IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 25 November 2022, accepted 11 December 2022, date of publication 15 December 2022,
date of current version 22 December 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3229767

SURVEY

Efficient Hardware Architectures for Accelerating Deep Neural Networks: Survey

PUDI DHILLESWARARAO[1], (Graduate Student Member, IEEE), SRINIVAS BOPPU[1], (Member, IEEE), M. SABARIMALAI MANIKANDAN[2], (Senior Member, IEEE), AND LINGA REDDY CENKERAMADDI[3], (Senior Member, IEEE)

[1]School of Electrical Sciences, Indian Institute of Technology Bhubaneswar, Bhubaneswar 752050, India
[2]Department of Electrical Engineering, Indian Institute of Technology Palakkad, Palakkad 678557, India
[3]Department of ICT, University of Agder, 4879 Grimstad, Norway

Corresponding author: Linga Reddy Cenkeramaddi (linga.cenkeramaddi@uia.no)

This work was supported in part by the Indo-Norwegian Collaboration in Autonomous Cyber-Physical Systems (INCAPS) of the International Partnerships for Excellent Education, Research and Innovation (INTPART) Program from the Research Council of Norway under Project 287918, and in part by the Seed Grant of IIT Bhubaneswar (TAML: Timing Analysis with Machine Learning) under Project SP088.

ABSTRACT In the modern-day era of technology, a paradigm shift has been witnessed in the areas involving applications of Artificial Intelligence (AI), Machine Learning (ML), and Deep Learning (DL). Specifically, Deep Neural Networks (DNNs) have emerged as a popular field of interest in most AI applications such as computer vision, image and video processing, robotics, etc. In the context of developed digital technologies and the availability of authentic data and data handling infrastructure, DNNs have been a credible choice for solving more complex real-life problems. The performance and accuracy of a DNN is a way better than human intelligence in certain situations. However, it is noteworthy that the DNN is computationally too cumbersome in terms of the resources and time to handle these computations. Furthermore, general-purpose architectures like CPUs have issues in handling such computationally intensive algorithms. Therefore, a lot of interest and efforts have been invested by the research fraternity in specialized hardware architectures such as Graphics Processing Unit (GPU), Field Programmable Gate Array (FPGA), Application Specific Integrated Circuit (ASIC), and Coarse Grained Reconfigurable Array (CGRA) in the context of effective implementation of computationally intensive algorithms. This paper brings forward the various research works on the development and deployment of DNNs using the aforementioned specialized hardware architectures and embedded AI accelerators. The review discusses the detailed description of the specialized hardware-based accelerators used in the training and/or inference of DNN. A comparative study based on factors like power, area, and throughput, is also made on the various accelerators discussed. Finally, future research and development directions, such as future trends in DNN implementation on specialized hardware accelerators, are discussed. This review article is intended to guide hardware architects to accelerate and improve the effectiveness of deep learning research.

INDEX TERMS Machine learning, field programmable gate array (FPGA), deep neural networks (DNN), deep learning (DL), application specific integrated circuits (ASIC), artificial intelligence (AI), central processing unit (CPU), graphics processing unit (GPU), hardware accelerators.

I. INTRODUCTION

Deep neural networks (DNNs), also known as deep learning, are a subset of the Artificial Intelligence (AI) discipline. The term AI was coined in 1956 by John McCarthy, who defined it as ''the science and engineering of making intelligent machines''. Machine learning is a broad topic of artificial intelligence that was first defined by Arthur Samuel in 1959 as the study of how computers may learn without being explicitly programmed. Machine Learning uses traditional

The associate editor coordinating the review of this manuscript and approving it for publication was Vivek Kumar Sehgal.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

techniques to perform tasks like classification, regression, and clustering. Deep learning is a subfield of machine learning that uses a multi-layered algorithm structure known as a neural network, which was developed mostly between 2006 and 2010. The relationship between deep learning, machine learning, and AI is illustrated in Fig. 1.

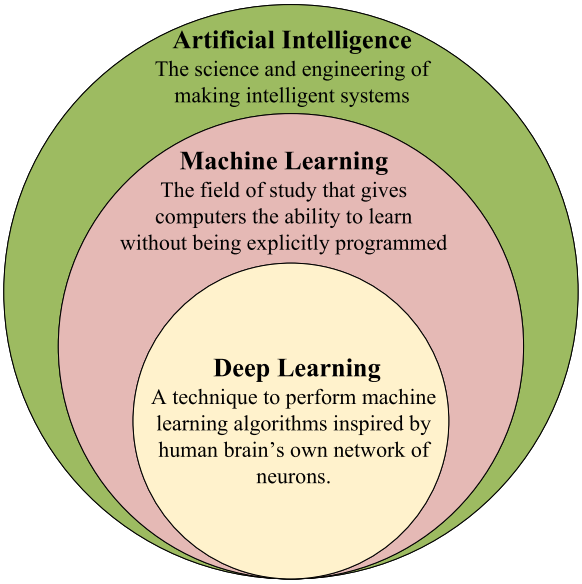


FIGURE 1. AI vs. Machine Learning vs. Deep Learning.

Nowadays, DNNs are used in many modern AI applications, including bioinformatics [60], natural language processing [147], image restoration [185], speech recognition [34], computer vision [194], machine translation [36], healthcare [43], finance [221], robotics [94], visual art processing [193], etc. Furthermore, the recent applications of DNN include aerospace and defence, automated driving, recommendation systems, and industrial automation [71], [86], [101], [215]. DNNs are also useful in a variety of applications, such as news aggregation and fraud detection [124], virtual assistants [61], chatbots [35], and customer relationship management systems [203]. In addition, DNNs have also been used to diagnose Covid-19 by classifying it based on different lung and chest imaging modalities [40].

DNNs contain many layers, and each layer is capable of detecting features at different levels. For instance, in pattern recognition, where the input is available in pixel form, the first layer of DNN extracts minor details of the image, such as curves and edges. The outputs of this first layer act as inputs to the second layer. The second layer extracts the image's primary details, such as squares and semi-circles. The outputs of the second layer act as inputs to the third layer. The third layer extracts the part of objects. Furthermore, the subsequent layer uses the previous layer's output and extracts more aspects of the objects. As the number of layers increases, the DNN extracts increasingly complicated features and complete objects [73]. DNNs provide superior accuracy and performance at the cost of high computational complexity. For instance, AlexNet [130] takes 1.4 Giga Operations Per Second (GOPS) to process a single image of size 224×224 with a top-1 accuracy of 61%, while ResNet-152 [108] takes 22.6 GOPS with a top-1 accuracy of 79.3%. DNN's superior accuracy and performance are due to its capacity to extract more complex high-level features, such as objects and facial structures, from raw input data.

DNNs are computationally expensive and need lots of computational resources and memory for training and inference. CPUs inherently support a limited number of parallel workloads, though they can context switch with hyper-threading. They are not sequential in nature. CPUs may have more resources than their counter architectures (like GPUs or FPGAs). CPUs have a limited number of registers to support concurrent threads. But they may have higher cache sizes, larger branch control logic, and higher on-chip bandwidth than GPUs. However, the limited number of cores on the CPU limits its ability to process large amounts of data in parallel, which is required for DNN acceleration. Although CPUs dominate the IoT industry in DNN inference on low-power edge devices, they struggle to realize complex DNNs. Therefore, specialized hardware designs are required for the acceleration of DNNs. DNNs can be implemented using customized hardware accelerators instead of a CPU. The heterogeneous computing platforms viz. Field Programmable Gate Array (FPGA), Application-Specific Integrated Circuits (ASIC), and Graphical Processing Units (GPU) are widely used to accelerate DNNs. The specialized hardware-based DNN accelerators can be categorized into two classes: the first class of accelerators efficiently implements the computational primitives, such as convolutional operations, fully connected operations, etc., for the DNNs [85], [175] and the second class of DNN accelerators efficiently optimize the data movement and memory access [56], [177]. These two generations of specialized hardware-based DNN accelerators improve the speed and energy efficiency of running DNNs. There are two ways to improve the performance of the DNN acceleration. The first method is optimizing the DNN algorithm, and the second is optimizing the hardware architecture. Therefore, we need to co-design the algorithm and the hardware to achieve superior performance.

Because of their high throughput and memory bandwidth, GPUs are one of the most often employed hardware accelerators for improving inference and training processes in DNNs [218]. In floating-point matrix-based calculations, GPU-based hardware accelerators are extremely efficient [205]. GPU-based hardware accelerators, on the other hand, consume a lot of power. ASIC and FPGA-based hardware accelerators have limited computational and memory resources compared to GPU-based accelerators. Nevertheless, they can achieve a moderate performance level while using less energy [153]. ASIC-based DNN accelerators provide superior performance compared to GPU and FPGA counterparts at the cost of reconfigurability. However, ASIC-based accelerators have some limitations, including the high cost of development, long time to market, inflexibility, etc [77], [103]. FPGA-based accelerators can be used as

an alternative to ASIC-based accelerators, and they can provide superior performance at an affordable cost with reconfigurability and low power dissipation [213]. FPGA, ASIC, and GPU-based AI accelerators have been the subject of numerous research [97], [150], [154], [155], [158], [210]. This survey, however, also looks at various embedded AI accelerators for DNN acceleration.

This survey supplements the existing work and contributes towards providing the complete background on DNN acceleration using various specialized hardware architectures. The contributions of this survey can be summarized as follows:

1) The survey discusses the various research works carried out on the development and deployment of DNN using FPGA-based accelerators.
2) The survey covers the work done in ASIC-based AI accelerators in the last decade, from 2012 to 2022.
3) The survey describes the various GPU-based DNN accelerators.
4) The survey provides a comprehensive overview of CGRA-based accelerators for DNN implementation.
5) The survey covers the research works carried out on the implementation of DNNs on edge using embedded AI accelerators.
6) The survey provides a comparative study of existing hardware architectures: FPGAs, GPUs, ASICs, and embedded AI accelerators.
7) The survey highlights the future research trends in DNN acceleration on specialized hardware architectures, including FPGA, ASIC, GPU, CGRA, and Edge AI accelerators.

A. SCOPE OF THE SURVEY

This paper focuses on research trends in FPGA, ASIC, and GPU-based accelerators for implementing DNNs. We have also briefly discussed the current trends in Arm-based machine learning processors and embedded edge AI accelerators. The review categorizes the FPGA-based accelerator into three categories and briefly discusses the key features of the accelerators, including the frameworks available. The three categories include accelerators for a specific application such as speech recognition, object detection, natural language processing, etc., accelerators for a specific algorithm such as CNN, RNN, etc., and accelerator frameworks with hardware templates. Furthermore, ASIC-based accelerators are categorized into three types: ALU-based accelerators, dataflow-based accelerators, and sparsity-based accelerators. A comparative study of these hardware accelerators based on performance metrics like power, throughput, and area has been presented. The review also focuses on the mapping frameworks available for these accelerators and briefly discusses the implementation details. In addition, the recent research contributions in Arm-based machine learning processors, a few embedded AI hardware accelerators, and CGRA-based accelerators are discussed and compared in terms of their cores, performance, power, availability of Software Development Kits (SDKs), and supported frameworks.

This survey is different and unique with respect to many existing papers in this area in the following ways. Few studies [44], [97], [126], [154], [210] focused only on the developments of FPGA-based accelerators. Few other studies [55], [136], [150], [158] have presented the details of ASIC-based accelerators. Some research reviews [48], [200], [201] have explored both FPGA and ASIC-based accelerators. Very limited studies [178], [201] have dealt with the progress of GPU-based accelerators. On the other hand, studies on embedded AI and CGRA-based accelerators haven't been explored much. Many of these reviews do not mention the compiler/mapping frameworks and SDKs available for these accelerators, making it difficult for someone to choose the appropriate accelerator. This review, therefore, aims to bring a comprehensive study of all the aforementioned hardware accelerators in the context of the implementation of DNNs. Furthermore, this survey uniquely classifies the FPGA- and ASIC-based accelerators and briefly discusses the key architectural features and the available compiler or mapping frameworks. Accelerators for each category are summarized and compared. A comprehensive survey of GPU-based accelerators by Nvidia is also presented. The need for edge AI computing is emphasized and state-of-the-art embedded AI accelerators, including Arm-based accelerators, are also discussed and compared. This survey also briefly discusses the recent developments in tinyML. Table 1 compares this survey paper with recently published review articles on DNN implementation using specialized hardware architectures. Researchers in the fields of artificial intelligence, system design, and hardware architecture are expected to benefit from this survey.

B. ORGANIZATION

This paper is organized as follows: Section II provides a brief overview of neural networks and DNNs, including the basic architecture of hardware for DNN acceleration. Section III describes various architectures implemented on the FPGA platform for DNN acceleration. Section IV describes various ASIC-based accelerator architectures for DNN acceleration. Section V shows a detailed review of GPU-based accelerators for the acceleration of DNN. Section VI discusses various CGRA-based accelerator architectures for DNN acceleration. Section VII discusses in detail the embedded edge AI accelerators for DNN acceleration. Section VIII provides the comparisons between the various hardware architectures used for the DNN acceleration. Section IX provides the future research directions of various hardware architectures for DNN acceleration. Finally, the conclusion of this review is presented in Section X.

II. BACKGROUND

A. NEURAL NETWORKS

A Neural Network (NN) is a computational model inspired by biological neural networks. It is also known as an Artificial Neural Network (ANN). An ANN comprises hundreds or

TABLE 1. Comparison among state-of-the-art surveys.

Paper	Year	Summary	Scope									
			FPGA-based accelerators	ASIC-based accelerators	GPU-based accelerators	Embedded AI accelerators	CGRA-based accelerators	Frameworks	Dataflow	Sparce DNN	Tiny ML	Applications
Sze et al. [200]	2017	Provided a comprehensive survey of various techniques to efficiently process deep neural networks on hardware.	✓	✓	✗	✗	✗	✓	✓	✓	✗	✓
Wang et al. [210]	2018	Provided a comprehensive survey for the acceleration of neural networks on FPGA and also discussed about advantages and disadvantages of FPGA-based accelerators	✓	✗	✗	✗	✗	✓	✗	✗	✗	✓
Guo et al. [97]	2019	Provided a comprehensive review of FPGA-based neural network inference accelerators.	✓	✗	✗	✗	✗	✓	✗	✓	✗	✓
Blaiech et al. [44]	2019	Reviewed the techniques and frameworks for the acceleration of deep learning algorithms on FPGA.	✓	✗	✗	✗	✗	✓	✗	✗	✗	✓
Mittal et al. [154]	2020	Provided detailed survey of techniques for implementing and optimizing CNN algorithms on FPGA.	✓	✗	✗	✗	✗	✓	✓	✓	✗	✓
Chen et al. [55]	2020	Summarized the recent advances in DNN accelerator, and also provided the future trends of DNN accelerators.	✗	✓	✗	✗	✗	✗	✓	✓	✗	✓
Capra et al. [48]	2020	Provided a comprehensive survey of state-of-the-art architectures for DNN acceleration.	✓	✓	✓	✗	✗	✓	✓	✓	✗	✗
Lee et al. [136]	2021	Provided a detailed survey of energy-efficient DNN processing on hardware and also summarized the key techniques of energy-efficient DNN training on ASICs	✗	✓	✗	✗	✗	✗	✓	✓	✗	✗
Talib et al. [201]	2021	Provided a systematic review of ASIC, FPGA, and GPU-based accelerators for AI and ML tools.	✓	✓	✓	✗	✗	✓	✗	✗	✗	✗
Reuther et al. [178]	2021	Summarized the current commercial accelerators that have been publicly announced with peak performance and power numbers.	✗	✗	✓	✗	✗	✗	✓	✗	✗	✓
Machupalli et al. [150]	2022	Provided a comprehensive review of optimization techniques used in the existing DNN accelerators	✗	✓	✗	✗	✗	✓	✓	✓	✗	✓
Moolchandani et al. [158]	2022	Provided a comprehensive survey of CNN accelerator architectures on custom hardware.	✗	✓	✗	✗	✗	✗	✓	✓	✗	✗
This survey	2022	Provided a detailed review of recent advancements in the area of DNN acceleration on specialized hardware architectures such as FPGA, ASIC, and GPU. Discussed the recent developments in embedded AI accelerators for edge environment and tinyML.	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

thousands of interconnected artificial neurons, also called processing units. Three or more interconnected layers are formed by these neurons. The input neurons are in the first layer. The input neurons receive external signals and pass them on to the subsequent layers, which eventually provide the final output data to the final output layer. The intermediate layers in the ANN are called as hidden layers. Fig. 2 depicts the architecture of a typical NN, which includes an input layer, an output layer, and two hidden layers.

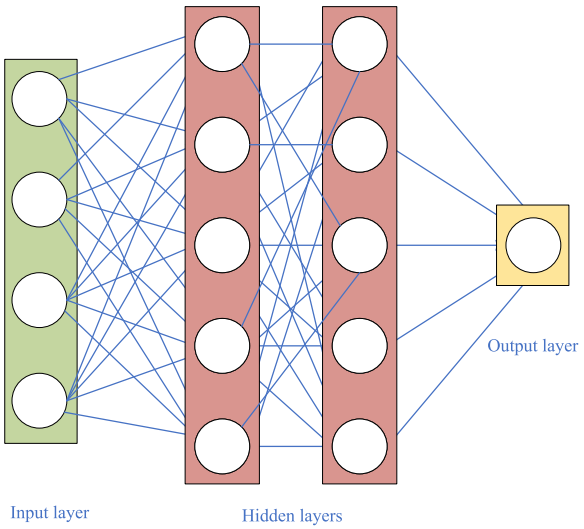


FIGURE 2. An architecture of NN.

In NN shown in Fig. 3, the input layer contains n inputs $(x_1, x_2, \ldots, x_n)$. The following layer (hidden layer) gets all n inputs from the input layer and generates the output y. These inputs are multiplied by the weight coefficients $(w_1, w_2, \ldots, w_n)$ and combined together with a bias value b for each neuron. A non-linear function $\sigma(.)$, also called as an activation function, is then used to calculate the neuron's output, see Eq. (1). In this scenario, the activation function causes a neuron to produce an output only if the input to it exceeds a specified threshold value. Common non-linear functions used in NN are Sigmoid, Rectified Linear Unit (ReLU), and Hyperbolic tangent. The graphical model and mathematical representation of artificial neuron is shown in Fig. 3 and Eq. (1), respectively.

$$y = \sigma\left(\sum_{n=1}^{N} x[n]w[n] + b\right) \tag{1}$$

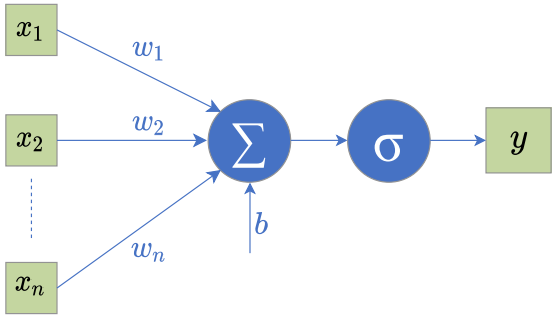


FIGURE 3. A single ANN neuron with its elements (inputs, weights, bias, summer, activation function, and output).

In neural networks, weights are initialized with some random values. However, during the training process, all

these weights get updated iteratively to predict the correct output. The weights are updated using the cost function, which is nothing more than the mean square error. The mathematical representation of mean square error is shown in Eq. (2). Here, *MSE* is mean squared error, n represents the number of input data points, y_i and $\hat{y}_i$ are true and predicted outputs, respectively. Once the neural network is trained, it may be used for classification problems.

$$MSE = \frac{1}{n}\sum_{i=1}(y_i - \hat{y}_i)^2 \tag{2}$$

B. DEEP NEURAL NETWORK (DNN)

The Deep Neural Network (DNN) is a type of neural network that has more than three hidden layers and is well-suited to complicated tasks [37]. In today's DNN, the typical number of layers used ranges from five to over a thousand. A DNN with N hidden layers is shown in Fig. 4. In DNNs, the model and its parameters are learned through an extensive training process.

