IEEE Open Journal of the
Solid-State Circuits Society

Received 29 June 2021; revised 14 September 2021; accepted 5 October 2021. Date of publication 13 October 2021; date of current version 21 October 2021.

Digital Object Identifier 10.1109/OJSSCS.2021.3119554

An Overview of Energy-Efficient Hardware Accelerators for On-Device Deep-Neural-Network Training

JINSU LEE (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 for Information & Communications Technology Promotion (IITP) grant funded by the Korea Government (MSIP, On-Device Instant Learning Complex Intelligence Processor Architecture and ADAS Design for Self-Driving Platform) under Grant 2019-0-01372.

ABSTRACT Deep Neural Networks (DNNs) have been widely used in various artificial intelligence (AI) applications due to their overwhelming performance. Furthermore, recently, several algorithms have been reported that require on-device training to deliver higher performance in real-world environments and protect users' personal data. However, edge/mobile devices contain only limited computation capability with battery power, so an energy-efficient DNN training processor is necessary to realize on-device training. Although there are a lot of surveys on energy-efficient DNN inference hardware, the training is quite different from the inference. Therefore, analysis and optimization techniques targeting DNN training are required. This article aims to provide an overview of energy-efficient DNN processing that enables on-device training. Specifically, it will provide hardware optimization techniques to overcomes the design challenges in terms of distinct dataflow, external memory access, and computation. In addition, this paper summarizes key schemes of recent energy-efficient DNN training ASICs. Moreover, we will also show a design example of DNN training ASIC with energy-efficient optimization techniques.

INDEX TERMS Application specific integrated circuit (ASIC), deep neural network (DNN), DNN training, on-device training, machine learning (ML), energy efficient hardware, digital hardware accelerator.

I. INTRODUCTION

DEEP neural network (DNN) [1] has been widely studied across all technology domains due to the superior accuracy in various applications such as computer vision [2]–[11], natural language processing (NLP) [12], [13], and autonomous system [14]–[16]. Moreover, along with the explosive growth of hardware, algorithms, and datasets, DNN has provided high-performance artificial intelligence (AI) to users in real life.

There are two phases of DNN process: training and inference. DNN training requires a significant amount of operations, so user's edge/mobile devices had provided only inference with downloaded DNN parameters which pre-trained on cloud servers. However, recently, several studies [17]–[24] have reported that require on-device DNN training to provide more advanced AI service. In order to protect user's private data, or achieve higher performance in real-world applications, these works tried to achieve their purpose by on-device DNN training as shown in Fig. 1. References [17]–[19] proposed DNN training scheme using private dataset stored on user devices. References [20]–[22] resolved accuracy degradation problems caused by domain difference between specific environment and dataset used for training on the cloud server, and [23]–[24] trained DNN to actuate robot using deep reinforcement learning.

However, DNN training iteratively processes three distinct steps to find high accuracy model parameters, inducing a large number of operations, external memory access, and various dataflow. And it makes the realization of on-device DNN training very challenging since edge devices contain only limited computation capability with battery power. Therefore, high energy-efficient hardware accelerator

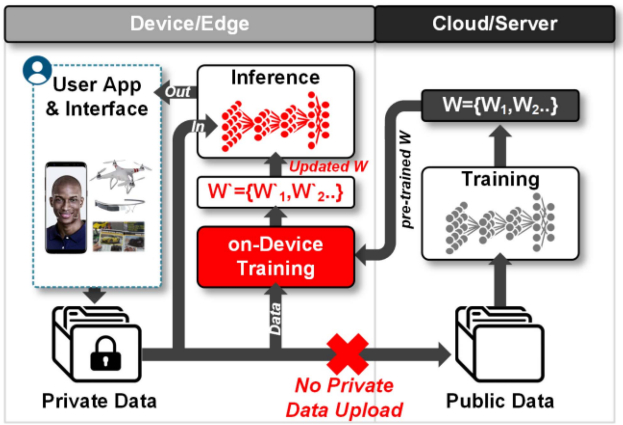


FIGURE 1. On-device DNN training.

is necessary to implement high-performance AI with DNN training on edge devices. Although there are previous survey papers [25]–[28] for energy-efficient DNN hardware, they have only focused on DNN inference. And the DNN training has quite different characteristics from the inference, limiting applying optimization techniques used in inference to training. Besides, existing surveys of optimization DNN training on hardware [29] are targeted at cloud servers with high computing and power capabilities [30], so they are not applicable to on-device DNN training.

In this article, we discuss energy-efficient optimization techniques for on-device DNN training. In particular, we analyze three design challenges for energy-efficient DNN training: dataflow, external memory access, and computation. First, the distinct dataflow of three DNN training steps are analyzed from the perspective of operational patterns and memory access patterns. Second, we break down external memory access and categorize compression methods for intermediate data that takes up the most significant amount. Third, computational optimization opportunities are summarized in terms of bit precision and sparsity. Moreover, we introduce the optimization techniques of recent DNN training hardware and describe an example of an energy-efficient DNN training ASIC design. This paper is organized as follows. Section II describes hardware design challenges for DNN training. Section III introduces recent DNN training ASICs and their key optimization techniques. Section IV shows a design example of an energy-efficient DNN training ASIC with a silicon chip. Section V summarizes DNN training ASICs. Finally, Section VI provides the key takeaways in conclusion.

II. HW DESIGN CHALLENGES FOR DNN TRAINING

The DNN training consists of three steps, as shown in Fig. 2, forward propagation (FP), backward propagation (BP), and weight gradient update (WG). First, FP produces the results of DNN with the given input data of the mini-batch. Second, BP calculates errors between labels and the results of the FP and transfers them to prior layers. This is performed by multiplying the weight of each layer with the error of the next layer. Then the calculated error is propagated to the prior layer. Third, weight gradient update (WG) is a process to update the weights of each layer to the direction of minimizing error. This process is performed by using features and errors for each layer. The DNN training process repeats the three steps until DNN accuracy converges, which results in many computations and memory accesses.

Although there are many studies about high energy-efficient DNN inference hardware optimizing memory access and computation, DNN training is quite different from DNN inference. Besides, the differences make several design challenges that have not been considered in DNN inference ASIC design. The following sections explain the design challenges of DNN training ASIC design in terms of dataflow, memory access, and computation.

A. DATAFLOW FOR THREE DNN TRAINING STEPS

The operations for each training step are commonly based on convolution or matrix multiplication operations, but each step has a distinct dataflow. During the FP step, it convolves each input channel of input features (X) with weights (W). After then, the results are added to produce a channel of output features (X). For the BP step, a channel of output gradients (δ) is convolved with rotated weights. Then, the output channel results are added to obtain a channel of input gradient (δ). the BP is similar to the FP, except that the W tensor is intra-channel-wise rotated and inter-channel-wise transposed. During the WG step, it convolves a channel of input features with a channel of output gradients to produce a weight gradient. Unlike the FP and the BP, there is no channel-to-channel accumulation.

The three steps of DNN training share a data array as an operand but have different dataflow. This also leads to different memory access patterns for the same data array. For example, Fig. 2 shows the dataflow of DNN training steps in a PE which has an output stationary data-path with memory layout optimized for only the FP. In order to maximize output reuse in the FP step, the input channels of input features and weights must be loaded firstly. Therefore, for efficient coalesced access, the input channel of input features and weights tensors is set to the inner-most dimension that allocates input channels to adjacent addresses.

The weights and input features that are already stored with the FP-optimized memory layouts are also accessed in the BP and the WG steps with different patterns. In the BP step, because transposed weights and output gradients are used to calculate an input gradient, output channels of weights must be fed first for output reuse. However, the inner-most dimension of weight array is already set to the input channel, so the output channels of the weights are located at intervals causing stride accesses. During the WG step, the stride access is also caused by loading input features and output gradients, as shown in Fig. 2. The input features and output gradients must be loaded spatially first to maximize the output reuse, but the neighbor pixels in the same channel are located far from each other in memory address space.

