Custom Hardware Architectures for Deep Learning on Portable Devices: A Review

Kh Shahriya Zaman, *Graduate Student Member, IEEE*, Mamun Bin Ibne Reaz, *Senior Member, IEEE*, Sawal Hamid Md Ali, *Senior Member, IEEE*, Ahmad Ashrif A Bakar, *Senior Member, IEEE*, and Muhammad Enamul Hoque Chowdhury, *Senior Member, IEEE*

***Abstract*— The staggering innovations and emergence of numerous deep learning (DL) applications have forced researchers to reconsider hardware architecture to accommodate fast and efficient application-specific computations. Applications, such as object detection, image recognition, speech translation, as well as music synthesis and image generation, can be performed with high accuracy at the expense of substantial computational resources using DL. Furthermore, the desire to adopt Industry 4.0 and smart technologies within the Internet of Things infrastructure has initiated several studies to enable on-chip DL capabilities for resource-constrained devices. Specialized DL processors reduce dependence on cloud servers, improve privacy, lessen latency, and mitigate bandwidth congestion. As we reach the limits of shrinking transistors, researchers are exploring various application-specific hardware architectures to meet the performance and efficiency requirements for DL tasks. Over the past few years, several software optimizations and hardware innovations have been proposed to efficiently perform these computations. In this article, we review several DL accelerators, as well as technologies with emerging devices, to highlight their architectural features in application-specific integrated circuit (IC) and field-programmable gate array (FPGA) platforms. Finally, the design considerations for DL hardware in portable applications have been discussed, along with some deductions about the future trends and potential research directions to innovate DL accelerator architectures further. By compiling this review, we expect to help aspiring researchers widen their knowledge in custom hardware architectures for DL.**

***Index Terms*— Application-specific integrated circuit (ASIC), deep learning (DL), deep neural network (DNN), energy-efficient architectures, field-programmable gate array (FPGA), hardware accelerator, machine learning (ML), neural network hardware, review.**

I. Introduction

DEEP learning (DL) has enabled sophisticated artificial intelligence (AI) applications, such as self-driving cars, natural language processing, virtual assistance, image recognition, image generation, music synthesis, financial forecasting, robot navigation, object detection, and legal practice [1]–[4]. DL has also made vital contributions to drug discovery, gene sequencing and splicing, healthcare monitoring, cancer diagnosis, and medical imaging [5]–[8]. Furthermore, DL has been shown to perform better than humans in image recognition, strategy games, and medical diagnosis [9]–[11].

The advancements in DL applications are fueled by the massive amounts of sensor data and the Internet of Things (IoT) devices, and the availability of extensive computational resources [12]. The data collected from the IoT sensors are sent to the cloud servers or external CPU for analysis. In most cases, these server PCs are usually equipped with high-performance general-purpose graphics processing units (GPUs) to accelerate deep neural network (NN) (DNN) computation. However, there is an increasing demand to introduce on-chip intelligence into the smart city and Industry 4.0 infrastructure to improve privacy, latency, and communication overheads [13], [14]. Most DNN models contain multiple neuron layers with complex structures that require extensive computational resources, thereby limiting AI applications' portability on low-power mobile devices. This limitation is intensified further by Moore's law and Dennard scaling [15], [16]. Currently, Samsung Electronics and Taiwan Semiconductor Manufacturing Company (TSMC) are developing the 3 nm fabrication process [17]. Scaling even further may pose fabrication challenges because of quantum effects [18]. Researchers are exploring new computation paradigms, such as quantum [19] and optical computing [20], to overcome transistor shrinking limitations. However, such technologies are still in their early phases. Therefore, substantial progress in hardware architecture is necessary to address these issues.

In recent years, researchers have developed specialized software and hardware architectures to increase instruction parallelization, optimize memory bandwidth, coordinate dataflow, and customize mathematical operations in NNs for resource-constrained devices. Researchers have evaluated their systems on various performance metrics, which include throughput, accuracy, scalability, and functionality [21], [22]. Several portable DNN accelerators are also available commercially in the form of development boards and system-on-chip (SoC) because of DL's popularity in research and applications. Even CPU vendors started including a dedicated neural processing unit or special vector instructions as part of their SoC.

Manuscript received June 25, 2020; revised October 15, 2020 and February 1, 2021; accepted May 17, 2021. This work was supported in part by the Research University Grant, Universiti Kebangsaan Malaysia, under Grant DPK-2021-001, Grant DIP-2020-004, and Grant MI-2020-002; and in part by the Qatar National Research Foundation (QNRF) under Grant NPRP12s-0227-190164. *(Corresponding authors: Muhammad Enamul Hoque Chowdhury; Mamun Bin Ibne Reaz.)*

Kh Shahriya Zaman, Mamun Bin Ibne Reaz, Sawal Hamid Md Ali, and Ahmad Ashrif A Bakar are with the Department of Electrical, Electronic and Systems Engineering, Universiti Kebangsaan Malaysia, Bangi 43600, Malaysia (e-mail: kh.shahriya.zaman@gmail.com; mamun@ukm.edu.my; sawal@ukm.edu.my; ashrif@ukm.edu.my).

Muhammad Enamul Hoque Chowdhury is with the Department of Electrical Engineering, Qatar University, Doha 27113, Qatar (e-mail: mchowdhury@qu.edu.qa).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TNNLS.2021.3082304.

Digital Object Identifier 10.1109/TNNLS.2021.3082304

The motivation behind this article is to enlighten DNN researchers with the fact that DNN hardware architectures are not limited to data routing systems, specialized processing elements (PEs), and efficient on-chip memory structures. In other words, these architectures can be extended to new computing paradigms, emerging devices, and storage-computing integration. We hope to inspire prospective researchers to explore unconventional computing and optimization techniques for DL hardware. The contributions of this article can be summarized as follows:

1) Compared to existing review articles [23], [24] that only focus on particular domains, such as the reconfigurability of field-programmable gate arrays (FPGAs), acceleration from approximate computing, and design space exploration, our review article explores hardware architecture based on neuromorphic computing, stochastic computing, and emerging devices, in addition to the aforementioned optimizations.
2) We highlight the specialty and limitations of several recent hardware architectures for DL acceleration in terms of dataflow, hierarchical memory design, array design, custom arithmetic units, and other unique functionalities. We also summarized the results in Table I, for a quick overview of hardware development progression for DL acceleration.
3) We point out the challenges in hardware architecture design for research and commercial purpose in DL applications. We also propose research areas to solve some of these issues.

The remainder of this article is organized into four sections. Section II provides a brief overview of DL, its components and models, and explains the software optimizations for efficient DL computation. Section III presents a review of several custom hardware architectures for DL applications, along with commercially available hardware solutions. Section IV summarizes our review and discusses some possible directions for DL accelerators. Section V presents some concluding remarks.

II. DL Overview

This section provides a brief overview of the components of DL, basic DNN architectures, and optimizations for DL computation. Additionally, we highlighted the areas where DL computation efficiency can be improved.

A. Basic Components of DL

There are several distinct architectures for DL, but all of them use similar mathematical algorithms to extract patterns from the input dataset. Each architecture has its benefits in a specific type of dataset. The simplest DL architecture is the multilayer perceptron (MLP). convolutional (CONV) NN (CNN) and recurrent NN (RNN) are two widely used DL architectures. CNN specializes in spatial data, such as image recognition, object detection, drug discovery, and video analysis. Conversely, RNN is advantageous with temporal data used in time-series prediction, anomaly detection, speech recognition, music composition, and other time-dependent applications. Long-short-term memory (LSTM) recurrent network has been introduced to address the problem with vanishing gradient [25]. However, the sequential nature of RNN and LSTM makes the parallelization of such networks challenging. Thus, the transformer architecture was proposed to allow parallelization and reduce training time for machine translation applications [26]. Lately, generative adversarial networks [27] and graph NNs [28] have been receiving considerable attention. However, further investigation on their application in edge devices is required.

For this article, we will mainly focus on CNN and the essential mathematical procedures involved. The reason is that CNNs contain computationally demanding convolution layers that can be used in the performance validation of hardware optimizations. Nevertheless, most hardware optimizations can also adapt to other DNN architecture because most of the mathematical operations present in different architectures can be generalized as matrix multiplication and vector operations.

A CNN model is mainly composed of different configurations of CONV, pooling, normalization, activation, and fully connected (FC) layers. The CONV layer is the basic building block of a CNN model. A convolution operation can be considered the matrix (output feature map) production by sliding the respective filter across each layer (channel) of the input matrix (input feature map). Thus, the same weights in the filter are reused on the entire input. This approach reduces the parameters of the layer significantly. Mathematically, convolution operation can be represented by several loops of matrix multiplications. The convolution operations often constitute 90% of the overall CNN operations [29]. CONV layers are the most intensive in terms of computation and memory. Therefore, accelerating the convolution operation can increase the overall performance of the DNN model effectively.

Pooling layers are usually inserted between convolution layers to reduce the spatial size of the output feature map from the previous layer. The normalization layer controls the input distribution and data bias for the next layer, thereby accelerating the training phase and reducing memory access and data caching. FC layers are regular NNs wherein each neuron is connected to all activation of the previous layers. As a result, the number of parameters in a single layer is larger than that of a CONV layer for the same dimensions, thereby significantly increasing the memory and computational requirements. However, on modern DL models, FC layers are usually applied only once for classification purposes. FC layers are placed after the convolution and pooling layers with few remaining parameters. Thus, this layer's outputs are typically connected to the activation layer, followed by the output layer's neurons. The rectified linear unit (ReLU) is a widely used activation function because of its minimal hardware requirement and nonsaturating property [30]. Leaky, parametric, and exponential ReLUs are some of the variants of ReLU [31]. The sigmoid and hyperbolic tangent functions are two types of conventional activation functions [32].

B. DL Models

Numerous DNN models have been proposed to date. Each model has different layer numbers, types, and shapes. The

TABLE I
SUMMARY OF CUSTOM HARDWARE ARCHITECTURES FOR DL

Architecture	Specialty	Platform	Training support	Computing paradigm	Dataflow	Data type	Scalability support	Validation DL model	Performance
DianNao [97] 2014	Exploited locality of data for efficient memory usage	3.02mm^2 ASIC on 65nm node (single NFU structure)	No	Conventional	OS	16-bit FXP	No	10 layer CNN	452 GOP/s at 485 mW
DaDianNao [98] 2014	Fat tree communication interface for improved parallelization	67.73mm^2 ASIC on 28nm node (16 tile structure)	Yes	Conventional	OS	16-bit FXP	Yes (by interfacing with same chips)	CNN	5.58 TOPS at 15.97 W
PuDianNao [100] 2015	Hierarchical memory for efficient reuse of data	3.51mm^2 ASIC on 65nm node (16 functional unit structure)	Yes	Conventional	OS	16-bit FLP	No	LeNet, several ML algorithms	1056 GOPS at 596 mW
ShiDianNao [99] 2015	On-chip computation without DRAM	4.86mm^2 ASIC on 65nm node (64 PE structure)	No	Conventional	OS	16-bit FXP	No	LeNet, CNP, ConvNN, etc.	194 GOPS at 320 mW
TrueNorth [128] 2015	Mixed Asynchronous and synchronous circuits for neuromorphic computing	4.3cm^2 ASIC on 28nm (4096 neurosynaptic tile structure)	No	Neuromorphic	WS	9-bit FXP for weights, 1-bit for synapse	Yes (by varying the no. of neurosynaptic tiles/cores)	Several ML algorithms	46 GOPS/W at 65 mW
EIE [75] 2016	DNN acceleration with Deep compression	40.8mm^2 ASIC on 45nm node (64 PE structure)	No	Conventional	OS	16-bit FXP	Yes (by varying no. of PEs)	AlexNet, VGG-16, NeuralTalk	102 GOPS at 590 mW
Cambricon-X [103] 2016	Leverages sparse network for DNN acceleration	6.38mm^2 ASIC on 65nm node (16 PE structure)	No	Conventional	NLR	16-bit FXP	Yes (by varying no. of PEs)	LeNet, AlexNet, VGG-16, etc	544 GOPS at 954 mW
ISAAC [117] 2016	In-memory computation with memristive crossbar arrays	85.4mm^2 ASIC on 32nm node (168 tile structure)	No	Conventional	WS	16-bit FXP (mixed signal)	Yes (by varying no. of tiles)	VGG, MSRA, DeepFace	40.8 TOPS at 65.8 W
PRIME [119] 2016	In-memory processing with ReRAM-based memory	Simulation on 65nm TSMC node (64 256×256 ReRAM cells per bank)	No	Conventional	WS	3-bit computation and 1-bit memory	Yes (by varying no. of ReRAM banks)	CNN, MLP	unspecified
Eyeriss [29] 2017	Implementation of row stationary dataflow	16mm^2 ASIC on 65nm node (168 PE structure)	No	Conventional	RS	16-bit FXP	No	AlexNet,	23.1 GMACS at 278 mW
								VGG-16	10.7 GMACS at 236 mW
Tetris [104] 2017	Integration of 3D hybrid memory cube	3.5mm^2 ASIC on 45nm node (14×14 PE, 3D structure)	No	Conventional	RS	16-bit FXP	Yes (by varying no. of PEs)	AlexNet, VGG-16, ResNet, etc.	39.2 GOPS at 8.42 W peak
Loihi [130] 2018	Neuromorphic processor with on-chip learning capabilities	60mm^2 ASIC on 14nm node (structure of 128 neuromorphic cores and 3 x86 cores)	Yes	Neuromorphic	WS	Up to 9-bit FXP for weights, 1-bit for spikes	Yes (by interfacing with same chips)	SNN	23.6 pJ/SOP
DYNAPs [135] 2018	Mixed-mode hierarchical routing	43.8mm^2 ASIC on 180nm node (quad core processor)	No	Neuromorphic	NLR	12-bit CAM (mixed signal)	Yes (by varying the no. of neural cores)	SNN, CNN	≈30pJ/SOP