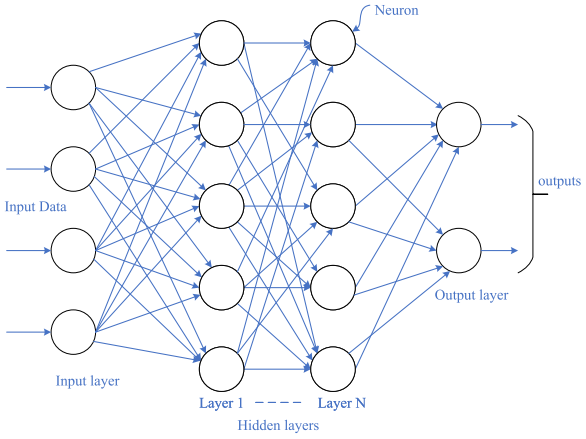


FIGURE 4. DNN with *N* hidden layers [157].

Training and inference are the two critical phases in accelerating a task using DNN. Specific tasks such as object detection, and pattern recognition etc. are part of the training, in which the DNN is taught to perform such specific tasks using available data. The known data is supplied to DNN throughout the training process, allowing the network to predict what the data represents. As a result, the prediction error is used to adjust the weights of the neurons. The weights are adjusted till the predictions are made with a considerable degree of accuracy. Backpropagation is a popular method for updating weights, as mentioned before in the training phase. DNN is ready to make predictions on fresh and unknown data once it has been fully trained. This stage is known as inference, and it involves testing the trained model with completely new and unknown data.

There are four types of deep learning approaches: supervised, unsupervised, reinforcement, and semi-supervised learning. The labeled data is used in supervised learning to train or model the network. The labeled data indicates that some input data has already been matched to the correct output. Unsupervised learning is another learning technique in which the network/model is trained using unlabeled data. The trained network generates the clusters or structures in the unlabeled data. Semi-supervised learning uses partially labeled data sets and it falls in between supervised and unsupervised learning approaches. Finally, reinforcement learning is a type of training that rewards positive behaviours while punishes undesirable ones. Reinforcement learning is bound to learn from its previous experience. The pictorial representation of the aforementioned deep learning approaches is shown in Fig. 5.

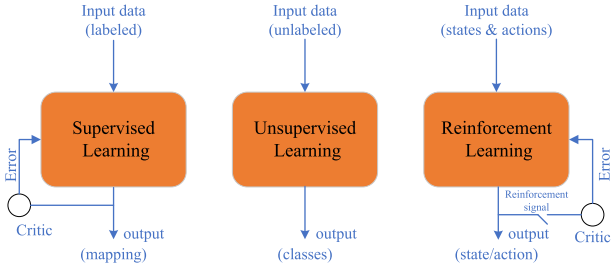


FIGURE 5. Deep learning approaches.

C. CONVOLUTIONAL NEURAL NETWORK (CNN)

Convolutional neural networks (CNNs) are a type of neural network which have been widely used for image recognition tasks. CNN is made up of several stages, each of which is referred to as a layer. Each layer extracts a feature from the data it receives. The identifying features get more sophisticated or complex as we proceed. CNN structure was first proposed by Fukushima in 1988 [87]. As shown in Fig. 6, a CNN consists of four layers: convolution, fully connected layer, pooling layer, and Rectified Linear Unit (ReLU) layer. Optionally, CNN might also have non-traditional layers such as dilated convolution layer [114] and deconvolution layer [52]. The CNN's overall design may be divided into two sections: feature learning and classification. Each layer of the CNN gets data from the layer before it as input and delivers its output to the following layer as input during the feature learning phase. The feature learning phase includes three types of layers: convolution, RELU, and pooling. At each node of the convolution layer, convolution operations on the input nodes detect features from the input feature maps. The output of the feature extraction phase's final layer is delivered to a fully connected network known as the classification layer. The following sections discuss each type of layer in the CNN in brief.

1) CONVOLUTION LAYER

The convolution layer is also known as the feature extraction layer since it extracts the features of images. The inputs and outputs of the convolution layer are defined as feature maps (FMs) which are organized in two-dimensional grids. The FM from the previous layers of the convolution layer is convolved with the filter coefficients. More than one input

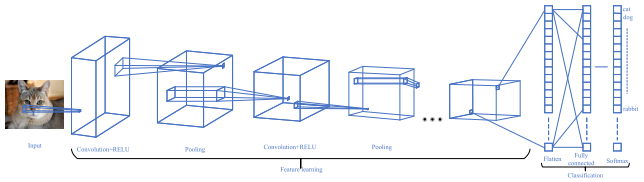


FIGURE 6. CNN architecture (adopted from [181]).

feature map can be paired with each output feature map. In a 2-D convolution operation between an input image matrix x (size $R \times C$) and a filter f (size $W \times L$), the convolution layer performs point-wise multiplication and addition of the corresponding pixels. The filter size is often smaller than the input matrix size. The filter multiplies the input matrix with the $W \times L$ sized block, accumulates the result, slides to the next block of the input matrix, and repeats the operation. The input matrix is processed one block at a time until it has processed all of the image's $R \times C$ elements. The 2-D convolution operation is given in Eq. (3) where $y(r, c)$ signifies one output pixel in the output matrix y, with each pixel's coordinates expressed as (r, c). The iterators over the filter's length (L) and width (W) are l and w, respectively, in Eq. (3). Finally, the resulting feature maps apply non-linear activation functions such as sigmoid, hyperbolic tangent, or rectified linear units.

$$y(r,c) = \sum_{w=0}^{W-1} \sum_{l=0}^{L-1} f(w,l)x(r+w-\left\lfloor \frac{W}{2} \right\rfloor, c+l-\left\lfloor \frac{L}{2} \right\rfloor) \tag{3}$$

2) POOLING LAYER

The pooling layer shrinks the spatial dimensions of the input image after convolution, thereby reducing the computation and number of parameters in the network. Pooling layers are also known as subsampling layers. In CNN, the pooling layer is used between two convolution layers. The MAX operation is used to resize each slice of the input image spatially, on which the pooling layers operate individually. A pooling layer with filters of size 2×2 is found in many CNN topologies. Over the four samples in the filter, the pooling operation, which is nothing but the MAX operation, is done. The operation yielding the maximum value is retained while discarding the other values [123]. It is noteworthy that additional operations like MIN operation and AVG operation can also be used in the pooling layer, particularly in some CNNs [197]. The MAX and AVG pooling operations for the filters of size 2×2 are shown in Fig. 7.

3) RECTIFIED LINEAR UNIT (ReLU) LAYER

In a CNN network, the ReLU layer is usually employed after the convolution and fully connected layers. The ReLU layer is generally used after the convolution and fully connected layers in the CNN network. By substituting all the negative valued outputs with 0, it introduces non-linearity into the CNN. Because of its computational simplicity, sparsity, and

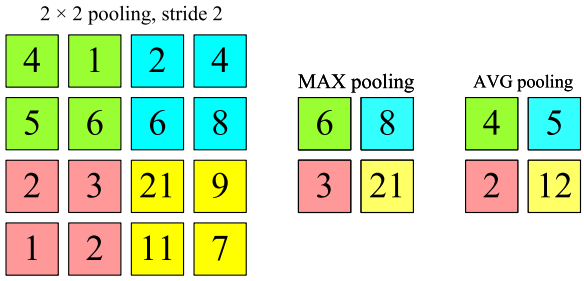


FIGURE 7. Various forms of pooling.

ability to converge faster than other activation functions like hyperbolic tangent and sigmoid [72], [197], ReLU [160] has gained a lot of traction in recent years. The mathematical representation of ReLU is shown in Eq. (4). Some popular extensions of ReLU, for instance, exponential LU [64], parametric ReLU [107], and leaky ReLU [149] are also being used in CNNs for improved performance and accuracy.

$$f(x) = max(0, x) \tag{4}$$

4) FULLY CONNECTED LAYER

Fully connected layers do the final classification in the CNN network after multiple convolutions, ReLU, and pooling layers. Weights, biases, and neurons are all part of the fully connected layer. All input and output neurons are connected in the fully connected layer. A CNN network typically has one or more fully connected layers. The final output of CNN comes from the last fully connected layer, often known as the classification layer. The fully connected layer in the CNN contains a large number of inputs and outputs. Therefore, it is challenging to implement fully connected layer operations on hardware platforms with limited resources.

5) DECONVOLUTION LAYER

To increase the size of the feature map, a deconvolution layer, also known as a transposed convolution layer, is employed [52]. Upsampling (inserting zeros in the feature map) and then convolving the upsampled feature maps with the kernel coefficients are used to accomplish this.

6) DILATED CONVOLUTION LAYER

The filter coefficients are up-sampled and convolved with the input image in a dilated convolution layer to capture a broader receptive field [114]. Image segmentation, for example, uses it to capture the larger global context in each output pixel.

With millions of weight coefficients, CNNs are extremely complex. They are computationally expensive and necessitate a significant amount of memory to store the input, output feature maps, and weight coefficients, causing CPUs to underperform. To boost the performance of the CNNs, specific hardware accelerators are used. As a result, different techniques for implementing CNNs efficiently on hardware platforms must be explored in order to reduce resource and memory requirements.

D. HARDWARE ARCHITECTURES FOR DNN ACCELERATION

DNNs have been increasingly popular in recent years, allowing for their development and deployment on a variety of hardware platforms. These hardware platforms are of various types, right from general-purpose architectures such as CPUs and GPUs, programmable architectures (FPGAs) to special-purpose chips (ASICs). In many DNN models, multiply-accumulate (MAC) operations are the most important computations, and they can be easily parallelized. Since these MAC operations can be executed in parallel, hardware architectures that enable parallel operations are required to process DNNs. To achieve superior performance, highly parallel computing models, encompassing both spatial and temporal computing architectures, are often employed for DNN acceleration. The spatial and temporal architectures have a similar computational structure, with a set of Processing Elements (PEs). However, processing units can have internal control in a spatial architecture, whereas control in a temporal architecture is centralized, as shown in Fig. 8. Each PE can have a register file (RF) to store data in spatial architecture; however, PEs do not have the memory capacity in a temporal architecture. The PEs can also be connected to exchange data in spatial computing designs. To summarize, the PEs in the temporal architectures contain only Arithmetic and Logic Units (ALUs). The PEs consist of ALU as a computation unit, RF to store the data, and a control unit in spatial architectures.

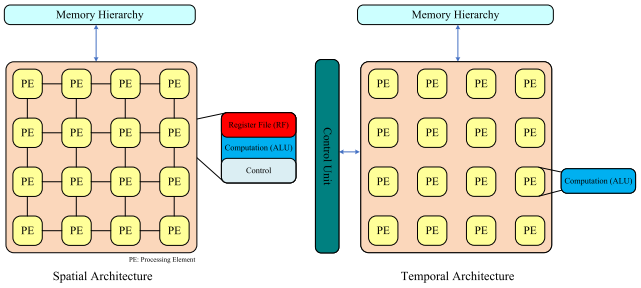


FIGURE 8. Spatial and temporal architectures.

1) TEMPORAL ARCHITECTURES

The temporal architectures exploit parallelism by supporting a variety of techniques, such as Single Instruction Multiple Threads (SIMT) or Single Instruction Multiple Data (SIMD). The temporal computing architectures appear mostly in CPUs and GPUs. In temporal designs, ALUs can only access data from the memory hierarchy and cannot communicate directly with one another. The memory (i.e., register file) and control are shared by all ALUs in the temporal architecture. In temporal architectures like CPUs or GPUs, all the convolution or fully connected operations are mapped to matrix multiplication. CPU cores are the least employed among the several temporal architectures for DNN training and inference. CPUs contain a small number of processing cores, ranging from one to ten. As a result, only a small number of processes can be performed in parallel, limiting throughput. GPUs are commonly used to train and infer DNNs. They have thousands of cores to run highly parallel algorithms efficiently, for instance, matrix multiplication. Throughput is enhanced by lowering the number of multiplications in both CPUs and GPUs. There are software libraries that optimize matrix multiplication for GPUs (e. g., cuBLAS, cuDNN [59], etc.) and CPUs (e. g., Intel MKL [2], OpenBLAS, etc.). Another well-known technique to reduce the matrix multiplications is Fast Fourier Transform (FFT) [80], [151]. Furthermore, several techniques, such as Winogra's algorithm [132] and Strassen's algorithm [67], are used to reduce the matrix multiplications and thereby reduce the resource and memory requirements.

2) SPATIAL ARCHITECTURES

In spatial architectures, each ALU can have its own local memory and control logic. The local memory is also referred to as the register file. The development and deployment of DNNs on Field-Programmable-Gate-Arrays (FPGA) and Application-Specific-Integrated-Circuits (ASIC) comes under the category of spatial architectures. FPGAs are less expensive and have a faster time to market than ASICs, and the design flow is simpler. However, FPGAs are less energy-efficient and consume more power than ASICs since FPGAs, unlike ASICs, contain a significant chip area dedicated to reconfigurability. ASICs, on the other hand, are mainly designed for a particular application and cannot support reconfigurability. The design flow of ASICs is more complex than FPGAs [46]. ASIC chips are expensive, but they are highly optimized and energy-efficient and provide superior performance than FPGAs. Memory accesses are the real bottleneck in DNN computations; therefore, off-chip DRAM accesses must be minimized, as they have a high energy cost and delay. The memory accesses (off-chip) can be reduced by reusing data stored in smaller, quicker, and low-energy memories. In spatial computing architectures, weight stationary, row stationary, output stationary, and other specialized processing dataflows can be designed to improve data reuse from memories in the memory hierarchy and reduce energy dissipation. At each level of the memory hierarchy, the dataflow defines what data is read and when it is processed. In spatial architectures, dataflows can be classified as follows:

a: WEIGHT STATIONARY (WS)

In weight stationary dataflow, the weights are kept fixed and are stored in the register files of the PEs, whereas the inputs and partial sums are distributed across the PEs. Weight stationary dataflow maximizes filter and convolutional reuse of weights. Weight stationary dataflow examples are found in [168], [182], [195], and [50].

b: OUTPUT STATIONARY (OS)

Each partial sum is held fixed in a PE in the output stationary dataflow, and accumulation is done until the final total is

obtained. In the meantime, the PEs' weights and inputs are dispersed in a variety of ways. The convolutional reuse is maximized with output stationary dataflow. This dataflow reduces the amount of energy used while writing and reading partial sums. Output stationary dataflow examples are found in [98] and [169].

c: ROW STATIONARY (RS)

The operations of a row of convolution are mapped to the same PE in row stationary dataflow, and the weights are kept stationary inside the register file of the PEs. Row stationary dataflow maximizes the convolutional reuse of input feature maps, weights, and partial sums. Row stationary dataflow examples are found in [53] and [57].

d: NO LOCAL REUSE (NLR)

In no local reuse dataflow, nothing is stationary inside the PEs, and it is used to reduce the accelerator area by eliminating the register file from PEs. No local reuse dataflow examples are found in [56] and [222].

All PEs in the spatial architectures can be connected in one of two ways: 1-D systolic or 2-D systolic. The PEs in a 1-D systolic architecture are arranged in one dimension, allowing systolic data flow, but the PEs in a 2-D systolic architecture are arranged in two dimensions and can receive data from both vertical and horizontal directions. Similarly, all PEs can be connected in temporal architectures in one of two ways: 1-D array or 2-D array. Data is received from the global buffer by the PEs in a 1-D array architecture, which are arrayed in one dimension. A 2-D array architecture has PEs that are arrayed in two dimensions and receive data only from the global buffer.

E. ROOFLINE MODEL

The roofline model is basically a visual performance model intended for floating point computations and multicore architectures [212]. The roofline model relates peak performance provided by the hardware platform and off-chip memory traffic with system performance. For a given compute-to-communication (CTC) ratio, the maximum attainable performance is the minimum of (1) peak computational performance and (2) peak memory performance. Here, the CTC ratio, also called operational intensity, means operations per byte of DRAM traffic. Eq. (5) formulates the attainable performance of an application on a specific hardware platform.

$$\text{Attainable Performance} = min \begin{cases} \text{Peak Floating Point Performance} \\ \text{Peak Memory Bandwidth} \times \text{CTC ratio} \end{cases} \quad (5)$$

The roofline model is illustrated in Fig. 9. Algorithm 2 in Fig. 9 has a better CTC ratio than Algorithm 1. As a result, Algorithm 2 performs better than Algorithm 1 because it effectively utilizes all of the hardware computation resources. In contrast, Algorithm 1 under-utilizes hardware computation resources due to inefficient off-chip communication.

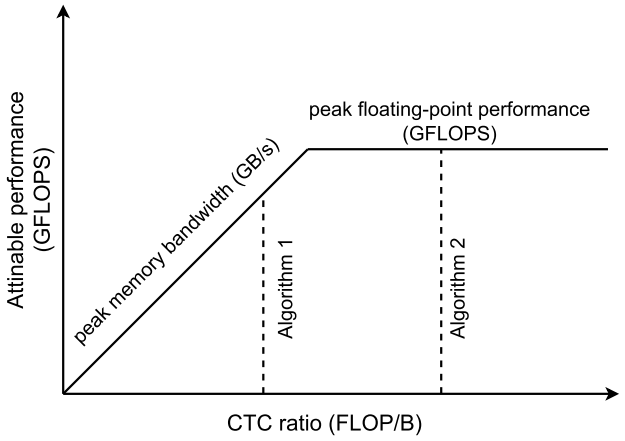


FIGURE 9. Roofline model, adopted from [222].

III. FPGA-BASED ACCELERATORS

The FPGA-based neural network accelerators are increasingly favored over CPUs because of their higher efficiency [164]. FPGA supports parallelism and accelerates the computations by mapping them to the parallel hardware; i. e., multiple DNN structures are executing in parallel on FPGA. FPGA-based accelerators deliver up to several orders of magnitude speedup compared to the baseline CPU [85]. FPGAs give designers the freedom to implement only the required logic in the hardware based on the target application. FPGA-based DNN accelerator architectures mainly contain a host computer and an FPGA part to implement DNN algorithms.

In this section, we would like to review FPGA-based DNN accelerators, which can be broadly categorized into three types: accelerators for a specific application, such as speech recognition, object detection, natural language processing, etc., accelerators for a specific algorithm, such as CNN, RNN, etc., and accelerator frameworks with hardware templates. For the first two categories, the design complexity of the accelerator is low, whereas the design complexity is relatively high for the final category.

A. ACCELERATORS FOR A SPECIFIC APPLICATION

There exists many FPGA-hardware accelerators for specific applications. Designing a custom accelerator for a given application is a good fit for the problem and has a low design complexity. Han et al. [102] proposed the FPGA-based accelerator named efficient speech recognition engine (ESE) to implement the LSTM algorithm for speech recognition. Load-balanced sensing pruning method is used in the proposed design to compress the LSTM model. The proposed accelerator uses a framework named Kaldi to implement LSTM algorithm for speech recognition. The ESE has a performance of 282 GOPS and is implemented in a Xilinx XCKU060 FPGA running at 200 MHz. The implementation

of speech recognition algorithms using FPGA-based accelerators is also presented in several earlier studies [62], [112], [137], [199].

Wang et al. [209] proposed a reconfigurable YOLOv3 FPGA hardware accelerator for object detection. In this context, YOLOv3 (You Only Look Once, Version 3) is a real-time object detection algorithm that detects specific objects in images or videos. The proposed accelerator is built using the ARM + FPGA architecture. Experiment results show that the FPGA-based YOLOv3 accelerator consumes less energy and achieves higher throughput than the GPU counterpart. The proposed accelerator is compatible with several frameworks, such as Tensorflow, Caffe, PyTorch, etc. The proposed accelerator is implemented on Xilinx ZCU104 running at a frequency of 300 MHz. Several previous works [82], [148], [161] also used the FPGA to implement object detection algorithms.

