCLs to complete the network architecture. Convolutional neural network (CNN) also utilizes FCLs for classification or channel attention. The FF stage of FCL is based on matrix multiplication of input activation (IA) and weight. In the EP stage, the computation method is similar but the input operands are replaced with error and transposed weight. After both FF and EP stages, IA and output error (OE) are multiplied to create the gradient of the weight. The WG stage needs elementwise multiplication in the single-batch gradient descent.

2) BP OF CL

Although the FCL needs matrix multiplication and elementwise multiplication for training, CL only needs the convolution operation. Instead, the shapes of the weight kernel are varied according to the training stages. It needs

TABLE 1. Typical batch size used in training processors.

Batch Size	1	1	4	32	40-100	256	2048
Papers	[24, 32]	[26]	[21]	[34]	[23]	[30]	[20]
DNN Type	CNN	CNN	CNN	CNN	CNN	MLP	MLP
Usage	GAN (Img2img)	Classification	Classification	Classification*	Classification	DRL	DRL

*On-chip training of classification DNN to realize tracking

a 180°-rotated kernel to propagate the errors. The position of the input and output channel is also switched. Similar to FCL, CL also needs the transpose-read due to this channel switching. After the EP stage, the gradient of the weight is calculated by the convolution of IA and OE. Unlike the FF and EP stages, the WG stage needs large-kernel-size convolution because the size of OE is almost the same as IA. As a result, the distance of the spatial data-reusing at the WG stage is not as long as in the other training stages.

B. CHALLENGES OF BP AT THE MICRO-AI SYSTEMS

The DNN training requires not only FF stage but also EP and WG stages. Additionally, optimization methods of the inference processors can be useless because they sometimes degrade the training quality. This difference induces new challenges to design training processors and they are summarized as follows.

1) READING OF TRANSPOSED WEIGHT

The EP stage reads the transposed form of the forward weight (FW) before it computes the convolution. The CL requires a 180°-rotated kernel as well as a transposed kernel. The reading of 180°-rotated data can be realized simply by reading data in reversed order. However, the reading of transposed weight is not that simple because it reads discontinuous data from the external memory where the weight is stored. Discontinuous data access pattern caused by transposed weight disturbs burst-mode read of DRAM, resulting in slow and inefficient DNN computing.

2) MAXIMIZING THROUGHPUT EVEN WITH THE LIMITED RESOURCES

DNN training requires a much larger amount of computations compared with inference. It needs to perform additional two training stages and repeats this training loop until the loss curve is converged. It also utilizes batch gradient descent which uses multiple inference results to generate a single gradient.

The required batch size can be varied according to its target training applications. Generally, the batch size becomes relatively small if the target network requires a large memory size such as image-to-image translation networks. On the contrary, as shown in Table 1, deep reinforcement learning (DRL) networks utilize large batch sizes because their network size is relatively small. In the object classification application, its training performance can be degraded significantly with the batch size < 4, but beyond 4, there is no significant difference in training performance [Fig. 3(a)]. For this reason, the conventional GP-GPU utilized large batch

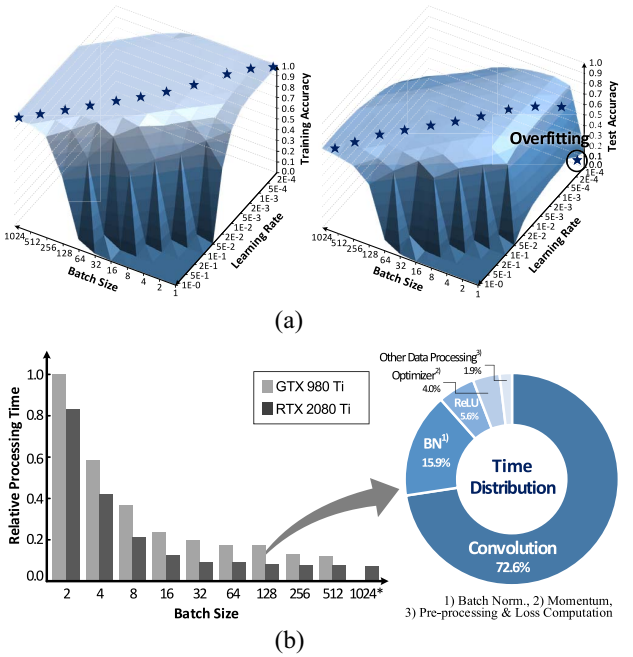


FIGURE 3. Relationship between batch size and DNN training. (a) Training and test accuracies of CIFAR-100 according to batch size and learning rate. (Blue star = Highest accuracy among various learning rates) batch size and its best learning rate show a positive linear correlation. (b) Analysis of DNN training at the GP-GPU (ResNet @ CIFAR-100).

parallelism to reduce the overall training time [Fig. 3(b)]. However, this acceleration method is challenging in mobile training processors with limited on-chip memory size and external memory bandwidth. Instead, a novel throughput improvement method is necessary for the training processors to accelerate training even with the small-batch parallelism.

3) HIGH-BIT PRECISION REQUIREMENT

During the inference, the final decision of the DNN is mainly affected by the large values of the IA [29]. For this reason, it is easy to decrease the bit-width during the inference acceleration. Unlike the inference, even the small values of IA cannot be ignored in DNN training because they may include important information for loss minimization. The naïve quantization methods can destroy this information and slow down the training. Furthermore, the errors computed in the EP stage have a wider distribution compared with the FF stage; thus, it is insufficient to be represented with low-bit precision. As a result, training requires high-bit-precision representation and this requirement becomes the main obstacle to the realization of DNN training on mobile devices because it totally decreases the efficiency of the processor.

4) MEMORY-INTENSIVE OPERATIONS DURING WG

As described in Fig. 2, intermediate IA and OE should be stored until the weight gradient is calculated. In addition, although data-reusing can be partially done in the CL, its reusing amount is much smaller than in the other training stages. Computing in the WG stage becomes similar to

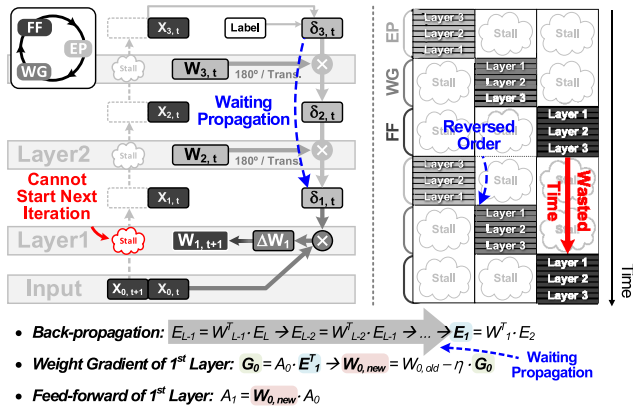


FIGURE 4. Low training speed caused by the backward locking problem.

TABLE 2. Typical batch size used in training processors.

Level	Paper	Features	Pros.	Cons.
SW or Algorithm Level	[43]	Feedback Alignment [44]	No W^T reading	Lossy training + extra BW
	[22,29]	Channel Re-arrangement	Off-the-shelf SRAM	Need time to re-arrange weights
	[23]	Diagonal Storage Pattern	Off-the-shelf SRAM	Need bit-rotator
Arch Level.	[20]	Transposable PE Array	Off-the-shelf SRAM	Efficiency drop of PE array
	[30]	Sparse Weight Transposer	Fast Decoding with Sparse Weight	Need additional register array
SRAM Level	[5,25]	Transpose SRAM	Simple PE array arch.	Low data density (Double BL/WL or > 6T cell)

matrix multiplication that appears in the FCL. After gradient calculation, it additionally loads the weight and momentum to finalize the weight update process. It causes high arithmetic intensity (the number of operations per byte) so that its performance highly depends on its external and internal memory bandwidth. Consequently, the reduction of memory access during the WG is the most important optimization method to accelerate its computation.

5) BACKWARD LOCKING PROBLEM

The BP propagates errors sequentially from the last layer to the first layers as depicted in Fig. 4. Therefore, the weight of the first layer can be updated only when the error propagation of the entire layers is completed. In other words, FF, EP, and WG stages should be computed sequentially and cannot be processed in parallel. It is called the backward locking problem [34] and this sequential process has two disadvantages. First, three training stages require the same weights but the weights should be reloaded every training stage. Second, it cannot hide the latency of the EP and WG stages and significantly reduce the framerate during online DNN training, such as object tracking [41] or temporal knowledge distillation (TKD, [42]).

IV. SOLUTION OF TRANSPOSE-READ DURING EP

As summarized in Table 2, there are three categories of transpose-read solutions: 1) software-level; 2) architecture-level; and 3) circuit-level solutions.

A. SOFTWARE-LEVEL SOLUTION

Han et al. [43] tried to avoid transposed weight by using a different EP method called feedback alignment [44]. This algorithm is the same as BP but the backward weight (BW) is substituted with the binary random matrix. However, since this algorithm modification may degrade the training accuracy significantly, it can only be applied to limited DNN applications.

Most DNN processors [22], [23], [29], instead, adopted software-level data prefetching. They rearranged the data stored in weight SRAM before it was used for the main convolution operation. This method is straightforward but needs additional time to rearrange the weight memory. Therefore, it can be efficient only when the reordered weights are reused repeatedly.

B. ARCHITECTURE-LEVEL SOLUTION

The transposable PE array suggested by Kim et al. [20] releases the burden of the software by simple architectural modification of the PE array. The transposable PE array utilized both broadcasting and unicasting data flow to perform matrix multiplication or convolution. It utilized the input feature reuse during the inference with a single image, but it adopted weight reuse by aggregating multiple images in the EP stage. This method can convert its computing type by using a simple instruction but it needs a local buffer for each MAC unit, resulting in an efficiency drop in the PE array.

Another paper [30] added a weight transpose-reading unit instead of modifying the main PE array. This additional unit generated the transposed weights before they were fetched to the PE array. Since the transposable PE array [20] assumed regular patterns of rectangular-shaped weight, there is no throughput improvement effect even with the pruned weight. The weight transposer suggested by Lee et al. [30] adopted hierarchical transpose-read and reduced memory access by excluding the fetch of pruned weights. Even though the weight transposer unit showed fast weight decoding speed, it can rather induce low area efficiency due to the additional large register file array.