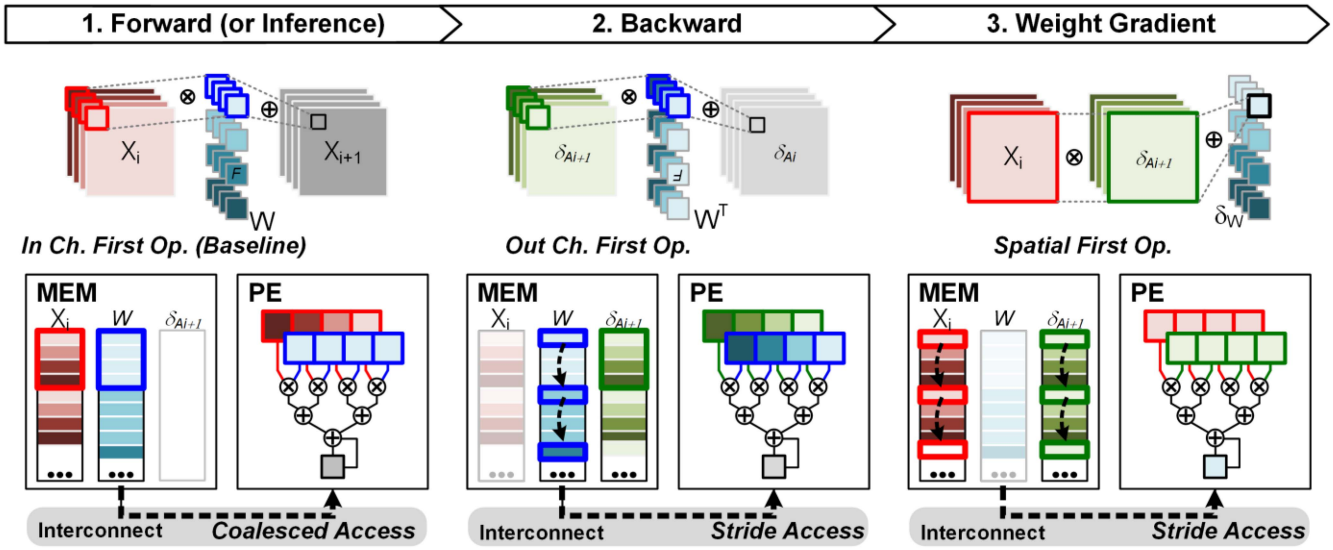


FIGURE 2. Dataflow of three DNN training steps and mapping to DNN hardware accelerator.

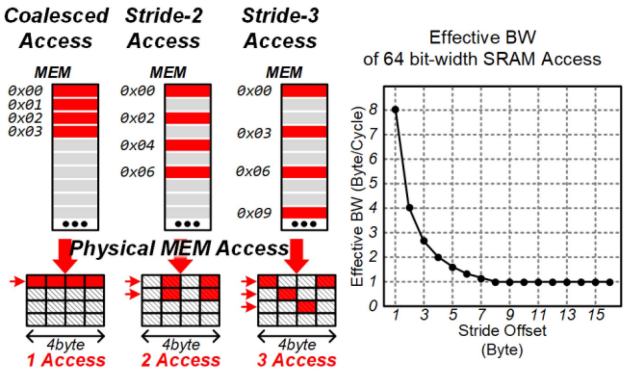


FIGURE 3. Effective bandwidth of different memory access patterns.

Fig. 3 shows effective bandwidth according to memory access patterns. Accessing data array from consecutive addresses allows utilizing all data in a row of the physical memory array, maximizing effective bandwidth and energy efficiency. However, non-consecutive address access, such as the stride access, causes bandwidth degradation and redundant memory array access. For example, to access 8×16bit data allocated in consecutive addresses on hardware with 64-bit width physical memory and data bus, only two memory accesses and data transfers are required. However, 8 times memory access and data transfers are needed if they are allocated at more than 8byte address intervals. In addition, access to data arrays of non-continuous addresses leads to increased access latency due to the inability to burst access when using external memory. Therefore, memory layouts and data-path must be determined by considering distinct dataflow for efficient processing. Section III-A will introduce and categorize dataflow optimization methods of recent DNN training ASICs.

B. EXTERNAL MEMORY ACCESS

For energy-efficient DNN training, lowering external memory access is one of the most important design challenges.

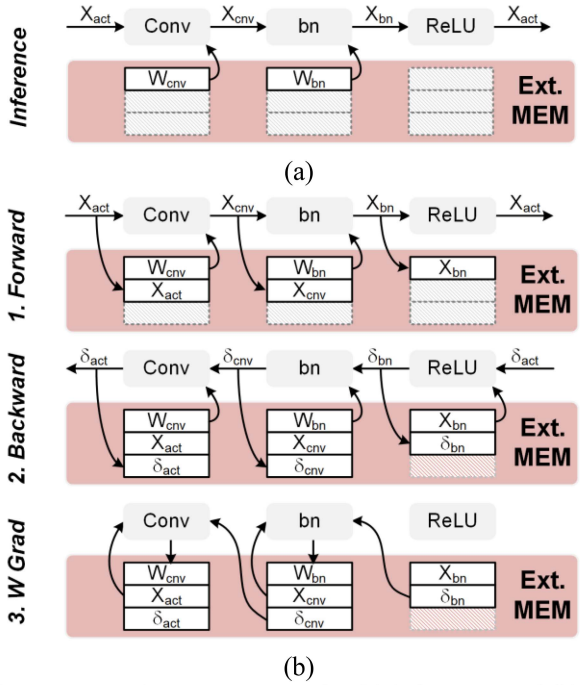


FIGURE 4. External memory usage for (a) inference, and (b) training.

Furthermore, limited memory capacity makes it more challenging to operate DNN training. Recently, lots of DNN inference algorithms and hardware have reduced external memory access for high energy efficiency. Many notable studies for the inference have focused more on lowering model parameters such as weight quantization and pruning [31], [32]. However, in DNN training, a different approach from DNN inference is needed to reduce external memory access. This is because the external memory access pattern is different from the inference. Fig. 4 shows the differences between external memory access patterns in DNN inference and training for the most commonly used CNN structure, Conv-BN-ReLU.

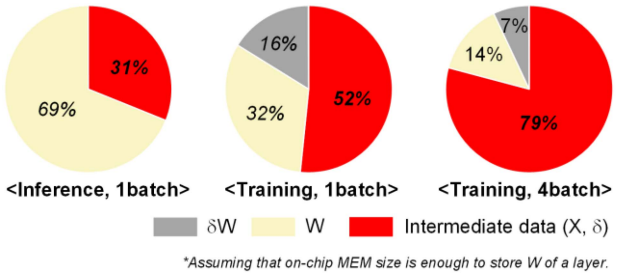


FIGURE 5. Breakdown of external memory access for ResNet-18 training.

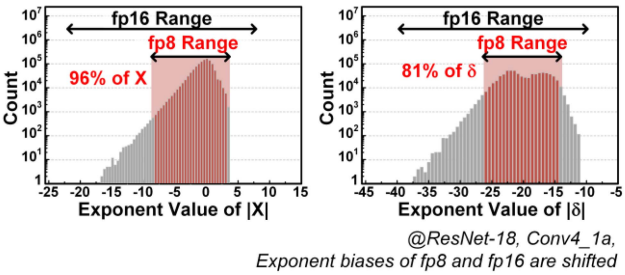


FIGURE 6. DNN training data distribution.

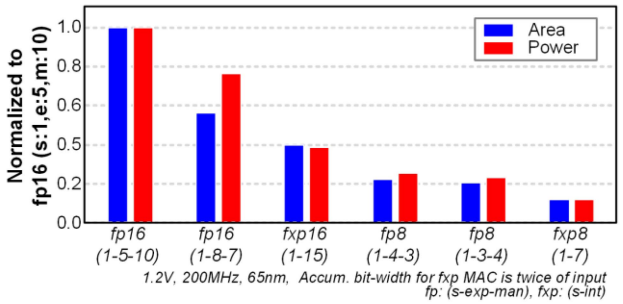


FIGURE 7. Area and power of MAC for bit-precisions.

In DNN inference, input features and weights are loaded from the external memory to calculate output features. Then, the output features are normalized with parameters of BN layer [33], and passed to the next layer after ReLU activation. Since the intermediate output features are never used again once each layer operation is completed, it is not required to allocate and keep the memory space for output features. Moreover, BN and ReLU are element-wise operations in the inference, so they can be merged with the convolution layer eliminating external memory access for intermediate features of BN and ReLU layers.

Unlike the inference, the BP and WG steps use the intermediate features produced in the FP for the training. So, the memory space for batches of intermediate features must be allocated causing a large amount of memory access and footprint. Besides, the mean and standard deviation of output features have to be calculated for the BN. As a result, the feature load/store accounts for the largest external memory access. Fig. 5 shows the breakdown for external memory access for the inference and training. The proportion of memory usage of features in training is higher than inference. Therefore, it is essential to lower the size of the intermediate features in the training. This paper will introduce how the recent DNN training ASICs have reduced external memory access in Section III.

C. COMPUTATION

DNN training iterates the FP, BP, and WG steps to optimize DNN parameters, causing many operations. Because each step has a similar contribution to the total amount of operations, DNN training hardware must efficiently compute all three steps. Many DNN inference hardware accelerators have explored sparsity [34]–[40] and bit-precision [41]–[45] for energy-efficient operation. However, there are limitations to borrowing energy-efficient inference hardware for training due to the different computational characteristics of the training and the inference. Most DNN inference ASICs have focused on optimizing bit-precisions using a fixed-point (fxp) with a low bit-width. Although the fixed-point with a low bit-width has high energy efficiency, it cannot cover the wide dynamic range of the gradients during the BP step, as shown in Fig. 6.

To cover a wide dynamic range of the gradients with a standard bit-precision, at least IEEE half-precision is required. Otherwise, custom bit-precision and hardware are needed. When designing a custom bit-precision, it is crucial to reduce the bit-width and determine the data format with the limited bit-width. Fig. 7 shows the relative area and power consumption for various data formats. Even if the same bit-width is used, the area and power consumption are different between data formats. In addition, since the dynamic range and resolution are diverse depending on the data format, it is necessary to determine an efficient format without losing the training accuracy. Recently, many hardware has proposed custom data formats, and they will be described in Section III.