Hamza et al. [125] proposed the FPGA-based accelerator named NPE to efficiently implement various Natural Language Processing (NLP) models. NPE provides a single framework for processing arbitrarily complex nonlinear functions with software-like programmability. NPE consumes 4× and 6× less power than CPU and GPU. NPE is implemented on the Xilinx Zynq Z-7100 FPGA running at a frequency of 200 MHz.

Serkan et al. [184] developed an FPGA-based CNN accelerator to classify malaria disease cells. The proposed accelerator is implemented on Xilinx Zynq-7000 FPGA running at a frequency of 168 MHz. The proposed accelerator achieves an accuracy of 94.76%. Zhu et al. [228] proposed an FPGA-based accelerator to recognize liver dynamic CT images. Xiong et al. [217] developed an FPGA-based CNN accelerator to improve the automatic segmentation of 3D brain tumors. FPGA-based accelerators are also used to implement various applications such as autonomous driving [105], [129], image classification [45], [70], fraud detection [128], cancer detection [186], etc. Table 2 summarizes the reviewed FPGA-based accelerators for specific applications.

B. ACCELERATORS FOR A SPECIFIC ALGORITHM

A prominent topic of research in the realm of accelerators is the use of FPGA-based accelerators for a particular neural network algorithm. Since the accelerator is intended to address a specific problem, its operation typically requires minimal adjustments to a few parameters to operate effectively. Cloutier et al. [65] proposed a hardware accelerator, referred to as *Virtual Image Processor (VIP)* to implement the CNNs. The Altera EPF81500 FPGA platform is used to implement the proposed design. VIP primarily consists of Processing Elements (PEs) connected by a 2-D systolic architecture and supports the SIMD paradigm. VIP is designed to perform the following vector and matrix operations: matrix multiplication, matrix-vector multiplication, scalar multiplication, matrix addition, matrix-vector addition, vector addition, 1-D convolution, 2-D convolution, etc. The host computer is used to provide the configuration data to the FPGA board, which is connected through Peripheral Component Interconnect (PCI) interface. VIP uses the low accuracy arithmetic because of the limitations of resources on Altera EPF81500 FPGA. Fortunately, recent FPGAs contains large numbers of computing units and memory resources and allow fast CNN implementations. FPGA implementations of DNNs mainly focused on accelerating the convolution operations, which are reported in [38] and [49].

Farabet et al. [85] presented ConvNet Processor (CNP): an FPGA-based accelerator to implement the CNNs. CNP uses dedicated hardware convolver for the data processing and also uses soft-processor for controlling. CNP is designed on the Virtex4 SX35 FPGA and also equipped with external memory to store the input and filter coefficients. CNP consists of *Vector Arithmetic and Logic Units (VALU)*, one of the main components in the architecture that implements the CNN operations viz. 2-D convolutions, sub-sampling, and non-linear activation functions. The implementation of 2-D convolution, represented using Eq. (6), is shown in Fig. 10 for $K = 3$, i. e. 3×3 kernel. In Eq. (6), x_{ij} is the data in the input plane, w_{mn} is the weight value in $K \times K$ kernel, y_{ij} is the partial sum, z_{ij} is the result in the output plane, and W is the width of the input image. At each clock cycle, the convolution module performs k^2 multiply-accumulate operations simultaneously. CNP uses the First In First Out (FIFO) buffers between the external memory and FPGA to provides the continuous flow of data in both directions. CNP uses the 32-bit soft processor that provides the macro instructions, generally higher level instructions than most traditional processors, to the VALU for implementing the basic CNN operations. CNP has a compiler that converts network implementations with Torch directly into CNP instructions. The proposed architecture has been used to implement the face detection system.

$$z_{ij} = y_{ij} + \sum_{m=0}^{K-1} \sum_{n=0}^{K-1} x_{i+m,j+n} \cdot w_{mn} \tag{6}$$

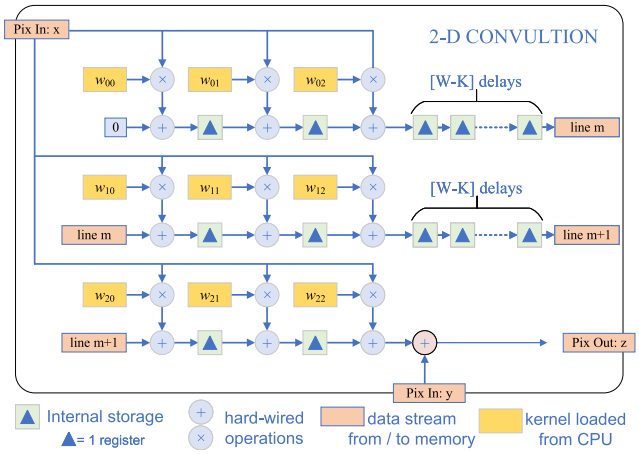


FIGURE 10. 2-D convolution module for 3 × 3 kernel, adopted from [85].

Sankaradas et al. [182] presented a massively parallel co-processor for accelerating CNNs. This co-processor is

TABLE 2. Summary of FPGA-based accelerators for specific application.

Application	DNN Type	FPGA Device	Frequency (MHz)	Year
Speech recognition [112]	CNN	Fudan Micro	-	2022
Object detection [209]	CNN	Xilinx ZCU104	300	2021
Natural Language Processing [125]	CNN	Xilinx Zynq Z-7100	200	2021
Liver CT image recognition [228]	CNN	-	-	2021
Image classification [45]	CNN	Xilinx xc7vx980t	225	2021
Fraud detection [128]	CNN	Xilinx Zynq 7000	-	2021
Brain tumor segmentation [217]	CNN	Xilinx Alveo U280	-	2021
Object detection [148]	CNN	Intel Arria 10 GX1150	240	2020
Object detection [161]	CNN	Xilinx VC707	200	2019
Malaria disease cell detection [184]	CNN	Xilinx Zynq-7000	168	2019
Autonomous driving [105]	DNN	Xilinx Ultra96	-	2019
Object detection [82]	CNN	Xilinx ZC706	100	2018
Autonomous driving [129]	CNN	Xilinx Zynq 7020	100	2018
Image classification [70]	CNN	Xilinx ZC702	100	2018
Speech recognition [102]	LSTM	Xilinx XCKU060	200	2017
Cancer detection [186]	MLPNN	Xilinx XC5VLX50TFFT1136	-	2016
Speech recognition [137]	LSTM	Xilinx XC7Z045	100	2016
Speech recognition [199]	-	Altera DE2-115	50	2015
Speech recognition [62]	-	Xilinx ML402	100	2010

designed using the Virtex5 LX330T FPGA platform and four DDR2 (Double Data Rate 2) memory banks totalling 1 GB. The proposed co-processor mainly consists of clusters of *Vector Processing Elements (VPE)* connected in parallel. Each cluster consists of 2-D convolver units, sub-samplers, Look-Up Tables (LUT) and performs convolution, pooling, and non-linear functions. The co-processor is coupled with DDR2 memory banks to store the intermediate data. Each VPE in the proposed co-processor exploits parallelism by supporting SIMD stream. The primitive 2-D convolver of the proposed design is shown in Fig. 11. It contains $k \times k$ convolution units along with $k^2 + k$ VPEs, and the final column of VPEs is used to add partial results. The coprocessor operates in collaboration with a host, which can control the coprocessor through an Application Programming Interface (API). The proposed design uses low precision data representation to improve the throughput and memory bandwidth. The proposed architecture has been used to implement the full face recognition application using CNN with four convolution layers. The proposed accelerator can not be used to realize the full CNNs, which contain convolution and fully connected layers. Graf et al. [93] used a similar approach to accelerate the *Support Vector Machines (SVM)* and the proposed design contains VPEs instead of VPE clusters. But the accelerator proposed in [93] provides low performance while accelerating DNNs compared to the co-processor proposed in [182].

A programmable parallel accelerator called MAPLE is presented in [47] to accelerate the several learning and classification algorithms such as Support Vector Machine (SVM), K_means, CNN, etc. MAPLE contains hundreds of simple PEs arranged in a 2-D grid fashion as shown in Fig. 12 MAPLE can be used to perform vector and matrix operations in parallel. In MAPLE, each PE has local storage to perform the computations efficiently. Each PE has two operands; one operand comes from its local storage, and another operand comes from the PE on its left,

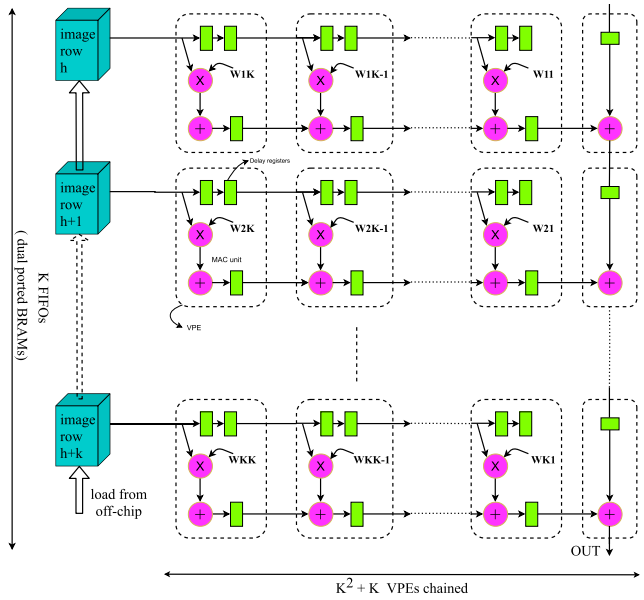


FIGURE 11. 2-D convolver unit of CNN co-processor, adopted from [182].

see Fig. 12. Furthermore, the output of each PE is connected to the PE on its right. The PEs are arranged as clusters, where each cluster has a separate off-chip memory block that creates independent data streams for memory-processor computations. MAPLE processing core can be organized as H clusters, and each cluster contains M PEs. So, the total number of PEs in MAPLE core equals $H \times M$. MAPLE also uses smart memory banks to process the intermediate data and to perform secondary reduction operations such as, aggregation, finding minimum or maximum, and array ranking. Authors developed a tool to map the applications on the MAPLE. For the given input matrices and reduction functions, the tool generates the assembly code needed to program MAPLE. The authors created a C++ simulator that estimates how long MAPLE will take to execute from

the input assembly code and an architectural configuration file that details the processor layout and off-chip memory architecture. MAPLE processor is connected to the host computer through Peripheral Component Interconnect (PCI). MAPLE is implemented on Xilinx Virtex 5 SX240T FPGA with 512 PEs organized as 2 cores, 32 chains per core, and 8 PEs per chain. The experimental results show that MAPLE with 512 PEs is 1.5 to 10 times faster than a quad-core Xeon processor with a clock frequency of 2.5 GHz despite running at 125 MHz.

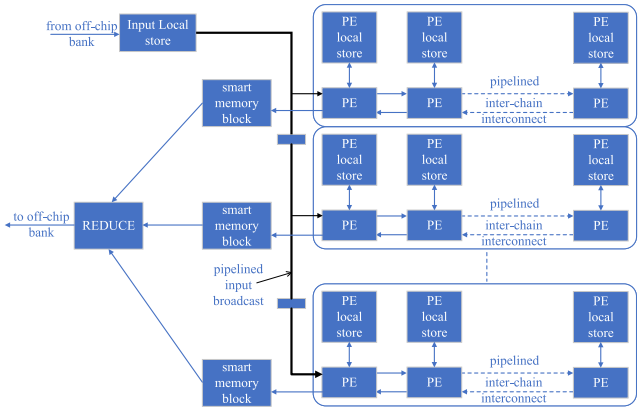


FIGURE 12. MAPLE's processing core architecture, adopted from [47].

Chakradhar et al. [51] presented a dynamically reconfigurable architecture for CNNs on Virtex 5 FPGA platform. The proposed system consists of a dynamically configurable CNN (DC-CNN) processing core and three bank memory sub-system. The DC-CNN processing core continuously communicates with the host computer that executes the main application. In the proposed accelerator, the host computer transfers the complete CNN structure and input images to the co-processor. The DC-CNN processing core, responsible for executing CNN applications, mainly contains computational units (2-D convolvers), subsampling, and non-linearity units, adders, input and output switches, as shown in Fig. 13. The co-processor uses the three bank memory sub-system to store input images, kernels, and intermediate data. The DC-CNN uses the Torch7 [66] software for CNN implementation. The proposed dynamically reconfigurable architecture supports ''inter-output'' and ''intra-output'' parallelism. The performance of the proposed dynamically reconfigurable architecture with 20 convolvers, 128-bit memory port width is 4 to 8 times faster than CNP presented in [85]. The proposed architecture can be used to accelerate CNN with only three convolutional layers. The proposed accelerator is not capable of realizing the full CNNs, which contain convolution and fully connected layers.

A CNN accelerator, referred to as NeuFlow is proposed in [84]. NeuFlow is implemented on a Xilinx Virtex 6 FPGA platform. NeuFlow contains a 2-D grid of *Processing Tiles (PTs)*, as shown in Fig. 14. A PT contains a bank of processing operators where an operator can be anything from memory (FIFO) to an arithmetic operator. All the operators

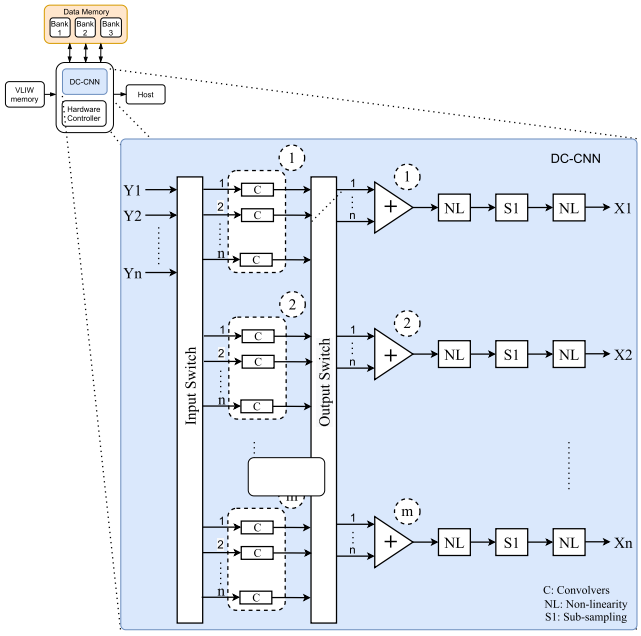


FIGURE 13. DC-CNN co-processor architecture, adopted from [51].

are connected to a local data line using reconfigurable routes. A multiplexer connects the local data line to a global data line by which a PT is connected to the four neighbouring PTs. Data is transferred from the off-chip memory to the tiles using a *Smart Direct Memory Access Module (Smart DMA)*. The control unit configures each tile for the computation and connections between the tiles. Data streams from the Smart DMA are processed in tiles, and the results are passed to the neighbouring tiles, or back to the Smart DMA. This 2-D grid can be used to perform arbitrary computations on streams of data and plain unary operations to complex nested operations. Using FIFOs input/output flow can be managed, and operators can easily be cascaded and connected across tiles. The NeuFlow accelerator uses a compiler named luaFlow to process CNNs. The *luaFlow* compiler converts high-level data flow graph representations of deep learning algorithms in the Torch5 environment into machine code for NeuFlow. The proposed accelerator has been used to implement a real-time street scene parser.

Peeman et al. [169] presented a memory-centric design method for CNN accelerator. The proposed memory-centric accelerator is implemented on a Virtex 6 FPGA board. This accelerator minimizes the bandwidth requirements by exploiting the data reuse in complex access patterns. The memory-centric accelerator uses the loop transformation and Block RAM (BRAM)-based multi-bank on-chip buffers to maximize the efficiency of on-chip memories for better data locality. The memory-centric accelerator uses SIMD type of PEs to accelerate the convolutional layers. The proposed accelerator design mainly focused on the maximization of the reuse of on-chip data. The proposed accelerator is connected to a MicroBlaze host processor and is communicated through Fast Simplex Link (FSL) connections. Vivado HLS tool is

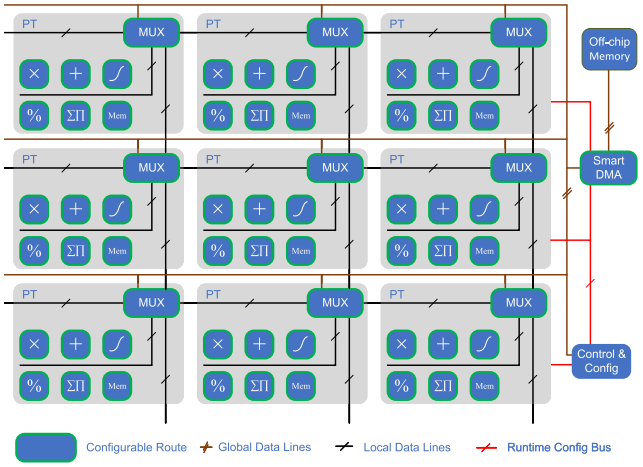


FIGURE 14. 2-D grid of Processing Tiles (PTs) in NeuFlow architecture, adopted from [84].

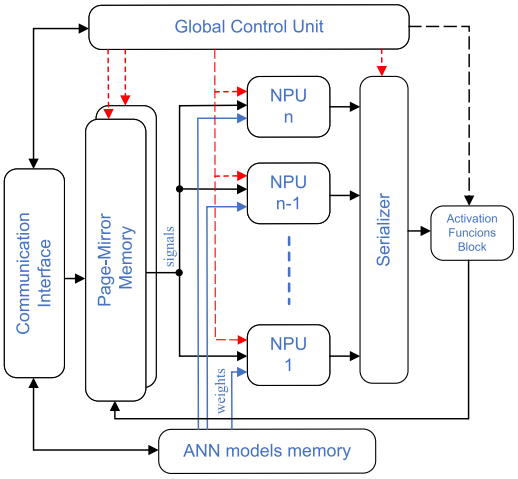


FIGURE 15. NPU-based neural network accelerator architecture, adopted from [171].

used to map the CNNs on the proposed accelerator, which enables the user to use the high-level accelerator description in C and to use HLS directives to specify the hardware configuration. The performance of the proposed accelerator will be improved with the use of the DMA controller.

In [171], an accelerator for DNNs is introduced, and it is implemented on the Xilinx Kintex 7 FPGA platform. It is built using a set of *Neural Processing Units (NPUs)*, see Fig. 15. The number of NPUs in the proposed design depends on the available FPGA resources. NPUs are mainly used to compute the majority of operations (multiplications and additions) in parallel. A multiply and accumulate (MAC) unit and control logic are the essential components of each NPU. The proposed accelerator utilizes the available FPGA resources efficiently by using pipelined architecture, time division multiplexing (TDM) processing scheme, and page-mirror algorithm. In the proposed accelerator, NPUs get the inputs from the host computer through the Ethernet interface, and weight coefficients are fetched from page mirror memory. The serializer sends the output of NPUs to the activation function blocks. For each sample, the proposed accelerator requires a long time to transfer the appropriate weight coefficient from the host computer to the accelerator core.

A scalable and low-power accelerator referred to as neural network next (nn-X) is presented in [91] to accelerate the DNNs. The nn-X accelerator mainly contains a co-processor, a host processor, and external memory as shown in Fig. 16. The host processor controls the input and configuration data transfer to the coprocessor, parses the DNN, and converts it into instructions for the coprocessor. The co-processor mainly contains an array of processing elements called collections, configuration bus, and memory router. The collections in the nn-X accelerator are mainly composed of convolution engines, pooling modules, and non-linear operators and are used to perform the most common CNN operations, such as convolution, sub-sampling, and activation functions. The memory router in the nn-X accelerator is used to transfer the data between the processing elements and the external memory, which provides independent data streams. The proposed architecture uses the weight stationary dataflow to improve energy efficiency. The nn-X accelerator is implemented using the Xilinx ZC706 platform, which has a dual ARM Cortex-A9 processor, Xilinx Zynq XC7Z045 chip, and 1 GB DDR3 memory. The experimental results show that the nn-X can achieve a peak performance of 240 GOPS.