TABLE I
(Continued.) SUMMARY OF CUSTOM HARDWARE ARCHITECTURES FOR DL

Architecture	Specialty	Platform	Training support	Computing paradigm	Dataflow	Data type	Scalability support	Validation DL model	Performance
Learning Memristive Architecture [118] 2018	Implementation of various DNN models on neuromorphic architecture	0.038mm^2 simulated ASIC on 180nm node (scalable crossbar array structure)	Yes (through analog BP)	Neuromorphic (with emerging devices)	WS	Analog weights	Yes (by varying crossbar size to adapt an NN model)	Hardware model on MNIST, face recognition	115.65 mW for 3 layer NN implementation
STT-MRAM based Accelerator for BNN [125] 2018	In-memory logic and 4-bit full-addition operation with STT-MRAM	Simulation on 45nm node (scalable STT-MRAM architecture)	No	Conventional (with emerging devices)	WS	Binary weights	Yes (by varying MRAM array size to adapt an NN model)	XNOR-Net (FC layer)	0.003 nJ per operation
								XNOR-Net (Conv layer)	0.059 nJ per operation
Eyeriss v2 [105] 2019	Efficient data reuse with hierarchical mesh	$\approx$32mm^2 ASIC on 65nm node (192 PE structure)	No	Conventional	RS	8-bit FXP	Yes (by varying no. of PEs)	AlexNet	253.2 GOP/W (dense) 962.9 GOP/W (sparse)
								MobileNet	193.7 GOP/W (dense) 251.7 GOP/W (sparse)
HEIF [127] 2019	Stochastic computing for DNN with custom ReLU hardware	ASIC of 22.9mm^2 and 24.7mm^2 for LeNet and AlexNet respectively, on 45nm node	No	Stochastic	WS	128-bit bitstream	Yes (by varying no. of feature extraction blocks)	LeNet	3.2M images per second at 2.6W
								AlexNet	2.5M images per second at 1.9W
ODIN [132] 2019	SDSP-based learning for digital neurons	0.086mm^2 ASIC in 28nm node (emulated crossbar architecture for digital neurons)	Yes (through SDSP)	Neuromorphic	WS	4-bit synapse (mixed signal)	No	SNN	12.7 pJ/SOp at 27.3 μW
Neuromorphic LIF Multi Convolution Processor [133] 2019	Event-based convolution engine	Zynq 7100 FPGA (247k LUT, 710 BRAM)	No	Neuromorphic	RS	8-bit FXP for weight	Yes (by varying the no. of convolution modules)	SCNN	348 MOPS for 64 convolution modules, 0.92 mW per convolution modules
MorphIC [134] 2019	Stochastic SDSP learning for binary weights	2.86mm^2 ASIC in 65nm node(4 core processor)	Yes (through S-SDSP)	Neuromorphic	WS	Binary weight, 1-bit synapse	Yes (by varying the no. of cores)	SNN	51 pJ/SOP at 5.45 mW
Fast Algorithm on FPGA [116] 2020	Implementation of efficient Winograd and FFT transformation	XilinxZCU102 FPGA platform (2520 DSP blocks, 600 and 600k logic cells used for AlexNet and VGG respectively)	No	Conventional	IS	16-bit FXP	Yes (by varying no. of specialized functional blocks)	AlexNet	854.6 GOPS at 23.6 W
								VGG-16	2940.7 GOPS at 23.6 W
MRIMA [126] 2020	Matrix operations with data organization and MixColumn transformations	Simulation on 45nm node(scalable STT-MRAM architecture)	No	Conventional (with emerging devices)	WS	4-bit data	Yes (by varying MRAM array size to adapt an NN model)	CNN with binary weights	1521.8 GOPS per mm^2
Tianjic [136] 2020	Unified architecture for SNN and DNN	14.4mm^2 ASIC on 28nm node	No	Conventional and neuromorphic	WS	8-bit FXP for weights, 1-bit for synapse	Yes (by interfacing with same chips)	AlexNet, SNN, LSTM, etc.	1278 GOPS/W and 649 GSOPS/W at 0.95 W

OS = output stationary, IS = input stationary, RS = row stationary, WS = weight stationary

CNN models may also contain additional specialized components, such as the inception layer of GoogLeNet [33] and shortcut module from ResNet [34]. AlexNet [35] is a popular CNN model for image recognition. However, its efficiency in terms of computational resources and model accuracy is low. SqueezeNet [36] is designed to be lightweight and is targeted for low-power mobile applications. ResNeSt [37] is a state-of-the-art DNN model that can obtain a top-1 accuracy of 81.1% on the ImageNet dataset. DNN models are usually designed for a specific application. For examples, YOLOv4 [38] and high resolution network with object contextual representation (HRNet-OCR) [39] are specialized in object detection and image segmentation, respectively. In some applications, results from several heterogeneous DNN models (i.e., ensemble network) are used to produce more accurate predictions [40]. A DL network can even be a combination of different architectures to exploit architecture-specific features. For example, the spatial characteristics of CNN and the temporal aspect of RNN can be combined in video processing applications [41], stock prediction [42], and drug analysis [43].

Generally, the use of a large number of parameters in a DL model improves accuracy. The specialized components of the models increase the computational efficiency in general-purpose systems. However, such complex DNN models pose challenges for specialized architectures in terms of external memory access, PE utilization, data flow, and instruction parallelization. MobileNet [44] and SqueezeNet [36] are compact DNN models that are optimized for portable and low-energy applications.

Designing DL models manually according to required specifications, such as the number of layers, types of layer, number of neurons, and types of activation, is possible. However, manually handcrafting a NN architecture can be tedious because these networks usually exhibit a black-box model [45]. To address this issue, researchers have introduced neural architecture search (NAS) [46]. NAS uses reinforcement learning to search for optimum NN hyperparameters and generate an efficient architecture by evaluating the computation cost, accuracy, and other constraints set by the designer. Various NAS architectures have been explored to optimize search space, search method, and performance evaluation [47], [48]. MobileNetV3 [44], and EfficientNets [49] are examples of such NAS-generated DL models. Both models offer efficient computation and can be scaled with accuracy and parameter size tradeoff depending on the available resources.

C. Training and Inference

In most cases, the DNN model structure remains unchanged for training and inference purposes. In the training phase, the network learns the values of weights and biases from the input data. The weights and biases are usually initialized randomly. These parameters are updated iteratively by using an optimization method called gradient descent. Adaptive moment estimation (ADAM) is usually used to direct gradient descent toward a fast and stable convergence. ADAM computes an adaptive learning rate from the decaying average of previous gradients for each parameter. Other well-known techniques with adaptive learning rates include root mean square propagation (RMSprop), adaptive gradient (AdaGrad), and AdaMax [50]. The partial derivatives can be calculated efficiently using the backpropagation technique, which uses the chain rule of calculus [25]. However, backpropagation requires each neuron's output to be stored for backward computation, thereby increasing the memory requirements. Moreover, the gradient calculations for updating weights require higher precision than the inference stage.

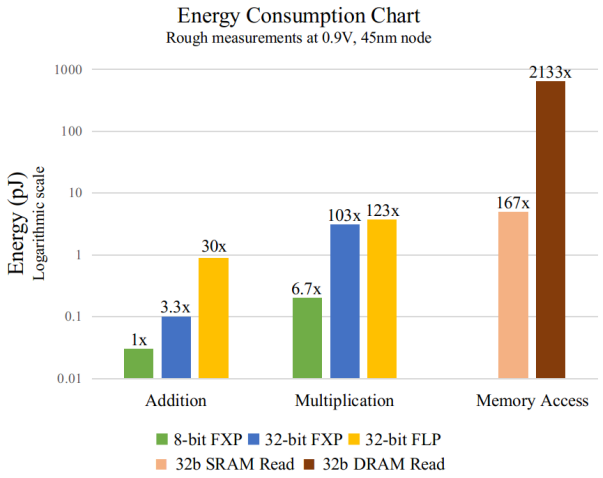

Fig. 1. Comparison of energy consumption for some FXP, FLP, and memory access instructions at 45 nm node. Adapted from [51].

For energy-conservative applications, the DNN models are usually trained offline on high-performance systems. The pretrained models are then optimized for low-power devices, wherein the inference takes place using a feedforward process. The memory footprint and computational requirements are reduced to meet the specifications of the inference hardware. Therefore, many optimization techniques only concentrate on inference because the performance of the feedforward process is crucial.

D. Optimizations for DL Algorithms

Deep and complex NNs often deliver high prediction accuracy in various applications at the cost of extensive computation. A large number of parameters for these models also require a substantial amount of memory bandwidth. Moreover, frequent access to external dynamic random access memory (DRAM) causes delays and requires a considerable amount of energy. Fig. 1 shows the energy consumed by various operations.

After selecting an appropriate model for a specific DL application, several optimization techniques can be applied to improve the model's efficiency. However, some optimizations are usually obtained at the expense of accuracy and increased complexity of the model. The optimization techniques used in DL are discussed in the following paragraphs.

1) Pruning: A typical NN contains several neuron connections, a portion of which have minor contributions to neurons' activation in the next layer [52], [53]. Furthermore, some of these weights contribute to model overfitting. Therefore, these connections can be removed or pruned without affecting

the accuracy of the overall network significantly. This step is performed by setting individual weights to zero (i.e., fine-grain pruning) or removing an entire vector, layer, channel, filter, or structure (i.e., coarse-grain pruning).

Pruning a DNN has several benefits in terms of computational efficiency. Pruning reduces the number of arithmetic instructions and the memory bandwidth requirement. However, fine-grain pruning creates a sparse matrix with unstructured data, which may not be implemented on hardware efficiently [54]–[56]. Most pruning techniques are performed after training the DNN. The pruned pretrained network is used for efficient inference on low-power hardware.

2) Quantization: Quantization is an important optimization for NNs because it not only minimizes the memory requirement for storing and transferring parameters but also reduces the computation and energy necessary for the multiplication and other mathematical instructions. In conventional CPUs, weights are usually calculated using high-precision 32-bit floating points. Higher precision yields better accuracy but also consumes more energy. Quantization is usually accomplished by lowering precision and reducing the number of storage bits. Several researchers concluded that DNNs with quantized weights could efficiently carry out predictions with acceptable accuracy loss [24]. Moreover, some quantization techniques can even replace regular multiplication operations with simple addition and shift operations. Such techniques are crucial for low-power and smaller-sized hardware implementation. However, most DNN quantization research only focuses on inference. The reason is that the gradient's numerical value requires high precision during the training phase [57]. Despite this shortcoming, some researchers also explored the quantization of activation, gradient, error, and batch normalization (BN) parameters for training [58]–[61]. The parameters can be quantized uniformly or according to a preset or learned nonuniform distribution to accommodate the allocated bit-width. For efficient hardware implementation, some studies have focused on different representations of numbers rather than the standard binary numerical representation [62], [63].

Low bit-width arithmetic can reduce the power consumption of NN accelerators substantially. Therefore, the overall efficiency of hardware, which supports low bit-width multipliers, can be increased by merely quantizing a DNN. Furthermore, eliminating multipliers altogether in hardware for binary and ternary weights can save a lot of logic circuit space and improve the parallelization of logical operations [59], [64]. However, in the case of complex quantization techniques, such as clustering and encoding weights with codeblocks, dedicated hardware is necessary to accelerate the process. Moreover, unconventional number representations require custom circuits for processing arithmetic instructions.

3) Weight Decomposition: The weight matrices (or tensors) in an uncompressed DNN usually contain many redundant parameters. Prior research suggests that some of these parameters do not contribute to the prediction of the others [52], [53], whereas others are correlated with one other [65]–[67]. Therefore, the overall size of the weight matrix (or tensor) can be decomposed, i.e., the weight matrix can be approximated into a smaller and simpler structure. Unlike pruning, retraining is often unnecessary in weight decomposition techniques for regaining lost accuracy. Singular value decomposition (SVD) is a popular low-rank matrix decomposition algorithm among DNN researchers [68]. This algorithm reduces the spatial complexity of the weight matrix, thereby enabling efficient computation on hardware. Truncated SVD further reduces storage space by truncating less significant values and approximating the decomposed matrix [69]. To address higher dimensional tensors, decomposition methods, such as block term [70], tucker [71], and canonical polyadic decomposition [72], are used.

Although weight decomposition reduces memory storage and bandwidth requirements in terms of hardware efficiency, most of the decomposition algorithms are designed for pretrained DNNs. As a result, these algorithms are only intended for inference. Furthermore, computations of the decomposed matrix during training are often complex and expensive. Hence such techniques are not suitable for online learning on portable devices.

4) Knowledge Distillation: Deep models with complex components and large parameter space obtain better accuracy at a given task than a shallow model. However, such networks also require massive computational resources. Thus, they are not suitable for low-power systems. By contrast, simpler shallow networks exhibit poor performance in generalizing a function from a multidimensional dataset. Knowledge distillation addresses this issue by transferring extracted features from the complex DNN (teacher) to the simpler model (student). Synthetic labels for input data created by the teacher network (or an ensemble of DNNs [73]) are used to train the student network. This also solves the overfitting problem as the teacher network has already learned a reasonable approximation from the dataset [74]. If the teacher model is noticeably larger than the student model, generally, it is not easy to transfer sufficient knowledge.

Knowledge distillation enables the deployment of different sized models on edge devices from a single high-performance DNN. The student networks can be tailored to meet the target hardware resources, resulting in efficient and reasonably accurate NN accelerator models.

5) Collective Compression: Almost all the mentioned approaches above contribute to some form of parameter compression for a DNN. These approaches can be combined to yield a substantially larger compression rate. Techniques, such as Huffman coding [75], compressed row entropy representation and [76], network stacking [77], can also be incorporated with pruning, quantization, and decomposition to provide further compression.

Although collective compression reduces model size significantly, in some cases, the model requires additional training to regain smaller models' lost accuracy. Some compression methods may even add overhead in terms of memory access on the general-purpose CPUs. Furthermore, most compression techniques are only designed for inference, and training must be performed offline on a high-performance machine. For online applications that require continuous training, such techniques may be rendered useless. Moreover, to reach the collective

compression method's full potential, a custom architecture is necessary in most cases.

6) Fast Algorithms: Fast algorithms transform the input data to reduce the computational complexity in the inference stage. The computed results must be transformed back to interpret the output. Fast Fourier transform (FFT) can be used to transform DNN weights from time-domain to frequency-domain, which converts convolution operations to simple element-wise multiplications effectively [78]. Meanwhile, Winograd algorithms reduce computational complexity by reducing the number of multiplications in convolution operations [79]. Spatial and depth-wise separable convolution reduces multiplication operation by manipulating the kernel matrix. However, spatial separable convolutions are limited to separable kernels along the length and width. Depth-wise separable convolution splits standard convolution into depth-wise and pointwise convolutions but may produce too few parameters to train the network properly [80], [81].

Despite reducing the computational complexity of a DNN model, some authors reported that fast algorithms increase memory accesses. As a result, the power consumption on hardware may increase considerably if off-chip memory is accessed frequently. Furthermore, some algorithms offer limited parallelization and are thus bottlenecked by the algorithm.

III. Hardware Architectures for DL

A brief overview of the general architectures for computing will be presented in this section, followed by a few recent innovation specialized architectures for high-performance computing (HPC). Finally, modern trends in custom hardware architecture for DL on portable devices will be reviewed.

A. General Architectures

The computational performance of DL models on general-purpose CPUs (GP-CPU) generally depends on the core count and the amount of cache memory. Modern DNNs may contain millions of parameters and over hundreds of layers. GP-CPU cannot provide adequate memory bandwidth and computational complexity to carry out DNN computation efficiently. GP-CPUs also suffer from numerous fetch instructions from external DRAM, which consumes significantly higher energy because of insufficient cache memory. This phenomenon is due to the architecture's limitation, known as the Von Neumann architecture bottleneck [82]. The universal processing nature of GP-CPUs also contributes to their inefficiency in DL computations, considering that most of the logic circuits are underutilized in the chip. Unsurprisingly, newer processors are offering support for special vector instructions [e.g., single instruction, multiple data (SIMD) and single instruction, multiple threads (SIMT)] for parallelization of multiply accumulate (MAC) operations in matrix multiplications [83]. Furthermore, the CPU vendors and DL researchers have provided several optimizations for most DL libraries and frameworks so that the CPU can take advantage of the data structure for faster processing [84]–[86]. However, training a DNN on a relatively large dataset may still take days or even weeks, depending on the workload. Clustering several CPUs on a wide area network can accelerate DNN computation by sharing the workload and parallelizing the process [87].