C. CIRCUIT-LEVEL SOLUTION

Custom SRAM design [5], [25] is also a good solution that can support both normal-read and transpose-read. It makes both the main core architecture and software much simpler but the memory density of the SRAM is degraded because the SRAM cell architecture should be modified by adding more MOSFETs or bit/word-line.

V. SPARSITY EXPLOITATION DURING TRAINING

The DNN training processor needs further optimization to realize high-speed DNN training in the micro-AI systems. It needs additional hardware features to maximize its throughput even with the limited resources. Sparsity exploitation is the key feature that increases throughput during the on-chip training and it is summarized in Table 3.

TABLE 3. Summary of sparsity exploitation DNN training processors.

Paper List	Sparsity Exploitation									Additional Sparsity -related Feature
	FF			EP			WG			
	IA	FW	OA	OE	BW	IE	IA	OE	G	
[48]	X	X	X	X	X	O	X	X	X	Stochastic error pruning
[30]	X	O	X	X	O	X	X	X	X	Group-sparse pruning
[47]	X	O	X	X	O	X	X	X	O	Early-training-step pruning
[50]	X	O	X	X	O	X	O	X	X	Continual weight decay for pruning
[21]	O	X	X	X	X	X	O	X	X	Feature Dividing for Mixed Precision
[24, 27]	O	X	O	X	X	O	O	X	X	Output sparsity prediction with MSBs (FXP) or exponent (FP)
[29]	O	X	O	O	X	O	O	X	X	In-out Slice Skipping for ReLU-robustness
[49]	O	O	X	X	O	X	O	X	X	-
[26, 31, 51, 78]	O	O	X	X	O	O	O	X	O	Adaptive in/out/weight skip + Structured Pruning

A. CONVENTIONAL INPUT/WEIGHT ZERO SKIPPING

Most of DNNs utilize the ReLU activation function and it makes the negative values of IA become zeros. Weight can also have many zero values when the pruning is applied. Zero operands do not affect the final output and many inference engines [33], [45], [46] tried to speed up their operations by skipping zeros in the IA or weight. Input and weight skipping techniques are still effective for DNN training but it shows limited performance improvement during the EP and WG stages. At the EP stage, input sparsity disappears due to the normalization layer. On the other hand, weight sparsity disappears at the WG stage because the gradient is calculated by the convolution between IA and OE. DNN training processor needs a different sparsity exploitation method compared with the inference processor to utilize the advantages of the zeros that appear during the training.

B. RELU-AWARE OUTPUT ZERO SKIPPING DURING THE EP

Input sparsity cannot be utilized for the acceleration of the EP stage. Instead, the derivative of ReLU activation, $\mathrm{dReLU}(x)/dx$, transfers only positive values to the prior layer and induces output sparsity. With this motivation, some DNN training processors [24], [27] exploit both input and output sparsity to improve their throughput both in FF and EP stages.

C. PRUNING-AWARE OUTPUT ZERO SKIPPING DURING THE WG

Weight sparsity exploitation core is still effective during the EP stage but it cannot be applied during the WG stage because it calculates matrix multiplication or convolution between IA and OE. Instead, the gradients of the pruned weight should be discarded and it gives the possibility of skipping that calculation. It results in output sparsity during the WG stages. Output sparsity exploitation during the WG stage has big benefits thanks to both useless computation avoidance and memory access removal. For this reason, recent energy-efficient training processors [26], [31], [51], [78] supported triple sparsity exploitation by combining iterative pruning.

D. IN- AND OUT-SLICE SKIPPING

Dual or triple sparsity exploitation is proposed recently for zero-skipping-based DNN training but they induce complex data-path leading to low efficiency at dense data computing. In addition, the activation functions used for a recent DNN architecture have been varied such as leaky-ReLU [52] and mish [53]. It wastes lots of energy if there is no sparsity during the DNN training. Han et al. [29], [80] focused on this drawback and utilized bit-slice (4 bit) level sparsity. Moreover, Han et al. [29] skipped partial accumulation slices which should be truncated before it is used for the next layer. Thanks to the in- and out-slice skipping it simplifies the sparsity exploitation data path and utilizes slice-level sparsity in the entire training stage.

VI. BIT-PRECISION OPTIMIZATION

Even if the training processor adopts sparsity-exploitation, its energy efficiency is still lower than the conventional mobile inference processors because it requires high-bit-precision representation. Previously, it was considered that the DNN training requires $\geq$ 32 bit but the required bit precision is now continuously decreasing thanks to the research on the low-bit-precision training. We will introduce recent research to reduce the required precision during the training.

A. NEW NUMBER REPRESENTATION

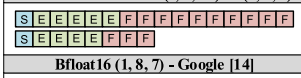
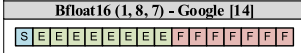
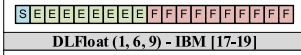
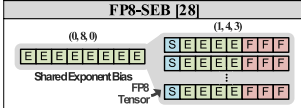
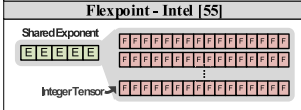
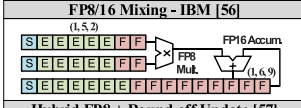
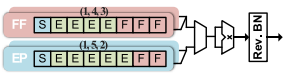
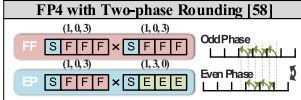
The first trial was the floating-point (FP) 16-bit IEEE 754 standard format. However, naïve FP16-based training sometimes degrades training accuracy because the exponent bit-width is too low to fully represent the weight gradient. There were many substitutions proposed for better DNN training.

1) NEW FLOATING-POINT REPRESENTATION

As shown in Table 4, there are three new FP representations that have a wider data representation range thanks to the high-bit-width exponent. The bit-width of the exponent in Bfloat16 [14] and TensorFloat [54] is the same as the original FP32 to give high compatibility with FP32. However, their energy efficiencies are still low because they need more than 16 bit for data accumulation. The DLFloat [17], [18], [19] was proposed to unify the data representation method of both input operand and accumulation. Flexpoint [55] tried to substitute FP with the fixed-point (FXP) representation using a shared exponent management algorithm together for the simplification of MAC design, but it failed to reduce the required bit precision to less than 16 bit.

Nowadays, FP-based training can be realized with less than 16 bit by utilizing the narrow distribution that appears in each of the tensors [28], chunks [56], or training stages [57], [58]. The required representation range can be reduced by dividing computation through the shared exponent [28], chunkwise accumulation [56], and hybrid representation among different training stages [57], [58]. Although the successful bit-precision reduction is observed

TABLE 4. Summary of FP representation for DNN training.

IEEE 754 FP Format (1, 5, 10) or (1, 4, 3)	**Pros. / Cons.**
	**Cons.** 1. Narrow representation range 2. Low representation resolution w/ FP8
Bfloat16 (1, 8, 7) - Google [14]	**Pros. / Cons.**
	**Pros.** High compatibility w/ IEEE 754 **Cons.** IEEE 754 FP32 for accum.
TensorFloat (1, 8, 10) - NVIDIA [54]	**Pros. / Cons.**
	**Pros.** High compatibility w/ IEEE 754 **Cons.** Relatively higher bit-precision
DLFloat (1, 6, 9) - IBM [17-19]	**Pros. / Cons.**
	**Pros.** Same format for Mult. and accum. **Cons.** Low compatibility w/ IEEE 754
FP8-SEB [28]	**Pros. / Cons.**
	**Pros.** 1. Securing tensor-wise enough representation range **Cons.** 1. FP30 (1-6-23) format for accum.
Flexpoint - Intel [55]	**Pros. / Cons.**
	**Pros.** 1. Securing tensor-wise enough representation range 2. High representation resolution **Cons.** 1. Relatively higher bit-precision
FP8/16 Mixing - IBM [56]	**Pros. / Cons.**
	**Pros.** 1. FP8 based Mult. **Cons.** 1. FP16 (1-6-9 format) for accum. 2. Need chunk-wise accum.
Hybrid FP8 + Round-off Update [57]	**Pros. / Cons.**
	**Pros.** 1. FP8 based Mult. **Cons.** 1. High-bit-precision for accum. 2. Need revised BN & WU
FP4 with Two-phase Rounding [58]	**Pros. / Cons.**
	**Pros.** 1. FP4 based Mult. **Cons.** 1. FP16 format for accum. 2. FP8 for 1x1 CONV

The number of bits: (S, E, F) S : Sign E : Exponent F : Fraction (Mantissa)

during the multiplication, accumulation still needs high-bit precision and it requires additional hardware units to manage hybrid representation [57] or two-phase rounding [58].

2) NEW FIXED-POINT REPRESENTATION

FXP representation needed much higher bit precision compared with the FP. The required bit precision can be reduced dramatically when dynamic FXP (DFXP) [3], [29], [34], [80] is adopted. It adjusts the required integer length to fit the layerwise narrow distribution instead of considering entire layers. Even though the DFXP shows a remarkable performance improvement, it still needs higher bit precision (> 8 bit) compared with the FP because the low-precision FXP has a limited representation resolution. Stochastic rounding (SR, [59]) was proposed to increase its virtual representation resolution. In addition to SR, stochastic thresholding (ST, [29]) was co-designed to further increase representation resolution by loosening its overflow judgment and removing outlier data. A combination of SR and ST successfully demonstrated a 43% bit-precision reduction in the CIFAR-100 dataset [29].

B. LOW BIT-PRECISION TRAINING ALGORITHM

Not only new number representation but also new algorithms were also proposed for low-bit-precision training. One of them is fine-grained-mixed-precision (FGMP, [21]) which divides IA into two different bit-width formats. The majority of data is placed near zero and it is enough to be represented by the low-bit-precision data. However, ignoring the precise representation of the outlier data degrades its training performance significantly. To solve this problem, the FGMP represents only outlier data with high-bit precision while maintaining low-bit precision for the majority of data.