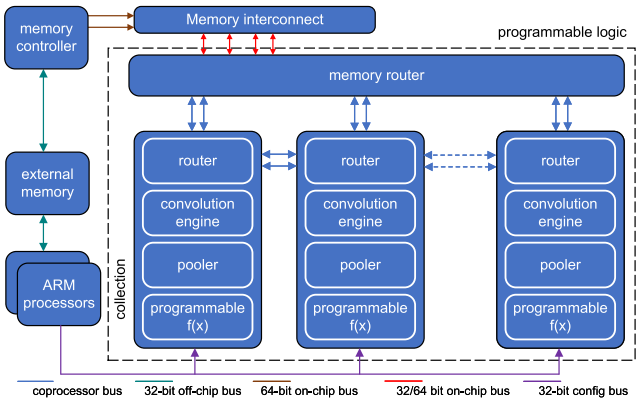


FIGURE 16. Architecture of nn-X system, adopted from [91].

Zhang et.al. [222] proposed a roofline-based model [212] to implement CNNs on FPGAs. The authors analyzed the throughput and required bandwidth for a given CNN design using various optimization techniques, such as loop tiling and loop transformation. With the help of the roofline model, they identified the solutions with the best performance and lowest FPGA resource requirement. This roofline-based model optimizes both the memory accesses as well as computations in the convolutional layers. The accelerator design is implemented with the Vivado HLS tool, which enables the accelerator implementation in C language. The proposed accelerator achieves a maximum throughput of 61.62 GFLOPS (Giga Floating-point Operations Per Second).

Implementing DNN in embedded devices is tough due to resource and power constraints. In this regard, authors in [172] have developed novel FPGA-based accelerators for implementing trained and fully connected DNNs. Since it is difficult to map a DNN with a large number of neurons and corresponding weights, directly onto an FPGA, the authors in [172] used a time division multiplexing scheme. Batch processing is used in the proposed architecture, which distributes different weights over many input samples. In addition, the suggested accelerator employs a pipelined architecture to make the most of the FPGA resources while staying within power and resource limits. The concept of pruning has also been incorporated into the proposed architecture to reduce data transfer from the external memory to the accelerator [173]. Both Batch processing and weight pruning can enhance the throughput of DNN accelerators.

Qiu et al. [177] proposed FPGA based CNN accelerator, which will efficiently accelerate all the layers of CNN, including the fully connected layers. The proposed accelerator improves bandwidth and resource usage by employing a dynamic-precision data quantization method and a unique design of the convolver hardware module. The proposed accelerator applies singular value decomposition (SVD) on weight coefficients to minimize the memory footprint at the fully connected layer. The convolver hardware module can be used for both convolutional and fully connected layers to reduce resource consumption. The adder tree, convolver complex, non-linearity, max-pooling, bias shift, and data shift are the main elements of the convolver hardware module, as shown in Fig. 17. Convolutions and fully connected layer operations are both performed using the convolver complex module. The max pooling action is carried out using the max-pooling module. CNN's non-linearity function is calculated using the non-linearity module. The convolver complex module generates partial sums, which are added by the adder tree. Finally, for dynamic quantization, bias shift and data shift modules are used. The proposed accelerator supports the Caffe deep learning framework. The proposed accelerator has been implemented on the Xilinx Zynq platform.

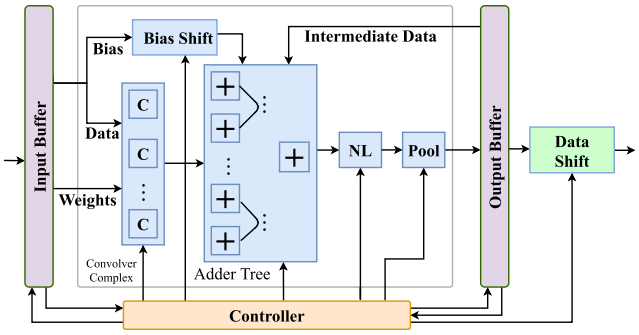


FIGURE 17. Convolver architecture, adopted from [177].

Wang et al. [208] proposed a scalable design called *Deep Learning Accelerator Unit (DLAU)* for accelerating deep learning algorithms. DLAU utilizes the tiling technique to produce a scalable architecture. The proposed accelerator mainly contains modules such as DMA, embedded processor, DLAU, and DDR3 memory controller as shown in Fig. 18. The DLAU module mainly contains three processing units, viz. Partial Sum Accumulation Unit (PSAU), Tiled Matrix Multiplication Unit (TMMU), and Activation Function Acceleration Unit (AFAU). TMMU is used to perform multiplication operations and also generate partial sums. PSAU is used to add the partial sums derived from TMMU. Finally, AFAU is used to perform the non-linear activation functions, for instance, the sigmoid function. The DLAU module reads the tiled input data through the DDR3 memory. The embedded processor provides the programming interface to the users and communicates with DLAU via JTAG-UART. The proposed architecture is implemented on Xilinx Zynq Zedboard with ARM Cortex-A9 processors operating at 667 MHz.

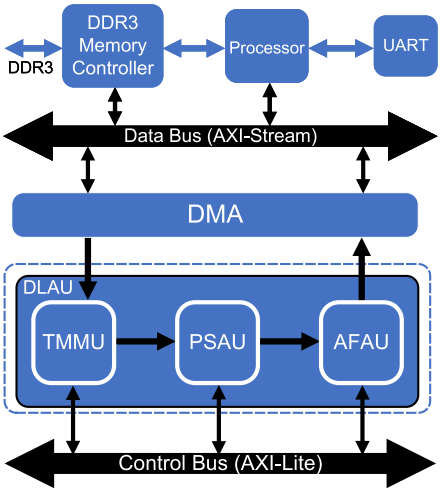


FIGURE 18. DLAU accelerator architecture, adopted from [208].

Lian et al. [140] proposed a block-floating-point (BFP) arithmetic-based CNN accelerator for DNN inference. The proposed accelerator mainly contains three elements: Processing Array (PEA), on-chip buffer, and external memory, as shown in Fig. 19. The onboard DDR3 modules receive input data and network parameters from the host computer via PCIe3.0 × 8. Conv PEA performs the convolutional operations, and FC PEA performs the fully connected layer operations. The proposed accelerator uses 8-bit and 16-bit formats to represent the feature maps and modal parameters (activations and weights), which can reduce off-chip bandwidth and memory compared to the 32-bit floating point counterpart with only a tiny accuracy loss. The accelerator design is implemented with the Vivado HLS tool, and the proposed BFP arithmetic is conducted on the Caffe [119] scheme. The proposed accelerator is implemented on the Xilinx VC709 evaluation board, running at a frequency of 200 MHz, and achieves a throughput of 760.83 GOP/s.

Xiao et al. [216] presented the DNN accelerator architecture specially designed for the sparse and compressed DNN models. The proposed DNN accelerator mainly contains

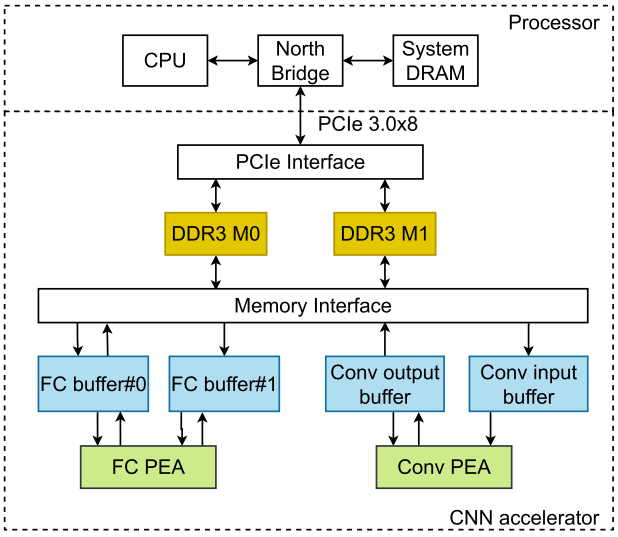


FIGURE 19. Block diagram of BFP arithmetic-based CNN accelerator, adopted from [140].

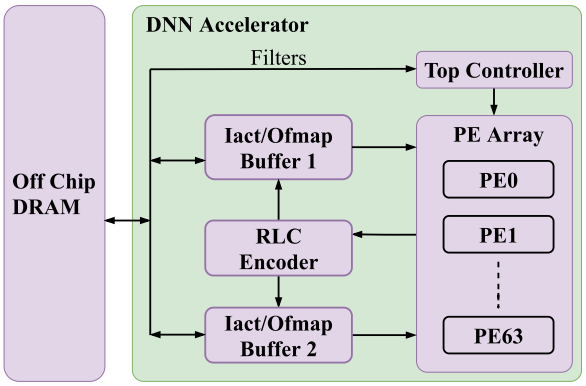


FIGURE 20. DNN accelerator architecture proposed in [216], adopted from [216].

a PE array, RLC encoder, controller, and on-chip buffers, as shown in Fig. 20. In the proposed DNN accelerator, all the weights and non-linear activation functions are kept in Run-length Coding (RLC) compressed form and are stored in off-chip DRAM memory. The PE array contains 64 PEs and performs the multiply-accumulate (MAC) operations of the fully connected layer. The proposed accelerator uses a novel circuit-level processing scheme to process the sparse data directly in the compressed domain without decompressing, which leads to improvement in efficiency and performance. The circuit-level process scheme used in this architecture is dataflow independent, and thus, applies to both CNN and fully connected layers. In this architecture, a new dataflow is proposed to facilitate the reuse of input activations across the fully connected layers, which leads to exploits parallelism and maximizes the utilization of PEs. In this work, the Xilinx Vivado HLS toolchain is used to convert C code to RTL implementation, and then Xilinx SDSoC is used to compile the source code to generate the bit stream. The proposed architecture is implemented on the Xilinx Virtex-7 FPGA platform and achieves the performance of 1.34 GOP/s.

Ahmed et al. [81] proposed an FPGA-based Low Power CNN (LP-CNN) accelerator based on GoogLeNet CNN. The proposed accelerator uses quantization and weight pruning techniques to reduce memory size. The LP-CNN accelerator is a time-sharing processor designed to process the CNN model layer by layer, and it enables pipelining. The proposed accelerator only uses the on-chip memory to store the activations and weights instead of offline DRAM memory. Moreover, the proposed architecture replaces multiplication operations with shifting operations and uses no DSP units. The LP-CNN accelerator is implemented in Verilog RTL, and the Vivado power analyzer has been used to calculate the power. The experimental results show that the LP-CNN accelerator provides 49.5 and 7.8 times power improvement over the Intel Core-i7 and NVidia GTX 1080Ti, respectively. The proposed accelerator has been implemented on the Virtex-7 FPGA running at a frequency of 200 MHz.

The low-power, energy-efficient FPGA-based accelerator is presented in [116] to accelerate the LeNet CNNs. The proposed accelerator uses 8-bit, 16-bit, and 32-bit fixed point formats to represent the weights, activations, and biases, respectively. The proposed accelerator supports pipelining and implements LeNet with the minimal resources possible without affecting the throughput. This work uses the Xilinx Vitis HLS tool to convert the C++ code to RTL implementation. The proposed accelerator is implemented on the Nexys DDR 4 FPGA evaluation board and achieves a throughput of 14K images/sec while using just 628 mW of power.

An FPGA-based dynamically reconfigurable architecture is presented in [117] to accelerate neural networks. Dynamic Partial Reconfiguration (DPR) is used in the proposed accelerator to realize different types of neural network architectures. Dynamic Partial Reconfiguration (DPR) allows the proposed architecture to switch between networks and applications without sacrificing precision or throughput. The proposed accelerator mainly contains a PE array and configurable switches, as shown in Fig. 21. PE is a high-level generic block that can implement the layers of a neural network accelerator and has three predefined interfaces: data interface, I/O interface, and memory interface. DPR allows each PE to implement many functionalities with the same hardware. Any PE can communicate with any other PE, CPU, or I/O port of the FPGA through configurable switches. The hard/soft processor controls all PE connections using the memory interface. This work uses the Xilinx Vitis HLS tool to convert the C++ code to RTL implementation. The proposed accelerator is implemented on the Xilinx Zynq 7020 FPGA board.

Gowda et al. [92] proposed an FPGA-based reconfigurable convolutional neural network (RCNN) accelerator. Unlike the existing structures, the RCNN accelerator contains configuration registers to reconfigure the architecture according to the configuration instructions stored in the Double Data Rate (DDR), as shown in Fig. 22. The image

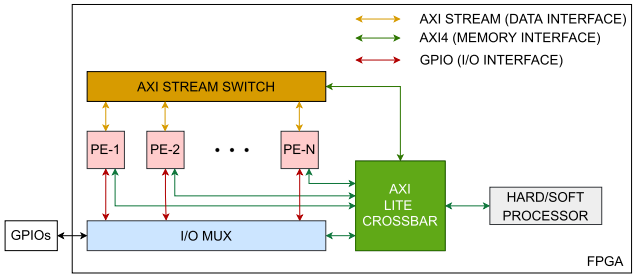


FIGURE 21. Accelerator architecture proposed in [117], adopted from [117].

and weight buffers are updated with input feature maps and weights. The PE arrays perform the convolution operations, whereas the special function buffer performs pooling, Batch Normalization (BN), and activation functions. The proposed accelerator uses the SOW (sparse optimization of weight) and CO (convolutional optimization) optimizations to reduce the sizes of weights and feature maps, respectively, which also minimizes the number of hardware resources needed. The proposed accelerator uses 16-bit, 8-bit, and 4-bit fixed point formats to represent the feature maps, convolution (CONV) layer weights, and fully connected (FC) layer weights, respectively. This work uses the Xilinx Vivado HLS toolchain to convert C++ code to RTL implementation. The proposed accelerator is implemented on Xilinx Zynq 7020 FPGA board.

Table 3 summarizes the reviewed FPGA-based accelerators for a specific algorithm. The year the accelerator was introduced, the deep learning model used, the FPGA platform used, the precision used for input feature maps and weights, the clock frequency, the number of resources available in terms of DSPs, LUTs, BRAMs, and FFs, the percentage of resources utilized, the performance in GOPS, and finally, the power efficiency (GOPS/W) are all listed for each accelerator. Fig. 23 shows the power efficiency and throughput of various FPGA-based accelerators listed in Table 3.

C. ACCELERATOR FRAMEWORKS WITH HARDWARE TEMPLATES

Several frameworks for mapping AI models onto FPGAs have been developed in recent years. Venieris et al. [206] developed a framework called *fpgaConvNet* to map CNNs on FPGAs. The fpgaConvNet framework employs the synchronous dataflow (SDF) paradigm to capture the CNN workloads. The processing flow of fpgaConvNet is shown in Fig. 24. Firstly, the Deep Learning expert uses a domain-specific language to provide a high-level description of a ConvNet architecture as well as information on the target FPGA-based platform as inputs. The ConvNet description is passed through a DSL (Domain-Specific Language) processor, which parses the input script and populates the ConvNet's semantic model as a Directed Acyclic Graph (DAG), and also extracts platform-specific resource constraints. The ConvNet

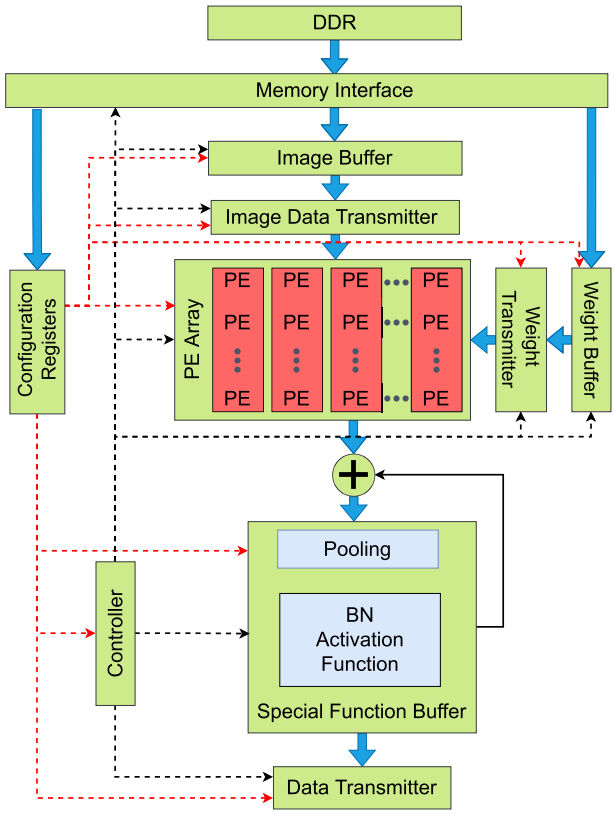


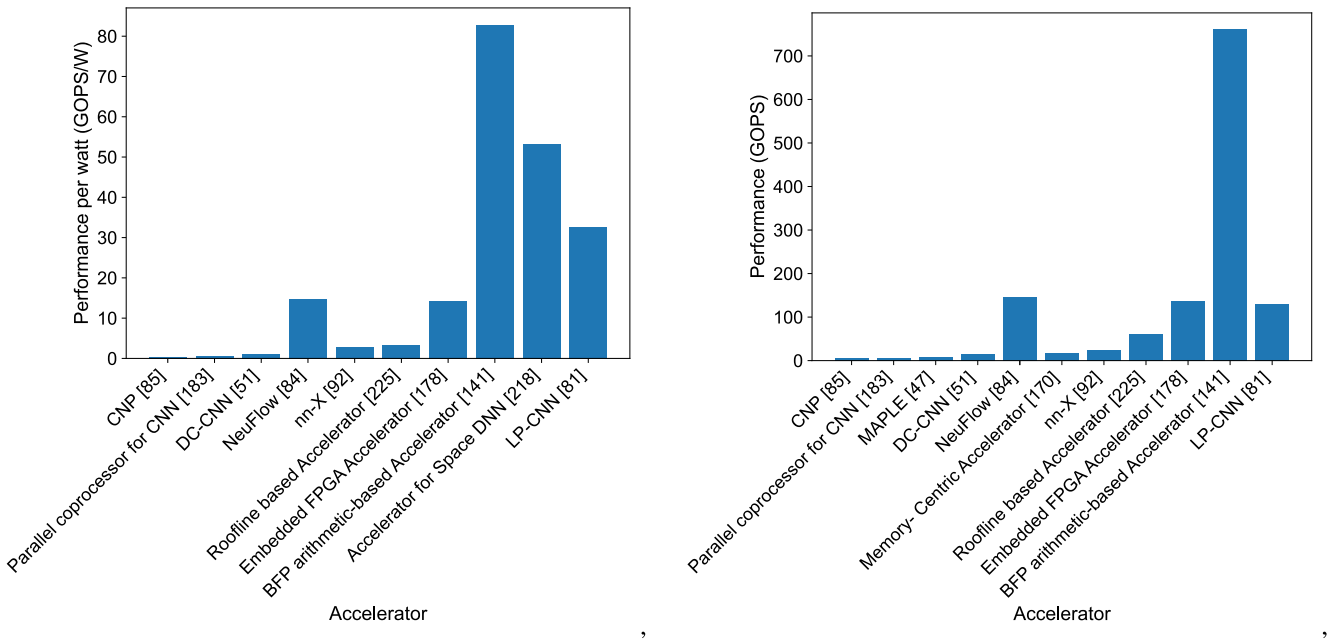
FIGURE 22. RCNN accelerator architecture, adopted from [92].