Unlike CPU, GPUs are a popular choice among DL researchers for the offline training of large DNNs. GPU utilizes single program multiple data architecture, which is beneficial for highly parallel computing. GPUs utilize larger memory bandwidth and efficient matrix operations in floating points, thereby resulting in high throughput. Compared with a standalone CPU, a GPU can synergize with the CPU to improve performance [88]. This phenomenon is partly due to the parallel computing platforms, such as compute unified device architecture (CUDA) and OpenCL, that enable general-purpose computing on GPUs (GP-GPU). GPU applications in DL are further reinforced by DL frameworks, such as Caffe and TensorFlow, which provide extensive library and resources for DL research [89]. These frameworks automatically optimize memory usages and multithreading tasks, depending on the hardware resources. Sun *et al.* [90] demonstrated that a cluster of 64 GPUs could train AlexNet with a full ImageNet dataset within only 1.5 min. Despite all the benefits, the use of GP-GPUs over a long period of time becomes inefficient because of their high energy consumption. Thus, such systems are mostly used as cloud servers for remote processing. NVIDIA recently announced the A100 Tensor Core GPU [91], which aimed explicitly at AI applications for data centers. The GPU incorporates unique features that accelerate DL computation, such as hardware acceleration for sparse DNN, mixed-precision arithmetic, and TensorFloat-32 (TF32) operations to accelerate data movement in the DL framework. A100 GPU can push out 624 tera floating point operations per second (TFLOPS) and 156 TFLOPS throughput on INT8 and TF32 operations at a power rating of 400 W, respectively.

Apart from the DL applications, GPUs are also responsible for various graphics processing tasks. They are not specialized solely for DL computation. This makes them somewhat inefficient in terms of energy and chip area, hence ineligible for low-power applications. Furthermore, DL optimizations such as pruning and weight decomposition cannot gain ideal acceleration on most general architectures. Compared to CPU and GPU systems, FPGA- and application-specific integrated circuit (ASIC)-based specialized architectures offer far better efficiency in DL tasks, thereby making them suitable for AI applications at the edge.

B. Custom Architectures for HPC in DL

To train a large DNN that contains millions of parameters, extensive computational resources are necessary to obtain results within a reasonable time frame. However, once a model is trained, such computation power solely for inference may not be required afterward. For this reason, cloud computing-based AI solutions are gaining popularity among smaller organizations. At the core of these cloud servers, high throughput is attained by a distributed system of custom architectures that can leverage various optimizations available for DL. The tensor processing unit (TPU) developed by Google has been designed explicitly for server-based machine learning (ML) applications. The first generation of TPU

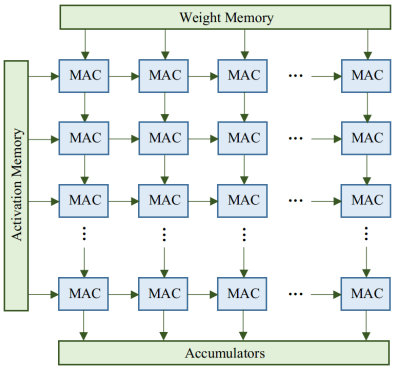

Fig. 2. In this systolic array architecture, the data are orchestrated to flow from the upper-left to the lower right in a temporal manner. In each time step, each MAC unit's result is transferred to the next one for further computation.

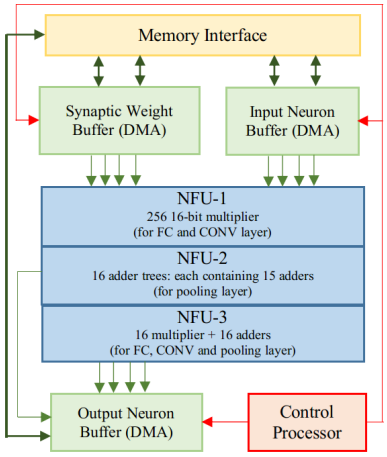

Fig. 3. Abstract view of DianNao architecture [97].

utilized several hardware architectural optimizations, such as a systolic array of quantized 8-bit matrix multiplication, large cache memory, high memory bandwidth for fast data transfer, and hardware acceleration for activation functions to achieve 23 TFLOPS performance at 45 W power usage [92], [93]. A simple representation of dataflow in a systolic array is shown in Fig. 2. Although the first-generation TPU only targeted inference, the second- and third-generation TPUs supported floating-point (FLP) operations and implemented high bandwidth memory (HBM), thereby making them applicable for training DNN. TPU v2 and v3 can generate 45 TFLOPS at 200 W and 90 TFLOPS at 250 W, respectively, [94]. Intel offered two distinct architectures for inference and training as a cloud server solution, namely, Goya and Gaudi [95]. The Goya architecture is a SIMD vector processor that consists of eight Tensor processing cores (TPC). Goya's AI capabilities were empowered by hardware acceleration for general matrix multiplication, support for the mixed-precision operation, and dedicated hardware for special NN functions. The Gaudi architecture incorporated all Goya features in conjunction with higher on-chip memories and four HBM blocks, which were essential for low latency and efficient DNN training. A single Gaudi could train up to 1650 images/s on ResNet-50 at 140 W, whereas Goya could classify more than 15 000 images/s at 120 W on the same DNN model.

Cerebras took a different approach to HPC in DL. They developed a large 46 255 mm^2 chip, called the wafer-scale engine (WSE) [96], which is 50× bigger than the largest GPU chip. WSE was equipped with sparse linear algebra computes (SLAC) that were connected with a 2-D communication mesh with 9 PB/s of bandwidth. On top of that, WSE contained 18 GB of on-chip memory that could be accessed simultaneously on a single clock cycle. Theoretically, such specifications should provide extremely high throughput from a single chip. However, powering 1.2 trillion transistors with a current of 20 MA, maintaining optimum operating temperatures, and obtaining a satisfactory yield in the fabrication process are the major challenges that must be overcome before deploying the technology into mainstream data centers.

C. Custom Architectures for Portable Applications

Several ASIC- and FPGA-based custom architectures have been proposed in recent years to take advantage of approximate computing, data reuse, large local memory, high memory bandwidth, and modified MAC units. This section will review several recent custom architectures and their key features that make them efficient for DL applications in terms of power and chip area. The red, green, and blue parts of the illustrations for hardware architectures in this section represent control, storage, and computation units, respectively. Green and blue arrows represent the relay of data and instructions, respectively.

1) Architectures With Reduced Off-Chip Memory Access: Several authors have reported that off-chip DRAM access contributes to a large portion of energy consumption during memory-intensive DL computation. Several custom memory-based architectures have been proposed to reduce off-chip DRAM access. Chen *et al.* focused on minimizing memory access usages and executing the unavoidable memory transfers as efficiently as possible for DNN computation in their DianNao [97] architecture. The authors implemented a custom storage structure to utilize the locality properties of NN layers. The storage was split into separate buffers for input-output and synapse. This approach allowed control over the memory access bandwidth depending on the data type, thereby orchestrating dataflow to minimize any conflict or bottlenecks. In this architecture, the neurons of a NN were modeled by a neural functional unit (NFU), which performed computation for the neuron and synapses. The NFU arrangements are shown in Fig. 3. To reduce area, power consumption, and memory bandwidth, 16-bit truncated fixed-point (FXP) multipliers were utilized. At 65 nm TSMC technology, DianNao performed 118× faster than a SIMD CPU with 21× less power. DaDianNao [98] is a multitiled version of DianNao (Fig. 4). DaDianNao integrated two central embedded DRAM (eDRAM) on the chip and surrounded by 16 tiles. Inside each tile, an NFU was connected to four eDRAM banks. The tiles were interconnected by a fat-tree structure, which improved communication among the tiles and the distributed eDRAM banks within the tiles. The whole chip could be set up as a node within a multinode cluster for massively parallel HPC. The communication among the nodes was enabled by the HyperTransport (HT) 2.0 interface through

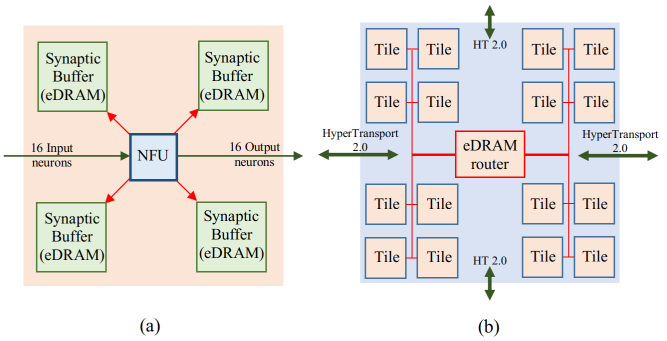

Fig. 4. Architecture inside a single tile of DaDianNao is shown in (a). The arrangement of tiles on a single chip is represented in (b) [98].

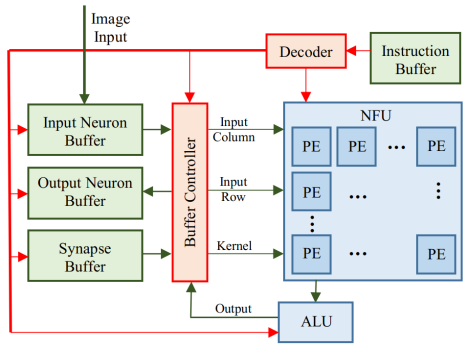

Fig. 5. ShiDianNao architecture [99].

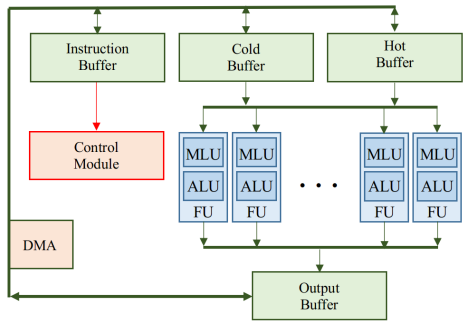

Fig. 6. PuDianNao architecture [100].

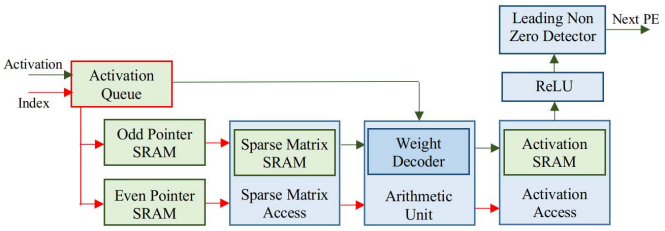

Fig. 7. Abstract view of dataflow inside PE of EIE architecture [101].

the central eDRAMs. The multichip architecture outperformed an NVIDIA K20M GPU by 430× at 150× reduced energy cost using 64 nodes.

ShiDianNao [99] completely eliminated DRAM access in image processing applications by placing a CMOS sensor next to the accelerator and enabling real-time processing. The architecture contained a customized NFU for matrix multiplication operations and an arithmetic logic unit (ALU) for computing activations. The customized NFU handled 2-D data more efficiently than DianNao. The NFU could communicate with input and output synapse buffers while distributing the data to internal PEs. Each PE was also equipped with local storage, which allowed data movement between PEs. The on-chip static random access memory (SRAM) stored all the weights and results during computation (Fig. 5). In comparison with DianNao, ShiDianNao was 1.87× faster and 60× more energy efficient in visual processing applications.

Among the DianNao family of accelerators, PuDianNao [100] was designed for generalized ML techniques, including DNNs. The authors mainly focused on the functional units and the memory hierarchy to generalize the hardware acceleration to multiple ML algorithms. Each functional unit contained an ML unit (MLU) that accelerated the commonly used ML operations and an ALU to handle general operations. The architecture also implemented a hierarchical structure of memory buffers (Fig. 6). The structure dedicated short-term storage space near computation for reusable data, larger space for long-term storage away from functional units, and finally, some storage space for temporary output data. On average, with 16 MLU, PuDianNao delivered 1.2× faster and 128.4× more energy-efficient performance than NVIDIA K20M GPU.

2) Architectures With Reduced Computation: A significant proportion of weight values often contain zero values. Thus, executing multiplication on such values wastes clock cycles, as well as energy consumed. Han *et al.* [101] exploited several DNN compression techniques and further accelerated the computation on their efficient inference engine (EIE). EIE performed customized matrix-vector multiplication and handled weight sharing on compressed DNN models efficiently. The architecture accommodated a scalable array of PEs, wherein each PE stored 131k 4-bit weight parameters of the compressed model. The control circuit parallelized matrix-vector computation by interleaving the rows of the weight matrix among the PEs. Groups of four PEs were equipped with leading nonzero detection circuits to exploit the input vector sparsity. Furthermore, EIE took advantage of the dynamic sparsity of activations to reduce computation. The dataflow in PE is shown in Fig. 7. By leveraging deep compression [102], the EIE architecture with 64 PEs performed 189× and 13× better with 24 000× and 3400× less energy than CPU and GPU, respectively.

Zhang *et al.* further developed the idea of leveraging sparse networks and proposed Cambricon-X [103] to compute sparse matrices efficiently. Cambricon-X utilized an architecture based on PEs combined with buffer controllers (Fig. 8). The buffer controller only indexed and transferred the appropriate neurons to connected PEs with minimum bandwidth overhead. The buffer controller used direct and step indexing schemes. Furthermore, multiple PEs performed computation to cope with the irregular distribution of synapses asynchronously. Compared to DianNao, Cambricon-X performs 7.2× and 6.4× better in terms of throughput and energy, respectively.

3) Optimized Dataflow Architectures: Cache data management can effectively accelerate DNN computation by relaying data to appropriate buffers, thereby reducing the instruction queue. Chen *et al.* proposed the row-stationary dataflow in

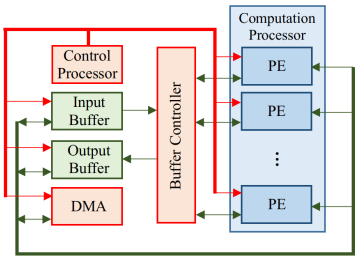

Fig. 8. Functional block in Cambricon-X processor [103].