Layerwise adaptive precision scaling (LAPS, [29]) further reduces the required bit precision by using automatic bit-precision tuning during the training. Required precision can be varied according to the training scenario but it is hard to predict the optimal bit precision at the beginning of training. The LAPS continuously monitors the similarity of high-precision and low-precision convolution results and increases the bit-width if the difference becomes larger. Bit precision found by LAPS can be varied according to the dataset and network complexity.

C. HARDWARE ARCHITECTURE FOR BIT-PRECISION OPTIMIZATION

Hardware architecture and PE circuit should be modified to support the new number representations and low-bit-precision optimization algorithms.

1) MULTIPLE-PRECISION CONFIGURABLE MULTIPLY–ADD UNIT

To accommodate the new FP representations, the design of the MAC unit should be modified. Since the required bit precision of the training is usually higher than the inference, training processors showed lower efficiency compared with the inference processors. Multiple-precision configurable multiply–add units can support both high-precision and low-precision computing and it becomes a common way to reduce the efficiency gap between inference and training processors.

2) ACTIVE TRAINING SUPPORTING UNIT

Both FGMP [21] and DFXP [3], [29], [34], [35], [80] need streaming data analysis units that can calculate the mean/variance or overflow ratio of the OA. Based on the analyzed tensorwise statistics, the FGMP converts more than 90% of FP16 accumulation results to FP8 operands during the ResNet-18 training. In DFXP-based computing, it increases the integer length of the corresponding layer if the overflow ratio exceeds the predetermined threshold [3]. The LAPS [29] performed it in a peripheral unit that calculates the difference and counts only large differences. The new bit precision is determined by an internal finite-state machine and then it is applied from the next training iteration. The bit-precision reduction algorithms usually require an additional unit to monitor output distributions and give useful information for runtime bit-precision optimization.

VII. WEIGHT GRADIENT STAGE ACCELERATION

Sparsity-aware WG acceleration assumes that it adopts weight pruning. Acceleration of the WG stage without weight pruning is still challenging because the bit precision of the gradient and primal weight should be higher than the bit precision used during the FF and EP stages.

Further optimization of the WG stage is essential for a high-performance training processor; thus, we summarize the existing methods for the WG stage.

A. GRADIENT BIT-PRECISION REDUCTION

Data compression is the mostly used WG acceleration method because the WG stage suffers from the memory-intensive computation. Seide et al. [60], Alistarh et al. [61], and Wen et al. [62] suggested ultralow-bit gradient quantization but their main purpose was to reduce the gradient communication among the multiple-edge devices during the distributed learning. Since the primal weight still needs full precision (16 bit), the quantized gradient should be converted to the original bit-width during the weight update operation. In addition, a large portion of memory access is still required due to the data movement of the primal weight and momentum from the memory.

B. SPARSE GRADIENT CALCULATION

Gradient compression can be realized by not only quantization but also gradient sparsification. Lin et al. [63] and Strom et al. [64] accumulated weight gradients every iteration and updated only when the accumulated gradient became larger than the predetermined threshold. These methodologies require a large internal buffer to store all weight gradients, which is not practical in training on micro-AI systems. Another sparsification [65] updated the weight only when it had a large gradient or the target confidence was low [66]. Large-gradient-only or low-confidence-only weight update showed great compression ratio but they can cause a slow training curve. Furthermore, they are not easy to be applied to other complicated DNN applications, such as object detection and image-to-image generation.

C. SELECTIVE DNN TRAINING

Although bit-precision reduction or sparsification during the WG stage shows a higher efficiency than the naïve gradient calculation, it still requires a large buffer to store all intermediate IA and OE that appear during the FF and EP stage. Cai et al. [83] suggested bias-only fine-tuning which freezes weights and updates new bias to remove storage of intermediate IAs. Another method is selective layer training [32] which focused on a few layers but skips memorizing the IA and OE of the useless layers. Both methods can reduce the required memory footprint significantly but drops training quality due to aggressive gradient calculation skipping. Lin et al. [84] newly suggested channelwise gradient skipping and also utilized both bias-only tuning and selective layer tuning to further optimize the training. To minimize the accuracy degradation, it predicts the accuracy loss and the reduced hardware cost during the compile time. Finally, selective training is a key enabler of on-device training by minimizing required memory access due to IA and OE.

TABLE 5. Comparison of typical BU methodologies.

	MEM of A&E	MEM of W	Proc. Time
BP [40]	$A_0+A_1+A_2+E_0+E_1+E_2$ $\approx \mathbf{A+E}$	$W_0+W_1+W_2$ $\approx \mathbf{W}$	$T_{A0}+T_{A1}+T_{A2}+T_{E0}+T_{E1}+T_{E2}$ $\approx \mathbf{T_A+T_E}$
DDG[1] [69]	$3A_0+2A_1+A_2+E_0+E_1+E_2$ $\approx \mathbf{2A+E}$	$3W_0+2W_1+1W_2$ $\approx$ **2W (W Access ▲)**	$T_{A0}+T_{A1}+T_{A2}+\max(T_{E0\sim E2})$ $\approx \mathbf{T_A+T_E/3}$
FDG[2] [70]	$6A_0+4A_1+2A_2+E_0+E_1+E_2$ $\approx$ **4A+E (A Access ▲)**	$W_0+W_1+W_2 \approx \mathbf{W}$	$\max(T_{A0\text{-}A2})+\max(T_{E0\text{-}E2})$ $\approx \mathbf{(T_A+T_E)/3}$
DFA [71-74]	$A_0+A_1+A_2+E_{last}$ $\approx \mathbf{A+E_{last}}$	$W_0+W_1+W_2+BW$ $\approx \mathbf{W+BW}$	$\max(T_{A0}, T_{E0})+\max(T_{A1}, T_{E1})$ $+\max(T_{A2}, T_{E2}) \approx \mathbf{\max(T_A, T_E)}$

1) Decoupled Delayed Gradient, 2) Fully Decoupled Gradients

VIII. BACKWARD UNLOCKING

The consideration of backward locking is still challenging and there are few proposals to solve this problem. Li et al. [68] proposed an out-of-order scheduling mechanism that performs inference of CL before the training of FCL is finished. However, balancing the workload between CL and FCL is challenging and FCL still suffers from backward locking problems. BU without algorithm modification is difficult because the BP itself has serial computation characteristics.

A. BACKWARD UNLOCKING WITH DELAYED BACKPROPAGATION

There are several BU methods [69], [70] using the delayed BP. Decoupled delayed gradient [69] divides DNN into multiple groups and performs BP only within a group to loosen the backward locking problems. The gradient calculated by each group is transferred to the prior group at the next iteration. Since it adopts an intergroup pipeline during the BP, multiple gradients can be computed at the same time, but each group utilizes the FF results obtained at a different time step. Thus, it should copy the weights of the multiple different time steps for parallel gradient computation.

Another method, fully decoupled gradient [70], copies intermediate IAs instead of weights. Not only EP but also the FF stage is divided into multiple groups and makes a big pipeline structure among them. Each group updates its weight with the delayed IAs and performs FF operation as soon as it completes the weight update. It can further decrease processing time thanks to the pipeline structure of both FF and EP stages.

In spite of their success, both methods are considered only in multi-GPU processing because they need a large buffer to store all weights or IAs of the delayed batches. Therefore, a memory-efficient BU method is necessary for the micro-AI systems.

B. DIRECT FEEDBACK ALIGNMENT-BASED APPROACHES

Direct feedback alignment (DFA, [71], [72], [73], [74]) described in Fig. 5 can propagate the errors of the last layer to the entire prior layers simultaneously. The problem of backward locking can be released in DFA-based training because the next inference can be started right after this

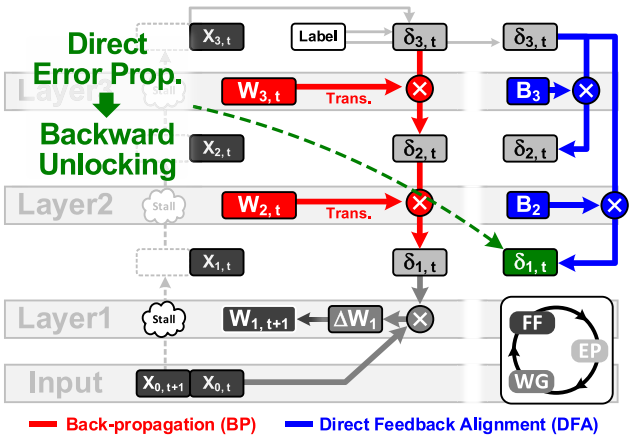


FIGURE 5. Typical BU method: DFA.

direct error propagation. As shown in Table 5, it shows the minimum memory accesses compared with the other two BU methods. The only extra cost is that it needs additional BW which is independent of the FW. It now turns out that the overhead of the BW can be reduced by a maximum of 96.9% when using binarization [72] or sparsification [73]. Thanks to the lowest memory requirement, the DFA algorithm was selected by the recent on-chip training ASIC [34], [80] and it successfully demonstrated DFA-based online training in the object tracking or object detection application. It constructed a pipeline structure among the three training stages and also hid latency caused by EP and WG. Thanks to the pipelined DFA (PDFA), it showed at least 2× higher training speed compared with the BP-based acceleration.

IX. DNN TRAINING ACCELERATOR EXAMPLES

In this section, the real chip implementation results of the DNN training processor will be discussed. First, we will introduce the examples of training processors and clarify their target applications. The reason why the algorithm co-design is necessary for mobile-oriented DNN training processors will be explained with practical examples. Finally, a design example of DNN training will be introduced with the key differences in accelerator design between inference and training.

A. APPLICATIONS AND EXAMPLES OF TRAINING PROCESSOR

1) APPLICATIONS OF DNN TRAINING

Training processors reported from the industry [14], [15], [16], [17], [18], [19] mainly focused on general-purpose DNN training. On the contrary, training processors from the academy [20], [21], [22], [23], [24], [25], [26], [27], [28], [29], [30], [31], [32], [33], [34], [35] mainly targeted local training which fine-tunes DNN to be more accurate in user-specific datasets. In-situ personalization is also suggested to fit the network to the user-specific tasks. However, these applications were not sensitive to the training speed because local dataset optimization and personalization should collect behaviors of users and experiences for a long time and train the network when the device is not in use, usually at midnight.