DAG is converted into an SDF hardware intermediate format, which corresponds to an utterly parallel hardware implementation. After several transformations on ConvNet's SDF hardware model, the design space is searched, and this procedure provides a set of hardware mappings of the ConvNet onto the specific FPGA-based platform. The fpgaConvNet front-end parser can examine models written in the Caffe and Torch machine-learning libraries. This framework accomplishes efficient design space explorations through graph segmentation, reconfiguration, folding, and weight reloading. This framework can be used to map small CNN models, for instance, LeNet-5 on FPGAs.

Wang et al. [211] developed a design automation tool referred to as *DeepBurning* that contains a library of building blocks that mimic the behavior of typical neural network components. The general design flow of the DeepBurning framework is shown in Fig. 25. The DeepBurning Neural Network Generator (NN-Gen) takes a model descriptive script (Caffe-compatible script) as input, which describes a high-level view of network topology and layer definition. The DeepBurning NN-Gen also takes user-specified constraints such as area and power as input. DeepBurning NN-Gen consists of a hardware generator and compiler that generate the control flow and data layout based on the user's specifications. The DeepBurning automation tool's hardware generator builds a neural network architecture for a given

TABLE 3. Summary of FPGA-based accelerators for specific algorithm.

Accelerator Name	Year	DNN Type	FPGA Platform	Frequency (MHz)	Precision	LUT Type (# inputs)	Resources BRAMs	Resources LUTs	Resources FFs	Resources DSPs	Resource Utilization BRAMs	Resource Utilization LUTs	Resource Utilization FFs	Resource Utilization DSPs	Performance (GOPS)	Power Efficiency (GOPS/W)
VIP [65]	1996	CNN	Altera EPF81500	16	fixed point	4	N/A	1500	1,500	N/A		N/A			N/A	N/A
CNP [85]	2009	LeNet-5	Virtex4 SX35	200	16-bit fixed point	4	192	30720	30720	192	N/A	90%	90%	28%	5.25	0.35
Parallel coprocessor for CNN [182]	2009	CNN	Virtex5 LX330T	115	16-bit fixed point	6	324	207360	207360	192	0.93%	17%	19.05%	55.73%	6.74	0.61
MAPLE [47]	2010	CNN	Virtex5 SX240T	125	fixed point	6	516	149760	149760	1056		N/A			7	N/A
DC-CNN [51]	2010	CNN	Virtex5 SX240T	120	48-bit fixed point	6	516	149760	149760	1056		N/A			16	1.14
NeuFlow [84]	2011	CNN	Virtex6 VLX240T	200	16-bit fixed point	6	416	150720	301440	768		N/A			147	14.7
Memory- Centric Accelerator [169]	2013	CNN	Virtex6 VLX240T	150	fixed point	6	416	150720	301440	768	45.50%	1.10%	N/A	6%	17	N/A
NPU based Accelerator [171]	2014	DNN	Xilinx Kintex 7	N/A	float point	6	1590	254200	508400	1540		N/A			N/A	N/A
nn-X [91]	2014	CNN	Zynq XC7Z045	142	16-bit fixed point	4	545	218600	437200	900		N/A			23.18	2.9
Roofline based Accelerator [222]	2015	AlexNet	Virtex7 VX485T	100	32-bit float point	4	2060	303600	607200	2800	50%	61.30%	33.87%	80%	61.62	3.31
Embedded FPGA Accelerator [177]	2016	VGG-16	Zynq XC7Z045	150	16-bit fixed point	4	545	218600	437200	900	86.70%	83.50%	29.20%	89.20%	136.97	14.22
DNN Acceleration using Batch Processing [172]	2016	DNN	Zynq-7000	100	16-bit fixed point	6	280	53200	106400	220		N/A			N/A	N/A
DLAU [208]	2017	DNN	Zynq XC7Z020	200	48-bit float point	6	280	53200	106400	220	12.50%	68.40%	26.60%	75.90%	N/A	N/A
DNN Acceleration using Batch Processing and Pruning **[173]**	2018	DNN	ZedBoard	100	16-bit fixed point	6	280	53,200	106,400	220		N/A			4.48	N/A
BFP arithmetic-based Accelerator [140]	2019	VGG-16	Xilinx VC709	200	8-bit BFP (feature maps) 16-bit BFP (weights and activations)	6	1470	433200	866400	3600	62.1%	53.5%	16.3%	28.5%	**760.83**	**82.88**
Accelerator for Space DNN [216]	2021	AlexNet/ VGG-16	Xilinx Virtex7	200	16-bit fixed point	6	1470	433200	866400	3600		N/A			1.34	53.14
LP-CNN [81]	2021	GoogLeNet	Virtex-7 VC709	200	12-bit fixed point	6	1470	433200	866400	3600	77%	94%	10%	0%	129.2	32.7
Energy-efficient CNN Accelerator [116]	2021	LeNet	Xilinx Artix XC7A100T	125	8-bit fixed point (weights) 16-bit fixed point (activations) 32-bit fixed point (biases)	6	135	63400	126800	240	21.48%	25.16%	13.93%	50%	N/A	N/A
Dynamically Reconfigurable Architecture [117]	2022	CNN, SNN	Xilinx Zynq 7020	200	N/A	6	280	53200	106400	220		N/A			N/A	N/A
RCNN Accelerator [92]	2022	AlexNet VGG-16 VGG-19	Xilinx Zynq 7020	200	16-bit fixed point (feature maps) 8-bit fixed point (CONV layer weights) 4-bit fixed point (FC layer weights)	6	280	53200	106400	220		N/A			N/A	N/A



(a) Power efficiency

(b) Throughput

FIGURE 23. Power efficiency and throughput of FPGA-based accelerators listed in Table 3.

network structure by selecting and instantiating blocks from the library with the required interconnections. DeepBurning supports a wide range of NN models and simplifies the design flow of NN-based accelerators for machine learning applications.

A framework referred to as DNNWeaver is presented in [187] that generates bitstream and host code to implement DNNs on various FPGA boards. DNNWeaver employs Caffe as its programming interface. DNNWeaver consists of three software components: translator, design weaver, and integrator. The translator transforms the Caffe specification of a DNN into a macro data flow graph. Design weaver accepts macro data flow graph as an input and generates a synthesizable Verilog implementation of the accelerator code. The integrator adds the memory interface code to the accelerator code. DNNWeaver generates accelerator

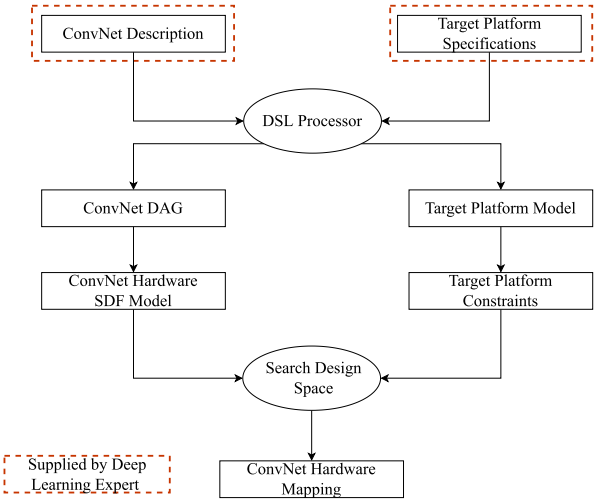


FIGURE 24. Processing flow of fpgaConvNet, adopted from [206].

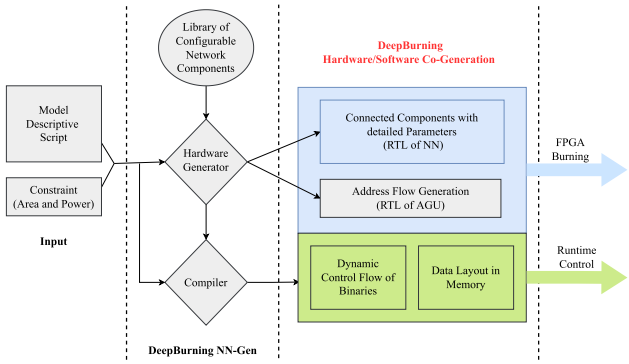


FIGURE 25. Design flow of DeepBurning framework, adopted from [211].

code from a series of scalable and customizable hand-optimized template designs, resulting in high performance and efficiency.

Guan et al. [95] proposed a framework called *Field Programmable DNN* (FP-DNN) to accelerate DNNs efficiently on FPGAs. FP-DNN Framework is shown in Fig. 26. The model description is generated by TensorFlow and is fed into a Symbolic Compiler. The compiler generates a C++ program and an FPGA programming bit stream for model inference, executed by the host and device, respectively. Model mapper examines the model description, extracts the target model's topological structure and operations, and sends the hardware kernel schedule and configuration to the code generators. Software generator generates the host code in C++ using the kernel scheduling. The host code is compiled using a commercial C++ compiler to create host programs. Hardware generator creates device codes by instantiating RTL-HLS hybrid templates based on kernel configuration. The hardware code is compiled using commercial synthesis tools to generate the programming files for the hardware implementation. With a high-performance computing engine and well-designed communication-optimized algorithms, FP-DNN performs model inference for DNNs.

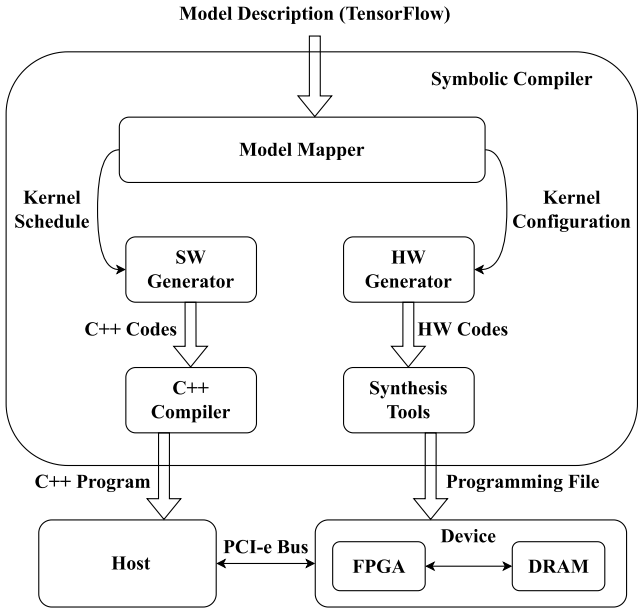


FIGURE 26. FP-DNN framework, adopted from [95].

Umuroglu et al. [204] proposed FINN, a framework that maps trained Binarized Neural Networks (BNNs) onto an FPGA. FINN generates a synthesizable C++ network description of a flexible heterogeneous streaming architecture. The architecture mainly contains pipelined compute engines that communicate via on-chip data streams. Each BNN layer has been implemented using dedicated compute engines with 1-bit values for FMs and weights. To evaluate FINN, the authors implemented CNV, a convolutional network topology inspired by BinaryNet [68] and VGG-16 [192], on a Xilinx Zynq-7000 FPGA board running at 200 MHz to accelerate BNN inference.

Guo et al. [96] proposed a flexible and programmable CNN accelerator, referred to as *Angle-Eye*, together with the compilation tool and the data quantization scheme. The data quantization scheme can be used to reduce the bit-width down to 8-bit with insignificant accuracy loss. The compilation tool is responsible for mapping a given CNN model efficiently onto the hardware architecture. The proposed accelerator supports the acceleration of various CNNs on different FPGA platforms. The overall architecture of the Angel-Eye is shown in Fig. 27. Angle-Eye accelerator mainly consists of a PE array, controller, on-chip buffer, and external memory. The PE array is used to perform the convolution operations, and it supports three levels of parallelism: input channel parallelism, kernel-level parallelism, and output channel parallelism. The on-chip buffer can isolate the PE array from external memory, allowing simultaneous convolution and data I/O operations. All network parameters and the results of each layer can be saved to external memory. The controller is responsible for receiving, decoding, and issuing instructions to the other three components and monitoring

each component's work status. Angle-Eye accelerator is implemented on the Zynq XC7Z045 platform.

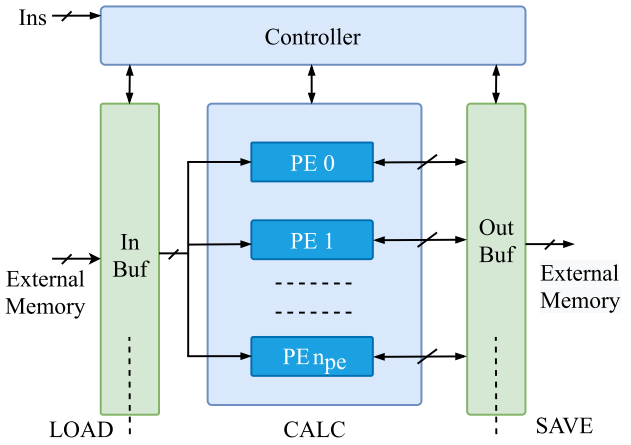


FIGURE 27. Angel-Eye accelerator architecture, adopted from [96].

Zhang et al. [223] proposed a software/hardware co-design library called *Caffeine* to accelerate CNNs efficiently on FPGAs. The authors developed a uniform convolutional matrix-multiplication representation for both convolutional and fully connected layers. Caffeine synthesizes Caffe models comprising convolutional layers and fully connected layers for FPGAs. The Caffeine framework effectively handles weights and biases reconfiguration in off-chip DRAM to maximize the underlying memory bandwidth utilization. The authors integrated the Caffeine with a deep learning framework Caffe and implemented AlexNet and VGG networks on multiple FPGA platforms, viz., Xilinx KU060 FPGA board, and Virtex7 690t FPGA board. Caffeine achieved better energy efficiency than 12-core CPU and GPU by 43.5 × and 1.5 ×, respectively.

Ghaffari et al. [90] developed a general framework called *CNN2Gate*, which allows mapping CNN models on FPGAs with automated design space exploration. The CNN2Gate overall architecture consists of an Open Neural Network eXchange (ONNX) format parser, a design-space exploration module, and leverages automated high-level synthesis is shown in Fig. 28. CNN2gate can parse CNN models using ONNX parser from several popular high-level machine learning libraries, such as Caffe2, Keras, TensorFlow, etc. The computation flow of network layers and their weights and biases are retrieved in CNN2Gate, and a fixed-point quantization is used. To undertake design space exploration for deeply pipeline OpenCL kernels of CNN, the authors used time-limited reinforcement learning.

Xilinx Vitis AI [32] is a framework for implementing deep learning inference on Xilinx FPGAs and SoCs. It uses an Intellectual Property (IP) core called the Deep Learning Processor Unit (DPU) to implement ample essential functions of deep learning on FPGAs, see Fig. 29. Xilinx Inc. released the DPU, a programmable engine designed for DNNs. Xilinx Vitis AI framework enables the compression of DNN models without sacrificing accuracy and compiling DNN models

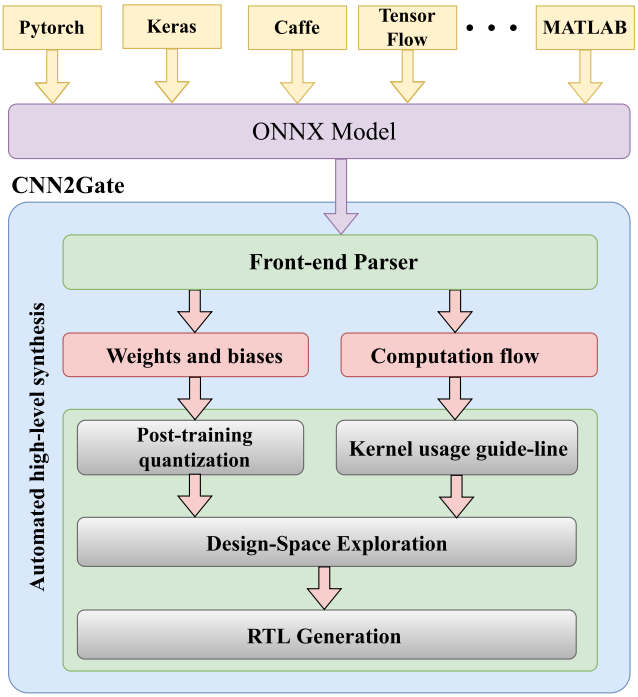


FIGURE 28. CNN2Gate, adopted from [90].

into DPU instruction code before deploying them to the target DPU platform. For efficient DNNs implementations, the Xilinx DPU offers a tailored and scalable overlay with ISA architecture. Xilinx Vitis AI framework supports various frameworks such as Caffe, PyTorch, TensorFlow, etc., and efficiently implements deep learning tasks, CNN, and RNN, see Fig. 29. The internal architecture of the DPU contains an *Instruction Unit (IU)*, a *Compute Array (CA)*, and a *Global Memory Pool (GMP)*. The IU fetches the DPU instructions associated with the model, decodes it, and drives the PEs present in compute array. It also manages the data/instructions transfer among the PEs and the memory. The GMP acts as buffer for the input and output data as well as intermediate output from the DPU, resulting in high throughput [113]. The DPU can be configured to meet the requirements of a specific CNN architecture, and the Vitis AI stack contains all the necessary libraries to generate the instructions for the DPU. The development flow is described in Fig. 29, where a trained model is compiled using the Vitis AI compiler. The Vitis AI tools provide a model quantizer to reduce the precision of weights without losing accuracy. An Xmodel file is generated by the Vitis Compiler consisting of domain-specific instructions for the DPU unit, which are used to configure the DPU. During inference, a Python script running on the PS acts as the interface, and it is responsible for transferring the data from the on-chip memory to the DPU memory buffers. Examples of CNNs that have been implemented using DPU include, but are not limited to, VGG, ResNet, GoogLeNet, YOLO, SSD, MobileNet, and FPN. Table 4 summarizes the reviewed FPGA-based accelerator frameworks for the implementation of DNNs.

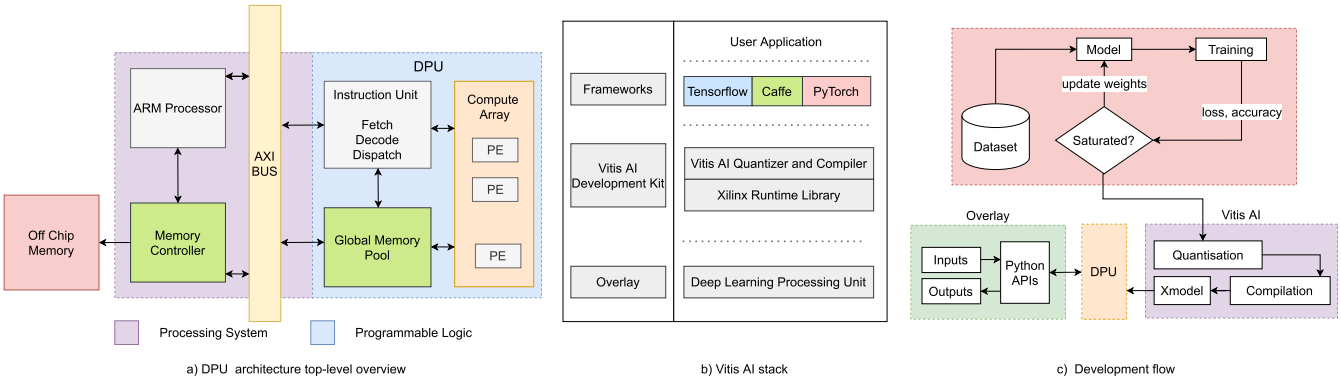


FIGURE 29. DPU architecture overview, adopted from [227], Vitis AI stack, and development flow.

TABLE 4. Summary of FPGA-based accelerator frameworks.