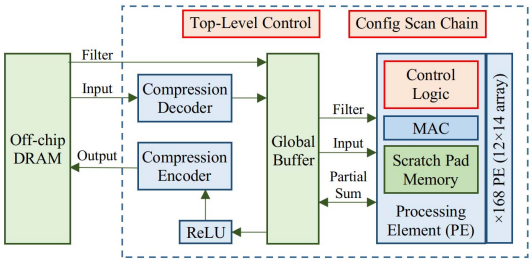

Fig. 9. Architecture for Eyeriss [29].

their CNN accelerator (i.e., Eyeriss [29]) to minimize data movement through DRAM by reusing NN parameters. They used a spatial architecture that contained 14 × 12 PEs, each containing respective MAC units and scratchpad memories (Fig. 9). They also incorporated a hierarchical network on chip (NoC), which controls communication with DRAM and with the global buffer, communication among neighbor PEs, and access to PE's local scratchpads. Data gating has also been implemented to leverage sparse networks and save processing power. The row stationary dataflow maximized the reuse of input feature maps, weights, and partial sums while considering the energy cost associated with different memory hierarchies. Eyeriss achieved a peak throughput of 33.6 giga multiply-accumulate operations per second (GMAC/s) at a clock speed of 200 MHz. Gao *et al.* interfaced a 3-D hybrid memory cube (HMC) in their Tetris [104] accelerator using the Eyeriss architecture. Tetris assigned more circuit area for computation (14 × 14 PEs) and less area for internal SRAM buffers. They also developed a partitioning scheme for HMC memory using row-stationary dataflow to improve parallelization of DNN computations. The high-memory bandwidth and high processing power achieved 4.1× better performance at 1.5× lower energy consumption than similar DRAM-based accelerators with 1024 PEs.

Several DNN optimization and compression techniques yield a sparse or irregular compact structure. This phenomenon results in the inefficient utilization of PEs and arrays, irregular memory access pattern, and imbalanced workload. Chen *et al.* presented the second generation of Eyeriss (Eyeriss v2 [105]) to address these issues. They introduced a hierarchical mesh that used a flexible NoC to meet the wide range of bandwidth requirements. The architecture had an array of 16 PEs and 16 global buffers in an 8 × 2 arrangement (Fig. 10). Each PE contained a MAC unit and a local scratchpad. The scratchpad stored short-term reuse data, whereas the global buffers acted as an additional memory hierarchy between the PEs and the off-chip DRAM. To support efficient networking through the

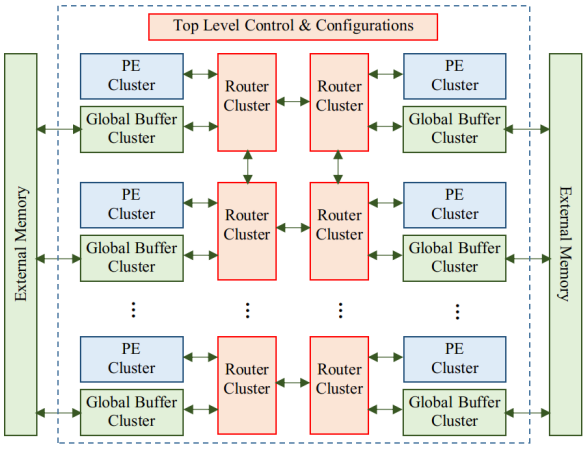

Fig. 10. Architecture for Eyeriss v2 [105].

hierarchical mesh, the PEs, and global buffers were clustered together into 3 × 4 arrays of PEs and 12 kB of SRAM, respectively. The hierarchical mesh network had four operating modes (Fig. 11). In high bandwidth mode, each global buffer transferred data to each PE independently. In high reuse mode, the data from the same buffer were routed to all the PEs. The grouped- and the interleaved-multicast mode was used when data reuse was not required in all the PEs. Moreover, each PEs operated in SIMD mode, thereby implementing further parallelism. Compared with Eyeriss v1, the architectural upgrades in Eyeriss v2 provided 42.5× and 11.3× more throughput and energy efficiency, respectively.

4) Reconfigurable Architectures: The inherent reconfigurability of FPGAs allows hardware architectural features such as dataflow, pipelining, and memory buffers to be tailored according to the application's needs. Furthermore, on-chip design space exploration (DSE) can be performed using high-level synthesis (HLS) to identify the optimal configurations based on the CNN parameters. Meloni *et al.* [106] prioritized parallelizing CNN computation by introducing the reconfigurability of filter-kernel size and strides of a specialized FPGA-based convolution engine during runtime. Their convolution engine contained reconfigurable arrays of MAC units and line buffers. Two GP-CPU cores handled the reconfiguration and instruction scheduling for the CNN accelerator, thereby adapting to the CNN task. Wei *et al.* [107] developed a framework that generated systolic arrays on FPGAs from a C program for CNN inference. PE mapping, shape selection, and data reuse strategy for the systolic array were implemented automatically by the proposed framework to optimize resource usage for maximum throughput. DiCecco *et al.* [108] designed a custom-precision FLP core using HLS to reduce lookup table (LUT) utilization and improve throughput for CNN operations. They relied on the fact that CNN accuracy does not suffer from reduced precision operations significantly and thus implemented a variable precision multiplier for FPGAs, which can be configured through HLS. Zhang and Li [109] focused on optimizing on-chip and off-chip memory access using a modified OpenCL kernel. The proposed kernel uses an analytical performance model to design a CNN accelerator on FPGA by analyzing the CNN classifier's resource requirement

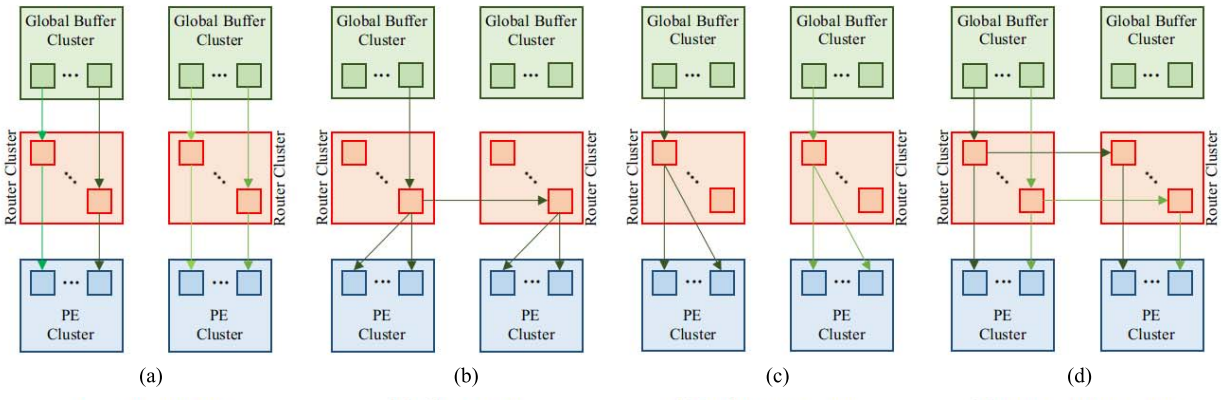

Fig. 11. Data movement with hierarchical mesh in Eyeriss v2 [105]. (a) high bandwidth mode. (b) high reuse mode. (c) grouped multicast mode. (d) interleaved multicast mode.

and available resources on the FPGA. Ma *et al.* [110] used Verilog scripts to parameterize their FPGA design to optimize loop operations and dataflow in CNN computations. Choi and Cong [111] also concentrated on optimizing FPGAs' loop operations by using an accurate DSE strategy on their proposed HLS-based framework. The framework developed by Chen *et al.* [112] quantized the weights of a NN to the power-of-2 and thus replaced all multiplications with shift operation for the efficient processing on FPGA. They also incorporated a reconstructed gradient function to minimize gradient noises caused by quantization. They reported up to 2.9× speed up and 68.7% reduction in energy consumption in inference compared with AlexNet on FPGAs. Wang *et al.* developed the FPDeep framework [113] to address the CNN workload distribution on FPGA clusters. FPDeep enabled workload balancing and model parallelism among FPGAs through fine-grained pipelining, distributing all weights to on-chip memories, and aligning the parameters using a 1-D topology. The authors reported that the performance of the FPGA clusters that used FPDeep was only limited by the inter-FPGA communication bandwidth.

Fast algorithms such as FFT use the frequency domain to compute data, while Winograd reduces the number of multiplication operations at the expense of several addition operations. Although this method improves parallelization in hardware, the number of memory access also increases. In such cases, further optimizations are required to manage the additional memory usage caused by additional summation operations. Aydonat *et al.* [114] presented an FPGA-based hardware architecture written in OpenCL language, which exploited Winograd transform to accelerate DL computation. The architecture reduced the overall resource usage by leveraging built-in digital signal processing (DSP) functions on the FPGA fully. Similarly, Zhang and Prasanna used FFT and Overlap-and-Add on FPGA design to reduce CNN computation and memory access latency by introducing double buffering and concurrent heterogeneous processing [115]. Liang *et al.* [116] developed an FPGA framework to efficiently apply Winograd and FFT algorithms for CNNs. The framework instantiated multiple specialized PEs, which transformed the input, applied a fast algorithm, and finally inverse transformed the output. In the case of PE for Winograd, the transformations of input feature maps and filter weights were performed in parallel.

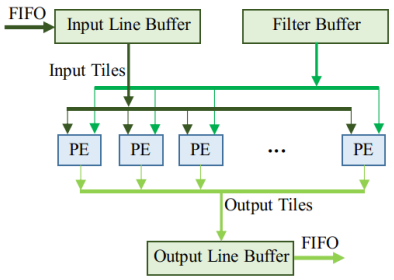

Fig. 12. PE arrangement on FPGA for proposed fast algorithm [116].

The multiplications were replaced by shift operations using LUT arrays. On the other hand, PE for FFT did not fully parallelize the operations because of the large input size. Similar to Winograd PE, multiplications were not used in FFT PE. Across multiple PEs, parallelization was achieved by using the loop unrolling method. The results of the fast algorithms were accumulated across all input channels and were transformed back to the spatial domain afterward. The architecture is shown in Fig. 12. The implementation on the Xilinx ZCU102 FPGA performed 2.5× more efficiently than NVIDIA TitanX GPU.

5) Storage-Computing Integration: In-memory computing can reduce energy consumption by eliminating data transfer between computation and storage units, thereby enabling a new generation of computer architectures to efficiently compute arbitrary arithmetic and control functions. Compared to DRAM cells, memristor-based emerging devices are considered for their high density and nonvolatile nature. Shafiee *et al.* [117] utilized memristor crossbar arrays to compute and store dot products from the input weights in an analog manner for their DNN accelerator, called in-situ analog arithmetic in crossbars (ISAAC). ISAAC was implemented as a tiled architecture. Each tile embedded several *in situ* MAC units and eDRAM buffers. The tiles also incorporated shift-and-add function, sigmoid function, and max-pool units. The MAC units were equipped with a few crossbar arrays and analog-to-digital converters (ADCs) connected by a shared bus. However, each array was dedicated to only one CNN layer because as they cannot be reprogrammed. The outputs of a layer were collected on the eDRAM buffer until the next layer's computation could be executed.

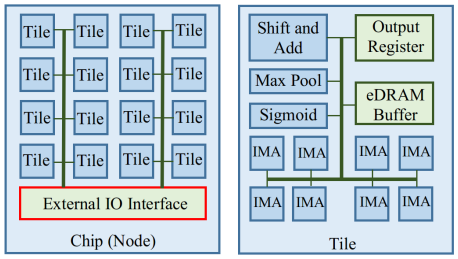

Fig. 13. Tile setup and *in situ* MAC arrangement for ISAAC [117] are shown in (a) and (b), respectively.

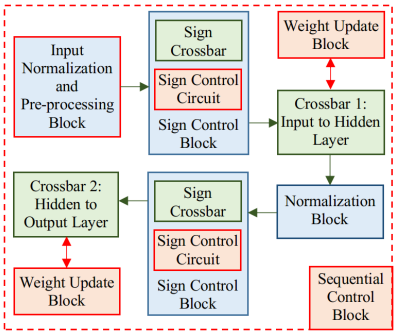

Fig. 14. Memristor-based DNN architecture with analog backpropagation [118].

The buffering limitation was mitigated by a balanced inter-layer and intratile pipelining scheme. Additionally, a bit-encoding scheme reduced stress in ADCs, digital-to-analog converter (DAC), and eDRAMs. The architecture is illustrated in Fig. 13. Using a 16-chip configuration, ISAAC achieved 14.8× higher throughput while consuming 5.5× lower energy than DaDianNao.

Krestinskaya *et al.* [118] proposed an analog circuit for the backpropagation learning with various memristor-based DNN architectures. The functional blocks of the architecture are shown in Fig. 14. The backpropagation circuit performs differential operations by using the memristor crossbar as the dot product operator. However, the authors mentioned that the parasitics, leakage current, and sneak paths in memristive devices limit crossbars arrays' scalability.

Resistive random access memory (ReRAM) is an emerging memristive memory technology that enables efficient computation in memory. Chi *et al.* [119] proposed the PRIME architecture, where a part of the ReRAM memory arrays can alternate between storage and compute units. They modified the standard wordline decoder and drivers (WDD), column multiplexers, and sense amplifiers for ReRAM, and partitioned the storage banks into memory subarrays (MS), full-function subarrays (FFS), and buffer subarrays (BS) (Fig. 15). The FFS can operate in computation mode for NNs, as well as in storage mode. Similarly, the BS also acts as storage when FSS is not in computation mode. The sense amplifier was reconfigured to detect the higher precision analog value for computation compared to storage requirements so that that matrix multiplication can be performed. In addition, extra hardware was added to detecting the sign bit and to implement a ReLU function. The modified column multiplexer executed analog substractions and nonlinear threshold functions. Although their implementation exerted a 60% area overhead, the computation energy was saved by 94% by reducing external memory accesses. Cheng *et al.* [120] augmented the PRIME architecture with training support and named it TIME. The control units and the sense amplifiers were slightly modified to incorporate additional steps required for backpropagation. However, enormous training iterations posed difficulty in constantly tuning ReRAM cells. So, they implemented a lookup table policy to avoid complex tuning transformations of voltage and conductance. They also applied a one-direction writing scheme (use of reset only) and smart memory refresh strategy to combat ReRAM technology limitations. The TIME architecture trained a CNN model 5.3× more efficiently than DaDianNao. To map a typical CNN model into the single-bit ReRAM technology, Zhu *et al.* [121] proposed a framework, which used a hardware-aware multiprecision quantization algorithm. Researchers have further innovated the ReRAM technology by developing multibit MAC units [122] and analog ASICs [123] for edge devices.

Spin-transfer torque (STT) magnetic RAM (STT-MRAM) is a nonvolatile memory technology that is expected to have almost no power consumption through leakage current. The technology also scales better than CMOS at lower fabrication nodes [124]. However, this technology is still in its infancy because of the high current requirements for reorienting magnetization. Nevertheless, several STT-MRAM-based architectures have been proposed for DL computation. Pan *et al.* [125] exploited binary NN (BNN) in STT-MRAM-based architecture for in-memory computing. Their architecture included a multilevel cell STT-MRAM structure, and each cell's resistance represented a 2-bit data value. Modified sense amplifiers were used to capture the resistance value and perform logical operations with the crossbar structure. The functional blocks are shown in Fig. 16. Their 4-bit full-add operation only consumed 3.4 pJ with just 9.4 ns latency. The architecture was evaluated on Modified National Institute of Standards and Technology (MNIST) dataset with binary CNN (BCNN) and was reported to require 48× less energy than memristor-based CNN architectures. Angizi *et al.* [126] extended the idea of STT-MRAM-based in-memory computation for BNN by enabling the in-memory computation of low bit-width CNNs. Their MRAM-based in-memory accelerator (MRIMA) design [126] utilized hardware-efficient bitline computation techniques to perform multiple Boolean logic functions parallelly within a single clock cycle. The overall memory was divided into banks, where each bank contained multiple memory matrices along with buffers and interfaces (Fig. 17). MRIMA reduced data movement across memory by maximizing spatial locality and allocating memory as close to their corresponding operands as possible. Therefore, it reduced operation latency and energy costs. Experimental results showed that MRIMA was 1.8× more energy-efficient and 2.4× faster than DRAM-based in-memory computing architectures.