The sparsity allows DNN hardware to reduce computation power and latency without accuracy loss, and most DNN inference hardware handles sparsity. However, DNN training and inference have different sparsity characteristics. So, the sparsity-aware hardware for the inference cannot be applied to the training. For the inference, the algorithms such as weight pruning and quantization produce lots of zeros in weights. In contrast, it is difficult to artificially create zeros in weights for DNN training because the weights are updated for every iteration. Although there are zeros in input features by ReLU in the FP and the WG steps, not in the BP step, which takes up a large amount of computation. Therefore, sparsity handling techniques for DNN training must newly be explored.

III. DNN TRAINING OPTIMIZATION METHODS

There are several challenges to design an energy-efficient DNN training hardware, because of distinct dataflow, large memory access and usage, and energy-consuming operators. Recently, energy-efficient DNN training ASICs [46]–[62]

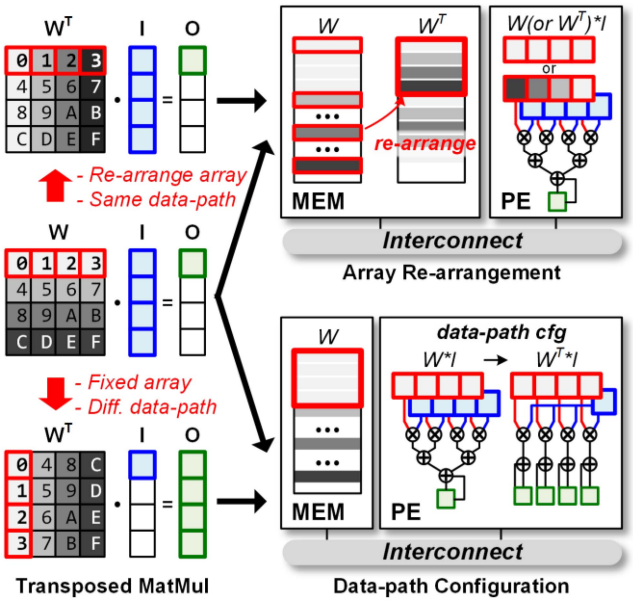


FIGURE 8. Transposed matrix multiplication with array re-arrangement and data-path configuration.

TABLE 1. Summary of dataflow optimizations in DNN training ASICs.

Optimization Method	Description	ASIC
Array re-arrangement (in memory)	Transpose in Custom SRAM	Su et al. [60]
	Transpose in Custom SRAM	Seo et al. [61]
	Diagonal storage pattern w/ bit-rotator	Yin et al. [55]
	Tiled W re-arrangement	Lu et al. [50]
	Tiled W re-arrangement w/ regfile array	LNPU [47]
Data-path configuration (in PE array)	2D torus & hierarchical memory	Fleischer et al. [46]
	Heterogeneous cores	Choi et al. [51]
	Heterogeneous cores w/ DFA algorithm	Han et al. [52]
	Transposable PE array data-path	Kim et al. [49]
	Reversible data-path w/ input zero-skip	LNPU [47]

have been presented to overcome the challenges. This Section introduces and categorizes optimization techniques of the recent DNN training ASICs in terms of dataflow, external memory access, and computation.

A. DATAFLOW OPTIMIZATIONS

In DNN training, the FP, BP, and WG have distinct dataflow. And the optimal memory layout is determined by each dataflow and hardware architecture. However, a tensor stored in the external memory are used for the three distinct dataflow, and this makes it difficult to optimize the operations for all three DNN training steps. For example, weights are loaded in both the FP and BP for matrix multiplication (MatMul) but only transposed in the BP. The incongruity between dataflow and memory layout causes redundant memory accesses and lowers core utilization, resulting in speed and energy efficiency degradation. Recent studies for energy-efficient DNN training ASICs have presented several methods to address this problem, and they can be divided into two categories; array re-arrangement and data-path configuration.

Fig. 8 shows an example of a transposed MatMul with dataflow optimization techniques. In Fig. 8, a row of weight matrix is the inner-most dimension to optimize MatMul for the output stationary and input channel parallel architecture. However, the weight matrix whose inner-most dimension is a row causes inefficient stride access during transposed MatMul, and it lowers PE utilization. First, memory re-arrangement can be applied to improve effective memory bandwidth and PE utilization. The memory re-arrangement modifies array structure to the optimized memory layout before feeding to PEs. In the example of Fig. 8, by re-arranging the inner-most dimension of weights array from a row to a column in advance to start MatMul, the weight matrix can be accessed as coalesced. The memory re-arrangement is simple to implement with DMA and software control. Furthermore, for small tensors, memory re-arrangement latency can be hidden in computation latency. However, it requires additional memory size and redundant memory accesses, besides the array re-arrangement takes a long time for large tensors, degrading overall hardware performance.

Table 1 shows the dataflow optimization techniques of recent DNN training ASICs. References [47], [50], [55], [60], [61] used array re-arrangement for dataflow optimization. References [47], [50] loads tiled weights from external memory and stores to on-chip SRAM or regfiles in transposed order. Thanks to the array re-arrangement, PE array can be shared between the FP and BP steps. It is the simplest design, but has a disadvantage when apply to lower bit-width data than the bit-width of data bus and physical memory, which consumes redundant memory access and long transfer latency. Reference [55] proposed a storage pattern that can be read array for both transposed and non-transposed manner without redundant memory access, but additional logic, a bit-rotator, is required. Reference [60], [61] proposed custom on-chip SRAM which can read data as transposed. The custom SRAMs allows to read tensors as transposed without redundant access. However, the custom SRAM has low SRAM cell density leading low area-efficiency.

A configurable data-path can also optimize the different dataflow of DNN training steps. The weights in a column of the transposed matrix are loaded first for the transposed MatMul without array re-arrangement. Because the weights in a column cannot be accumulated, PE utilization is severely degraded when running on the channel parallel and output stationary PE architecture. However, the input stationary

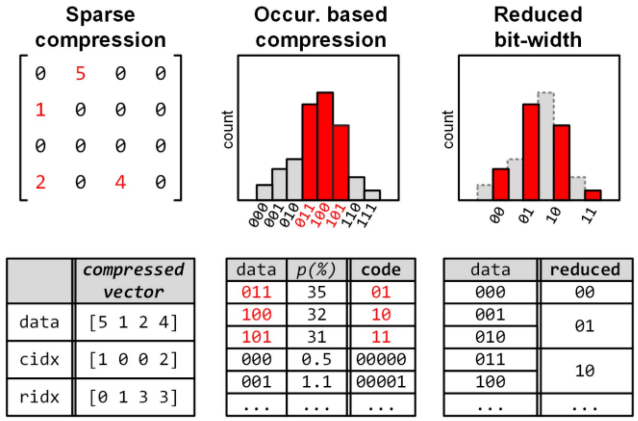


FIGURE 9. Compression methods for the DNN tensor.

TABLE 2. Summary of key features for DNN tensor compression.

HW	Sparse compression	Occur. based compression	Reduced bit-width
Agrawal et al. [57]	-	-	fp8 + bias
Part et al. [59]	-	-	fp8 + bias
PNPU [56]	ZVC	-	fp8 or fp16
GANPU [54]	ZVC	-	fp8 or fp16
GIST [64]	CSR	-	1b for ReLU
LNPU [47]	RLE	-	fp16/fp8 + bias
Kim et al. [49]	-	Exp. bits encoding	Feature: bfloat W: fxp 4,8,16
JPEG-ACT [63]	RLE	Huffman code	8b quant. (freq. domain)

data-path allows high PE utilization thanks to each weight's independent output accumulation. As such, high performance can be maintained by the configuration of data-path without changing the memory layout.

References [46], [47], [49], [51], [52] supports multiple data-paths in a PE array. Reference [46] can configure data-path to weight stationery, output stationery, and row-stationary with 2D torus interconnector and memory hierarchy. The configurable data-path allows high PE utilization for all DNN training steps. Reference [47] can reverse data-paths for weight and output. And thanks to the fixed input data-path which handles input zeros, input zero MAC operation skipping is supported for all DNN training steps. Reference [49] provides transposed and non-transposed MatMul without redundant memory access by exchangeable feeding paths of input and weight. References [51], [52] optimized different dataflow of DNN training steps with heterogeneous architectures.

B. EXTERNAL MEMORY ACCESS AND FOOTPRINT REDUCTION

During DNN training, the intermediate features which account for the largest portion of memory access have characteristics of sparsity, locality, and loss robustness. The natures of the intermediate features allow the tensor to be compressed reducing memory access and footprint. Fig. 9 shows three types of compression methods for DNN tensors: sparse compression, probability of occurrence based compression, and reduced bit-width.