Framework Name	Year	DNN Type	Interface
Xilinx Vitis AI [32]	2022	CNN, RNN	Caffe, PyTorch, TensorFlow,
CNN2Gate [90]	2020	Alexnet, VGG-16	Caffe2, Keras, TensorFlow
Caffeine [223]	2019	Alexnet, VGG-16	Caffe
Angle-Eye [96]	2018	VGG-16	Caffe
FP-DNN [95]	2017	VGG-19, Res-152	TensorFlow
FINN [204]	2017	CNV	Caffe
DNNWeaver [187]	2016	LeNet, Siamese	Caffe
DeepBurning [211]	2016	Alexnet, NiN	Caffe
fpgaConvNet [206]	2016	CNN	Caffe, Torch

IV. ASIC BASED ACCELERATORS

Application Specific Integrated Circuit (ASIC) is a powerful platform to accelerate the DNNs. ASICs are customized chips designed for a specific application. They are smaller in size, consume less power, and provide higher speeds, making them suitable solutions for DNN acceleration [76]. ASIC based hardware accelerators have limited computing resources, memory resources, and I/O bandwidths compared with GPU based accelerators, but they can achieve moderate performance and consume less power [165]. Furthermore, ASIC exhibits the best computation speed and energy efficiency than GPU and FPGA at the cost of reconfigurability. Many researchers are focused on building custom ASICs for accelerating CNNs inference workloads to achieve the best performance and energy efficiency. In this section, we would like to review the recent ASIC-based DNN accelerators.

There are three broad types of ASIC-based DNN accelerators depending on how the architecture has been optimized/designed: ALU (Arithmetic Logical Unit), Dataflow, and Sparsity-based accelerators. The main building block, the MAC unit (or an array of MAC units), in ALU-based accelerators is modified to have ample computational resources and flexibility to obtain the best performance with varying bit accuracy. In dataflow-based accelerators, the activations, weights, and partial sums are managed to reduce the energy needed to move data within the chip and achieve high arithmetic intensity. In Sparsity-based accelerators, the unstructured sparse data is handled in such a way that the matrix multiplication units (2-D array of MAC units) can prevent zero multiplications. The following sections provide a comprehensive overview of ALU, Dataflow, and Sparsity-based accelerators.

A. ALU BASED ACCELERATORS

NeuFlow is the ASIC based CNN accelerator presented in [170] to accelerate the NNs and other ML algorithms. The architecture of the proposed accelerator is the same as the accelerator discussed in [84] and shown in Fig. 14, but is implemented using IBM 45 nm Silicon-On-Insulator (SOI) process. The NeuFlow accelerator uses a compiler named luaFlow to process CNNs. The luaFlow compiler converts high-level data flow graph representations of deep learning algorithms in the Torch5 environment into machine code for Neuflow. The proposed architecture provides higher power efficiency and is suitable for vision-based applications, such as autonomous vehicle navigation, driving assistance, etc. The proposed architecture achieves the maximum throughput of 320 GOPS with a power consumption of 0.6 W; in contrast, the NeuFlow architecture implemented on Xilinx Virtex6 FPGA presented in [84] has a maximum throughput of 16 GOPS with power consumption of 10 W.

Chen et al. [53] proposed the ASIC-based hardware accelerator, also called DianNao, to accelerate the large-scale CNNs and DNNs. The proposed architecture provides the quick and energy-efficient execution of the inference of large-scale CNNs and DNNs. The architecture contains the Neural Functional Unit (NFU), buffers, and control processor (CP), see Fig. 30. The NFU module is used to

perform the computations needed to determine the output of the neuron in the fashion, i. e., in the first stage, NFU performs the multiplication of input neuron values with weight coefficients. In the second stage, NFU accumulates these products using the adder trees. In the third stage, NFU calculates the activation functions. Buffers are used to store the input/output neuron values and weights. The proposed architecture contains three buffers viz., an input buffer to store the input neuron values (NBin), an output buffer to store the output neuron values (NBout), and a third buffer to store the weights (SB). Different computational operators are invoked in each stage depending on the type of the layer (convolution, activation function, pooling, etc.) For architecture exploration, the author developed a C++ simulator that evaluates execution time and serves as a specification for the Verilog implementation. The Verilog version of the accelerator is synthesized using Synopsys' design compiler, and the generated design is placed and routed by Synopsys' ICC compiler. The design is simulated using Synopsys' VCS, while PrimeTime PX is used to determine the power. The proposed architecture was implemented using 65 nm CMOS technology. The experimental results show that DianNao achieves an average performance of 452 GOPS with 485 mW of power consumption. The proposed accelerator has scalability issues due to the bandwidth constraints of the memory system. The DaDianNao accelerator [54] and [146] are extensions of the DianNao accelerator [53]. DaDianNao has enough on-chip memory to hold all of CNN's weights. DaDianNao also uses 16-bit fixed-point representation in the inference process like DianNao. However, it is implemented using 28 nm CMOS technology. The design compiler synthesizes the Verilog version of the DaDianNao accelerator, and the ICC compiler is used to generate the layout. The energy, area, and critical path are obtained after the layout. The design is simulated using VCS, while PrimeTime PX is used to determine the power. The proposed architecture uses eDRAM to store all the data related to a CNN, i. e., input feature maps, weight kernels, output kernels, etc. The DaDianNao accelerator gives better performance while accelerating the CNNs, but provides moderate to low performance while accelerating large-scale CNNs.

Liu et al. [141] proposed the machine learning accelerator referred to as PuDianNao, that supports multiple machine learning scenarios (e.g., regression, classification, and clustering) as well as many machine learning techniques, including k-means, k-nearest neighbors, linear regression, classification tree, naive bayes, support vector machine, and DNNs. The PuDianNao mainly contains various Functional Units (FUs) and three types of data buffers: ColdBuf, HotBuf, and OutputBuf, an instruction buffer (InstBuf), and a DMA, and a control module, see Fig. 31. The FU contains a *Machine Learning Functional Unit (MLU)* and an *Arithmetic Logic Unit (ALU)*. The MLU can be used to perform several computational primitives, including dot product, counting, sorting, distance calculations, non-linear functions, for instance, sigmoid and so on. The ALU has an

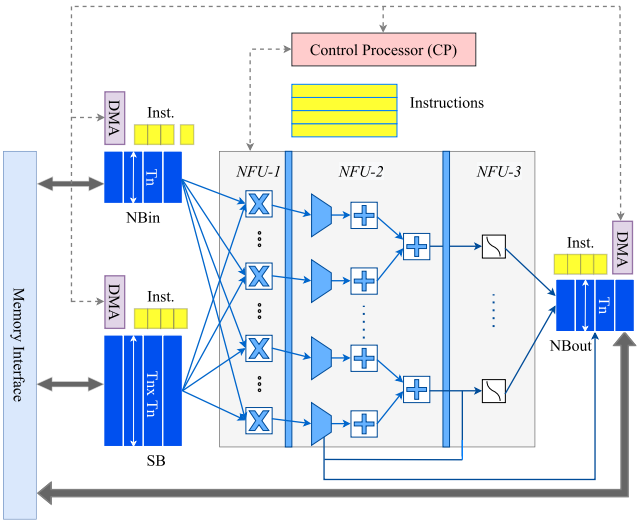


FIGURE 30. DianNao accelerator architecture, adopted from [53].

adder, divider, multiplier, and converters for the 16-bit float to 32-bit float and 32-bit float to 16-bit float. It may also be used to compute estimates using the Taylor expansion of log (1-x). HotBuf (8 KB) and ColdBuf (16 KB) store the input data with short and longer reuse distances, respectively. OutputBuf (8 KB) is used to store the output data or intermediate results. The authors implemented an in-house C simulator of PuDianNao; it acts as a specification for the Verilog implementation and also measures the performance of PuDianNao on large-scale datasets. The design compiler synthesizes the design, and the ICC compiler is used to generate the layout. The energy, area, and critical path are obtained after the layout. The design is simulated using Synopsys VCS, and PrimeTime PX is used to determine the power using the Value Change Dump (VCD) file. The proposed architecture has been implemented using TSMC 65 nm CMOS technology.

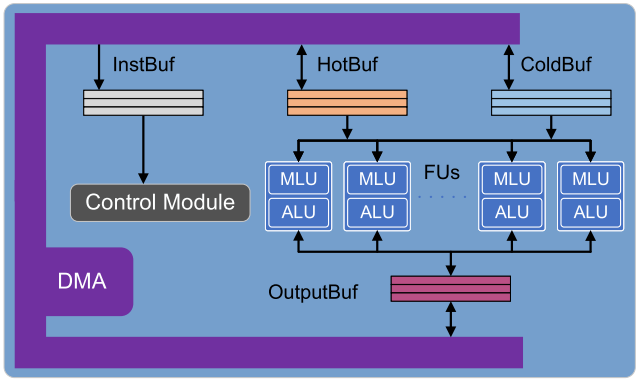


FIGURE 31. PuDianNao accelerator architecture, adopted from [141].

Du et al. [78] proposed a CNN accelerator referred to as ShiDianNao to improve the energy efficiency and scalability of DianNao [53] design discussed above. The ShiDianNao accelerator does not access the main memory while executing a CNN and achieves more energy efficiency

compared to DianNao. The design is implemented in Verilog and synthesized by the design compiler, and IC compiler is used to place and route the synthesized design. The energy cost of DRAM accesses is calculated using CACTI 6.0 [159]. The ShiDianNao accelerator will not support the acceleration of large-scale CNNs. The ShiDianNao accelerator is implemented using 65 nm CMOS technology. DianNao [53], DaDianNao [54], [146], PuDianNao [141], and ShiDianNao [78] are not built utilizing reconfigurable hardware, hence they cannot be adapted to changing application demands such as NN sizes.

Lu et al. [145] proposed a flexible dataflow architecture called FlexFlow to accelerate the CNNs, exploiting all kinds of parallelisms viz., inter-kernel, intra-kernel, and inter-output on a two-dimensional array of PEs. FlexFlow has additional interconnections between on-chip memories and PEs, which provides the flexibility to fetch any neuron from any feature map. The proposed accelerator minimizes the interconnections between the PEs at the cost of energy because of data movement from on-chip memory to PEs. In FlexFlow, all the PEs are operated in parallel, therefore, helping in improving the overall throughput. The proposed architecture has high scalability and supports different sizes of CNNs with stable resource utilization. FlexFlow only implements CNNs and is confined to within a layer rather than across layers. The design is simulated, synthesized, placed & routed using Synopsys' tools. The FlexFlow accelerator is implemented using TSMC 65 nm technology.

Hardik et al. [188] developed a bit-level dynamically composable architecture called *Bit Fusion* for accelerating DNNs. Bit fusion mainly consists of an array of bit-level computation elements, called BitBricks, that dynamically fuse to match the bit width of individual DNN layers and execute DNN operations with the required bit width, without any loss of accuracy. Furthermore, Bit Fusion supports the multiplication of 2, 4, 8, and 16 bits spatially. Bit Fusion decomposes a 16-bit multiplication into multiple 2-bit multiplications to achieve the flexibility to efficiently map various layers of CNN with different bit widths and minimize the computation and the communication with no loss of accuracy. Bit Fusion architecture comes with an Instruction Set Architecture (ISA) that minimizes the data transfer and maximizes the parallelism in computations. The proposed design is implemented in Verilog and is synthesized using the Design Compiler, which estimates the area, frequency, and power. The proposed accelerator architecture is implemented on 45 nm CMOS technology. Bit Fusion accelerator achieves 5.1× energy saving and 3.9× speedup over Eyeriss accelerator.

Shin et al. [190] proposed Deep Neural Processing Unit (DNPU) architecture to process CNNs and Recurrent Neural Networks (RNNs). DNPU is a SIMD MAC-based CNN/RNN accelerator that uses dynamic precision control to minimize kernel data size. DNPU consists of a convolutional layer processor (CP), a fully connected and RNN-LSTM layer processor (FRP), and a RISC controller. CP performs convolutional operations, and FRP performs matrix multiplication operations. DNPU is the first CNN/RNN accelerator with the highest energy efficiency of 8.1 TOPS/W on 65 nm CMOS technology. DNPU has some limitations; for instance, its area limits the number of processing elements (PEs) for convolutional layers (CL) and recurrent layers (CL). As a result, performance was sub-optimal in cases that just required CLs or RLs. Furthermore, DNPU only supports a limited number of weight-bit precisions, such as 4 bits, 8 bits, or 16 bits. Lee et al. [133] proposed the Unified Neural Processing Unit (UNPU) architecture to process CNNs and RNNs. UNPU contains a bit-serial MAC unit to perform the required computations. UNPU supports CLs, RLs, and fully connected layers (FCLs) with fully-variable weight bit-precision from 1 to 16 bits. UNPU achieves an energy efficiency of 3.08, 11.6, and 50.6 TOPS/W for the case of 16-bit, 4-bit, and 1-bit weights, respectively. UNPU achieves 1.43× higher energy efficiency than the DNPU for convolutional layers with 4-bit weights.

B. DATAFLOW BASED ACCELERATOR

The accelerators based on dataflow put a special emphasis on data management to minimize off-chip memory reads/writes. When it is feasible, reusing parameters between layers can enhance dataflow. For instance, in a convolutional layer, both activations and weights can be reused. In a fully connected layer, each neuron has a unique set of weights; as a result, weights cannot be reused, but input data may. In order to minimize data movement between a computing unit and higher-level memory, the reusable parameters are kept in local registers.

Cavigelli et al. [50] proposed the Origami CNN accelerator, which is scalable to different network sizes. The proposed architecture uses the Weight Stationary (WS) dataflow to improve energy efficiency during the acceleration process. WS dataflow minimizes energy consumption by maximizing the access of weight coefficients. WS dataflow used in the Origami maximizes the convolution and filter reuse of weights. The proposed accelerator was implemented using UMC 65 nm CMOS technology and having a core area of 3.09 mm^2. The proposed CNN accelerator can achieve a throughput of 274 GOPS and a power efficiency of 369 GOPS/W with an external memory bandwidth of 525 MB/S full-duplex. The proposed architecture is only used to perform the convolution operation and is unsuitable for implementing the fully connected layer operations.

Eyeriss [56] is an ASIC based CNN accelerator that uses a row-stationary (RS) dataflow that minimizes data movement energy consumption on a spatial computing architecture. RS dataflow is adaptable to various CNN shapes and minimizes energy consumption by reusing the filter coefficients and input feature maps. The proposed accelerator mainly contains a 12 × 14 PE array, feature map compression units, and a 108 KB global buffer; ReLU as shown in Fig. 32. The global buffer enables the reuse of loaded data from off-chip DRAM and the generated results by PEs and is also

responsible for returning the final results to the off-chip DRAM. In the Eyeriss accelerator, the PEs are connected via a Network on Chip (NoC). The NoC used in Eyeriss only supports multi-cast. The authors proposed an analysis framework for calculating the energy efficiency of various CNN data flows under the same hardware constraints. The proposed accelerator is implemented using 65 nm CMOS technology.

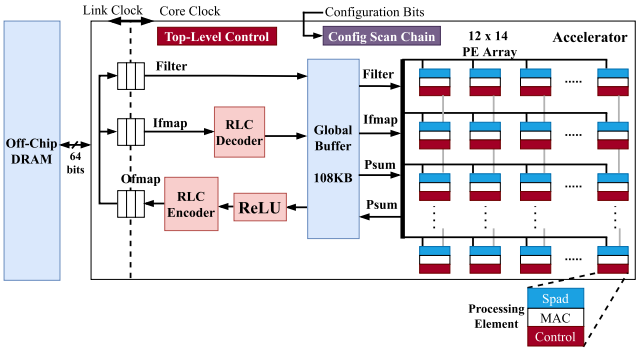


FIGURE 32. Eyeriss DNN accelerator, adopted from [56].

Chen et al. [57] proposed a DNN accelerator architecture referred to as Eyeriss v2 to accelerate compact and sparse DNNs. Like Eyeriss [56], Eyeriss v2 is composed of an array of PEs to perform MAC operations, global buffers, and local scratchpad (SPad) memory to support data reuse. In the Eyeriss v2 accelerator, PEs and global buffers (GLB) are grouped into clusters to support a flexible Network On Chip (NoC), as shown in Fig. 33. The main difference between Eyeriss and Eyeriss v2 is that Eyeriss v2 uses a hierarchical mesh NoC (HM-NoC) to connect the global buffers to the PEs; in contrast, the Eyeriss uses multicast NoC between the global buffer and PEs. Furthermore, the Eyeriss v2 accelerator uses separate NoCs to transfer the input activations, weights, and partial sums between the global buffer and PEs. The hierarchical mesh NoC used in the Eyeriss v2 accelerator supports unicast, multicast, and broadcast. The HM-NoC can be configured into various modes ranging from high data reuse to high bandwidth. The proposed architecture supports various CNN layer dimensions and sizes because of the flexible hierarchical mesh NoC. The authors proposed an analysis framework named EYEXAM for evaluating the performance of various CNN dataflows. The Eyeriss v2 accelerator has higher hardware utilization than Eyeriss but has a large area overhead. The experimental results show that Eyeriss v2 reaches 11.3× and 42.5× improvement in energy efficiency and throughput, respectively, with the sparse AlexNet, compared to Eyeriss running with the AlexNet. It also achieves 2.5× and 12.6× improvement in energy efficiency and throughput, respectively, with sparse MobileNet compared to Eyeriss running with MobileNet.

Multiply-Accumulate Engine with Reconfigurable Interconnect (MAERI) is a DNN accelerator containing a set of configurable building blocks to support various CNN partitions and mapping by configuring the tiny switches

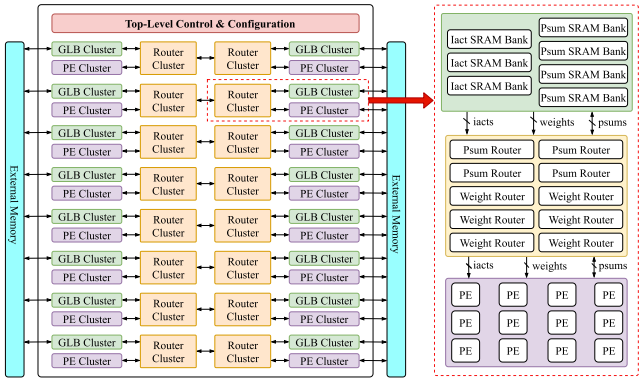


FIGURE 33. Eyeriss v2 top-level architecture, adopted from [57].

presented in [131]. MAERI contains a set of multiply adder computation units, each augmented with tiny configurable switches that can be configured to support various kinds of dataflows, see Fig. 34. The prefetch buffer stores the input activations, intermediate partial sums, weights, and output activations. Acceleration units mainly contain look-up tables (LUT) and perform activation functions. MAERI uses two configurable interconnect networks, namely, distributed network and augmented reduction network. To assist the effective mapping of the irregular dataflows and to provide high resource utilization, MAERI offers non-blocking communication via reconfigurable links with large bandwidth. The proposed accelerator can accelerate various operations viz., convolution, pooling, fully connected layer, and LSTM. The proposed accelerator also supports sparsity and cross-layer mapping. MAERI is implemented in Bluespec System Verilog (BSV) [163] and is synthesized with TSMC 28 nm standard cell and SRAM library at 200 MHz.

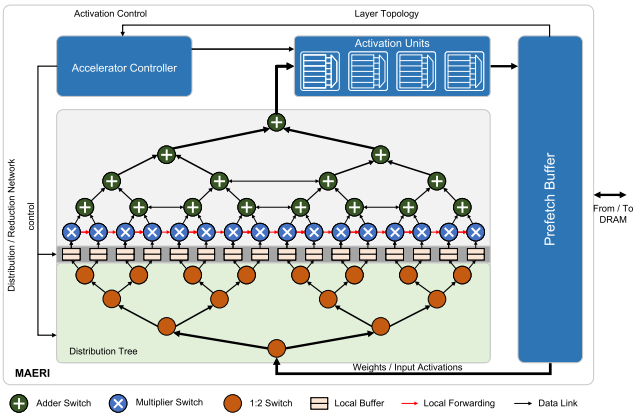


FIGURE 34. MAERI architecture, adopted from [57].