6) Stochastic Computing Architectures: Stochastic computing (SC) is an emerging field among DL researchers. Its popularity stems from the fact that complex computations

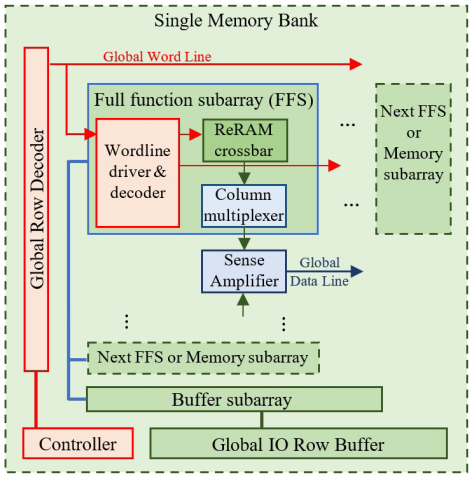

Fig. 15. Structure inside a single memory bank of PRIME architecture [119].

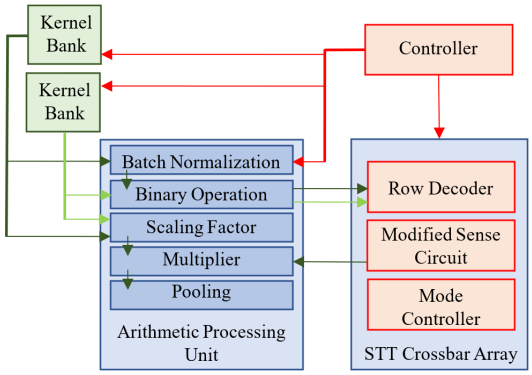

Fig. 16. Functional blocks for in-memory computing architecture using STT-MRAM [125].

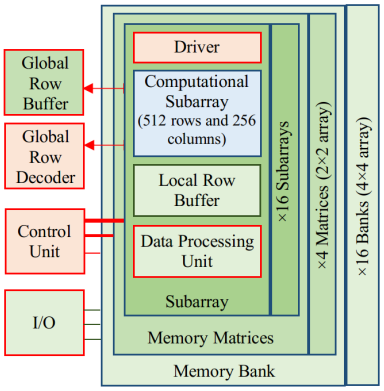

Fig. 17. Architecture for MRIMA [126].

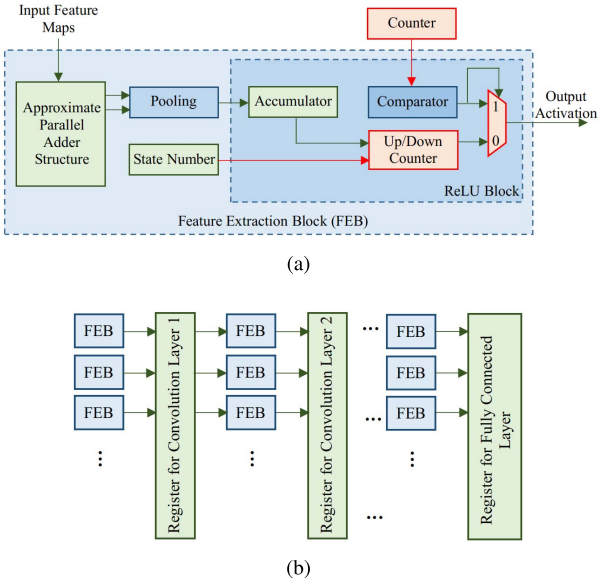

Fig. 18. Architectural design for stochastic computing with HEIF. (a) Functional design in FEB. (b) FEB arrangement for proposed HEIF framework [127].

can be converted into simple bit-wise operations using continuous numbers represented by streams of random bits. Furthermore, stochastic computing is tolerant of noises in operands. The results' accuracy in stochastic computing increases with the number of bits in the stream, which is advantageous for NN operations. Li *et al.* applied stochastic computing in their highly efficient inference framework (HEIF) [127] for DNN. The architecture represented a deep CNN that contained multiple feature extraction blocks (FEBs). Each FEB contained serially connected inner-product, max pooling, and ReLU functional blocks. The authors designed a stochastic computing-based ReLU block for hardware implementation. The ReLU block accumulated the bit-stream and compared the accumulated value with a reference value for dynamic estimation. This approach mitigated additional latency incurred for the sign estimation of the bitstream. An approximate parallel counter for inner-product block was designed with an adder tree using inverse mirror full adders, which reduced area, power, and energy consumption. The storage space for weights was reduced by using k-means clustering. The FEB (Fig. 18) represented each layer in a CNN, and d-flip flops were placed in-between the layers to hold the data temporarily. An 8-bit and 16-bit FEB implementation was fabricated, which achieved energy efficiencies of 1.2 M images/J and 1.3 M images/J, and throughput of 3.2 M images/s and 2.5 M images/s, respectively.

7) Neuromorphic Computing Architectures: Neuromorphic computers are brain-inspired processors that simulate the neuro-biological architectures present in our nervous system. Memristive devices are gaining considerable attention in neuromorphic computing because of their synapse-like analog behavior. TrueNorth [128] developed by International Business Machines (IBM) corporation incorporated a neuromorphic architecture that demonstrated high efficiency in DL application by using spiking NNs (SNNs). The chip contained 1 million digital neurons distributed among 4096 cores that were connected by 256 million synapses through an event-driven crossbar infrastructure (Fig. 19). Communications among the neurons were achieved by transmitting spikes. The communication data were encoded using the frequency, time, or spatial distribution of spikes. An asynchronous design was used for communication and control systems. All the computations were handled synchronously. The merge-split blocks enclosed an outer array of cores and combined the spikes from multiple buses into a single stream. Fig. 20 illustrates the overall architecture of TrueNorth, which only required 65 mW for the computation of 58 Giga synaptic

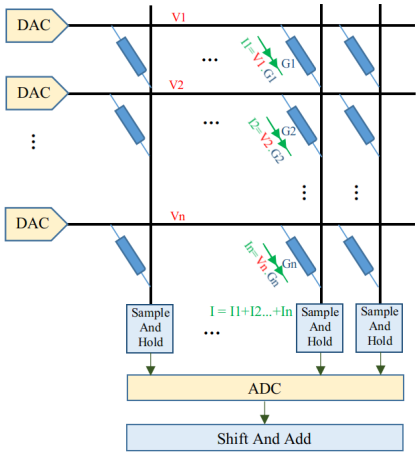

Fig. 19. General crossbar architecture for neuromorphic computation.

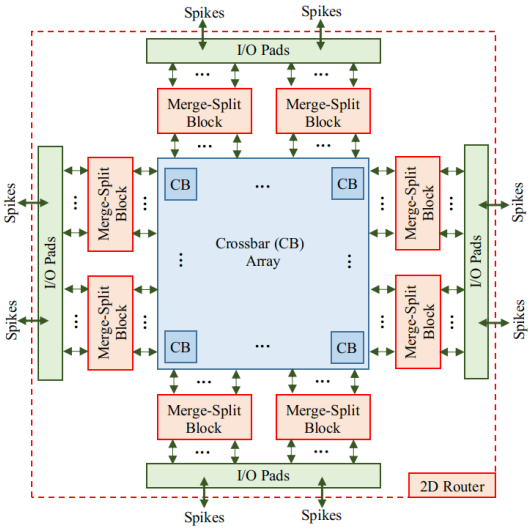

Fig. 20. Neuromorphic architecture for TrueNorth [128].

operations per second (GSOPS) using 5.4 billion transistors present in the chip. However, the TrueNorth chip suffered from limited precision of synaptic weights, and experimental results showed that although TrueNorth was very energy efficient compared with GPUs and FPGAs, the number of cores utilized to obtain a comparable accuracy was relatively high [129]. Intel's neuromorphic chip Loihi [130] also used asynchronous SNN to implement fine-grain parallelization in learning and inference with high efficiencies. Loihi consisted of 130 000 neurons within 128 cores, along with three embedded $\times$86 processor cores. Additionally, the chip supported sparse network compression, 1-9-bit variable weight precision, and hierarchical connectivity of neurons. Despite having lower neuron density than TrueNorth, Loihi performed more efficiently on various DL applications [131].

Frenkel *et al.* presented an online-learning digital spiking neuromorphic processor (ODIN) [132] that utilized spike-driven synaptic plasticity (SDSP) for training SNN models efficiently. As illustrated in Fig. 21, their design contained a 256 $\times$ 256 crossbar array containing 256 neurons and 64 k synapses. The individual states and parameter values of neurons and synapses were stored in 4 and 32 kB of high-density SRAMs. A conventional serial peripheral interface bus

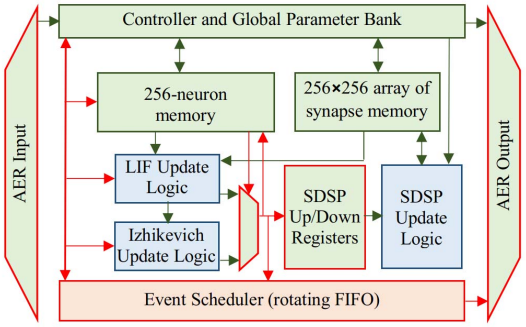

Fig. 21. Neuromorphic architecture of ODIN [132].

was used to read and write these memories. A multiplexing approach updated the neurons and synapses sequentially. The address-event representation bus was equipped with various event types to experiment with emulated biological neuron behavior. An event scheduler arbitrated and directed external and internally generated events. The array architecture resulted in a very high neuron and synapse density with 0.68 μm^2 per 4-bit synapse. The evaluation of ODIN on the classification tasks with the MNIST dataset showed an energy consumption of only 15 nJ during inference.

Tapiador-Morales *et al.* [133] synthesized an event-driven multi-CONV processor on an FPGA based on a leaky integrate-and-fire neuron model (LIF). The kernel values were applied over the neuron models, which fired spikes when the membrane potential values in embedded memory reached a threshold value. This phenomenon reduced the number of memory accesses, thereby saving bandwidth. The convolution engine applied masks to indicate decaying neurons and neurons that met the refractory period. Then, this engine processed the kernel values row-by-row until the entire kernel was applied. This step increased data reuse and decreased memory access even more. A host microcontroller configured the accelerator through a 32-bit digital interface. In the case of spiking CNNs (SCNN), their FGPA model computed 64 convolutions in parallel, with kernel sizes ranging from 1×1 to 7×7. The functional block is shown in Fig. 22. The row-by-row data access from memory reduced the latency to convolve a square kernel between 11.3 μs and 9.01 μs on a Kintex-7 FPGA running at 100 MHz.

Frenken *et al.* [134] investigated the use of binary weights for SNN on their quad-core neuromorphic processor called MorphIC. They implemented a stochastic version of the SDSP (S-SDSP) learning algorithm suitable for binary weights. The MorphIC embedded 2000 LIF neurons and more than 2 million plastic synapses that were interconnected by crossbar routing synchronously. Interchip connectivity was enabled by mesh-based routing, whereas star-based routing was used for intercore connectivity. An additional mid-level router handled intrachip and intercore communication within all four local cores. All connectivity information was stored locally in the neurons. Thus, access to the local mapping table was not required. The architecture is illustrated in Fig. 23. Although only 27 connectivity bits per neuron were available, the low-memory hierarchy allowed up to 1000 and 2000 connections for inputs and outputs, respectively. Therefore, 2.86 mm^2

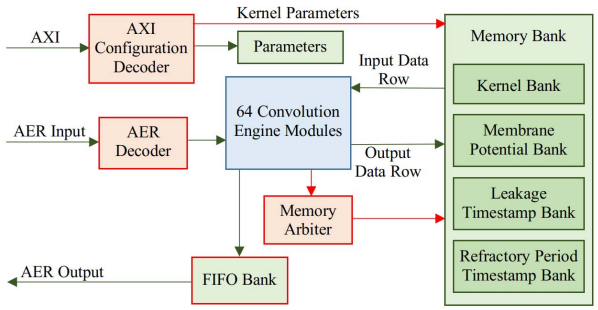

Fig. 22. Implementation of multiconvolutional processor on FPGA [133].

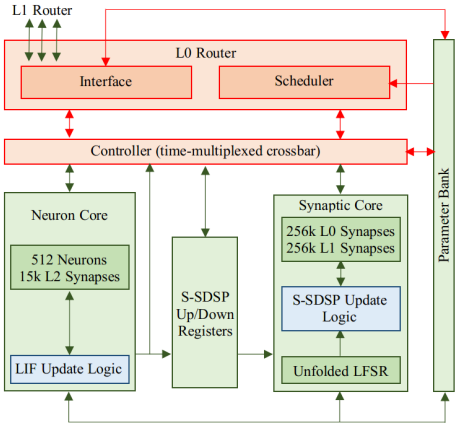

Fig. 23. Architecture for MorphIC [134].

MorphIC fabricated with 65 nm technology obtained a high density of 738 k 1-bit synapses per mm^2. MorphIC consumed 51 picojoules per synaptic operation (pJ/SOP), although a better fabrication process could lower this value.

Moradi *et al.* proposed a dynamic neuromorphic asynchronous processor (DYNAPs) [135] that implemented a two-stage asynchronous routing scheme within the array of neurons and heterogeneous memory structures to store data as programmable routing tables. They minimized the memory requirement in SNN by combining point-to-point routing and multicast routing in a three-level hierarchical mesh architecture analytically. The lowest level of routing mesh, which is R1, controlled local communications within a core. The intermediate-level router R2 communicated with lower and higher-level routers R3 if any events were triggered between cores. Finally, R3 mainly routed data packets over long distances in the 2-D mesh. The routing scheme is shown in Fig. 24. The authors fabricated DYNAPs using 180 nm very large-scale integration (VLSI) technology, which performed on par with TrueNorth and Loihi, despite using inferior fabrication technology.

The Tianjic [136], [137] processor was presented to unify digital artificial NN (ANN) computing with neuromorphic computing. The Tianjic architecture not only supported SNN but could also implement DL on conventional architectures such as CNNs and RNNs. Unified functional cores (UFC) were the building blocks of a multicore Tianjic architecture, which could either be homogeneous or heterogeneous. Each UFC consisted of a hybrid activation buffer (HAB) to receive and organize inputs, a local synapse memory (LSM) to

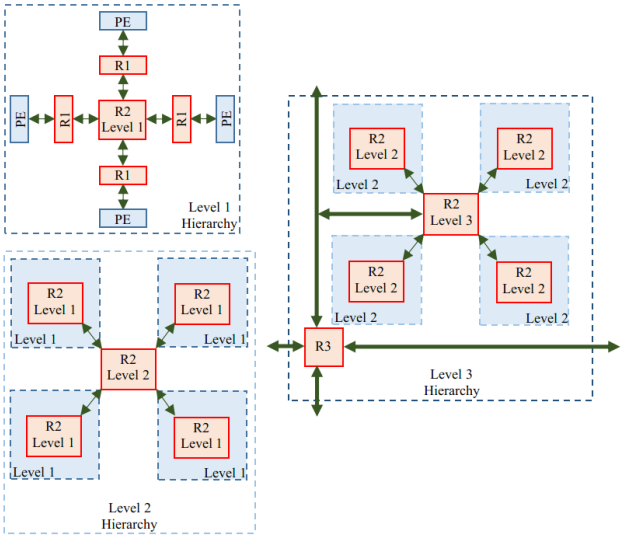

Fig. 24. 3-level routing system in the DYNAPs architecture. At lowest level R1 performs local communication, R2 communicates between levels, and R3 performs interchip communication [135].