ReLU, a derivative of ReLU, and a derivative of max-pooling cause redundant zeros in the intermediate features. Sparse compression that represents the tensors into a non-zero vector and an index vector reduces the byte size of data without any loss. There are several ways to present the index vector. References [54], [56] adopted zero-value compression (ZVC). The index vector for ZVC is a binary mask that has the same number of elements as the tensor. The index vector indicates whether the element is zero or not. Reference [64] uses compressed sparse row (CSR) which has two index vectors, rows and column indices. Reference [47] compresses intermediate features with run-length encoding (RLE). The index vector of RLE represents a number of consecutive zeros between non-zero values. In addition, intermediate features can be compressed more efficiently using the locality characteristics. Because of the locality characteristic, zeros are concentrated. So, [69]–[71] utilized this characteristic and reduced the index size of sparse compression by hierarchically compressing features. References [69]–[71] divided the features into grids and compressed them in grid-level first. After that, only non-zero grids are compressed at a feature level. The hierarchical sparse compression can be effectively applied to training to reduce intermediate features which take most external memory access.

The probability of occurrence-based compression reduces data size by encoding more likely to occur values into low bit-width code. In the example of Fig. 9, only three values, 3b011, 3b100 and 3b101, account for 98% of the total data. And, encoding three values into 2bits and the remaining five values into 5bits can reduce the data size by 31%. There are several works that use the probability of occurrence-based compression to reduce the features. Reference [49] uses bfloat which allocates 8bit for an exponent value, and replaced 3 most-frequent exponent values with 2bit code. Although the bit-width of less frequent values is increased, the overall byte size of the intermediate feature size is decreased. Reference [63] adopts Huffman coding, which encodes high likely to occur values into low bits.

Due to the loss robustness, reduced bit-width can be used for DNN training, so most DNN hardware uses fp16. Furthermore, some ASICs aggressively reduced bit-width, such as fp8, with algorithmic assistance [47], [57], [59]. In addition, depending on the purpose of the stored data, the bit-width can be reduced without loss. For example, [64] uses 1 bit for ReLU-Pool layer pairs. Because in the BP, ReLU only needs whether the feature value is positive or not.

Further from applying each three compression methods to DNN data, the compression methods can be combined to minimize the external memory access and footprint. Reference [63] adopts JPEG encoding. In more detail, [63] quantizes the intermediate features to 8bit in the frequency

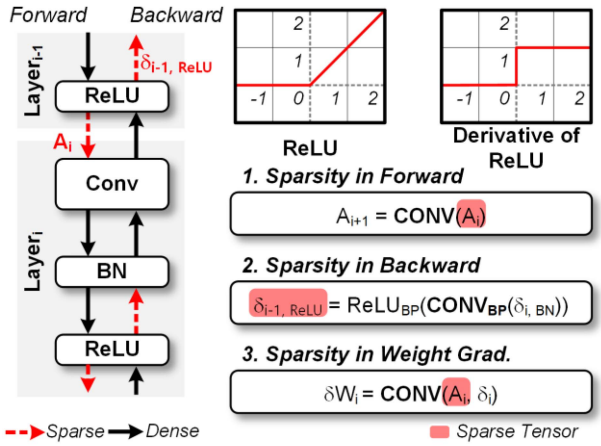


FIGURE 10. External memory access reduction schemes.

domain, producing lots of zeros. Then, the zeros are eliminated by RLE. Moreover, Huffman encoding is used to reduce the bit-width of frequent data at the end of the JPEG encoding. Reference [47] suggested FGMP which uses both fp8 and fp16 for a tensor. The fp8 is used for small and high occurrence, and the high bit-width, fp16, is used for large values which cannot be covered by fp8. In addition, zeros produced by dividing a tensor into two fp8/fp16-groups are compressed into RLEs.

C. COMPUTATION

1) SPARSITY HANDLING

DNN training ASICs have implemented energy-efficient compute cores with sparsity handling and bit-precision optimization. The zero input MAC operations that do not affect output can be eliminated leading energy efficiency improvement, and many DNN training accelerators support skipping native zeros [47], [54], [56], [58], [62], [73]. However, the energy-efficiency improvement is limited because there is no zero in the BP which accounts for a large portion of the total number of operations. Several DNN training ASICs breakthrough the limitation without any loss by layer merging and output zero-skipping logics. As shown in Fig. 10, the derivative of ReLU is determined by the features calculated in the FP step. Therefore, regardless of the back-propagated gradients in the BP step, it is possible to know in advance whether the input gradient of the ReLU is zero. Thus, unnecessary convolution operations that results are zeros can be pre-detected by merging ReLU of the pre-layer, and eliminated with the help of output zero-skipping logics. Reference [72] proposed hardware architecture that skips output which will be zero. In FP step, the architecture stores bit-vector that indicates whether the output of the derivative of ReLU is zero. And in BP step, the architecture skips all related convolution operations that generate zero-output by the derivative of ReLU layer. Reference [54], [56], [58] also improved the energy-efficiency and performance for BP step with output skipping supported ASIC.

TABLE 3. Summary of zero-skipping HW for DNN training.

	Forward			Backward			Weight Gradient		
	X_i	W	X_{i+1}	δ_{i+1}	W	δ_i	X_i	δ_{i+1}	δW
Native Sparsity	O	-	-	-	-	O*	O	-	-
Lee, G** [72]	-	-	-	-	-	O*	-	-	-
SIGMA** [73]	O	O		-	O	-	O	-	-
Eager pruning**[74]	-	Prun.	-	-	Prun.	-	-	-	Prun.
LNPU [47]	O	-	-	FGMP	-	-	O	-	-
GANPU [54]	O	-	Pred.	-	-	O*	O	-	-
PNPU [56]	O	Prun.	-	-	Prun.	O*	O	-	Prun.
Evolver [58]	O	-	Pred.	-	-	O*	O	-	-
HNPU [62]	O	-	P_{SUM} Skip	-	-	P_{SUM} Skip	O	-	P_{SUM} Skip

* Sparsity by ReLU backprop. of pre-layer
** Architecture Paper
w/ Algorithm co-design

The algorithm and hardware co-design allows to produce more zeros for efficiency, but maintain the training accuracy. Reference [47] replaces most energy-consuming fp16 operations with zero input MAC operations, and compensates with high energy-efficient fp8 operations. Reference [54], [58] realized high efficiency with zero output prediction and speculative skipping. Reference [56] proposed a method of determining the weight to be pruned in the training phase and skipping all related operations. Reference [62] focuses on a partial sum (P_{sum}) of small values which does not significantly affect accumulation results, and proposed a way to skip the accumulation operations for small P_{sum}. Reference [74] proposed an algorithm that enables pruning in an earlier stage of DNN training, and enhanced performance and energy-efficiency with weight skipping hardware architecture.

2) QUANTIZATION

Table 4 shows optimized bit-precisions for DNN training and hardware to accelerate custom bit-precisions. Most studies on bit-precision have focused on covering the wide data distribution within reduced bit-width such as 8bit and 16bit. A floating-point with custom bits assignment is most commonly adopted. Compared to conventional floating-point formats such as half-precision and mini-precision, more bits are assigned to exponent value expending dynamic range. For example, [65], [66] assigned 8, 6bit to exponent while half-precision has 5 of exponent bit. In addition, there are studies to use fixed-point to take advantage of high energy-efficient hardware. Reference [62] tunes integer-fraction bits depending on data statistics to cover a wide dynamic range with minimum data loss.

TABLE 4. Optimized bit-precisions for DNN training and HW to support the bit-precisions.

Data-type	Opt. Level	Format	Custom HW	
Mixed-Precision [75]	Network	@FP,BP fp32 fp16 @WG fp32	[30]	fp16/32MAC
			[78]	NMP
bfloat [65]	Network	Extended exponet	[68]	fp MAC w/ e:8, m:7
DLfloat [66]	Network	Extended exponet	[53]	fp MAC w/ e:6, m:9
Hybrid fp8 [67]	Network	forward backward	[57]	exp. cfg fp MAC
fp8-SEB [59]	Layer	Shared exp. bias	[59]	fp8 MAC
Flexpoint [76]	Layer	Shared exp. fxp	-	fxp MAC
SDFXP [62]	Layer	Fixed Point &Dynamic Frac/Int	[62]	bit cfg fxp MAC
LDQ [77]	Neuron	$Block_{N-1}$ $Block_0$ $\theta_{blockN-1}$ θ_{block0}	[77]	fxp MAC & NMP
FGMP fp8/fp16 [47]	Neuron	100%-p% p%	[47]	fp8/fp16 cfg MAC

There is a limit to reducing bit width by uniformly optimizing bit precision for all DNN training operations. To overcome the limitation, each DNN training data characteristic should be analyzed, and bit-precision should be optimized for each data depending on their characteristic. The bit-precision can be optimized at each training step, layer, or even neuron to minimize data loss. Reference [65], [66], [75] used a different bit-precision for each step of training. The weight-gradient, which has a relatively wide distribution and requires high precision, is calculated with fp32, and in FP and BP, fp16 is used to improve energy-efficiency. So, they store weight in high precision, fp32, in external memory, and quantize weight to fp16 for FP and BP steps. In particular, [65] assigned the same exponent bit-width as fp32, reducing the conversion cost between fp16 and fp32. Reference [59] optimizes bit-precision for FP and BP steps respectively to enable training with 8 bits. Data for FP step with a relatively narrow distribution is represented by 4bit-exponent and 3bit-mantissa, and in BP step, expended exponent bit-width format, 5bit-exponent and 2bit-mantissa, is used for wider dynamic range. References [59], [62], [76] perform quantization based on each layer's statistical analysis to minimize data loss. Reference [59] adjusts exponent bias of fp8 based on overflow and underutilization monitoring. Reference [62], [76] use fxp and [76] shares exponent values within a tensor, and [62] adjusts int and fraction bit-width depends on each layer's overflow monitoring results.