Super-fast training is required when online DNN training is necessary. One of the ASICs [34] showed an example that utilized online training to extend the role of object classification networks to object tracking. It learned the shape of the first bounding box and decided whether the surrounding boxes were similar to the target. Since it updates the network to learn the continuously moving target, it can achieve robust tracking under the shape deformation and illumination changes. Another example was TKD suggested by Farhadi and Yang [42] and Han et al. [80]. It realized energy-efficient but accurate object detection by adopting online knowledge distillation to the lightweight DNN. The lightweight network itself has been considered impractical due to its poor performance. This problem can be solved if the network is trained by data with labels generated by heavyweight teacher network inference. It needs the inference of a heavy teacher network, but this process is performed infrequently and consumes significantly less energy compared to the direct use of the teacher network. These examples show that DNN training can be used as not only for long-term user customization but also for short-term domain adaptation to further improve its functionality and performance.

2) EXAMPLES OF TRAINING PROCESSOR DESIGN

The majority of processors [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25], [26], [27], [28], [29], [30], [31], [32] adopted the homogeneous core design and they were programmable to be applied in various types of networks and applications. Moreover, they also emphasized a new FP-number-representation-based PE design by utilizing a new number representation method and adopted precision-configurable MAC for energy-efficient inference and training [15], [16], [17], [18], [19], [21], [22], [24], [26]. Mobile training processors are mainly developed by the academy and they were continuously evolved. First-generation mobile training processors [20], [22], [23], [25], [35] focused on giving on-chip training functionality by supporting transpose-read and reconfigurable data path. The second-generation processors [21], [24], [27], [31] were designed by combining sparsity exploitation or low-bit-precision training schemes to improve their throughput and efficiency. It is now evolved into third-generation processors [26], [29], [30] which further improved the performance by proposing the adaptive methods of bit precision or pruning ratio control even when the training is in progress.

B. ALGORITHM-HARDWARE CO-OPTIMIZATION IN TRAINING PROCESSOR

Most training processors adopted algorithm-hardware co-design and it becomes an inevitable trend for realizing DNN

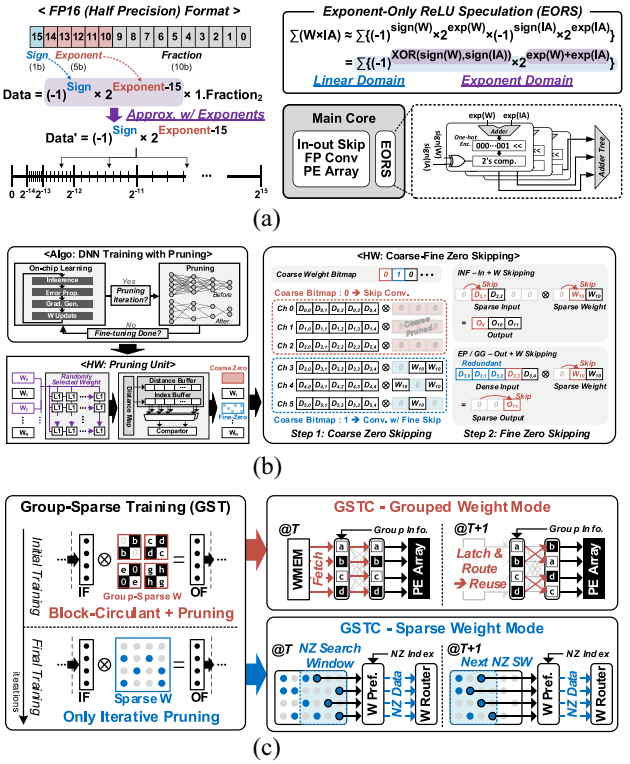


FIGURE 6. New algorithm and its dedicated hardware architecture for training. (a) GANPU [24]: Exponent-only ReLU speculation algorithm and adder-only light convolution supporting hardware. (b) PNPU [26]: New iterative pruning algorithm and core architecture to exploit both coarse and fine-grained sparsity. (c) OmniDRL [30]: GST algorithm with reconfigurable core architecture to support both weight reuse and skipping.

training in micro-AI systems. Since the naïve BP-based training requires large hardware resources, the algorithmic co-optimization is essential for high-speed and energy-efficient training.

1) NEW ALGORITHM AND ITS DEDICATED HARDWARE

Some algorithms are not that efficient in conventional DNN accelerators and require completely different hardware designs as shown in Fig. 6 Nevertheless, when they are co-designed with their optimized hardware, it shows much higher efficiency compared with the naïve implementation of DNN training.

The GANPU [24] of Fig. 6(a) sought co-optimization by proposing a hardware-friendly output sparsity prediction algorithm for the FP number system to support fast training. Sign and exponent bits, which represent the scale of data, are used to speculate whether the ReLU result will be zero. Given that multiplications between exponents are mathematically identical to integer additions, a dedicated speculation unit was able to be integrated on-chip with minimal resource overhead and it helped the main convolution core to exploit both input and output sparsity simultaneously at the FF stage.

The PNPU [26] of Fig. 6(b) proposed a new iterative pruning algorithm that utilized both coarse-level and fine-level pruning for efficient training acceleration. It approximated similarity calculation among the output channels through random channel sampling. The PNPU designed its dedicated pruning core and construct a pipeline structure between the main core and the pruning core to support iterative pruning without PE utilization drop. In addition to the pruning unit, the main core is designed to support triple sparsity exploitation and its hardware complexity could be reduced by utilizing coarse-zero skipping.

Pruning-aware training processors [26], [31] were efficient only when the pruning ratio became high enough. The low pruning ratio that appeared at the beginning of training disturbed pruning-aware acceleration, resulting in significant performance degradation. OmniDRL [30] of Fig. 6(c) solved this problem by utilizing block-circulant-based weight grouping and the pruning of small weights together in the initial training phase. The new algorithm, group-sparse training (GST), induced both repeated patterns and sparsity in the weight. GST core (GSTC) suggested by Lee et al. [30] improved training performance by enabling both grouped weight reuse and zero-weight skipping through a weight router and prefetcher.

2) HARDWARE PERFORMANCE IMPROVEMENT THROUGH THE ALGORITHM COMBINATION

Algorithms introduced in Fig. 5 were newly proposed for energy-efficient DNN training acceleration but it is effective only when its dedicated hardware exists. Unlike these examples, there were ASICs that have greater synergy when the algorithm is combined.

The first example was LNPU [21] of Fig. 7(a) which combined the input-zero-skipping core and FGMP-based training. The required computation is linearly increasing when the naïve FGMP is applied. However, the IA of each bit precision shows high input sparsity because of the zero paddings. Computation including zeros wastes many computing resources because zero paddings do not affect final computing results. The LNPU [21] adopted the input sparsity exploitation core to skip zeros induced by ReLU or FGMP. It not only exploits data sparsity but also reduces the burden of high-bit-precision computing to obtain a great synergy effect.

The second examples were ASICs [27], [29] which adopted bit-scalable core architecture. Bit precision can be varied according to layer number and the bit-scalable core architecture could remove useless operations during low-bit-precision training and maximize both training speed and efficiency. In addition to these purposes, it was able to realize a ReLU prediction algorithm [27] by performing MSB-only computing. After MSB-only prediction, it skipped the LSB computations of which results would have the zero-expected value. Output-slice skipping [29] was another algorithm that shows synergy with the bit-scalable architecture. It could compensate for the throughput degradation problem during

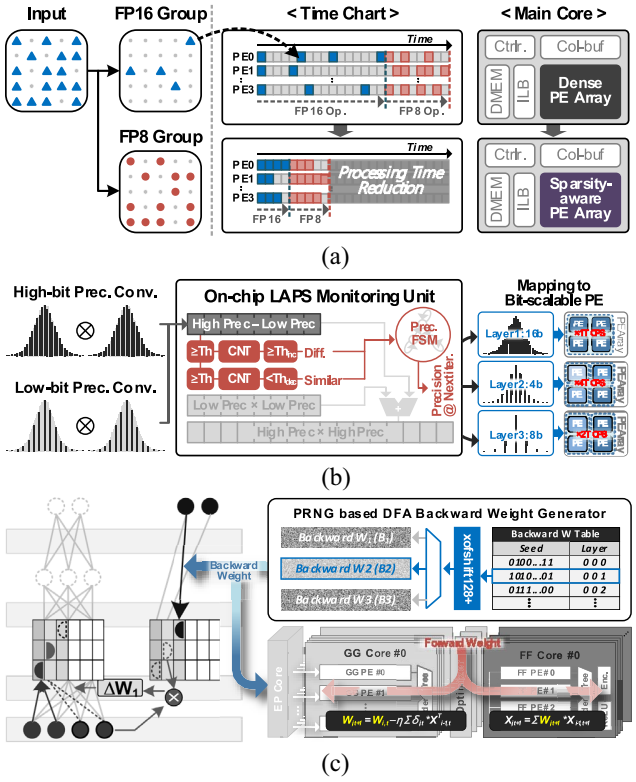


FIGURE 7. Examples of algorithm-HW co-design in DNN training ASICs. (a) LNPU [21]: Combining IA zero-skipping core and FGMP. (b) HNPU-V1 [29]: Combining bit-scalable core and LAPS. (c) DF-LNPU [34]: Combining fully heterogeneous core and PDFA.