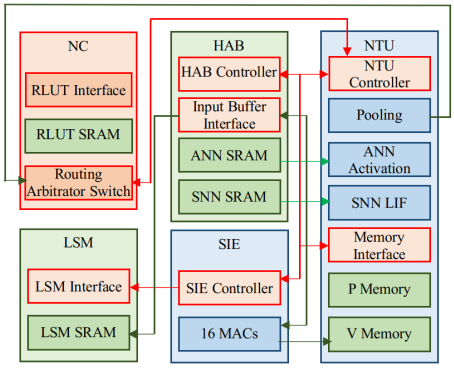

Fig. 25. Unified architecture of Tianjic [136].

store weights, a shared integration engine (SIE) to perform MAC operations, a nonlinear transformation unit (NTU) to generate output, and a network connector (NC) to route the data packets. The HAB and NTU operated independently to perform in SNN or ANN modes to work with spikes or digital data, respectively. Therefore, they also functioned as ANN to SNN or SNN to ANN converter. LSM and SIE could adapt to the data types depending on the ANN or SNN mode. SIE could perform six different vector-matrix operations to accommodate different types of DL architectures. HAB and NTU prepackaged the data routed by NC. Thus, its functionality did not change. Although each UFC contained its hierarchical memory, data communication was faster because of near-memory computing. The NC skipped the routing of matrix rows or columns if they contained zero, thereby saving more energy in sparse networks. The NC also routed data to multiple neurons to improve parallelism. All UFCs could operate in a single-mode (homogeneous) or at different modes (heterogeneous) in the multicore paradigm. The functional blocks utilized in the architecture are depicted in Fig. 25. A single core Tianjic chip fabricated at 28 nm mode operated at 0.95 W and outputs 1278 giga operations per watt (GOPS/W) and 649 GSOPS/W.

8) Custom Architecture Frameworks: The rapid growth of IoT and the demand for integrating DL in IoT infrastructure are pushing the boundaries of hardware architectures toward specialization. Many different applications have specific requirements. To meet these requirements, customizing and fine-tuning an architecture depending on the application may deliver significant efficiency gains. However, most applications also require a GP-CPU along with an accelerator for smart IoT devices. Furthermore, commercial microprocessors and microcontrollers do not allow the fine-tuning of instruction set architecture (ISA). Therefore, RISC-V ISA [138] was developed as an open-source ISA that could be customized depending on the client's needs. Researchers have also contributed several softcore designs [139], [140] that could be implemented on FPGA or taped out as ASICs. The Chipyard framework [141] was built on top of the RISC-V project to allow customization of prebuilt CPUs and their integration with customizable accelerators. Several implementations of RISC-V accelerators for DL applications have been proposed [142]–[145]. Furthermore, researchers at Google recently announced that they used deep reinforcement learning to automate chip placement, which is a complicated and time-consuming process in ASIC design. They completed the task within six hours, whereas human experts require several weeks. The resulting chip design was more efficient in terms of area and power than a similar design with manual placement and commercial tools [146].

IV. Discussion and Future Trends

In this section, we highlight the recent trends in hardware architecture development for DL. We also emphasize some key innovations in architectures that enable low-resource DL applications on portable devices.

Over the past few years, tremendous progress in algorithms and hardware for AI applications has been observed. The availability of numerous distinct datasets and various DL models makes hardware comparison and analysis a tedious process. The reproduction of the published results is further complicated by the use of different generations of CPUs and GPUs as a baseline, proprietary fabrication technology for ASICs, and the use of vendor dependent DSP tools for FPGAs. The DL optimization techniques approached in Section II-D allow DNN to be deployed on resource-constrained devices with reasonable accuracy–performance tradeoff. However, these optimizations may present some challenges in terms of hardware implementation on different platforms. For example, sparse matrices resulting from pruning and weight decomposition contain numerous weight parameters with the value zero. Typical MAC units waste energy by performing multiplication operations with zero.

From the hardware perspective, general-purpose CPUs are not capable enough to handle complex DNN models efficiently. The performance increase from a newer generation of CPUs is essentially a result of smaller transistor nodes, which are almost at their limit. Although modern GPUs are adopting special hardware features for DL computation, GPUs generally consume far too much energy and generate excessive heat to be considered for portable DL applications. Therefore, specialized devices are necessary to accelerate DL models with high efficiency and low latency.

Several DNN accelerators for portable devices have been reviewed in Section III. Their key aspects have been summarized in Table I. At first glance, we observe that the researchers are moving away from conventional computing and are adopting brain-inspired neuromorphic computing for AI applications. Researchers are also exploring various emerging devices on their hardware architecture for low-power and low-latency applications. However, whether these technologies will end up in widespread adoption is still an open question. The hardware features used in the current technology, computation with the emerging technologies, and future research directions for efficient DL computation are discussed below.

A. Optimized Computation

Our literature review found that the overall efficiency and performance of an accelerator can be improved substantially by reducing off-chip memory access. In comparison, the fetch and store instructions in conventional CPUs consume the most energy in performing a simple arithmetic operation. Although accelerators can generally benefit from increasing memory bandwidth and enhancing parallelization of instruction through multiprocessing and multithreading, they come at the cost of higher energy and circuit area requirements.

For low-resource computation, the use of low-precision mathematics (e.g., 16-bit or 8-bit fixed point values) has been adopted as a standard because of the fact that DNN can tolerate such approximation without significant degradation in model accuracy. However, for training purposes, the calculation of the gradient requires higher precision. Therefore, the accelerators are either designed for inference only, or a custom mixed-precision solution is implemented on the hardware for training. ALUs and PEs can also be integrated with leading zero detection to skip redundant operations. This is beneficial in processing sparse matrices that result from pruning or weight decomposition techniques. For further optimization, DNN architecture-specific accelerators can be implemented with software-hardware codesign. Such specialized hardware for CNN or LSTM architectures provides excellent results on specific tasks. For example, the EIE accelerator leverages the Huffman coding of Deep Compression techniques to efficiently process pruned, quantized, and coded DNN parameters. However, such specialized architectures are not feasible for long-term use as DL is a rapidly evolving academic field, and an efficient algorithm today may become obsolete in the next year.

B. Dataflow Architectures

Dataflow-based architectures promote data reuse so that data movement can be minimized. Among these architectures, weight and output stationary dataflow are popular choices. In weight stationary, the weight parameters are stored in the same programming elements (PE), whereas the input feature maps are sent to all the PE. The weight values are used multiple times during matrix multiplication in each iteration of training and inference. Therefore, keeping a weight value

stationary in the same PE reduces the cost of fetching it again from off-chip memory for the next operation in the same parameter. In such cases, the architecture temporarily mimics near-data computation. The duration of stationary weights depends on the dimensions of the PE structure. Output stationary and other dataflows are implemented similarly. However, the performance gain in each case depends on the DNN architecture and its workload, as the number of parameters to be kept stationary affects data cache period within available PEs. In the case of no local reuse, a large global on-chip memory is shared by all parameters. Some accelerators even include high-level data routing schemes that allow parameter sharing across multiple clusters of PEs. These routing mechanisms (NoC) require a sophisticated control unit, which may increase hardware complexity. In contrast, systolic array-based architectures orchestrate data movement among the PEs in an orderly manner to perform a predefined sequence of operations. The timed processes can be handled using a simple control unit. However, the PEs are dependent on predefined input connections, so fine-grain parallelization is not possible.

C. Reconfigurable Versus Specialized Hardware

FPGAs offer remarkable fine-grain and coarse-grain optimizations because of their hierarchical storage structure and flexible scheduling mechanism. This instance enables complex data access modes, thereby improving memory and energy efficiency. Furthermore, their reconfigurability makes them ideal for long-term use if any update is necessary to the synthesized architecture. The architecture can be configured according to the data structure for maximum efficiency. By contrast, ASICs provide the highest performance per watt on a smaller footprint, however, at the expense of almost no reconfigurability, slow tape-out process, and expensive development cost.

Specialization is a critical aspect in obtaining maximum efficiency in a particular DL task. The demand for custom hardware, including general-purpose processors, has increased due to emerging markets, such as the IoT, Industry 4.0, and smart city infrastructure. Some applications may require high performance for security purposes, whereas others may need long battery life for real-time online applications. To meet the large variety of demand, custom architectures for target DL application is an obvious solution. However, as we are continually witnessing rapid innovations in DL algorithms, fully specialized nonreconfigurable hardware may not be a viable solution. Therefore, the lifetime and the investments involved must be considered before deploying DL hardware.

D. Emerging Computing Paradigms

Large data structures require large cache memory to minimize external DRAM access for low-energy computation. The data movement among off-chip DRAM also increases the latency of the processor. 3-D memory structures, such as HBM and HMC, can eliminate the memory bandwidth bottleneck. However, such structures also increase the overall energy consumption. Processing-in-memory is another viable solution for reducing data movement. Therefore, it improves energy efficiency and mitigates architecture latency. Such storage-computing integration is often designed as a crossbar structure that contains weight parameters at each node. However, the commonly used DRAM-based in-memory computing solutions are very inefficient. As a result, various emerging devices, such as ReRAM and STT-MRAM, are being investigated. Such systems use mixed signals (e.g., analog and digital) for in-memory computing. The simulation results show that these systems consume very low energy than DRAM-based solutions. However, this is still an ongoing research area. Several other companion devices, such ADCs and DACs, require significant due to their intricate design and unreliable error tolerance. With the successful implementation of these emerging devices, future sensors can incorporate built-in AI capabilities. Such sensors will directly process input data in real-time with minimum latency and data movement.

Theoretically, quantum and optical computing have the potential to outperform currently available conventional processors at AI tasks. However, they are still in their very early stages and require further research. Meanwhile, stochastic and neuromorphic computing can also be beneficial for DL accelerators with current technology. SC uses bitstreams as data, which only requires AND gates (opposed to multipliers) for NN computation. However, SC requires an unbiased random number generator, which is challenging to implement on digital circuits. Neuromorphic computing mimics brain functionality by using SNNs. SNN can be emulated on current CMOS technology. However, proper neuromorphic computation requires asynchronous functionality, which is quite challenging for digital circuits. Furthermore, the data exchange between neuromorphic and conventional computing presents another hurdle. Neuromorphic architectures with emerging devices are better at implementing SNN due to their analog computing nature. For now, the unified architecture in Tianjic, which incorporates digital neurons and emulated spiking neurons, may provide the best of both computing technologies. This approach is a positive leap toward artificial general intelligence.

E. Prospective Research Directions

Although the DL optimization techniques and architectural innovations show positive results individually, the software-hardware codesign holds a better prospect that can enable fine-tuning the dataflow, memory hierarchy, and even instruction-level parallelization depending on the available resources and user requirements. A unified, stable framework that can compare and test various DL models on different ASIC accelerators and FPGAs would play a vital role in developing custom hardware architectures. Therefore, developing a standard open-source hardware ISA and hardware design flow is necessary for a unified framework.

To take advantage of different hardware features, heterogeneous architectures can also be an interesting research area. Such architectures may include partial reconfigurability like FPGAs, while common logic circuits are hardwired for saving circuit area and power consumption. Memristive technology can also be incorporated into heterogeneous systems, but current technology has several communication limitations between digital logic and analog circuits. On the other hand,

completely transferring computation from the digital to analog domain deserves attention. For example, SNNs can be trained to perform general-purpose computation along with NN computation, thus eliminating communication bottlenecks in heterogeneous architectures.

V. Conclusion

In this review, we highlighted the recent innovations in DL hardware architectures. We began by briefly explaining modern DL optimization and compression techniques that can be leveraged by low-power portable DL accelerators. Then, a thorough review of numerous hardware architectures, which illustrate the functionalities of various memory structures, PEs, and hierarchical routing, has been presented. This work includes custom architectures based on FPGA, ASIC, and simulated architectures with emerging devices. We have also explored computing paradigms, such as neuromorphic and stochastic computing for DL acceleration. As CNNs are utilized in most DL applications, we mainly focused on the optimizations and hardware architectures of CNNs. However, the architectural features and optimization techniques can also be generalized to other DNN types. Finally, a summary of the software optimizations and custom hardware architectures has been provided, followed by a discussion about the future trends in software and hardware for DL applications on portable devices.

Our review discusses the increasing research interest for neuromorphic computing-based accelerators. However, this technology may not be fully ready for adoption yet because it still requires the repurposition of digital computation into the analog domain for efficient implementation. The rapid innovations in DL and the rising demand for low-power AI applications are still driving researchers to push the boundaries of conventional computing and explore other computing paradigms. A stable framework for software-hardware codesign is necessary to enable society to embrace new architectural changes and software optimizations. In conclusion, despite all the innovations in custom architectures for DL, many opportunities with computing methods and emerging devices still need to be explored.

References

[1] Z.-Q. Zhao, P. Zheng, S.-T. Xu, and X. Wu, "Object detection with deep learning: A review," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 11, pp. 3212–3232, Nov. 2019.

[2] W. G. Hatcher and W. Yu, "A survey of deep learning: Platforms, applications and emerging research trends," *IEEE Access*, vol. 6, pp. 24411–24432, 2018.

[3] M. A. Zulkifley, "Two streams multiple-model object tracker for thermal infrared video," *IEEE Access*, vol. 7, pp. 32383–32392, 2019.

[4] N. S. Jaddi and S. Abdullah, "Optimization of neural network using kidney-inspired algorithm with control of filtration rate and chaotic map for real-world rainfall forecasting," *Eng. Appl. Artif. Intell.*, vol. 67, pp. 246–259, Jan. 2018.

[5] G. Murtaza *et al.*, "Deep learning-based breast cancer classification through medical imaging modalities: State of the art and research challenges," *Artif. Intell. Rev.*, vol. 53, no. 3, pp. 1655–1720, Mar. 2020.

[6] J. M. Stokes *et al.*, "A deep learning approach to antibiotic discovery," *Cell*, vol. 180, no. 4, pp. 688–702, Feb. 2020.

[7] J. Wang, X. Zhang, L. Cheng, and Y. Luo, "An overview and meta-analysis of machine and deep learning-based CRISPR gRNA design tools," *RNA Biol.*, vol. 17, no. 1, pp. 13–22, Jan. 2020.