Custom PE design is essential to operate the optimized bit-precisions with high energy efficiency. A configurable PE data-path is needed to support various bit-precisions

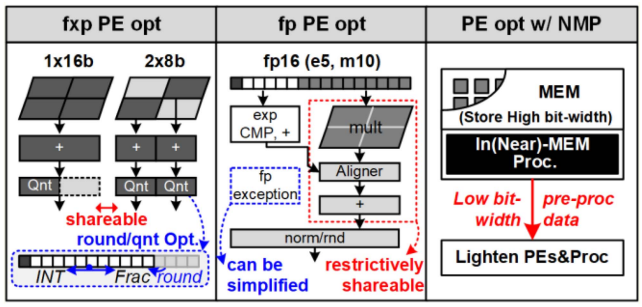


FIGURE 11. PE micro architecture optimizations.

optimized on each DNN training data. Moreover, some functionality of PE can be eliminated or added to achieve high energy efficiency with the help of the algorithm. Fig. 11 shows hardware optimization techniques for high energy-efficient custom bit-precisions. A fxp based PE has a very high area- and energy- efficiency. In addition, because bit configurable MAC can be simply implemented, the various bit-precision for each different DNN training data can be efficiently supported. However, quantization and rounding should be carefully done to compensate for the narrow cover range of fxp. Reference [79] minimizes data loss by implementing local maximum quantization logic. Reference [62] dynamically adjusts the fraction and integer bit-width ratio with overflow monitoring and stochastic thresholding. A fp-based PE can handle a wide range of data but has a lack of reconfigurability and disadvantages of large area and energy consumption. Reference [47] implemented a 2xfp8/1xfp16 configurable FMA. For minimizing the area cost, a mantissa multiplier, an aligner, and an adder are shared for 2xfp8 and 1xfp16. In the case of [57], the adder part is shared for hybrid-fp8 and fp16. Although some sub-blocks are not shared to support multiple bit-precisions, area efficiency can be improved by sharing a mantissa multiplier and an adder, which occupy a large area. In addition, area- and energy-efficiency can be improved by removing fp functions that are not required during DNN training, such as NaN, Inf, and subnormal [53]. References [77], [78] suggested near-memory-processing (NMP) that offloads PE's task and improves computational efficiency by performing simple operations in memory. Reference [77] stores data with high bit-precision in external memory, and the data is fetched to DNN hardware accelerator after quantization in the memory. Since high bit-precision data is stored in the memory, quantization can be adjusted as necessary while maintaining DNN training accuracy. Reference [78] also supports processing in memory, which enables to efficiently accelerate memory-bound operations such as weight-update and weight quantization.

IV. DNN TRAINING ASIC DESIGN EXAMPLE: LNPU

Many optimization techniques for energy-efficient DNN training processing have been proposed to solve the design challenges. However, the dependences of optimization

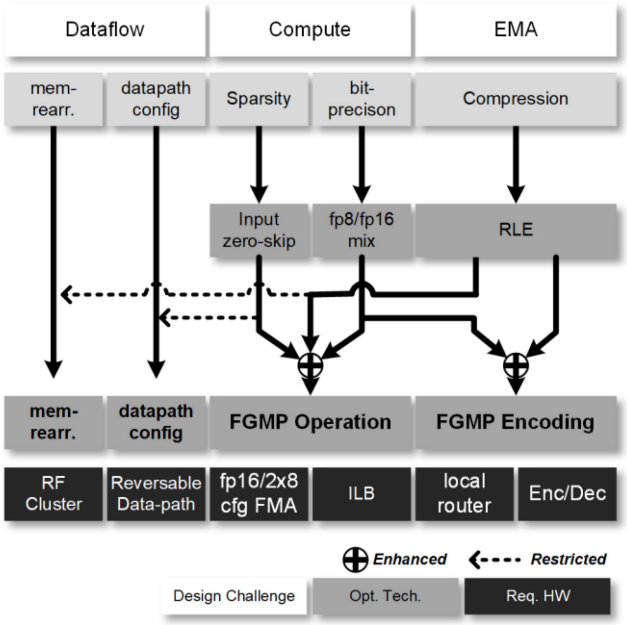


FIGURE 12. Energy-efficient DNN training ASIC design example and dependency between optimization techniques.

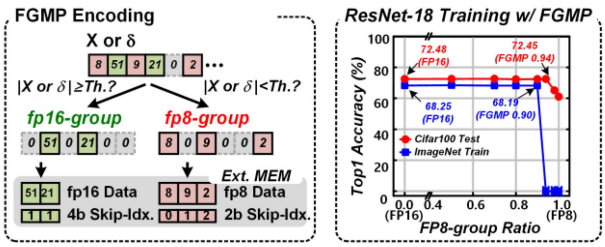


FIGURE 13. Fine-grained mixed precision (FGMP) encoding.

techniques have to be taken into account to design the DNN training ASIC. In this Section, a design example of energy-efficient DNN training ASIC, LNPU [47], [48] will be introduced.

A. DEPENDENCY BETWEEN OPTIMIZATION TECHNIQUES.

Optimization techniques for dataflow, external memory access, and computation affect each other, so dependency between the techniques must be considered to design DNN training ASIC. Implementing multiple techniques can enhance each other's performance, but they can also interfere with each other. Fig. 12 shows the optimization techniques of LNPU and their dependence. For the various dataflow of DNN training, tiled weight array re-arrangement and reversible data-path are supported. For lowering external memory access, a spare compression, RLE, is adopted. Additionally, input zero-skipping and fp8/fp16 are supported for high computational efficiency. Combining the techniques has advantages but also gives some restrictions to each other.

Fig. 13 shows fine-grained-mixed-precision (FGMP) encoding. In DNN training, a required minimum bit-width

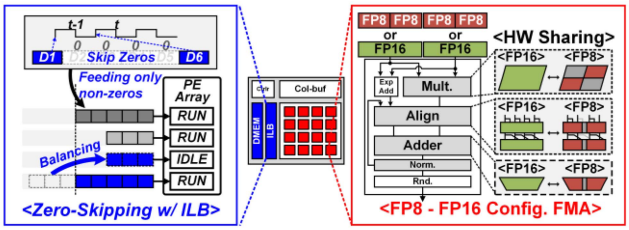


FIGURE 14. Zero-skipping & fp8/fp16 configurable FMA.

for each network and data is different. So, the energy efficiency can be maximized when the bit-width of each data is optimized according to their characteristics. So, it is essential to support bit-precision re-configurable PE for high energy-efficient DNN training ASIC. However, it is tricky to support bit-precision configuration in fp based PE. Therefore, instead of using various bit-precision and designing a PE that supports many data fp formats, LNPU optimizes bit-precision by adjusting the ratio of fp8 and fp16. LNPU divides the input into fp8- and fp16-group, and each group is compressed with RLE. The ratio between fp8- and fp16-group determines energy-efficiency and training accuracy, but only fp8/fp16 re-configurable PE is required. Besides, due to the lots of fp8, external memory access can be more reduced than only using RLE. As a result, FGMP increases the compression ratio by combining reduced bit-width and RLE and helps implement energy-efficient hardware.

Fig. 14 describes LNPU's PE array that supports zero-skipping and 2xfp8/1xfp16 configuration. FGMP encoded data composed of RLE-compressed fp8- and fp16-group is directly fed to the PE array. Because decoding for the encoded data is not required and LNPU skips unnecessary operation, hardware performance can be improved. Moreover, a PE shares a mantissa multiplier, an adder, and an aligner in a 2xfp8/1xf16 configurable FMA to improve area efficiency. And an input load balancer is implemented to balance parallel PE's irregular workload caused by random sparsity. The hardware features further improve computational energy efficiency when used with FGMP encoding. As such, in the design of LNPU, combining optimization techniques for EMA and computation improved each of their efficiency.