Pruning Ratio

	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
DFXP16	74.0	73.8	72.9	72.1	71.1	69.7	67.7	63.6	43.6	27.5
DFXP14	74.0	73.4	72.5	71.8	70.5	69.6	67.7	63.3	53.0	33.4
DFXP12	73.3	73.1	72.7	71.9	71.5	70.0	68.0	63.0	43.5	33.0
DFXP10	73.3	73.4	72.7	72.3	71.5	70.1	67.6	63.3	51.0	29.3
DFXP8	73.9	73.7	73.4	72.5	71.7	70.0	67.5	63.4	54.1	39.2
DFXP6	73.1	72.7	72.3	71.9	70.6	69.7	67.1	61.9	52.3	30.3
DFXP4	32.3	31.3	28.0	23.6	24.7	21.5	13.3	14.2	5.4	2.1

(a)

Pruning Ratio

	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
DFXP16	78.8	78.4	77.9	77.1	76.6	75.1	73.7	71.4	64.1	50.1
DFXP14	78.8	78.4	78.0	77.1	76.1	75.0	73.8	71.9	65.0	50.4
DFXP12	78.6	78.5	77.8	77.4	76.4	75.1	74.2	71.3	65.3	50.0
DFXP10	78.8	78.7	78.2	77.6	76.6	75.5	74.0	71.4	65.6	49.8
DFXP8	78.5	78.1	77.9	76.9	76.5	75.3	74.0	71.2	64.5	48.0
DFXP6	77.7	77.4	76.9	76.3	75.4	74.7	72.8	70.5	63.3	47.0
DFXP4	44.9	44.3	46.3	43.7	42.7	40.9	39.0	34.7	34.7	27.2

(b)

Pruning Ratio

	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
DFXP16	78.5	78.4	78.0	77.9	77.4	76.6	75.8	73.6	70.0	54.4
DFXP14	78.4	78.3	78.3	77.7	77.5	76.8	75.5	74.2	69.5	54.5
DFXP12	78.6	78.6	78.4	77.3	77.3	76.6	75.4	73.7	69.2	54.1
DFXP10	78.1	78.1	78.3	77.5	77.4	76.5	75.6	73.7	70.6	55.3
DFXP8	78.5	78.5	78.2	78.0	77.5	76.6	75.8	73.6	68.6	52.1
DFXP6	78.3	77.9	77.7	77.5	76.8	75.5	74.5	73.0	68.7	51.5
DFXP4	62.4	63.3	61.8	62.7	62.8	59.9	57.4	54.0	40.6	15.9

(c)

FIGURE 8. PQ plot of ResNet-9, ResNet-18, and ResNet-34 (Green box: >73.0% and Blue box: >77.0 %). (a) Test accuracy of ResNet-9 after retraining for pruning and quantization. (b) Test accuracy of ResNet-18 after retraining for pruning and quantization. (c) Test accuracy of ResNet-34 after retraining for pruning and quantization.

high-precision computing by excluding LSB accumulations. Finally, it could be combined with the runtime bit-precision optimization algorithm, LAPS, as shown in Fig. 7(b). The HNPU-V1 [29] minimized the overhead of difference calculation between high-precision and low-precision convolution results by performing only LSB accumulations. The bit-scalable architecture used in [29] finally minimized the extra time required for bit-precision searching, and at the same time, optimized training with the discovered precision numbers.

The last example is DF-LNPU [34] of Fig. 7(c) which adopted a heterogeneous core design. The heterogeneous core design can optimize each training stage and minimize hardware resources to support training but has the problem of core utilization drop. The DF-LNPU [34] adopted a heterogeneous core design but no longer suffered from low core utilization problems by utilizing PDFA-based BU. The PDFA-based optimization enabled parallel processing of three different training stages. Combination with the BU algorithm compensated for throughput degradation caused by backward locking while retaining the advantage of the heterogeneous core design.

The main challenges illustrated in Section III-B can be solved effectively when the algorithm is co-designed with the training hardware. These examples clarify that algorithm-hardware co-optimization plays an important role in realizing DNN training at the edge device, even with limited hardware resources.

C. RELATIONSHIP OF SPARSITY AND QUANTIZATION DURING DNN TRAINING

Sections V and VI introduce typical examples of sparsity exploitation and quantization method for on-device DNN training. In this section, we will discuss the relationship among these optimization techniques. We construct the experiment of three different ResNet training in the CIFAR-100 dataset. In this experiment, we pretrained the network with FP32 and retrain them after weight pruning and DFXP-based quantization. During the quantization, we modified bit precision of weight, IA and OE but maintains high accumulation bit width. To analyze the relationship between pruning and quantization, we introduce a new graphical display, Pruning-quantization (PQ) plot, which visualizes the test accuracy according to varying training conditions to help designers to select optimal pruning and quantization level during the on-device training. Fig. 8 shows three PQ plots obtained from our experiment. Green and blue boxes in Fig. 8 indicate acceptable conditions with two different accuracy thresholds. Within the accuracy boundary, the on-device training designer can choose the best condition

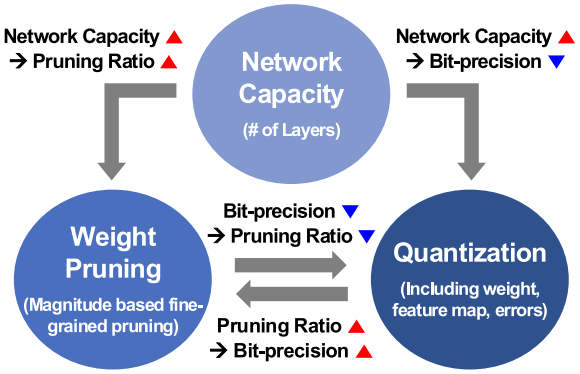


FIGURE 9. Tradeoff between weight pruning and quantization.

based on the hardware characteristics of the training processor.

There are two key observations obtained from the PQ plot as summarized in Fig. 9. First, DNN shows a low pruning ratio and high bit precision if the network capacity is small. Second, there is a tradeoff relationship between pruning and quantization. If a designer selects a high pruning ratio, high-bit precision must be used to maintain original high accuracy. The optimal design point can be varied whether the target processor has the functionality of zero-weight skipping or bit-reconfigurability.

Another minor observation is about IA sparsity. Fig. 10 summarizes the layerwise average IA sparsity after pruning or quantization. As shown in this figure, IA sparsity increases when it has a higher pruning ratio or lower bit precision. However, the IA sparsity within the accuracy boundary (indicated as a green or blue box) shows a small variance; thus, there is no expectation of additional throughput improvement through input sparsity exploitation when considering both pruning and quantization.

Fig. 11 shows an example of a DNN training execution comparison. We compare the ideal performance of the two DNN accelerators which support both weight-skipping and bit-reconfigurability but have different bit-granularity. The optimal design point of the coarse-grained bit-reconfigurable accelerator shows a higher pruning ratio than the fine-grained bit-reconfigurable accelerator (FGBRA). Instead, the FGBRA shows faster training in this experiment even with the lower pruning ratio. To sum up, the PQ plot can be useful for training processor designers to select optimal design hyperparameters. Utilizing both pruning and quantization is the best way to improve efficiency but needs to avoid lopsided optimization.

D. DESIGN EXAMPLES: HNPU-V1 AND HNPU-V2

As illustrated in Section II, the design of the on-device training processor can be varied according to their target training scenarios. There were two typical processors: 1) HNPU-V1 [29] and 2) HNPU-V2 [80], which utilized the same amount of hardware resources but were optimized differently due to their target application. Their key characteristics are

Pruning Ratio

	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
DFXP16	0.53	0.53	0.56	0.58	0.61	0.65	0.69	0.72	0.78	0.82
DFXP14	0.52	0.53	0.55	0.58	0.62	0.65	0.68	0.72	0.77	0.82
DFXP12	0.52	0.53	0.55	0.58	0.61	0.65	0.69	0.73	0.78	0.82
DFXP10	0.52	0.53	0.55	0.59	0.62	0.65	0.69	0.73	0.79	0.82
DFXP8	0.53	0.54	0.56	0.59	0.62	0.66	0.69	0.73	0.78	0.82
DFXP6	0.57	0.61	0.63	0.65	0.67	0.7	0.72	0.76	0.79	0.82
DFXP4	0.76	0.78	0.76	0.79	0.79	0.79	0.8	0.83	0.84	0.85

(a)

Pruning Ratio

	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
DFXP16	0.59	0.6	0.6	0.62	0.63	0.66	0.69	0.73	0.79	0.84
DFXP14	0.59	0.6	0.61	0.62	0.63	0.66	0.68	0.73	0.78	0.84
DFXP12	0.58	0.6	0.6	0.62	0.63	0.66	0.69	0.73	0.78	0.84
DFXP10	0.59	0.6	0.61	0.62	0.64	0.66	0.69	0.73	0.78	0.84
DFXP8	0.61	0.61	0.62	0.63	0.64	0.66	0.69	0.73	0.79	0.85
DFXP6	0.65	0.65	0.66	0.67	0.67	0.69	0.73	0.76	0.81	0.85
DFXP4	0.83	0.83	0.83	0.84	0.85	0.86	0.86	0.87	0.89	0.91

(b)

Pruning Ratio

	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
DFXP16	0.6	0.61	0.63	0.64	0.66	0.68	0.7	0.74	0.79	0.84
DFXP14	0.6	0.61	0.63	0.64	0.66	0.68	0.7	0.73	0.78	0.84
DFXP12	0.6	0.61	0.62	0.64	0.65	0.67	0.7	0.73	0.78	0.84
DFXP10	0.61	0.62	0.63	0.65	0.67	0.68	0.71	0.74	0.78	0.84
DFXP8	0.62	0.63	0.64	0.66	0.67	0.69	0.71	0.75	0.79	0.84
DFXP6	0.68	0.69	0.7	0.71	0.72	0.73	0.75	0.78	0.81	0.86
DFXP4	0.85	0.85	0.85	0.86	0.87	0.87	0.89	0.9	0.91	0.94

(c)

FIGURE 10. IA sparsity of ResNet-9, ResNet-18, and ResNet-34. (a) IA sparsity of ResNet-9 after retraining for pruning and quantization. (b) IA sparsity of ResNet-18 after retraining for pruning and quantization. (c) IA sparsity of ResNet-34 after retraining for pruning and quantization.