Tensor Processing Unit (TPU) is developed by Google in order to implement machine learning algorithms. A matrix multiplication unit as a systolic array of 256 × 256 units is used in the TPU architecture [120]. Fig. 35 shows the block diagram of the TPU. The mentioned systolic array structure is basically built with weight-stationary dataflow and as a 2-D SIMD architecture. Extracting from the DRAM, the weights can then be stored in the weight FIFO (First-In,

First-Out) register. The results from the previous layers and the input activation function are stored in the unified local buffer. In order to perform a convolution operation on a matrix multiply unit, a systolic data setup block is used in order to rearrange the data. Efficient running of machine learning model tasks and inference tasks like search and image recognition, and language translation have been the focus of the first version of TPU, called TPU1. Since 2015, TPU1 has been operational in Google's data center. A second version TPU2, also called Cloud TPU is operational in data centers for the purpose of training and interference. Cloud TPU supports several frameworks, including TensorFlow, PyTorch, and JAX/FLAX.

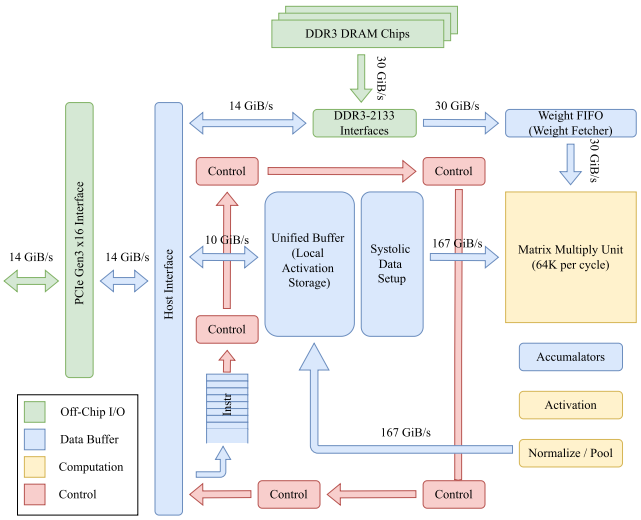


FIGURE 35. Block Diagram of TPU, adopted from [120].

C. SPARSITY BASED ACCELERATORS

The fraction of zeros in a CNN layer's weights and input activation matrices is called sparsity. Since multiplying by zero should produce a zero, there should be no effort required. As a result, typical layers can cut work by a factor of four, and in some instances, by a factor of ten. Also, the addition is not needed because the zero products won't add anything to the total of which they are a part. Moreover, data with many zeros can be compressed—these traits, when combined, open up a lot of possibilities for improvement. This section provides a comprehensive overview of accelerators that explore sparsity.

A CNN accelerator referred to as Sparse CNN (SCNN) is presented in [167] for inference of CNNs. SCNN employs a novel dataflow referred to as sparse *Planar-Tiled Input-Stationary Cartesian Product (PT-IS-CP-sparse)* dataflow that maximizes the reuse of activations and weights and removes needless data transfers and reduces storage and power requirements. The dataflow used in SCNN eliminates all multiplications with a zero and keeps both activations and weights in compressed form. SCNN mainly contains an array of processing elements arranged in a 2-D fashion with systolic connections to transfer partial sums. The proposed dataflow efficiently delivers activations and weights to the multiplier array to perform the required MAC operations. SCNN exploits all three kinds of parallelisms viz., inter-kernel, intra-kernel, and inter-output. SCNN requires additional optimization circuitry to implement the fully connected layer operations. SCNN improves the performance by skipping the zeros in the input feature maps and weights. SCNN is implemented in system C and Catapult High-Level Synthesis (HLS) [30] tool is used to generate the Verilog RTL. Synopsys Design Compiler synthesizes the Verilog version of the design. SCNN is implemented using TSMC 16 nm FinFET technology.

Eyeriss [56] also looked into input sparsity as a way to save energy. The gating mechanism deactivates MAC units that correspond to zero inputs. Gating saves energy while not increasing throughput. With sparse models, The processing speed and energy efficiency of Eyeriss V2 [57] have improved due to its ability to process sparse data directly in compressed format for both the weights and activations.

Zhang et al. [225] developed Sparse Neural Acceleration Processor (SNAP) to exploit unstructured sparsity in DNNs. To ensure that data is distributed evenly throughout the MAC units, SNAP employs parallel associative search. SNAP is fabricated using 16 nm CMOS technology and achieves a peak energy efficiency of 21.55 TOPS/W (FP16) for CONV layers with 10% weight and activation density.

Lee et al. [135] proposed an energy-efficient on-chip accelerator called LNPU for sparse DNN model learning. In the LNPU accelerator, Sparsity is exploited with intra-channel as well as inter-channel accumulation. The input load buffer module of the LNPU evenly distributes workload among the PEs while considering irregular sparsity. LNPU uses the fine-grained mixed precision (FGMP) of FP8-FP16 that optimizes data precision while maintaining training accuracy. LNPU maintains an average hardware utilization of 100%. LNPU is fabricated using 65 nm CMOS technology and has an energy efficiency of 3.48 TFLOPS/W (FP8) at 0% sparsity and 25.3 TFLOPS/W (FP8) at 90% sparsity.

SIGMA is a scalable and flexible accelerator proposed in [176] to implement large, irregular, and sparse general matrix-matrix multiplications (GEMMs). The basic building block in SIGMA is the Flexible Dot Product Engine (Flex-DPE). All the Flex-DPE modules can be interconnected via simple NoC. In SIGMA, all the Flex-DPE multipliers are arranged in a 1-D fashion, and it performs the multiple variable-sized dot-products in parallel. SIGMA uses scalable interconnects to efficiently map the GEMMs of different dimensions and sparsity levels to the PEs. SIGMA outperforms systolic array architectures by 5.7× for irregular sparse matrices. SIGMA is implemented using the 28 nm CMOS technology and achieves a throughput of 10.8 TFLOPS with a power dissipation of 22.33 W.

Zhang et al. [224] proposed an accelerator called GAMMA to perform the Sparse matrix-sparse matrix multiplication (spMspM) operations. The proposed accelerator uses Gustavson's algorithm [99] to compute the spMspM operations. GAMMA accelerator mainly consists of an array of

processing elements(PEs), on-chip storage referred to as FiberCache, and a scheduler, as shown in Fig. 36. The PEs are used to perform the required spMspM operations that combine sparse input rows to produce each output row. FiberCache is a specialized memory structure that stores the non-zero elements and their coordinates. The scheduler distributes computational workloads among PEs to maximize resource efficiency while reducing unnecessary access to shared memory. GAMMA is implemented using 45nm CMOS technology.

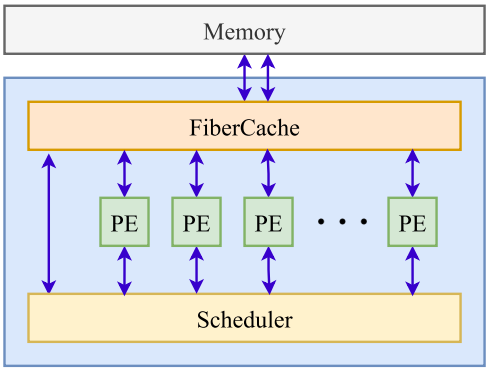


FIGURE 36. Block Diagram of GAMMA, adopted from [224].

We summarized the reviewed ASIC-based accelerators for DNN in Table 5. For each accelerator, we list the year the accelerator was introduced, the process technology, the clock frequency, the dataflow, the architecture type, the power dissipation, the area, the performance in GOPS, and finally, the power efficiency. Fig. 37 shows the plots of various metrics, such as power, throughput, area, and power efficiency of ASIC-based accelerators.

V. GPU BASED ACCELERATORS

Over the last few decades, Graphics Processing Units (GPUs) are widely used in training DL algorithms or CNNs for face recognition [109], object detection [220], [226], data mining [88], and other AI applications. GPU supports parallelism due to lots of parallel cores in the architecture and offers significant computation speed. GPU exploits large degrees of data-level parallelism in the applications through the Single Instruction Multiple Thread (SIMT) execution models. The high computational capacity of the GPUs makes them the primary choice for DNN acceleration. In this section, we would like to review some of the recent GPU-based DNN accelerators.

The study of implementing a standard backpropagation algorithm for training multiple perceptrons simultaneously on GPU using NVIDIA CUDA technology is presented in [100]. For a given program, GPU-based implementation on NVIDIA GTX 260 GPU achieves 50× to 150× speedup compared to the CPU-based implementation. A neurally accelerated architecture for GPU, called NGPU (neurally accelerated GPU) is presented in [218] to enable scalable integration of neural acceleration with a large number of GPU cores. The proposed architecture brings the neural and GPU accelerators together without hampering the SIMT execution model. NGPU provides significant energy and performance benefits at the cost of reasonably low hardware overhead. NGPU achieves 2.44× average speedup and 2.8× average energy reduction compared to the baseline GPU architecture across different sets of benchmarks.

Danial et al. [196] presented a framework for accelerating the training and classification of arbitrary CNNs on the GPU. The proposed method improves the performance by moving the computationally intensive tasks of a CNN to the GPU. Training and classification of CNN on the GPU performs 2 to 24 times faster than on the CPU based on the network topology. Li et al. [139] proposed an efficient GPU implementation to accelerate the training process of large-scale Recurrent Neural Networks (RNN). When compared to the CPU-based solution with Intel's Math Kernel Library (MKL), the proposed method yields a speedup of 2 to 11 times. Kim et al. [127] proposed a new memory management scheme to enhance the overall GPU memory utilization in multi-GPU systems for deep learning algorithms acceleration. The authors extended the concept of vDNN to a multi-GPU environment employing PCIe-bus, where vDNN [179] virtualizes the GPU and memory of the CPU so that it can be used simultaneously to train DL algorithms in a hybrid fashion. The suggested memory scheme increases batch size by 60% in multi-GPU systems and enhances training throughput by 46.6%. High-performance GPU dedicated architecture referred to as *TResNet* is presented in [180] to accelerate CNNs. The proposed architecture effectively utilizes GPU resources and achieves better accuracy and efficiency.

Nvidia GPUs are the most popular for Deep Learning (DL) implementations. Table 6 lists the accelerators that Nvidia has released, which are used for the inference and training of deep learning (DL) algorithms and have both a Central Processing Unit (CPU) and a GPU integrated on a single chip.

VI. CGRA-BASED ACCELERATORS

Coarse Grain Reconfigurable Architectures (CGRAs) primarily consist of an array of Processing Elements (PEs) connected using reconfigurable interconnects. When compared to FPGAs, CGRAs often have a shorter reconfiguration time. CGRAs have emerged as a popular option for real-time computing due to their low power consumption, high efficiency, fast reconfiguration time, and ability to perform both spatial and temporal calculations. In recent years, CGRAs have become increasingly significant in accelerating DNNs, particularly CNNs, thanks to their ability to combine FPGAs' flexibility with ASICs' efficiency. In this section, we would like to review some of the recent CGRA-based DNN accelerators.

Jafri et al. [118] proposed a CGRA-based accelerator named NeuroCGRA to realize neural networks and digital signal processing applications. The authors have opted to investigate the viability of deploying neural networks on an actual CGRA using a Dynamically Reconfigurable Resource

TABLE 5. Summary of ASIC-based accelerators.

Accelerator Name	Year	Process Technology	Frequency (MHz)	Dataflow	Architecture	Power Dissipation (W)	Area (mm^2)	Performance (GOPS)	Power Efficiency (GOPS/W)
NeuFlow [170]	2012	IBM 45 nm SOI	400	Flexible	2-D systolic	0.6	12.5	320	490
DianNao [53]	2014	TSMC 65 nm CMOS	980	NLR	1-D array	0.485	3.02	452	932
DaDianNao [54]	2014	TSMC 28 nm CMOS	606	NLR	1-D array	15.97	**0.78**	5580	350
PuDianNao [141]	2015	TSMC 65 nm CMOS	**1000**	NLR	1-D array	0.596	3.51	1056	1752
ShiDianNao [79]	2015	TSMC 65 nm CMOS	**1000**	OS	2-D matrix	0.32	4.86	194	606
Origami [50]	2015	UMC 65 nm CMOS	700	WS	2-D array	0.744	3.09	274	369
FlexFlow [145]	2017	TSMC 65 nm CMOS	**1000**	flexible	2-D matrix	6.8	3.89	420	500
SCNN [167]	2017	**TSMC 16 nm FinFET**	**1000**	N/A	2-D systolic	N/A	7.9	2000	N/A
Eyeriss [56]	2017	TSMC 65 nm CMOS	200	RS	2-D array	**0.278** (for AlexNet)	12.25	N/A	N/A
TPU [120]	2017	28 nm CMOS	700	WS	2-D systolic	N/A	< 331	N/A	N/A
Bit Fusion [188]	2018	TSMC 45 nm CMOS	500	N/A	2-D array	0.895	5.87	N/A	N/A
MAERI [131]	2018	TSMC 28 nm CMOS	200	WS, RS, OS	2-D array	N/A	6	N/A	N/A
DNPU [190]	2018	65 nm IP8M CMOS	200	N/A	2-D array	**0.279**	16	300	**3900**
UNPU [133]	2018	65 nm IP8M logic CMOS	200	N/A	2-D array	0.297	16	345.6	3080
LNPU [135]	2019	65 nm IP8M CMOS	200	N/A	2-D array	0.367	16	> 300	3480
SNAP [225]	2019	**16 nm CMOS**	33-480	N/A	2-D array	0.0163-0.364	2.4	N/A	3860
Eyeriss2 [57]	2019	TSMC 65 nm CMOS	200	RS	2-D array	N/A	N/A	153.6	253.2 (for AlexNet)
SIGMA [176]	2020	28 nm CMOS	500	WS	2-D array	22.33	65.1	**10800**	480
GAMMA [224]	2021	28 nm CMOS	**1000**	WS, RS, OS	1-D array	N/A	30.6	N/A	N/A

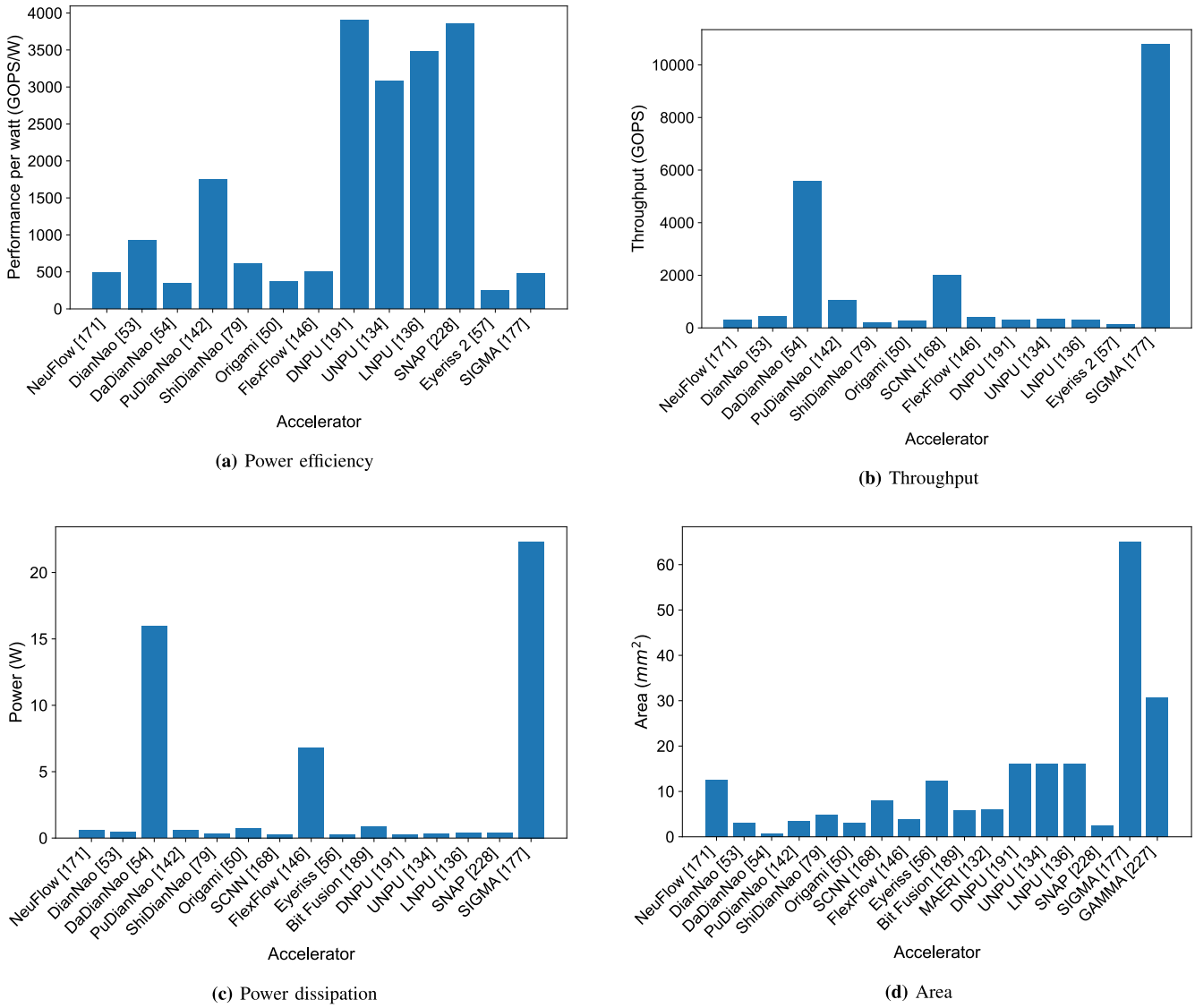
TABLE 6. GPU-based accelerators developed by Nvidia.

Accelerator	DNN Model	Precision	Memory	Memory Bandwidth (GB/s)	Thermal Design Power (W)	Performance (GOPS)	Applications
Jetson Xavier NX [26]	ResNext-50	int8	16 GB	59.7	10	21	Image Classification, object detection, natural language processing
Jetson AGX Xavier [28]	ResNext-50	int8	32 GB	136.5	10	32	Image Classification, object detection, natural language processing
T4 [31]	ResNet-50	int8	16 GB	320	70	130	Natural language interpretation
V100 [74] [121]	ResNet-50	fp32	16 GB	900	300	15700	Natural language interpretation
A100 [198] [63]	ResNext-50	fp32	**40 GB**	**1555**	250	**19500**	High performance computing

Array (DRRA). DRRA mainly consists of four elements, viz., Data Path Units (DPUs), register files (Reg-files), Switch Boxes (SB), and sequencers, as shown in Fig. 38. The DPUs are the functional units that perform the required computations. The Reg-files store the data for the DPUs. Interconnectivity between various DRRA components is provided through SBs. The sequencers configure the DPU, switch boxes, and register files. Distributed Memory Architecture (DiMArch) is essentially a scratch pad providing enough data to the DRRA. The authors have embedded dedicated hardware, known as neuroDPU, with each DPU of DRRA to implement neural networks on it. The authors proposed a neural network translator that provides a framework for mapping neural networks onto CGRAs. The translator takes three inputs, viz., network model, weights, and network specifications, and generates three outputs: DPU, Reg-file, and SB instructions. NeuroCGRA is synthesized using 65nm technology running at a frequency of 500 MHz. A framework called FIST is presented in [162] that allows the NeuroCGRA [118] to realize both DSP applications and neural networks, depending on the target applications. The authors have implemented edge detection on DRRA using the proposed framework.

EMAX is an energy-efficient, low-power CGRA architecture with on-chip distributed memory proposed in [202] to implement CNNs. EMAX supports both CNN training and inference. EMAX is composed primarily of an array of PEs and an interconnection network, as shown in Fig. 39. Each PE is connected to its neighbors by local interconnections, and each row of the PE array has a shared bus. The results of calculations performed on the PEs are passed on to the PEs that exist in the next row. The PEs can access external memory (DRAM) via the memory interface. Each PE has two execution units that perform the arithmetic and logical operations. Each PE also has a local memory to store the required data, reducing the memory bandwidth pressure. Experimental results show that EMAX performs better than GPUs in terms of per memory bandwidth and per area.