However, LNPU's computation and EMA optimization techniques restrict to implementation of dataflow optimization techniques. First, the compression limits array re-arrangement. An encoded data array is difficult to re-arrange due to irregular data addresses. Therefore, it cannot be applied to the FGMP encoded input feature. Instead, as shown in Fig. 15, tiled weight arrays are re-arranged for BP step. In addition, to efficiently support tiled weight re-arrangement, weight buffer is implemented as an RF cluster, enabling writing in horizontal or vertical order. Second, data-path configuration is limited by a control path for input zero-skipping. The data-path for input is fixed to support input zero-skipping for all DNN

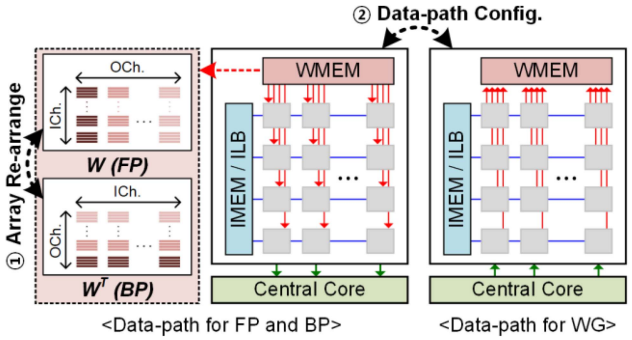
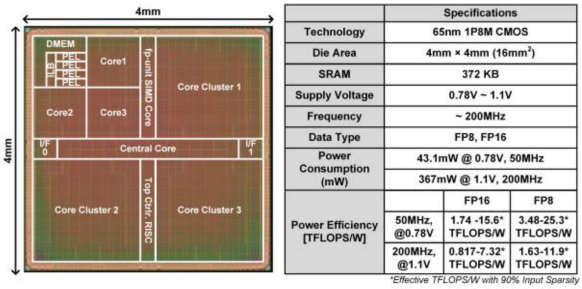


FIGURE 15. Data-path configuration and array re-arrangement for three DNN training steps.



	Specifications		
Technology	65nm 1P8M CMOS		
Die Area	4mm × 4mm (16mm^2)		
SRAM	372 KB		
Supply Voltage	0.78V ~ 1.1V		
Frequency	~ 200MHz		
Data Type	FP8, FP16		
Power Consumption (mW)	43.1mW @ 0.78V, 50MHz		
	367mW @ 1.1V, 200MHz		
Power Efficiency [TFLOPS/W]		FP16	FP8
	50MHz, @0.78V	1.74 -15.6* TFLOPS/W	3.48-25.3* TFLOPS/W
	200MHz, @1.1V	0.817-7.32* TFLOPS/W	1.63-11.9* TFLOPS/W

*Effective TFLOPS/W with 90% Input Sparsity

FIGURE 16. Chip photograph and summary.

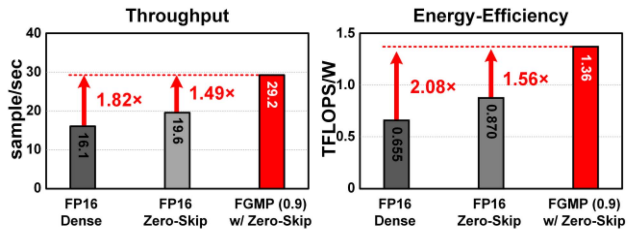


FIGURE 17. Throughput and energy-efficiency for ResNet-18.

training steps, and the data-paths for weight and output can be reconfigured to support WG step in which output and weight are reversed.

B. CHIP IMPLEMENTATION RESULTS

The proposed LNPU [47] is fabricated using 65nm 1P8M CMOS technology, and Fig. 16 shows the LNPU chip micrograph. LNPU occupies 16mm^2 and has 372KB of on-chip SRAM. LNPU can operate at 0.78-to-1.1V supply with a maximum 200MHz clock frequency. The power consumption at 0.78V and 1.1V are 43.1mW and 367mW, respectively. The energy efficiency is 3.48TFLOPS/W (FP8) for 0.0% sparsity and 25.3TFLOPS/W (FP8) for 90% sparsity. Fig. 17 shows the throughput and energy efficiency improvement for ResNet-18 training with LNPU. Thanks to the FGMP and sparse-DL-core, throughput and energy efficiency are improved 1.82 times and 2.08 times, respectively, compared to dense FP16 operation while maintaining training accuracy for ResNet-18.

V. SUMMARY OF DNN TRAINING ASICS

Table 5 shows DNN training ASICs and their optimization techniques for distinct dataflow, external memory access, and computation. Several energy-efficient DNN training ASICs are reported for running various applications such as private local training, personalization, DRL, and visual tracking. Most studies on DNN training ASIC have optimized hardware and proposed hardware-friendly algorithms for high energy efficiency.

The configurable data-path [37], [39], [46], [51]–[53], [57], [59] and array re-arrangement [47], [50], [55], [60] have been implemented to support the various data flows of DNN training with high PE utilization. In the ASIC design with a configurable data-path, it is essential to consider feasibility, such as routing complexity at the back-end process. Therefore, most ASICs that support configurable data paths focus on simple, flexible interconnect (or routing) designs between PEs and implement memory buffers such as LRF to minimize computation stall by data transaction latency [46], [47], [53], [59]. The dataflow issue can be solved with hardware-algorithm co-design. In addition, [52] adopted DFA to run FP, BP, and WG simultaneously. And thanks to the DFA algorithm, three DNN training steps could be pipelined, achieving high PE utilizations with heterogenous cores optimized to each FP, BP, and WG. ASICs that support array re-arrangement load small-sized tile tensors as input to minimize latency for re-arrangement. Besides, [60] implemented re-arrangement functions in memory circuits to remove redundant memory access or latency.

In ASIC design, EMA optimization techniques are closely related to computational optimization techniques. As shown in Table 5, ASICs tend to process data as compressed to avoid encoding and decoding latency delays. Moreover, they also optimized compute and EMA together by accessing and processing quantized and compressed data. However, DNN layers having lots of external memory than operations such as FC layer based DRL [49], should focus more on reducing data size. So, [49] implemented encoding/decoding cores to prevent memory bound.

Sparsity handling has been adopted by many DNN training ASICs because removing zeros does not affect results but improving efficiency. All ASICs that support sparsity handling skip native zeros such as ReLU and Pooling. In addition, recent DNN training ASICs tend to explore sparsity with algorithmic assistance. They predict output zeros or proactively eliminate data that will be less valuable. For example, [54] proposed an output prediction algorithm based on exponent values and implemented dedicated cores for prediction. In addition, [56] proposed weight pruning which is performed during training and skips operations related to the pruned weights.

A bit-precision is essential in determining DNN training accuracy and the ASIC's energy efficiency. Therefore, the algorithm verification and PE micro-architecture design must be done simultaneously. Starting with fp16, recently, fp8- and fxp-based DNN training ASICs have been reported. The

TABLE 5. Summary of DNN training ASICs.

Paper	DNN training data-flow Opt.	EMA Opt.	Compute Opt. (Sparsity)	Compute Opt. (Bit-precision)	Process	Area	Tops/W
SOVC18[46] IBM	2D torus & hierarchical mem[1)]	16/32bit[a)] (train)	-	fp16/32 (train) binary/ternary (inf)	14nm	9mm^2	-
ISSCC19[47] LNPU	Reversible data-path[1)] Tiled W re-arrangement[2)]	RLE[b)] & 8bit/16bit[a)]	in zero-skipping & input load balancer	FGMP of fp8/fp16 2xfp8/1x16 cfg FMA	65nm	16mm^2	0.817 ~25.3*
ISSCC19[49]	Transposable PE array data-path[1)]	exp. encoding[c)] & 4/8/16bit[a)]	-	bfloat (feature) fxp4/8/16 (w)	65nm	16mm^2	2.16
ASSC19[50]	Tiled W re-arrangement[2)]	16bit[a)] (train)	-	fp16(train) fxp5,10(inf) fxp & fp MAC sharing	40nm	5mm^2	2.25
SOVC19[52]	Heterogenous Arch w/ DFP[1)] (pipelined FP, BP, WG core)	13/16bit[a)], 1bit[a)] W at BP	in zero-skipping (for Conv Layer)	fxp13/16 binary W for BP	65nm	5.76mm^2	0.77 ~1.32
ASSC19[51]	Heterogenous Arch[1)] (FP/BP, WG core)	1bit enc for ReLU and Pooling[a)] & 8bit[a)]	-	dynamic fxp8	65nm	10.24mm^2	1.03
SOVC20[53] IBM	flexable 2D interconnect & hierarchical mem[1)]	16/32bit[a)] (train)	-	DLfloat16 FMA w/ Simplified fp exception	14mn	9.8mm^2	1.1 ~ 1.4
SOVC20[56] PNPU	-	ZVC[b)] & 8/16bit[a)]	In/W/Out zero-skipping (w/ W Pruning Algm.)	2xfp8/1xfp16 cfg FMA	65nm	16mm^2	1.57 ~82.04*
ISSCC20[54] GANPU	-	ZVC[b)] & 8/16bit[a)]	In/Out zero-skipping (w/ Output Pred. core)	2xfp8/1xfp16 cfg FMA	65nm	32.4mm^2	0.83 ~135*
ISSCC20[60]	Transpose in Custom SRAM[2)]	2~20bit[a)]	-	in: fxp2/4/8, w:fxp4/8, o:fxp10/12/16/20 & CIM	28nm	64kB CIM	7~61.1** (Analog)
SSCL20[55]	Diagonal storage pattern w/ bit-rotator[2)]	16bit[a)]	In zero clock-gating	fxp16	65nm	16.9mm^2	0.50 ~2.60
ISSCC21[57] IBM	Flexable datapath MUXs[1)]	8/16bit[a)]	In zero clock-gating	Cfg fp FMA for hybrid-fp8 & fp16	7nm	36mm^2	0.98 ~1.8
ISSCC21[59]	Flexable In/Out 2D Routing[1]	8bit + shared exponent bias[a)]	-	SEB-fp8	40nm	6.25mm^2	~4.81
JSSC21[58] Evolver	-	2/4/8bit[a)]	In/Out zero-skipping	fxp2/4/8 QVF	28nm	5.64mm^2	~173**
JSSC21[62] HNPU	-	4/8/12/16bit[a)]	In-/Out-slice zero-skipping	SDFXP	28nm	12.96mm^2	~50.3***

1) data-path configuration technique 2) array re-arrangement technique *a) bit-width reduction b) sparse compression c) occurrence based compression*

algorithm HW co-designed **with Sparsity, **@fxp2, ***@fxp4*

lower the bit precision, the more precisely controlled quantification techniques for each DNN data type must be used to maintain accuracy. So, recent ASICs support various bit-precision to support optimized bit-precision for each DNN data type. Furthermore, recent DNN training ASICs using low bit-precision are assisted by algorithms that analyze statistical characteristics of DNN data and adjust bit-precision in real-time. Reference [51], [59], [62] monitor overflow and maximum value to adjust bit-precision and shared scaling values in run-time.