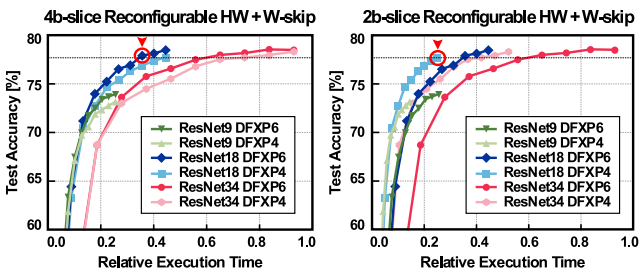
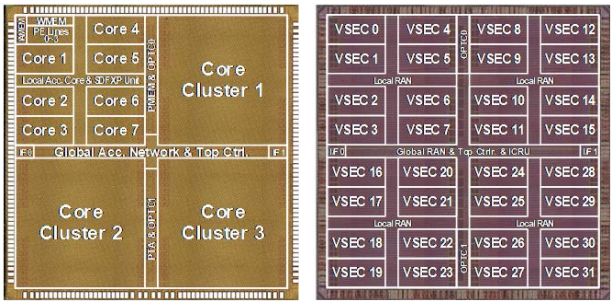


FIGURE 11. ResNet retraining experiment in the CIFAR-100 dataset.

summarized and the differences are highlighted in Fig. 12. The HNPU-V1 mainly focused on long-term and medium-term DNN training, thus, it introduced automatic precision search during the training for minimization of total training time and energy consumption. Moreover, it combined not only DFXP and SR but also ST to make a stable training curve even with the precision change caused by the LAPS. On the contrary, the HNPU-V2 targeted short-term DNN training, which performs online tuning for real-world environmental adaptation. It adopted the predetermined static bit precision during the online DNN tuning and removed ST-related circuits in its PE architecture. The higher bit-precision requirement of HNPU-V2 due to input data uncertainty could



HNPU-V1 [29]	Processor	HNPU-V2 [80]
Training from the Scratch & Personalization	Target Application	Run-time Online DNN Tuning @ Object Detection
Samsung 28nm 1P8M CMOS	Technology	Samsung 28nm 1P8M CMOS
3.6mm × 3.6mm	Die Area	3.6mm × 3.6mm
0.58V (@ 2MHz) ~ 1.04V (@ 250MHz)	Op. Condition	0.63V (@ 10MHz) ~ 1.0V (@ 250MHz)
DFXP+SR+ST	Data Type	DFXP+SR
(4/8/12/16)-bit × (4/8/12/16)-bit	Data Bit-precision	(4/8/12/16)-bit × (4/8/12/16)-bit
X	Pruning-aware Training	O (Channel-pruning)
X	Backward Unlocking	O
FF: In-out Slice Skip EP: In-out Slice Skip WG: In-slice Skip	Sparsity Exploitation	FF: In-slice Skip + FW Skip EP: In-slice Skip + BW Skip WG: In-slice Skip + G Skip
LAPS	Advanced Feature	Low-cost SR
955-to1162 mW	Power	850-to-1032 mW
4.53 TOPS	Peak Performance	7.07 TOPS
4.74 TOPS/W	Energy Efficiency	7.89 TOPS/W

FIGURE 12. Examples of algorithm-HW co-design in DNN training ASICs.

disturb the realization of low-latency training. To resolve this problem, it adopted pruning-aware training and DFA-based BU to reduce training latency. Since it does not require ST logic, it further optimized SR circuits by using the low-cost random number generator. In the same online tuning scenario, the HNPU-V2 can achieve 56.1% higher throughput and 66.5% higher energy efficiency than HNPU-V1. Although the training method of HNPU-V2 can be lossy to be used in other training tasks, it successfully realized low-latency DNN tuning for fast environmental adaptation. In contrast, the HNPU-V1 can support any long-term or medium-term training without accuracy loss by adopting changing precision during the training. Thanks to precision search, it can minimize required internal/external memory access compared with HNPU-V2 while training most of the network without loss. As shown in this example, the design methodology of the DNN training processor can be varied according to the target application, which may require different optimization schemes.

E. COMPARISON OF THE INFERENCE & TRAINING PROCESSOR DESIGN

Table 6 compares the main features of inference and training processors. Conventional inference processors focused only on the fast and efficient execution of well-trained networks while they do not consider whether the training of the network becomes slow or not. In contrast, the training processor should achieve high performance both in inference and training without compromising the training accuracy.

TABLE 6. Comparison of inference and training processor.

Type	Inference Processor	Training Processor
Purpose	• Fast / Efficient Inference • (Training can be slow)	• Training for Inference • Fast / Efficient Training
Algorithm Level Feature	• Fine-grained Mixed-precision [21] • Sparsity Encoding: Zero-length Encoding [21] / CSC [33, 47, 48] / Bitmap [24]	
	• Binary/Ternary Quant. [9, 10] • DNN Early-stopping [75, 76] • Output ReLU Speculation [24, 27]	• Backward Unlocking [69-74] • Quant-aware Training [27, 29, 79] • Pruning-aware Training [26, 31, 47] • Selective Layer Training [32, 84]
Architecture Level Feature	• Input or Weight Zero Skipping [13, 21, 24, 26, 27, 29, 30, 31, 34, 47-51] • Distributed Memory Architecture / Systolic Array [14, 32]	
	• Fused-layer based Acceleration [12, 75]	• Output Zero Skipping [24, 26, 27, 29, 31, 49] • Training-stage-pipeline [34, 68] • Transposable PE Array [20] • Sparse Weight Transposer [30] • Norm. Layer Reform [31, 32]
PE/Circuit Level Feature	• Dynamic Fixed-point MAC [3, 29, 34, 35, 43] • Voltage-Frequency Scaling	
	• Look-up Table based PE [3, 4, 10] • Mixed-mode Computing [5, 8, 9]	• Stochastic Rounding Circuit [29] • Binary BW Computing MAC [34] • FP-FXP Fused Multiply-Add Unit [16, 18, 19, 21, 22, 24, 26, 31]

1) PE/CIRCUIT-LEVEL DIFFERENCES

The inference processor can take advantage of ultralow-bit quantization because the network can be retrained multiple times in advance to compensate for accuracy degradation. Ultralow-bit quantization enables look-up-table-based computing [3], [4], [10] for energy-efficient inference acceleration. Furthermore, an inference processor can utilize analog-domain computing [8] if it repeats retraining after adding noise components to a prewell-trained network. However, both ultralow-bit quantization and mixed-mode computing can cause accuracy degradation during the DNN training and slow down the training curve. For this reason, the majority of training processors adopt precision-configurable MAC units which can also support high-bit-precision FP. It sometimes uses BW binarization [34], [43] but only in limited applications.

2) ARCHITECTURE-LEVEL DIFFERENCES

The training processor shows differences also in the architecture-level features. In the inference processor, they sometimes adopt fused-layer-based acceleration [12], [75] to remove external memory access that appears during layer-by-layer computing. The fused layer is less effective in the training processor because all intermediate IAs and OEs should be stored and reloaded at the WG stage. Moreover, the training processor shows a more complex data-path architecture due to the transpose-read of weights or WG operation. It utilizes additional features, such as the transposable PE array [20] and output sparsity exploitation [24], [26], [27], [31] to compensate for efficiency drop caused by complex data path.

3) ALGORITHM-LEVEL DIFFERENCES

Training processors also show different design methods from inference processors during the algorithm-hardware co-optimization. Recent application-specific inference processors [75], [76] adopt DNN early-stopping. It skips computations of the posterior layers if it judges them useless

at the prior computation steps. Branch-net-based inference prediction [75] uses layerwise binary classifiers to determine whether the remaining layers should be computed or not. Similarity maps generated in the Siamese network inform important spatial positions and it can also exclude the computation of useless tiles in the remaining layers [76]. Another feature is output ReLU prediction [24], [27] which predicts virtual output zeros by using only the minimum amount of precomputation. It helps the sparsity exploitation core to skip both input and output sparsity during the inference scenario. However, these two features cannot be used during the training because the wrong prediction can disturb the training of the original backbone network. Instead, the training processor utilizes features to compensate for the drop in energy efficiency during the inference. Runtime quantization [27], [29] and pruning automation [26] are unique features of training processors even though they are useless to inference processors.

X. CONCLUSION

The current AI system is smart only for the given applications but what we expect for AI is not to simply provide the predetermined routine passive processing. AI will cooperate together with human beings through active interaction. The purpose of the DNN training processor is to provide an AI system with dynamic adaptability. However, the design of the training processor is not that easy because it has more challenges compared with the inference processors. Training processors should support transpose-read of the weight during the EP stage. It needs to exploit three different types of sparsity: 1) input; 2) weight; and 3) output sparsity. The new bit-precision optimization methods are also required to realize training in the micro-AI systems. Unlike inference, it needs to reduce the required storage for IA and OE for efficient training on compact devices. Furthermore, the additional EP and WG stages should be finished as soon as possible to minimize the delay of next inference. As analyzed in Section IX, the design of the training processor should be accompanied by the algorithm-hardware co-optimization and it has unique design methodologies compared with the conventional inference processors.

In spite of the many realizations of DNN training, there are still challenges. First, current training processors showed few optimization methods during the WG stages. Primal weight needs high bit precision to maintain its training performance. Moreover, it shows memory-intensive computation because of the elementwise multiplication and large kernel convolution. To release the memory burden of the WG stage, both a new algorithmic approximation and hardware architecture are needed. The second issue is backward locking. Even if the acceleration of each layer becomes much faster, backward locking will eventually be the major obstacle to fast training. BU should be studied with the new algorithm to replace conventional BP-based training.

The majority of training processors adopted long-term personalization or customization for their target applications by training with personal datasets. However, in future AI research, online training will become mainstream. Han et al. [34], [80] already revealed that the DNN trained by general knowledge showed a poor performance in an unexpected situation. Moreover, in the 6G wireless communication network, the online training will be essential because the pretrained network cannot respond to the nonstationary nature of real-world situations to find optimal spectrum and base station [77]. Only online training can achieve such complicated real-time adaptation to realize the reliable 6G system. The application shift to online training will further highlight the necessity of an ultrahigh-speed and ultralow-power DNN training processor and system-level optimization.

In conclusion, the DNN training processors for micro-AI systems establish their own research area and show unique characteristics compared with the conventional inference processors. In order to improve the intelligence of the device, research should be extended to the training processor. The DNN training may not be the only answer for AI to reach human intelligence, but it will lead to the harmonious coexistence of AI and human beings.

REFERENCES

[1] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017.

[2] S. Choi, J. Lee, K. Lee, and H.-J. Yoo, "A 9.02mW CNN-stereo-based real-time 3D hand-gesture recognition processor for smart mobile devices," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2018, pp. 220–222.