[8] M. E. H. Chowdhury *et al.*, "Can AI help in screening viral and COVID-19 pneumonia?" 2020, *arXiv:2003.13145*. [Online]. Available: http://arxiv.org/abs/2003.13145

[9] P. J. Phillips *et al.*, "Face recognition accuracy of forensic examiners, superrecognizers, and face recognition algorithms," *Proc. Nat. Acad. Sci. USA*, vol. 115, no. 24, pp. 6171–6176, Jun. 2018.

[10] F.-Y. Wang *et al.*, "Where does AlphaGo go: From church-turing thesis to AlphaGo thesis and beyond," *IEEE/CAA J. Autom. Sinica*, vol. 3, no. 2, pp. 113–120, Apr. 2016.

[11] P. Ruamviboonsuk *et al.*, "Deep learning versus human graders for classifying diabetic retinopathy severity in a nationwide screening program," *Npj Digit. Med.*, vol. 2, no. 1, p. 25, Dec. 2019.

[12] M. Mohammadi, A. Al-Fuqaha, S. Sorour, and M. Guizani, "Deep learning for IoT big data and streaming analytics: A survey," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 4, pp. 2923–2960, 4th Quart., 2018.

[13] J. Chen and X. Ran, "Deep learning with edge computing: A review," *Proc. IEEE*, vol. 107, no. 8, pp. 1655–1674, Aug. 2019.

[14] W. Zhou, Y. Jia, A. Peng, Y. Zhang, and P. Liu, "The effect of IoT new features on security and privacy: New threats, existing solutions, and challenges yet to be solved," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 1606–1616, Apr. 2019.

[15] S. S. BintiMdSallah, H. Mohamed, M. M. I. Reaz, and M. S. Amin, "CMOS downsizing: Present, past and future," *J. Appl. Sci. Res.*, vol. 8, no. 8, pp. 4138–4146, 2012.

[16] E. P. DeBenedictis, "It's time to redefine Moore's law again," *Computer*, vol. 50, no. 2, pp. 72–75, Feb. 2017.

[17] P. Ye, T. Ernst, and M. V. Khare, "The last silicon transistor: Nanosheet devices could be the final evolutionary step for Moore's law," *IEEE Spectr.*, vol. 56, no. 8, pp. 30–35, Aug. 2019.

[18] M. Lapedus and E. Sperling. (2020). *Making Chips at 3 nm and Beyond*. [Online]. Available: https://semiengineering.com/making-chips-at-3nm-and-beyond/

[19] A. Perdomo-Ortiz, M. Benedetti, J. Realpe-Gómez, and R. Biswas, "Opportunities and challenges for quantum-assisted machine learning in near-term quantum computers," *Quantum Sci. Technol.*, vol. 3, no. 3, Jun. 2018, Art. no. 030502.

[20] M. A. Nahmias, T. F. de Lima, A. N. Tait, H.-T. Peng, B. J. Shastri, and P. R. Prucnal, "Photonic multiply-accumulate operations for neural networks," *IEEE J. Sel. Topics Quantum Electron.*, vol. 26, no. 1, pp. 1–18, Jan. 2020.

[21] Z. Zhang and A. Z. Kouzani, "Implementation of DNNs on IoT devices," *Neural Comput. Appl.*, vol. 32, no. 5, pp. 1327–1356, Mar. 2020.

[22] A. Shawahna, S. M. Sait, and A. El-Maleh, "FPGA-based accelerators of deep learning networks for learning and classification: A review," *IEEE Access*, vol. 7, pp. 7823–7859, 2019.

[23] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, "A survey of FPGA-based neural network accelerator," 2017, *arXiv:1712.08934*. [Online]. Available: http://arxiv.org/abs/1712.08934

[24] E. Wang *et al.*, "Deep neural network approximation for custom hardware: Where we've been, where we're going," *ACM Comput. Surv.*, vol. 52, no. 2, May 2019, Art. no. 40.

[25] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[26] A. Vaswani *et al.*, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 5998–6008.

[27] A. I. Miller, *Ian Goodfellow's Generative Adversarial Networks: AI Learns to Imagine*. Cambridge, MA, USA: MIT Press, 2020, pp. 87–98.

[28] Z. Liu and J. Zhou, *Introduction to Graph Neural Networks*, vol. 14, no. 2. San Rafael, CA, USA: Morgan & Claypool, 2020.

[29] 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.

[30] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *Proc. 27th Int. Conf. Mach. Learn. (ICML)*, 2010, pp. 807–814.

[31] Z. Qiumei, T. Dan, and W. Fenghua, "Improved convolutional neural network based on fast exponentially linear unit activation function," *IEEE Access*, vol. 7, pp. 151359–151367, 2019.

[32] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation functions: Comparison of trends in practice and research for deep learning," 2018, *arXiv:1811.03378*. [Online]. Available: http://arxiv.org/abs/1811.03378

[33] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 1–9.

[34] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[35] 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.

[36] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50× fewer parameters and <0.5 MB model size," 2016, *arXiv:1602.07360*. [Online]. Available: https://arxiv.org/abs/1602.07360

[37] H. Zhang *et al.*, "ResNeSt: Split-attention networks," 2020, *arXiv:2004.08955*. [Online]. Available: http://arxiv.org/abs/2004.08955

[38] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YOLOv4: Optimal speed and accuracy of object detection," 2020, *arXiv:2004.10934*. [Online]. Available: http://arxiv.org/abs/2004.10934

[39] A. Tao, K. Sapra, and B. Catanzaro, "Hierarchical multi-scale attention for semantic segmentation," 2020, *arXiv:2005.10821*. [Online]. Available: http://arxiv.org/abs/2005.10821

[40] P. Musikawan, K. Sunat, Y. Kongsorot, P. Horata, and S. Chiewchanwattana, "Parallelized metaheuristic-ensemble of heterogeneous feedforward neural networks for regression problems," *IEEE Access*, vol. 7, pp. 26909–26932, 2019.

[41] Y. Huang, W. Wang, and L. Wang, "Video super-resolution via bidirectional recurrent convolutional networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 4, pp. 1015–1028, Apr. 2018.

[42] R. Zhang, Z. Yuan, and X. Shao, "A new combined CNN-RNN model for sector stock price analysis," in *Proc. IEEE 42nd Annu. Comput. Softw. Appl. Conf.*, Jul. 2018, pp. 546–551.

[43] A. Paul, D. Jha, R. Al-Bahrani, W.-K. Liao, A. Choudhary, and A. Agrawal, "CheMixNet: Mixed DNN architectures for predicting chemical properties using multiple molecular representations," 2018, *arXiv:1811.08283*. [Online]. Available: http://arxiv.org/abs/1811.08283

[44] A. Howard *et al.*, "Searching for MobileNetV3," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 1314–1324.

[45] C. Rudin, "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead," *Nature Mach. Intell.*, vol. 1, no. 5, pp. 206–215, May 2019.

[46] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv:1611.01578*, 2016. [Online]. Available: https://arxiv.org/abs/1611.01578

[47] B. L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, "Model compression and hardware acceleration for neural networks: A comprehensive survey," *Proc. IEEE*, vol. 108, no. 4, pp. 485–532, Apr. 2020.

[48] Y. Weng, T. Zhou, L. Liu, and C. Xia, "Automatic convolutional neural architecture search for image classification under different scenes," *IEEE Access*, vol. 7, pp. 38495–38506, 2019.

[49] M. Tan and Q. V. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *Proc. 36th Int. Conf. Mach. Learn. (ICML)*, Jun. 2019, pp. 10691–10700.

[50] S. Ruder, "An overview of gradient descent optimization algorithms," 2016, *arXiv:1609.04747*. [Online]. Available: http://arxiv.org/abs/1609.04747

[51] M. Horowitz, "Computing's energy problem (and what we can do about it)," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2014, pp. 10–14.

[52] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural network," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 1135–1143.

[53] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," 2016, *arXiv:1611.06440*. [Online]. Available: http://arxiv.org/abs/1611.06440

[54] B.-C. Lai, J.-W. Pan, and C.-Y. Lin, "Enhancing utilization of SIMD-like accelerator for sparse convolutional neural networks," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 5, pp. 1218–1222, May 2019.

[55] X. Zeng *et al.*, "Addressing irregularity in sparse neural networks through a cooperative software/hardware approach," *IEEE Trans. Comput.*, vol. 69, no. 7, pp. 968–985, Jul. 2020.

[56] J. Wang *et al.*, "Performance of training sparse deep neural networks on GPUs," in *Proc. IEEE High Perform. Extreme Comput. Conf.*, Sep. 2019, pp. 1–5.

[57] J. L. McKinstry *et al.*, "Discovering low-precision networks close to full-precision networks for efficient embedded inference," 2018, *arXiv:1809.04191*. [Online]. Available: http://arxiv.org/abs/1809.04191

[58] Y. Yang, L. Deng, S. Wu, T. Yan, Y. Xie, and G. Li, "Training high-performance and large-scale deep neural networks with full 8-bit integers," *Neural Netw.*, vol. 125, pp. 70–82, May 2020.

[59] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 6869–6898, Jan. 2017.

[60] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients," 2016, *arXiv:1606.06160*. [Online]. Available: http://arxiv.org/abs/1606.06160

[61] F. Zhu *et al.*, "Towards unified INT8 training for convolutional neural network," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2020, pp. 1969–1979.

[62] J. Choi, B. Y. Kong, and I.-C. Park, "Retrain-less weight quantization for multiplier-less convolutional neural networks," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 67, no. 3, pp. 972–982, Mar. 2020.

[63] P.-C. Lin, M.-K. Sun, C. Kung, and T.-D. Chiueh, "FloatSD: A new weight representation and associated update method for efficient convolutional neural network training," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 9, no. 2, pp. 267–279, Jun. 2019.

[64] S. Jung *et al.*, "Learning to quantize deep networks by optimizing quantization intervals with task loss," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2019, pp. 4350–4359.

[65] K. Posch and J. Pilz, "Correlated parameters to accurately measure uncertainty in deep neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 3, pp. 1037–1051, Mar. 2021.

[66] B. O. Ayinde, T. Inanc, and J. M. Zurada, "On correlation of features extracted by deep neural networks," in *Proc. Int. Joint Conf. Neural Netw.*, Jul. 2019, pp. 1–8.

[67] X. Dai, H. Yin, and N. K. Jha, "NeST: A neural network synthesis tool based on a grow-and-prune paradigm," *IEEE Trans. Comput.*, vol. 68, no. 10, pp. 1487–1497, Oct. 2019.

[68] T. Bouwmans, N. S. Aybat, and E.-H. Zahzah, *Handbook of Robust Low-Rank and Sparse Matrix Decomposition: Applications in Image and Video Processing*. Boca Raton, FL, USA: CRC Press, 2016.

[69] S. Swaminathan, D. Garg, R. Kannan, and F. Andres, "Sparse low rank factorization for deep neural network compression," *Neurocomputing*, vol. 398, pp. 185–196, Jul. 2020.

[70] P. Wang, Q. Hu, Z. Fang, C. Zhao, and J. Cheng, "DeepSearch: A fast image search framework for mobile devices," *ACM Trans. Multimedia Comput., Commun., Appl.*, vol. 14, no. 1, pp. 1–22, Jan. 2018.

[71] J.-T. Chien and Y.-T. Bao, "Tensor-factorized neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 5, pp. 1998–2011, May 2018.

[72] Q. Zhang, L. T. Yang, Z. Chen, and P. Li, "An improved deep computation model based on canonical polyadic decomposition," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 48, no. 10, pp. 1657–1666, Oct. 2018.

[73] T. Fukuda, M. Suzuki, G. Kurata, S. Thomas, J. Cui, and B. Ramabhadran, "Efficient knowledge distillation from an ensemble of teachers," in *Proc. Interspeech*, 2017, pp. 3697–3701.

[74] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv:1503.02531*, 2015. [Online]. Available: https://arxiv.org/abs/1503.02531

[75] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," in *Proc. 4th Int. Conf. Learn. Represent. (ICLR)*, 2016, pp. 1–14.

[76] S. Wiedemann, K.-R. Müller, and W. Samek, "Compact and computationally efficient representation of deep neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 3, pp. 772–785, Mar. 2020.

[77] S. Moon, Y. Byun, J. Park, S. Lee, and Y. Lee, "Memory-reduced network stacking for edge-level CNN architecture with structured weight pruning," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 9, no. 4, pp. 735–746, Dec. 2019.

[78] T. Abtahi, C. Shea, A. Kulkarni, and T. Mohsenin, "Accelerating convolutional neural network with FFT on embedded hardware," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 9, pp. 1737–1749, Sep. 2018.

[79] Y. Zhao, D. Wang, L. Wang, and P. Liu, "A faster algorithm for reducing the computational complexity of convolutional neural networks," *Algorithms*, vol. 11, no. 10, p. 159, Oct. 2018.

[80] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jul. 2017, pp. 1251–1258.

[81] L. Kaiser, A. N. Gomez, and F. Chollet, "Depthwise separable convolutions for neural machine translation," 2017, *arXiv:1706.03059*. [Online]. Available: http://arxiv.org/abs/1706.03059

[82] D. Efnusheva, A. Cholakoska, and A. Tentov, "A survey of different approaches for overcoming the processor–memory bottleneck," *Int. J. Comput. Sci. Inf. Technol.*, vol. 9, no. 2, pp. 151–163, Apr. 2017.

[83] *Intel Architecture Instruction Set Extensions Programming Reference*, Intel Corp., Santa Carla, CA, USA, 2012.

[84] J. Huang *et al.*, "A parallel optimization of the fast algorithm of convolution neural network on CPU," in *Proc. 10th Int. Conf. Measuring Technol. Mechatronics Autom.*, Feb. 2018, pp. 5–9.

[85] N. G. Greeneltch and J. Xu, "Maximize TensorFlow performance on CPU: Considerations and recommendations for inference workloads," Intel Corp., Santa Carla, CA, USA, Tech. Rep., 2019. [Online]. Available: https://software.intel.com/content/www/us/en/develop/articles/maximize-tensorflow-performance-on-cpu-considerations-and-recommendations-for-inference.html

[86] Z. Li, J. Eichel, A. Mishra, A. Achkar, and K. Naik, "A CPU-based algorithm for traffic optimization based on sparse convolutional neural networks," in *Proc. IEEE 30th Can. Conf. Electr. Comput. Eng.*, Apr. 2017, pp. 1–5.

[87] D. D. Kalamkar *et al.*, "Training Google neural machine translation on an Intel CPU cluster," in *Proc. IEEE Int. Conf. Cluster Comput.*, Sep. 2019, pp. 1–10.

[88] K. Aida-Zade, E. Mustafayev, and S. Rustamov, "Comparison of deep learning in neural networks on CPU and GPU-based frameworks," in *Proc. IEEE 11th Int. Conf. Appl. Inf. Commun. Technol.*, Sep. 2017, pp. 1–4.

[89] H. Kim, H. Nam, W. Jung, and J. Lee, "Performance analysis of CNN frameworks for GPUs," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, Apr. 2017, pp. 55–64.