VI. CONCLUSION

DNN training iteratively processes three distinct steps, causing a lot of external memory accesses and operations. Therefore, high energy-efficient DNN training hardware is necessary to realize DNN training on an edge device which has limited computational capability and power supply. For the high energy-efficient DNN training hardware, three challenges must be optimized: distinct dataflow, external memory access, and computation.

Three DNN training steps, FP, BP, and WG, share a tensor but have different dataflow. In detail, FP and BP steps share a weight tensor, however, BP step uses transposed weights in out- and in-channel dimensions. In addition, WG step produces weight gradients from features and output gradients obtained in FP and BP, respectively, but WG step does not perform channel accumulation, unlike FP and BP. The different dataflow of the three training steps makes designing DNN training hardware very challenging. This is because operating different dataflow with the same data-path architecture and memory layout reduces PE utilization and reduces memory access efficiency. There are two ways to solve this problem: data array re-arrangement and data-path configuration. The array re-arrangement modifies the memory layout before feeding to PEs. The array re-arrangement is simple to implement but requires redundant memory access and additional memory buffer; moreover, it takes long latency for large size of tensor. Many schemes have been proposed to implement re-arrangement more efficiently, such as tiled transpose, diagonal storage pattern with bit-rotator, and custom SRAM. In contrast, the data-path configuration modifies data-path of the PE array to maximize data reuse and PE utilization while still using the same memory layout of a tensor. The data-path configuration uses the same memory

layout for different training steps but modifies the data-path of the PE array to maximize data reuse and PE utilization. This method has been used in DNN training hardware to efficiently process a tensor that is difficult to apply array re-arrangement, such as large size tensor and compressed tensor.

External memory access must be considered for energy-efficient DNN training. Unlike DNN inference, during the training, the intermediate features of all layers are stored in external memory. In addition, min-batch and batch normalization layer also make features a high proportion of external memory access. Therefore, DNN training accelerators have focused on compressing features to reduce external memory access. There are three methods of compression: sparse compression, occurrence-based compression, and bit-width reduction. First, sparse compression eliminates zeros by representing tensor with a non-zero value vector and an index vector. Second, occurrence-based compression reduces the tensor's byte size by encoding the most likely occurrence values to the lower bit-width. Third, reduced bit-width utilizes the loss robust property of DNN to lower bit-width of data while maintaining training accuracy. The compression methods can be used alone or combined depending on the feature's data characteristics.

The iterative operations of DNN training and the wide dynamic range of operands cause a lot of energy consumption, so computation has to be optimized to enable DNN training on edge devices. In contrast to inference, in DNN training, the back-propagated gradients have a wide range of data distributions, so floating-point ALU is commonly used to handle them. However, floating-point ALU has high area consumption and energy consumption, making it challenging to design highly energy-efficient training hardware. There are two ways to implement high-energy efficient processing elements: zero-skipping operation and bit-precision optimization. Unlike inference, training has limitations in utilizing sparsity because there is no input sparsity in BP steps and no zeros in weights. Thus, using zero-skipping accelerator for DNN inference can not significantly improve efficiency. So, many studies have proposed DNN training accelerators that explore output sparsity and detect unnecessary operations to skip. The bit-precision within the limited bit-width determines energy efficiency and accuracy. Bit precision can be optimized for each DNN training step, layer level, and neuron level. In addition, the PE must be configured to support various bit-precisions optimized to each level.

Many optimization techniques have been presented to overcome three design challenges. However, the dependencies between optimization techniques must also be taken into account in order to design energy-efficient DNN training hardware. For example, the encoded tensors for reducing external memory access are difficult to re-arrange which is a dataflow optimization technique. In addition, a compression tensor can be fed to a customized MAC array without decoding to maximize energy efficiency. Section IV introduces an example of energy-efficient DNN training ASIC design that takes into account dependencies between optimization techniques.

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[3] K. Simonyan and A. Zisserman, "Two-stream convolutional networks for action recognition in videos," 2014. [Online]. Available: arXiv:1406.2199.

[4] T. Wang, D. J. Wu, A. Coates, and A. Y. Ng, "End-to-end text recognition with convolutional neural networks," in *Proc. IEEE 21st Int. Conf. Pattern Recognit. (ICPR)*, 2012, pp. 3304–3308.

[5] S. S. Farfade, M. J. Saberian, and L.-J. Li, "Multi-view face detection using deep convolutional neural networks," in *Proc. 5th ACM Int. Conf. Multimedia Retrieval*, 2015, pp. 643–650.

[6] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 3431–3440.

[7] Z. Wang, J. Chen, and S. C. H. Hoi, "Deep learning for image super-resolution: A survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 10, pp. 3365–3387, Oct. 2021.

[8] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila, "Analyzing and improving the image quality of StyleGAN," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 8110–8119.

[9] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," 2015. [Online]. Available: arXiv:1511.06434.

[10] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, pp. 2961–2969.

[11] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," 2018. [Online]. Available: arXiv:1804.02767.

[12] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *J. Mach. Learn. Res.*, vol. 12, pp. 2493–2537, Mar. 2011.

[13] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018. [Online]. Available: arXiv:1810.04805.

[14] N. Smolyanskiy, A. Kamenev, J. Smith, and S. Birchfield, "Toward low-flying autonomous MAV trail navigation using deep neural networks for environmental awareness," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, 2017, pp. 4241–4247.

[15] A. Loquercio, A. I. Maqueda, C. R. Del-Blanco, and D. Scaramuzza, "DroNet: Learning to fly by driving," *IEEE Robot. Autom. Lett.*, vol. 3, no. 2, pp. 1088–1095, Apr. 2018.

[16] E. Çetin, C. Barrado, G. Muñoz, M. Macias, and E. Pastor, "Drone navigation and avoidance of obstacles through deep reinforcement learning," in *Proc. IEEE/AIAA 38th Digit. Avion. Syst. Conf. (DASC)*, 2019, pp. 1–7.

[17] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," 2016. [Online]. Available: arXiv:1610.05492.

[18] L. Li, Y. Fan, M. Tse, and K.-Y. Lin, "A review of applications in federated learning," *Comput. Ind. Eng.*, vol. 149, Nov. 2020, Art. no. 106854.

[19] W. Y. B. Lim *et al.*, "Federated learning in mobile edge networks: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 3, pp. 2031–2063, 3rd Quart., 2020.

[20] X. Peng, Z. Huang, Y. Zhu, and K. Saenko, "Federated adversarial domain adaptation," 2019. [Online]. Available: arXiv:1911.02054.

[21] G. Csurka, "Domain adaptation for visual applications: A comprehensive survey," 2017. [Online]. Available: arXiv:1702.05374.

[22] S. Kornblith, J. Shlens, and Q. V. Le, "Do better ImageNet models transfer better?" in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 2661–2671.

[23] H. Zhu *et al.*, "The ingredients of real-world robotic reinforcement learning," 2020. [Online]. Available: arXiv:2004.12570.

[24] L. Pinto and A. Gupta, "Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2016, pp. 3406–3413.

[25] 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.

[26] 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.

[27] 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, 2020.

[28] 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, 2020.

[29] S. Mittal and S. Vaishay, "A survey of techniques for optimizing deep learning on GPUs," *J. Syst. Architect.*, vol. 99, Oct. 2019, Art. no. 101635.

[30] *NVIDIA A100 TENSOR CORE GPU*. Accessed: Jun. 21, 2021. [Online]. Available: https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datasheet.pdf

[31] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," 2016. [Online]. Available: arXiv:1611.06440.

[32] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized convolutional neural networks for mobile devices," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 4820–4828.

[33] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 448–456.

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

[35] 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*, 2018, pp. 33–34.

[36] A. Parashar *et al.*, "SCNN: An accelerator for compressed-sparse convolutional neural networks," *ACM SIGARCH Comput. Architect. News*, vol. 45, no. 2, pp. 27–40, 2017.

[37] P. Judd, A. Delmas, S. Sharify, and A. Moshovos, "Cnvlutin2: Ineffectual-activation-and-weight-free deep neural network computing," 2017. [Online]. Available: arXiv:1705.00125.