[3] D. Shin, J. Lee, K. Lee, and H.-J. Yoo, "DNPU: An 8.1TOPS/W reconfigurable CNN-RNN processor for general-purpose deep neural networks," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2017, pp. 240–241.

[4] J. Lee, D. Shin, and H.-J. Yoo, "A 21mW low-power recurrent neural network accelerator with quantization tables for embedded deep learning applications," in *Proc. IEEE Asian Solid-State Circuits Conf. (A-SSCC)*, Nov. 2017, pp. 237–240.

[5] K. Bong, S. Choi, C. Kim, D. Han, and H.-J. Yoo, "A low-power convolutional neural network face recognition processor and a CIS integrated with always-on face detector," *IEEE J. Solid-State Circuits*, vol. 53, no. 1, pp. 115–123, Jan. 2018.

[6] K. Ueyoshi et al., "QUEST: A 7.49 TOPS multi-purpose log-quantized DNN inference engine stacked on 96 MB 3D SRAM using inductive-coupling technology in 40nm CMOS," in *Proc. IEEE Int. Solid-State Circuits Conf.*, Feb. 2018, pp. 216–218.

[7] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H.-J. Yoo, "UNPU: A 50.6TOPS/W unified deep neural network accelerator with 1b-to-16b fully-variable weight bit-precision," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, San Francisco, CA, USA, 2018, pp. 218–220.

[8] A. Amravati, S. B. Nasir, S. Thangadurai, I. Yoon, and A. Raychowdhury, "A 55nm time-domain mixed-signal neuromorphic accelerator with stochastic synapses and embedded reinforcement learning for autonomous micro-robots," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2018, pp. 124–126.

[9] Z. Jiang, Z. Jiang, J.-S. Seo, and M. Seok, "XNOR-SRAM: In-memory computing SRAM macro for binary/ternary deep neural networks," in *Proc. IEEE Symp. VLSI Technol.*, 2018, pp. 173–174.

[10] S. Kang, J. Lee, C. Kim, and H.-J. Yoo, "B-Face: 0.2 MW CNN-based face recognition processor with face alignment for mobile user identification," in *Proc. IEEE Symp. VLSI Circuits*, Honolulu, HI, USA, 2018, pp. 137–138.

[11] J. Song et al., "7.1 An 11.5TOPS/W 1024-MAC butterfly structure dual-core sparsity-aware neural processing unit in 8 nm flagship mobile SoC," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2019, pp. 130–132.

[12] J. Lee, D. Shin, J. Lee, J. Lee, S. Kang, and H.-J. Yoo, "A full HD 60 fps CNN super resolution processor with selective caching based layer fusion for mobile devices," in *Proc. Symp. VLSI Circuits*, Kyoto, Japan, 2019, pp. C302–C303.

[13] J.-H. Kim, J. Lee, J. Lee, H.-J. Yoo, and J.-Y. Kim, "Z-PIM: An energy-efficient sparsity aware processing-in-memory architecture with fully-variable weight precision," in *Proc. IEEE Symp. VLSI Circuits*, 2020, pp. 1–2.

[14] *Tensor Processing Unit–Second Generation (TPU-V2)*, Google, Menlo Park, CA, USA.

[15] "GTX 1080 Ti." NVIDIA. Accessed: Mar. 10, 2017. [Online]. Available: https://www.geforce.com/hardware/desktop-gpus/geforce-gtx-1080-ti/specifications

[16] "Tesla V100." NVIDIA. Accessed: Jun. 21, 2017. [Online]. Available: https://www.nvidia.com/kokr/data-center/v100

[17] B. Fleischer et al., "A scalable multi-teraOPS deep learning processor core for AI trainina and inference," in *Proc. IEEE Symp. VLSI Circuits*, Honolulu, HI, USA, 2018, pp. 35–36.

[18] J. Oh et al., "A 3.0 TFLOPS 0.62V scalable processor core for high compute utilization AI training and inference," in *Proc. IEEE Symp. VLSI Circuits*, Honolulu, HI, USA, 2020, pp. 1–2.

[19] A. Agrawal et al., "9.1 A 7nm 4-core AI chip with 25.6TFLOPS hybrid FP8 training, 102.4TOPS INT4 inference and workload-aware throttling," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, San Francisco, CA, USA, 2021, pp. 144–146.

[20] 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.

[21] J. Lee, J. Lee, D. Han, J. Lee, G. Park, and H.-J. Yoo, "7.7 LNPU: A 25.3 TFLOPS/W sparse deep-neural-network learning processor with fine-grained mixed precision of FP8-FP16," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2019, pp. 142–144.

[22] C.-H. Lu, Y.-C. Wu, and C.-H. Yang, "A 2.25 TOPS/W fully-integrated deep CNN learning processor with on-chip training," in *Proc. IEEE Asian Solid-State Circuits Conf. (A-SSCC)*, Macau, Macao, 2019, pp. 65–68.

[23] S. Yin and J.-S. Seo, "A 2.6 TOPS/W 16-bit fixed-point convolutional neural network learning processor in 65-nm CMOS," *IEEE Solid-State Circuits Lett.*, vol. 3, pp. 13–16, 2020.

[24] 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.

[25] J.-W. Su et al., "15.2 A 28 nm 64 Kb inference-training two-way transpose multibit 6T SRAM compute-in-memory macro for AI edge chips," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2020, pp. 240–242.

[26] S. Kim, J. Lee, S. Kang, J. Lee, and H.-J. Yoo, "A 146.52 TOPS/W deep-neural-network learning processor with stochastic coarse-fine pruning and adaptive input/output/weight skipping," in *Proc. IEEE Symp. VLSI Circuits*, Honolulu, HI, USA, 2020, pp. 1–2.

[27] F. Tu et al., "Evolver: A deep learning processor with on-device quantization–voltage–frequency tuning," *IEEE J. Solid-State Circuits*, vol. 56, no. 2, pp. 658–673, Feb. 2021.

[28] J. Park, S. Lee, and D. Jeon, "9.3 A 40 nm 4.81 TFLOPS/W 8b floating-point training processor for non-sparse neural networks using shared exponent bias and 24-way fused multiply-add tree," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, San Francisco, CA, USA, 2021, pp. 1–3.

[29] D. Han et al., "HNPU: An adaptive DNN training processor utilizing stochastic dynamic fixed-point and active bit-precision searching," *IEEE J. Solid-State Circuits*, vol. 56, no. 9, pp. 2858–2869, Sep. 2021.

[30] J. Lee et al., "OmniDRL: A 29.3 TFLOPS/W deep reinforcement learning processor with dualmode weight compression and on-chip sparse weight transposer," in *Proc. Symp. VLSI Circuits*, 2021, pp. 1–2.

[31] Y. Wang et al., "A 28nm 276.55 TFLOPS/W sparse deep-neural-network training processor with implicit redundancy speculation and batch normalization reformulation," in *Proc. Symp. VLSI Circuits*, 2021, pp. 1–2.

[32] S. Kim, S. Kang, D. Han, S. Kim, S. Kim, and H.-J. Yoo, "An energy-efficient GAN accelerator with on-chip training for domain-specific optimization," *IEEE J. Solid-State Circuits*, vol. 56, no. 10, pp. 2968–2980, Oct. 2021.

[33] Z. Yuan et al., "Sticker: A 0.41-62.1 TOPS/W 8 bit neural network processor with multi-sparsity compatible convolution arrays and online tuning acceleration for fully connected layers," in *Proc. IEEE Symp. VLSI Circuits*, Jun. 2018, pp. 33–34.

[34] D. Han, J. Lee, and H.-J. Yoo, "DF-LNPU: A pipelined direct feedback alignment-based deep neural network learning processor for fast online learning," *IEEE J. Solid-State Circuits*, vol. 56, no. 6, pp. 1630–1640, May 2021.

[35] S. Choi, J. Sim, M. Kang, Y. Choi, H. Kim, and L.-S. Kim, "An energy-efficient deep convolutional neural network training accelerator for in situ personalization on smart devices," *IEEE J. Solid-State Circuits*, vol. 55, no. 10, pp. 2691–2702, Oct. 2020.

[36] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec. 2017.

[37] Y. Chen, Y. Xie, L. Song, F. Chen, and T. Tang, "A survey of accelerator architectures for deep neural networks," *Engineering*, vol. 6, no. 3, pp. 264–274, Mar. 2020.

[38] M. Capra, B. Bussolino, A. Marchisio, M. Shafique, G. Masera, and M. Martina, "An updated survey of efficient hardware architectures for accelerating deep convolutional neural networks," *Future Internet*, vol. 12, no. 7, p. 113, Jul. 2020.

[39] J. Lee, S. Kang, J. Lee, D. Shin, D. Han, and H.-J. Yoo, "The hardware and algorithm co-design for energy-efficient DNN processor on edge/mobile devices," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 67, no. 10, pp. 3458–3470, Oct. 2020.

[40] D. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, Oct. 1986. [Online]. Available: https://doi.org/10.1038/323533a0

[41] H. Nam and B. Han, "Learning multi-domain convolutional neural networks for visual tracking," 2015, *arXiv:1510.07945*.

[42] M. Farhadi and Y. Yang, "TKD: Temporal knowledge distillation for active perception," in *Proc. IEEE Winter Conf. Appl. Comput. Vis. (WACV)*, Snowmass Village, CO, USA, 2020, pp. 942–951.

[43] D. Han, J. Lee, J. Lee, S. Choi, and H.-J. Yoo, "A 141.4 mW low-power online deep neural network training processor for real-time object tracking in mobile devices," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Florence, Italy, 2018, pp. 1–5.

[44] T. P. Lillicrap, D. Cownden, D. B. Tweed, and C. J. Akerman, "Random synaptic feedback weights support error backpropagation for deep learning," *Nat. Commun.*, vol. 7, Nov. 2016, Art. no. 13276.

[45] S. Han et al., "EIE: Efficient inference engine on compressed deep neural network," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, 2016, pp. 243–254.

[46] S. Han et al., "ESE: Efficient speech recognition engine with sparse LSTM on FPGA," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays (FPGA)*, 2017, pp. 75–84.