[90] P. Sun, W. Feng, R. Han, S. Yan, and Y. Wen, "Optimizing network performance for distributed DNN training on GPU clusters: ImageNet/AlexNet training in 1.5 minutes," 2019, *arXiv:1902.06855*. [Online]. Available: http://arxiv.org/abs/1902.06855

[91] *NVIDIA A100 Tensor Core GPU Architecture*, NVIDIA Corp., Santa Clara, CA, USA, 2020. [Online]. Available: https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf

[92] N. Jouppi, C. Young, N. Patil, and D. Patterson, "Motivation for and evaluation of the first tensor processing unit," *IEEE Micro*, vol. 38, no. 3, pp. 10–19, May 2018.

[93] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proc. 44th Annu. Int. Symp. Comput. Archit.*, no. 2, Jun. 2017, pp. 1–12.

[94] Y. E. Wang, G.-Y. Wei, and D. Brooks, "Benchmarking TPU, GPU, and CPU platforms for deep learning," 2019, *arXiv:1907.10701*. [Online]. Available: http://arxiv.org/abs/1907.10701

[95] E. Medina and E. Dagan, "Habana labs purpose-built AI inference and training processor architectures: Scaling AI training systems using standard Ethernet with Gaudi processor," *IEEE Micro*, vol. 40, no. 2, pp. 17–24, Mar. 2020.

[96] M. James, M. Tom, P. Groeneveld, and V. Kibardin, "ISPD 2020 physical mapping of neural networks on a wafer-scale deep learning accelerator," in *Proc. Int. Symp. Phys. Design*, Mar. 2020, pp. 145–149.

[97] T. Chen *et al.*, "DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," in *Proc. Int. Conf. Archit. Support Program. Lang. Oper. Syst. (ASPLOS)*. New York, NY, USA: Association for Computing Machinery, 2014, pp. 269–283.

[98] Y. Chen *et al.*, "DaDianNao: A machine-learning supercomputer," in *Proc. 47th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Dec. 2014, pp. 609–622.

[99] Z. Du *et al.*, "ShiDianNao: Shifting vision processing closer to the sensor," in *Proc. 42nd Annu. Int. Symp. Comput. Archit.*, Jun. 2015, pp. 92–104.

[100] D. Liu *et al.*, "PuDianNao: A polyvalent machine learning accelerator," *ACM SIGPLAN Notices*, vol. 50, no. 4, pp. 369–381, May 2015.

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

[102] S. Han *et al.*, "Deep compression and EIE: Efficient inference engine on compressed deep neural network," in *Proc. IEEE Hot Chips 28 Symp. (HCS)*, Aug. 2016, vol. 44, no. 3, pp. 243–254.

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

[104] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis, "TETRIS: Scalable and efficient neural network acceleration with 3D memory," in *Proc. 22nd Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, 2017, pp. 751–764.

[105] Y.-H. Chen, T.-J. Yang, J. S. Emer, and V. Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 9, no. 2, pp. 292–308, Jun. 2019.

[106] P. Meloni, G. Deriu, F. Conti, I. Loi, L. Raffo, and L. Benini, "A high-efficiency runtime reconfigurable IP for CNN acceleration on a mid-range all-programmable SoC," in *Proc. Int. Conf. ReConFigurable Comput. FPGAs*, Nov. 2016, pp. 1–8.

[107] X. Wei *et al.*, "Automated systolic array architecture synthesis for high throughput CNN inference on FPGAs," in *Proc. 54th Annu. Design Autom. Conf.*, Jun. 2017, pp. 1–6.

[108] R. DiCecco, L. Sun, and P. Chow, "FPGA-based training of convolutional neural networks with a reduced precision floating-point library," in *Proc. Int. Conf. Field Program. Technol. (ICFPT)*, Dec. 2017, pp. 239–242.

[109] J. Zhang and J. Li, "Improving the performance of OpenCL-based FPGA accelerator for convolutional neural network," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2017, pp. 25–34.

[110] Y. Ma, Y. Cao, S. Vrudhula, and J.-S. Seo, "Optimizing loop operation and dataflow in FPGA acceleration of deep convolutional neural networks," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays (FPGA)*. New York, NY, USA: Association for Computing Machinery, Feb. 2017, pp. 45–54.

[111] Y.-K. Choi and J. Cong, "HLS-based optimization and design space exploration for applications with variable loop bounds," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, Nov. 2018, pp. 1–8.

[112] J. Chen, L. Liu, Y. Liu, and X. Zeng, "A learning framework for n-bit quantized neural networks toward FPGAs," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 3, pp. 1067–1081, Mar. 2021.

[113] T. Wang, T. Geng, A. Li, X. Jin, and M. Herbordt, "FPDeep: Scalable acceleration of CNN training on deeply-pipelined FPGA clusters," *IEEE Trans. Comput.*, vol. 69, no. 8, pp. 1143–1158, Aug. 2020.

[114] U. Aydonat, S. O'Connell, D. Capalija, A. C. Ling, and G. R. Chiu, "An OpenCL™ deep learning accelerator on Arria 10," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2017, pp. 55–64.

[115] C. Zhang and V. Prasanna, "Frequency domain acceleration of convolutional neural networks on CPU-FPGA shared memory system," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2017, pp. 35–44.

[116] Y. Liang, L. Lu, Q. Xiao, and S. Yan, "Evaluating fast algorithms for convolutional neural networks on FPGAs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 4, pp. 857–870, Apr. 2020.

[117] A. Shafiee *et al.*, "ISAAC: A convolutional neural network accelerator with *in-situ* analog arithmetic in crossbars," *ACM SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 14–26, 2016.

[118] O. Krestinskaya, K. N. Salama, and A. P. James, "Learning in memristive neural network architectures using analog backpropagation circuits," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 66, no. 2, pp. 719–732, Feb. 2019.

[119] P. Chi *et al.*, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," *ACM SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 27–39, Jun. 2016.

[120] M. Cheng *et al.*, "TIME: A training-in-memory architecture for RRAM-based deep neural networks," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 5, pp. 834–847, May 2019.

[121] Z. Zhu *et al.*, "A configurable multi-precision CNN computing framework based on single bit RRAM," in *Proc. 56th ACM/IEEE Annu. Design Autom. Conf.*, Jun. 2019, pp. 1–6.

[122] C.-X. Xue *et al.*, "A 22 nm 2 Mb ReRAM compute-in-memory macro with 121-28TOPS/W for multibit MAC computing for tiny AI edge devices," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2020, pp. 244–246.

[123] Q. Liu *et al.*, "A fully integrated analog ReRAM based 78.4TOPS/W compute-in-memory chip with fully parallel MAC computing," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2020, pp. 500–502.

[124] Y. Shi, S. Oh, Z. Huang, X. Lu, S. H. Kang, and D. Kuzum, "Performance prospects of deeply scaled spin-transfer torque magnetic random-access memory for in-memory computing," *IEEE Electron Device Lett.*, vol. 41, no. 7, pp. 1126–1129, Jul. 2020.
[125] Y. Pan *et al.*, "A multilevel cell STT-MRAM-based computing in-memory accelerator for binary convolutional neural network," *IEEE Trans. Magn.*, vol. 54, no. 11, pp. 1–5, Nov. 2018.
[126] S. Angizi, Z. He, A. Awad, and D. Fan, "MRIMA: An MRAM-based in-memory accelerator," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 5, pp. 1123–1136, May 2020.
[127] Z. Li *et al.*, "HEIF: Highly efficient stochastic computing-based inference framework for deep neural networks," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 8, pp. 1543–1556, Aug. 2019.
[128] F. Akopyan *et al.*, "TrueNorth: Design and tool flow of a 65 mW 1 million neuron programmable neurosynaptic chip," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 34, no. 10, pp. 1537–1557, Oct. 2015.
[129] H.-P. Cheng, W. Wen, C. Wu, S. Li, H. H. Li, and Y. Chen, "Understanding the design of IBM neurosynaptic system and its tradeoffs: A user perspective," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2017, pp. 139–144.
[130] M. Davies *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, Jan. 2018.
[131] N. Imam and T. A. Cleland, "Rapid online learning and robust recall in a neuromorphic olfactory circuit," *Nature Mach. Intell.*, vol. 2, no. 3, pp. 181–191, Mar. 2020.
[132] C. Frenkel, M. Lefebvre, J. Legat, and D. Bol, "A 0.086-mm^2 12.7-pJ/SOP 64 k-synapse 256-neuron online-learning digital spiking neuromorphic processor in 28-nm CMOS," *IEEE Trans. Biomed. Circuits Syst.*, vol. 13, no. 1, pp. 145–158, Feb. 2019.
[133] R. Tapiador-Morales, A. Linares-Barranco, A. Jimenez-Fernandez, and G. Jimenez-Moreno, "Neuromorphic LIF row-by-row multiconvolution processor for FPGA," *IEEE Trans. Biomed. Circuits Syst.*, vol. 13, no. 1, pp. 159–169, Feb. 2019.
[134] C. Frenkel, J. Legat, and D. Bol, "MorphIC: A 65-nm 738k-synapse/mm^2 quad-core binary-weight digital neuromorphic processor with stochastic spike-driven online learning," *IEEE Trans. Biomed. Circuits Syst.*, vol. 13, no. 5, pp. 999–1010, Oct. 2019.
[135] S. Moradi, N. Qiao, F. Stefanini, and G. Indiveri, "A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (DYNAPs)," *IEEE Trans. Biomed. Circuits Syst.*, vol. 12, no. 1, pp. 106–122, Feb. 2018.
[136] L. Deng *et al.*, "Tianjic: A unified and scalable chip bridging spike-based and continuous neural computation," *IEEE J. Solid-State Circuits*, vol. 55, no. 8, pp. 2228–2246, Aug. 2020.
[137] J. Pei *et al.*, "Towards artificial general intelligence with hybrid Tianjic chip architecture," *Nature*, vol. 572, no. 7767, pp. 106–111, Aug. 2019.
[138] A. Waterman, Y. Lee, D. Patterson, and K. Asanovic, "The RISC-V instruction set manual, volume I: User-level ISA," RISC-V Int., San Francisco, CA, USA, Tech. Rep., 2014, Version 2. [Online]. Available: https://riscv.org/technical/specifications/
[139] K. Asanovic *et al.*, "The rocket chip generator," Dept. EECS, Univ. California, Berkeley, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2016-17, 2016.
[140] C. Celio, P.-F. Chiu, B. Nikolic, D. A. Patterson, and K. Asanovic, "BOOMv2: An open-source out-of-order RISC-V core," in *Proc. 1st Workshop Comput. Archit. Res. RISC-V*, 2017, pp. 1–8.
[141] A. Amid *et al.*, "Chipyard: Integrated design, simulation, and implementation framework for custom SoCs," *IEEE Micro*, vol. 40, no. 4, pp. 10–21, Jul. 2020.
[142] A. Garofalo, M. Rusci, F. Conti, D. Rossi, and L. Benini, "PULP-NN: Accelerating quantized neural networks on parallel ultra-low-power RISC-V processors," *Phil. Trans. Roy. Soc. A, Math., Phys. Eng. Sci.*, vol. 378, no. 2164, Feb. 2020, Art. no. 20190155.
[143] W. Lou, C. Wang, L. Gong, and X. Zhou, "RV-CNN: Flexible and efficient instruction set for CNNs based on RISC-V processors," in *Proc. Int. Symp. Adv. Parallel Process. Technol.* Tianjin, China: Springer, 2019, pp. 3–14.
[144] H. Genc *et al.*, "Gemmini: An agile systolic array generator enabling systematic evaluations of deep-learning architectures," 2019, *arXiv:1911.09925*. [Online]. Available: http://arxiv.org/abs/1911.09925
[145] F. Farshchi, Q. Huang, and H. Yun, "Integrating NVIDIA deep learning accelerator (NVDLA) with RISC-C SoC on FireSim," 2019, *arXiv:1903.06495*. [Online]. Available: http://arxiv.org/abs/1903.06495
[146] A. Mirhoseini *et al.*, "Chip placement with deep reinforcement learning," 2020, *arXiv:2004.10746*. [Online]. Available: http://arxiv.org/abs/2004.10746

Kh Shahriya Zaman (Graduate Student Member, IEEE) is currently pursuing the Ph.D. degree with the Department of Electrical, Electronic and Systems Engineering, Universiti Kebangsaan Malaysia, Bangi, Malaysia, where he is involved in developing specialized hardware architectures for artificial neural networks.

His current research interests include deep learning, hardware accelerators for deep learning, and custom hardware architectures for low-power devices.

Mamun Bin Ibne Reaz (Senior Member, IEEE) is currently a Professor with the Department of Electrical, Electronic and Systems Engineering, Universiti Kebangsaan Malaysia, Bangi, Malaysia. He is involved in teaching, research, and industrial consultation. He is also a Senior Associate with The Abdus Salam International Center for Theoretical Physics (ICTP), Trieste, Italy. He has published more than 350 research articles in design automation and integrated circuit (IC) design for biomedical applications.

Sawal Hamid Md Ali (Senior Member, IEEE) is currently an Associate Professor with the Department of Electrical, Electronics, and System Engineering, Universiti Kebangsaan Malaysia, Bangi, Malaysia. He has authored and coauthored more than 80 publications and has been granted three patents. He is involved in mixed-signal systems, system-on-chips, wearables, and embedded systems. He is the Research Group Leader of the Micro and Nano Electronic Systems Group.

Dr. Md Ali is a member of the IEEE Circuit and Systems Society.

Ahmad Ashrif A Bakar (Senior Member, IEEE) is currently an Associate Professor with the Department of Electrical, Electronics, and System Engineering, Universiti Kebangsaan Malaysia, Bangi, Malaysia. He is actively involved in the Optical Society of America and Fiber Optic Association Inc., Santa Monica, CA, USA. His research works include optical sensors, plasmonic waveguide sensors, polymeric electrooptic modulator waveguide, interferometer, evanescent field sensors, and devices based on nanoparticles and nanostructures.

Muhammad Enamul Hoque Chowdhury (Senior Member, IEEE) received the Ph.D. degree from the University of Nottingham, Nottingham, U.K., in 2014.

He worked as a Post-Doctoral Research Fellow with the Sir Peter Mansfield Imaging Centre, University of Nottingham. He is currently working as an Assistant Professor with the Department of Electrical Engineering, Qatar University, Doha, Qatar. He is also running several QNRF grants and internal grants from Qatar University along with academic and government projects along with different national and international projects. He has worked as a Consultant for the projects entitled, Driver Distraction Management Using Sensor Data Cloud from 2013 to 2014 and Information Society Innovation Fund (ISIF) Asia. He has two patents and published around 80 peer-reviewed journal articles, conference papers, and four book chapters. His current research interests include biomedical instrumentation, signal processing, wearable sensors, medical image analysis, machine learning, and embedded system design.

Dr. Chowdhury received the ISIF Asia Community Choice Award 2013 for a project entitled, Design and Development of Precision Agriculture Information System for Bangladesh. He has recently won the COVID-19 Dataset Award for his contribution to the fight against COVID-19. He is also serving as an Associate Editor for IEEE ACCESS and a Topic Editor for *Frontiers in Neuroscience*.