[38] S. Zhang *et al.*, "Cambricon-X: An accelerator for sparse neural networks," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, 2016, pp. 1–12.

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

[40] S. Han *et al.*, "EIE: Efficient inference engine on compressed deep neural network," *ACM SIGARCH Comput. Architect. News*, vol. 44, no. 3, pp. 243–254, 2016.

[41] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H.-J. Yoo, "UNPU: An energy-efficient deep neural network accelerator with fully variable weight bit precision," *IEEE J. Solid-State Circuits*, vol. 54, no. 1, pp. 173–185, Jan. 2019.

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

[43] B. Moons, R. Uytterhoeven, W. Dehaene, and M. Verhelst, "14.5 envision: A 0.26-to-10 TOPS/W subword-parallel dynamic-voltage-accuracy-frequency-scalable convolutional neural network processor in 28nm FDSOI," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2017, pp. 246–247.

[44] 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.

[45] D. Shin, J. Lee, J. Lee, and H.-J. Yoo, "14.2 DNPU: An 8.1 TOPS/W reconfigurable CNN-RNN processor for general-purpose deep neural networks," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2017, pp. 240–241.

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

[47] 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 *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2019, pp. 142–144.

[48] J. Lee, J. Lee, D. Han, J. Lee, G. Park, and H.-J. Yoo, "An energy-efficient sparse deep-neural-network learning accelerator with fine-grained mixed precision of FP8–FP16," *IEEE Solid-State Circuits Lett.*, vol. 2, no. 11, pp. 232–235, Nov. 2019.

[49] C. Kim, S. Kang, D. Shin, S. Choi, Y. Kim, and H.-J. Yoo, "A 2.1 TFLOPS/W mobile deep RL accelerator with transposable PE array and experience compression," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2019, pp. 136–138.

[50] 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)*, 2019, pp. 65–68.

[51] S. Choi, J. Sim, M. Kang, Y. Choi, H. Kim, and L.-S. Kim, "A 47.4 μJ/epoch trainable deep convolutional neural network accelerator for *in-situ* personalization on smart devices," in *Proc. IEEE Asian Solid-State Circuits Conf. (A-SSCC)*, 2019, pp. 57–60.

[52] D. Han, J. Lee, J. Lee, and H.-J. Yoo, "A 1.32 TOPS/W energy efficient deep neural network learning processor with direct feedback alignment based heterogeneous core architecture," in *Proc. IEEE Symp. VLSI Circuits*, 2019, pp. C304–C305.

[53] J. Oh *et al.*, "A 3.0 tFlops 0.62 v scalable processor core for high compute utilization Ai training and inference," in *Proc. IEEE Symp. VLSI Circuits*, 2020, pp. 1–2.

[54] 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)*, 2020, pp. 140–142.

[55] 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, Nov. 2019.

[56] 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*, 2020, pp. 1–2.

[57] A. Agrawal *et al.*, "A 7nm 4-core AI chip with 25.6 TFLOPS hybrid FP8 training, 102.4 TOPS INT4 inference and workload-aware throttling," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, vol. 64, 2021, pp. 144–146.

[58] 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.

[59] J. Park, S. Lee, and D. Jeon, "A 40nm 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)*, vol. 64, 2021, pp. 1–3.

[60] J.-W. Su *et al.*, "15.2 a 28nm 64kb 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.

[61] J.-S. Seo *et al.*, "A 45nm CMOS neuromorphic chip with a scalable architecture for learning in networks of spiking neurons," in *Proc. IEEE Custom Integr. Circuits Conf. (CICC)*, 2011, pp. 1–4.

[62] 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.

[63] R. D. Evans, L. Liu, and T. M. Aamodt, "JPEG-ACT: Accelerating deep learning via transform-based lossy compression," in *Proc. ACM/IEEE 47th Annu. Int. Symp. Comput. Architect. (ISCA)*, 2020, pp. 860–873.

[64] A. Jain, A. Phanishayee, J. Mars, L. Tang, and G. Pekhimenko, "GIST: Efficient data encoding for deep neural network training," in *Proc. ACM/IEEE 45th Annu. Int. Symp. Comput. Architect. (ISCA)*, 2018, pp. 776–789.

[65] *BFloat16: The Secret to High Performance on Cloud TPUs*. Accessed: Jun. 21, 2021. [Online]. Available: https://cloud.google.com/blog/products/ai-machine-learning/bfloat16-the-secret-to-high-performance-on-cloud-tpus
[66] A. Agrawal *et al.*, "DLFloat: A 16-b floating point format designed for deep learning training and inference," in *Proc. IEEE 26th Symp. Comput. Arithmetic (ARITH)*, 2019, pp. 92–95.
[67] X. Sun *et al.*, "Hybrid 8-bit floating point (HFP8) training and inference for deep neural networks," in *Proc. 32nd Adv. Neural Inf. Process. Syst.*, 2019, pp. 4900–4909.
[68] T. Norrie *et al.*, "Google's training chips revealed: TPUV2 and TPUV3," in *Proc. Hot Chips Symp.*, 2020, pp. 1–70.
[69] J. S. Park *et al.*, "9.5 A 6K-MAC feature-map-sparsity-aware neural processing unit in 5nm flagship mobile SoC," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, vol. 64, 2021, pp. 152–154.
[70] K. Kanellopoulos *et al.*, "SMASH: Co-designing software compression and hardware-accelerated indexing for efficient sparse matrix operations," in *Proc. 52nd Annu. IEEE/ACM Int. Symp. Microarchit.*, 2019, pp. 600–614.
[71] Y. Park, Y. Kang, S. Kim, E. Kwon, and S. Kang, "GRLC: Grid-based RUN-length compression for energy-efficient CNN accelerator," in *Proc. ACM/IEEE Int. Symp. Low Power Electron. Design*, 2020, pp. 91–96.
[72] G. Lee, H. Park, N. Kim, J. Yu, S. Jo, and K. Choi, "Acceleration of DNN backward propagation by selective computation of gradients," in *Proc. 56th Annu. Design Autom. Conf.*, 2019, pp. 1–6.
[73] 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. Architect. (HPCA)*, 2020, pp. 58–70.
[74] 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. Architect. (ISCA)*, 2019, pp. 292–303.
[75] P. Micikevicius *et al.*, "Mixed precision training," 2017. [Online]. Available: arXiv:1710.03740.
[76] U. Köster *et al.*, "FlexPoint: An adaptive numerical format for efficient training of deep neural networks," 2017. [Online]. Available: arXiv:1711.02213.
[77] Y. Zhao *et al.*, "Cambricon-Q: A hybrid architecture for efficient training," in *Proc. ACM/IEEE 48th Annu. Int. Symp. Comput. Architect. (ISCA)*, 2021, pp. 706–719.
[78] H. Kim *et al.*, "GradPIM: A practical processing-in-DRAM architecture for gradient descent," in *Proc. IEEE Int. Symp. High Perform. Comput. Architect. (HPCA)*, 2021, pp. 249–262.
[79] S. Choi, J. Shin, Y. Choi, and L. S. Kim, "An optimized design technique of low-bit neural network training for personalization on IoT devices," in *Proc. 56th Annu. Design Autom. Conf.*, 2019, pp. 1–6.

JINSU LEE (Member, IEEE) received the B.S. degree from the School of Electrical and Electronic Engineering, Yonsei University, Seoul, South Korea, in 2015, and the M.S. and Ph.D. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 2017 and 2020, respectively.

He currently holds a Postdoctoral position with KAIST, where he is involved in the NPU hardware architecture and system-on-chip hardware architecture for AI applications. His current research interests include energy-efficient deep-neural-network inference/training processor design, intelligent vision system-on-chip, and embedded system development. He received the IEEE International Solid-State Circuits Conference 2019 Demonstration Session Certificate of Recognition.

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

Since 1998, he has been on the faculty of the Department of Electrical Engineering, KAIST, where he is currently a Full Professor. From 2001 to 2005, he was the Director of the Korean System Integration and IP Authoring Research Center. From 2003 to 2005, he was the full-time Advisor to Minister of the Korea Ministry of Information and Communication and the National Project Manager for system-on-chip (SoC) and computers. In 2007, he founded the System Design Innovation and Application Research Center, KAIST. Since 2010, he has been the General Chair of the Korean Institute of Next Generation Computing. He has coauthored *DRAM Design* (South Korea: Hongrung, 1996), *High Performance DRAM* (South Korea: Sigma, 1999), *Future Memory: FRAM* (South Korea: Sigma, 2000), *Networks on Chips* (Morgan Kaufmann, 2006), *Low-Power NoC for High-Performance SoC Design* (CRC Press, 2008), *Circuits at the Nanoscale* (CRC Press, 2009), *Embedded Memories for Nano-Scale VLSIs* (Springer, 2009), *Mobile 3-D Graphics SoC: From Algorithm to Chip* (Wiley, 2010), *Bio-Medical CMOS ICs* (Springer, 2011), *Embedded Systems* (Wiley, 2012), and *Ultra-Low-Power Short-Range Radios* (Springer, 2015). His current research interests include computer vision SoC, body area networks, and biomedical devices and circuits.