[47] J. Zhang, X. Chen, M. Song, and T. Li, "Eager pruning: Algorithm and architecture support for fast training of deep neural networks," in *Proc. ACM/IEEE 46th Annu. Int. Symp. Comput. Archit. (ISCA)*, Phoenix, AZ, USA, 2019, pp. 292–303.

[48] P. Dai et al., "SparseTrain: Exploiting dataflow sparsity for efficient convolutional neural networks training," in *Proc. 57th ACM/IEEE Des. Autom. Conf. (DAC)*, San Francisco, CA, USA, 2020, pp. 1–6.

[49] E. Qin et al., "SIGMA: A sparse and irregular GEMM accelerator with flexible interconnects for DNN training," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, 2020, pp. 58–70.

[50] D. Yang, A. Ghasemazar, X. Ren, M. Golub, G. Lemieux, and M. Lis, "Procrustes: A dataflow and accelerator for sparse deep neural network training," in *Proc. 53rd Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, Athens, Greece, 2020, pp. 711–724.

[51] M. Mahmoud et al., "TensorDash: Exploiting sparsity to accelerate deep neural network training," in *Proc. 53rd Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, 2020, pp. 781–795.

[52] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolution network," in *Proc. ICML Deep Learn.*, 2015, pp. 1–5.

[53] D. Misra, "Mish: A self regularized non-monotonic activation function," 2019, *arXiv:1908.08681*.

[54] "NVIDIA A100 tensor core GPU architecture." NVIDIA. Accessed: Jun. 28, 2021. [Online]. Available: https://www.nvidia.com/content/dam/en-zz/Solutions/DataCenter/nvidia-ampere-architecture-whitepaper.pdf
[55] U. Köster et al., "Flexpoint: An adaptive numerical format for efficient training of deep neural networks," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst. (NIPS)*, Red Hook, NY, USA, 2017, pp. 1740–1750.
[56] N. Wang, J. Choi, D. Brand, C.-Y. Chen, and K. Gopalakrishnan, "Training deep neural networks with 8-bit floating point numbers," in *Proc. Int. Conf. Adv. Neural Inf. Process. Syst.*, 2018, pp. 7685–7694.
[57] X. Sun et al., "Hybrid 8-bit floating point (HFP8) training and inference for deep neural networks," in *Proc. Int. Conf. Adv. Neural Inf. Process. Syst.*, 2019, pp. 4901–4910.
[58] X. Sun et al., "Ultra-low precision 4-bit training of deep neural networks," in *Proc. Int. Conf. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 1796–1807.
[59] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," Feb. 2015, *arXiv:1502.02551*.
[60] F. Seide, H. Fu, J. Droppo, G. Li, and D. Yu., "1-bit stochastic gradient descent and its application to data-parallel distributed training of speech DNNs," in *Proc. INTERSPEECH*, 2014, pp. 1058–1062.
[61] D. Alistarh, D. Grubic, J. Z. Li, R. Tomioka, and M. Vojnovic, "QSGD: Communication-efficient SGD via gradient quantization and encoding," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst. (NIPS)*, 2017, pp. 1707–1718.
[62] W. Wen et al., "TernGrad: Ternary gradients to reduce communication in distributed deep learning," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst. (NIPS)*, 2017, pp. 1508–1518.
[63] Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally, "Deep gradient compression: Reducing the communication bandwidth for distributed training," 2017, *arXiv:1712.01887*.
[64] N. Strom, "Scalable distributed DNN training using commodity GPU cloud computing," in *Proc. 16th Annu. Conf. Int. Speech Commun. Assoc.*, 2015, pp. 1–5.
[65] A. F. Aji and K. Heafield, "Sparse communication for distributed gradient descent," in *Proc. Empir. Methods Nat. Lang. Process. (EMNLP)*, 2017, pp. 440–445.
[66] D. Shin, G. Kim, J. Jo, and J. Park, "Prediction confidence based low complexity gradient computation for accelerating DNN training," in *Proc. 57th ACM/EDAC/IEEE Des. Autom. Conf. (DAC)*, 2020, pp. 1–6.
[67] J. Shin, S. Choi, Y. Choi, and L.-S. Kim, "A pragmatic approach to on-device incremental learning system with selective weight updates," in *Proc. 57th ACM/IEEE Des. Autom. Conf. (DAC)*, San Francisco, CA, USA, 2020, pp. 1–6.
[68] J. Li et al., "TNPU: An efficient accelerator architecture for training convolutional neural networks," in *Proc. 24th Asia South Pac. Des. Autom. Conf. (ASPDAC)*, 2019, pp. 450–455.
[69] Z. Huo, B. Gu, Q. Yang, and H. Huang, "Decoupled parallel backpropagation with convergence guarantee," in *Proc. ICML*, 2018, pp. 2103–2111.
[70] Z. Huiping, Y. Wang, Q. Liu, and Z. Lin, "Fully decoupled neural network learning using delayed gradients," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 10, pp. 6013–6020, Oct. 2022.
[71] A. Nøkland, "Direct feedback alignment provides learning in deep neural networks," in *Proc. 30th Conf. Neural Inf. Process. Syst. (NIPS)*, 2016, pp. 1037–1045.
[72] D. Han and H.-J. Yoo, "Efficient convolutional neural network training with direct feedback alignment," 2019, *arXiv:1901.01986*.
[73] B. Crafton, A. Parihar, E. Gebhardt, and A. Raychowdhury, "Direct feedback alignment with sparse connections for local learning," 2019, *arXiv:1903.02083*.
[74] D. Han, G. Park, J. Ryu, and H.-J. Yoo, "Extension of direct feedback alignment to convolutional and recurrent neural network for bio-plausible deep learning" 2020, *arXiv:2006.12830*.
[75] Y. Kim, D. Han, C. Kim, and H.-J. Yoo, "A 0.22–0.89 mW low-power and highly-secure always-on face recognition processor with adversarial attack prevention," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 67, no. 5, pp. 846–850, May 2020.
[76] S. Kim, S. Kim, S. Kim, D. Han, and H.-J. Yoo, "A 64.1mW accurate real-time visual object tracking processor with spatial early stopping on siamese network," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 68, no. 5, pp. 1675–1679, May 2021.
[77] C. She et al., "A tutorial on ultrareliable and low-latency communications in 6G: Integrating domain knowledge into deep learning," *Proc. IEEE*, vol. 109, no. 3, pp. 204–246, Mar. 2021.
[78] Y. Wang, Y. Qin, L. Liu, S. Wei, and S. Yin, "SWPU: A 126.04 TFLOPS/W edge-device sparse DNN training processor with dynamic sub-structured weight pruning," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 69, no. 10, pp. 4014–4027, Oct. 2022.
[79] S. Song, S. Kim, G. Park, D. Han, and H.-J. Yoo, "A 49.5 mW multi-scale linear quantized online learning processor for real-time adaptive object detection," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 69, no. 5, pp. 2443–2447, May 2022.
[80] D. Han et al., "A 0.95 mJ/frame DNN training processor for robust object detection with real-world environmental adaptation," in *Proc. IEEE 4th Int. Conf. Artif. Intell. Circuits Syst. (AICAS)*, 2022, pp. 37–40.
[81] 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.
[82] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," in *Proc. NIPS Workshop Private Multi-Party Mach. Learn.*, 2016, pp. 1–5.
[83] H. Cai, C. Gan, L. Zhu, and S. Han, "TinyTL: Reduce memory, not parameters for efficient on-device learning," in *Proc. 34th Int. Conf. Neural Inf. Process. Syst. (NIPS)*, 2020, pp. 1–13.
[84] J. Lin, L. Zhu, W.-M. Chen, W.-C. Wang, C. Gan, and S. Han, "On-device training under 256KB memory," in *Proc. 36th Int. Conf. Neural Inf. Process. Syst. (NIPS)*, 2022, pp. 1–47.

DONGHYEON HAN (Graduate Student Member, IEEE) received the B.S. and M.S. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology, Daejeon, South Korea, in 2017 and 2019, where he is currently pursuing the Ph.D. degree.

His current research interests include low-power system-on-chip design, especially focused on deep neural network accelerators and hardware-friendly algorithms for deep learning.

SANGHOON KANG (Member, IEEE) received the B.S., M.S., and Ph.D. degrees from the School of Electrical Engineering, Korea Advanced Institute of Science and Technology, Daejeon, South Korea, in 2016, 2018, and 2022, respectively.

His research interests include energy-efficient deep learning processor design, hardware-friendly deep learning algorithm optimizations, and embedded intelligence system development.

SANGYEOB KIM (Graduate Student Member, IEEE) received the B.S. and M.S. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology, Daejeon, South Korea, in 2018 and 2020, respectively, where he is currently pursuing the Ph.D. degree.

His current research interests include energy-efficient system-on-chip design, especially focused on deep neural network accelerators, neuromorphic hardware, and computing-in-memory accelerators.

JUHYOUNG LEE (Graduate Student Member, IEEE) received the B.S. and M.S. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology, Daejeon, South Korea, in 2017 and 2019, respectively, where he is currently pursuing the Ph.D. degree.

His current research interests include energy-efficient multicore architectures/accelerator ASICs/systems, especially focused on artificial intelligence, including deep reinforcement learning and computer vision, energy-efficient computing-in-memory accelerator, and deep learning algorithm for efficient processing.

HOI-JUN YOO (Fellow, IEEE) graduated from the Electronic Department, Seoul National University, Seoul, South Korea, in 1983, and received the M.S. and Ph.D. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology, Daejeon, South Korea, in 1985 and 1988, respectively.

Prof. Yoo served as a member for the Executive Committee of ISSCC, Symposium on VLSI, and A-SSCC, the TPC Chair for A-SSCC 2008 and ISWC 2010, the IEEE Distinguished Lecturer from 2010 to 2011, the Far East Chair for ISSCC from 2011 to 2012, the Technology Direction Sub-Committee Chair for ISSCC in 2013, the TPC Vice Chair for ISSCC in 2014, and the TPC Chair for ISSCC in 2015. For more details, see http://ssl.kaist.ac.kr.