A CGRA-based accelerator referred to as stream dual-track CGRA (SDT-CGRA), which targets the implementation of object inference algorithms, is presented in [83]. SDT-CGRA employs stream processing and uses both static and



(a) Power efficiency

(b) Throughput

(c) Power dissipation

(d) Area

FIGURE 37. **Performance metrics of ASIC-based accelerators.**

dynamic configurations for stream processing. The SDT-CGRA accelerator mainly contains an array of PEs known as reconfigurable cells (RS) and stream buffer units (SBUs), as shown in Fig. 40. The SDT-CGRA architecture is divided into two sections: global memory and computing array. The global memory section is dynamically configured and stores data streams. On the other hand, the computing array section operates in a static configuration mode. It comprises several RC columns and one special RC column. The Special RC is used for operations like power (represented as PRC in Fig. 40) and piece-wise functions (represented as IRC in Fig. 40). The crossbar switch serves as a bridge to connect the RC array and SBUs. Data can be transferred from off-chip memory to SBUs using the external direct memory access interface. Static and dynamic interfaces are used for static and dynamic configurations, respectively. The proposed SDT-CGRA is realized in Verilog HDL, and Synopsys design compiler is used to synthesize the design. The proposed SDT-CGRA is implemented using SMIC 55nm CMOS technology. Experimental results show that SDT-CGRA outperforms EMAX by three times in terms of operations per memory bandwidth.

In [110], the authors proposed mapping of CNNs onto Tightly Coupled Processor Array (TCPA) efficiently. TCPA belongs to the class of CGRA, containing an array of tightly coupled VLIW Processing Elements (PEs) [104]. TCPA offers multiple levels of parallelism, for instance, task-level, loop-level, iteration-level, instruction-level parallelism, etc. TCPAs are suited for accelerating computationally expensive nested loop programs exhibiting a high degree of parallelism, such as CNNs. CNN layers are based on matrix multiplications which can be written as 6-dimensional nested loops, making them suitable for acceleration. It was demonstrated that TCPAs use techniques such as loop

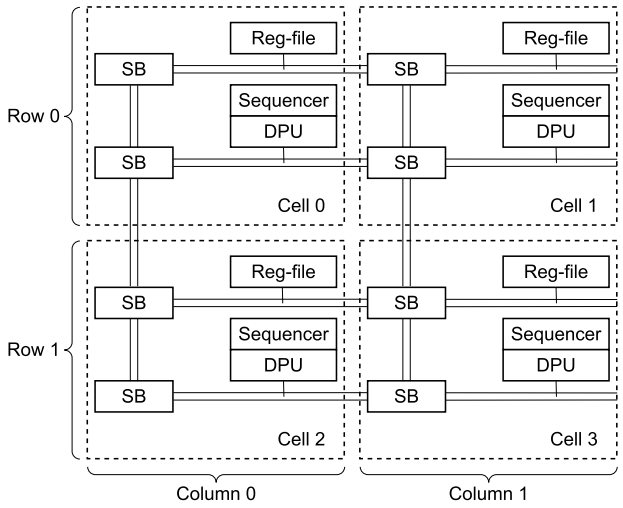


FIGURE 38. DRRA computation layer [118].

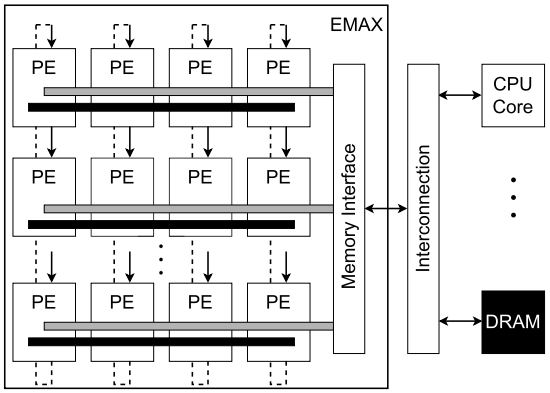


FIGURE 39. EMAX architecture, adopted from [202].

permutation, loop unrolling, and layer-parallel processing to exploit the parallelism offered by the TCPA architecture. Layer fusion allows the processing of multiple layers of CNN in the overlapped fashion [33], which was exploited by TCPA to save the intermediate memory needed between the layers. Loop permutation allows the computation of multiple convolution filters in an interspersed way. TCPA allows the parallel execution of multiple layers by different PEs. A CNN model for the MNIST benchmark on an array of size 4×4 was evaluated and the performance of the layer-parallel approach over layer-by-layer processing was compared.

A CGRA-based accelerator called Neural Processing CGRA (NP-CGRA) is presented in [134] to accelerate lightweight CNNs. The authors have proposed a set of extensions to the baseline CGRA [152] to improve the performance of CGRAs and to efficiently implement depth-wise convolution (DWC) and pointwise convolution (PWC). The authors have presented three architectural extensions: a crossbar-style memory bus, dual-mode MAC unit, and operand reuse network. The crossbar-style memory bus contains horizontal and vertical buses, and each bus is accessible to all the PEs connected to it. Dual-mode MAC unit works in MAC mode and MUL/ALU mode. The multiplication and accumulation operations are chained together in the MAC mode to realize the function. On the other hand, in the MUL/ALU mode, a PE can choose either a multiplication or an addition operation for each cycle. Operand reuse network offers input-to-input routing instead of output-to-input routing. The proposed NP-CGRA is realized in Verilog HDL, and Synopsys design compiler is used to synthesize the design. The proposed NP-CGRA is implemented using Samsung 65nm CMOS technology. Experimental results show that the area-delay product of NP-CGRA is 8-18 times better than that of baseline CGRA.

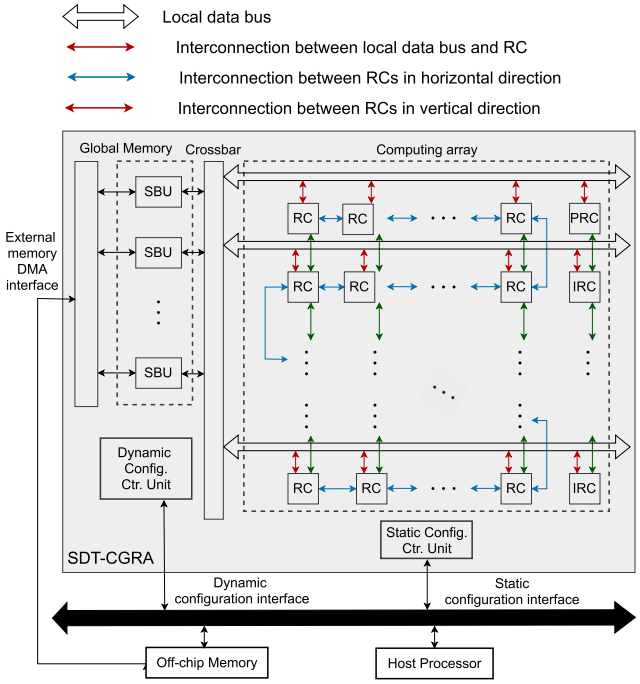
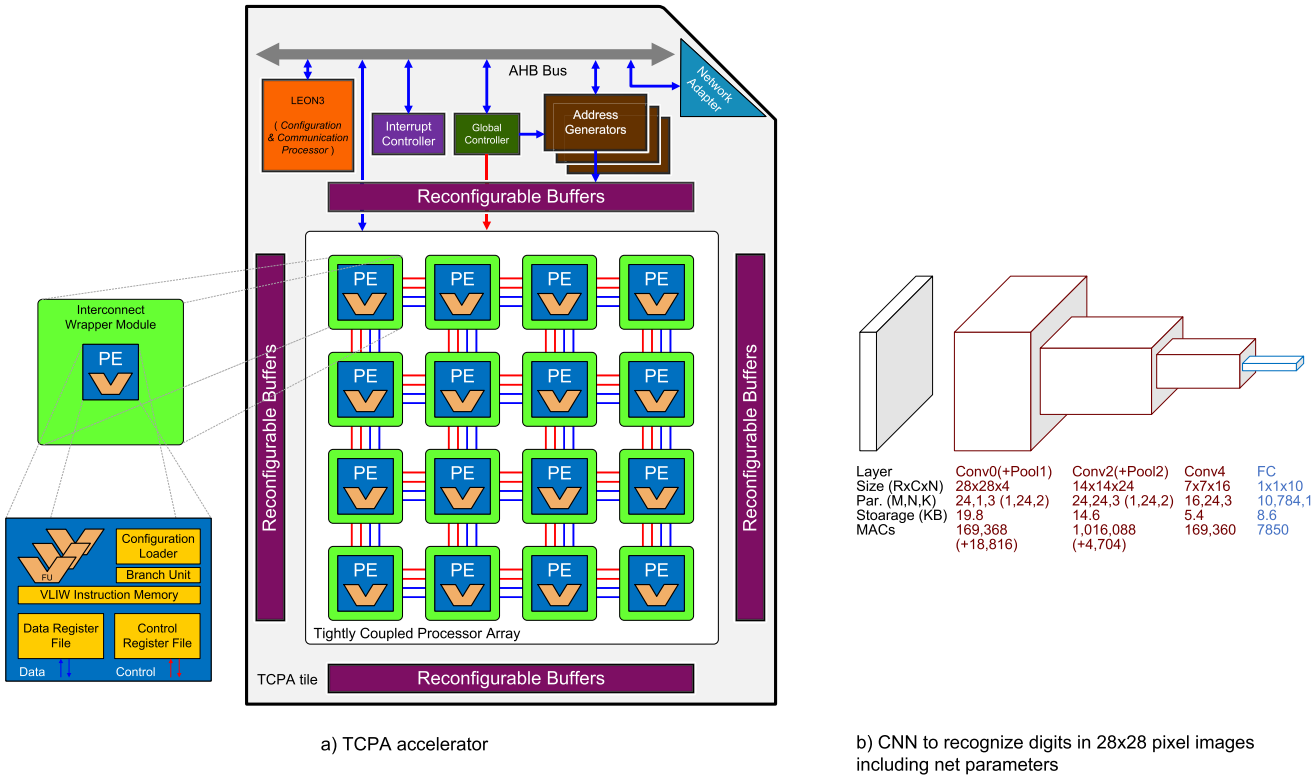


FIGURE 40. SDT-CGRA architecture, adopted from [83].

There is a lot of room for CGRA research to develop and expand as a topic of study for future architecture; this is especially true when developing high-performance CGRAs tailored to specialized or general-purpose computing. Some key issues that require further research in this area include developing tools to program the architecture efficiently, memory management, scalability, adaptability, productivity, virtualization, etc.

VII. EMBEDDED AI ACCELERATORS

The AI hardware requirements are more critical in the edge environment, typically represented by the Internet of Things (IoT) devices (e.g., smart speaker, mobile, sensors and actuators) with limited computing resources, as opposed to cloud infrastructure with relatively sufficient computing capability. For the sake of real-time immediacy, latency, offline capabilities, security, and privacy, AI models are increasingly required to be implemented on edge. In this context, *Small Form Factor (SFF)* devices such as micro-controllers, which dominate the market, are of particular interest, and having AI capabilities on these devices can



a) TCPA accelerator

b) CNN to recognize digits in 28x28 pixel images including net parameters

FIGURE 41. TCPA accelerator showing PE array of size 4 × 4 and a CNN that is mapped onto it for recognizing digits from MNIST database.

help many applications. Many industrial solutions require products with SFF and Size, Weight, and Power (SWaP) enhanced embedded systems. In this section, we review some of the latest embedded AI accelerators.

Fig. 42 shows the architecture of Edge TPU from Google, which is used in products such as Coral and Pixel Phones [111]. Edge TPUs are designed to give high-performance acceleration while staying within strict physical and power constraints [219]. Edge TPU is organized in a 2-D array of *Processing Elements (PEs)* where each PE performs computations in a SIMD fashion. Data is transferred from off-chip memory and PEs via an on-chip controller. Activation and parameters are loaded into the on-chip staging buffers by the controller. In addition, the controller reads in the low-level instructions executed on the PEs (e. g., convolution, pooling, etc.). Each PE may contain single or multiple cores, each having multiple compute lanes to support operation in SIMD fashion. A memory is shared across all cores, *PE Memory* is used to model activations, partial results, and outputs are all stored in a shared memory, which is labeled *PE Memory*, see Fig. 42. Each PE's cores have a *core memory* that is mostly used to store model parameters. Each compute lane has multi-way MAC units to perform computations between activations and model parameters. A few prototyping boards, see Fig. 43 from Coral are available for the community to try and deploy ML apps at the edge, including the Dev Board, USB accelerator, Dev Board Mini, and Dev Board Macro [27]. TensorFlow Lite

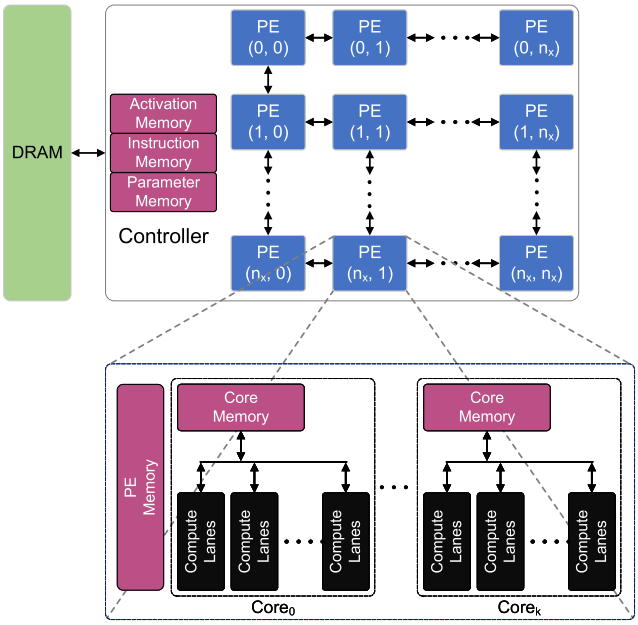


FIGURE 42. Overall architecture of Edge TPU [219].

framework is particularly developed for mapping various neural network operations onto the Edge TPU [29]. The Edge TPU coprocessor can compute 4 trillion operations per second (TOPS) while consuming just 0.5 watts for each TOPS (2 TOPS per watt) [27].

FIGURE 43. Prototyping boards from Coral [27] having edge TPU.

FIGURE 44. NVIDIA's Jetson Nano.

NVIDIA's *Jetson Nano* [3], [183] is an embedded board suitable for edge AI applications. It contains a 64-bit quad-core Arm Cortex-A57 CPU running at 1.43 GHz, NVIDIA Maxwell GPU with 128 CUDA cores, and 4GB LPDDR4 memory. Jetson Nano runs Linux and offers 472 GFLOPS of FP16 computation performance while consuming only 5-10 W of power. NVIDIA also provides the developer kit with examples to map the multiple neural networks applications such as object detection, segmentation, image classification, and speech processing [183]. NVIDIA also provides many embedded boards such as Jetson AGX Orin, Jetson Orin NX, Jetson Xavier NX Series, Jetson TX2 Series in various combinations of form-factor, power-efficiency, and performance to address various industry segments [3], [11].

BeagleBone AI [4], built around *Texas Instruments' (TI) AM 5729 Sitara SoC* [115], is yet another board for AI at the edge. This SoC has two 32-bit Arm Cortex-A15 cores, two Image Processing Units (IPUs) that each having two Cortex-M4 cores, two C66x DSP cores, two PowerVER SGX5443D GPUs, and four Embedded Vision Engines (EVEs). It also has 15 GB of eMMC flash, 1 GB of RAM, Wi-Fi, Bluetooth support, and USB connectors for power and data transfer. The BeagleBone AI runs Linux, and TI Deep Learning (TIDL) framework can be used to develop real-time ML applications.

A *Vision Processing Unit (VPU)* [10] is a processor optimized to perform inference tasks at the edge with ultra-low power without compromising performance. Movidius Myriad 2 VPU is based on the *Intel Neural Compute Stick (NCS) platform*, designed as a 28 *nm* co-processor that provides the high-performance tensor acceleration,

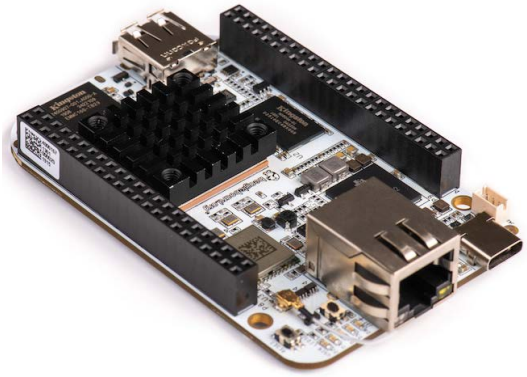


FIGURE 45. BeagleBone AI board [4].

see Fig. 46. The *Streaming Hybrid Architecture Vector Engines (SHAVE)* are 12 highly parallelizable vector processors in the Myriad 2 VPU, whose parallelism and ISA allow good performance efficiency across a range of computer vision applications, even those with low latency requirements. The *Neural Compute Engine*, a dedicated hardware AI accelerator for deep neural network deep-learning inferences, is included in the Myriad X VPU, Movidius' third generation of VPUs. Myriad X is popular for on-device DNNs and computer vision applications thanks to the Neural Compute Engine, 16 SHAVE cores, and ultra-high throughput. The Myriad X VPU includes a native 4K image processor pipeline and can directly link up to eight HD sensors. The *Myriad Development Kit (MDK)*, which offers development tools, frameworks, and APIs to implement computer vision, imaging, and DNN workloads on the chip, can be used to program both the Myriad 2 and Myriad X VPUs.

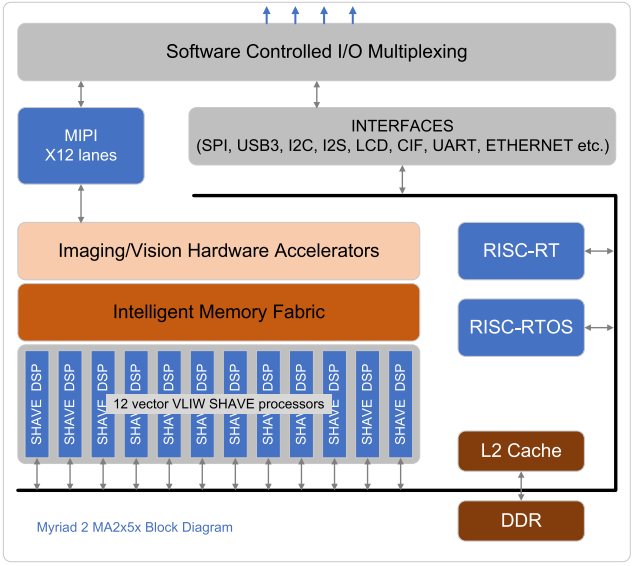


FIGURE 46. Myriad 2 VPU architectural block diagram, adopted from [15].

Sipeed Maixduino is like an Arduino for machine learning projects. It has MAIX SoC [13], which includes Kendryte

K210 KPU (Knowledge Processing Unit, also called Network Processing Unit), a powerful chip suited for visual and semantic recognition [12]. MAIX SoC block diagram is shown in Fig. 47, which includes K210 featuring two RISC-V 64-bit CPU cores, an APU (Audio Processing Unit, also called Audio Accelerator), and KPU optimized for running CNNs. KPU offers 0.25 TOPS@0.3 W,400 MHz; when overclocking to 800 MHz, it offers 0.5 TOPS. It means, we can do object recognition 60 fps@VGA. MAIX also includes a Fast Fourier Transform (FFT) unit, making it useful for signal processing. In addition, it supports a wide range of other peripherals, see Fig. 47. TensorFlow Lite framework is supported by this board, and platforms such as Arduino IDE and PlatformIO can be used for development.

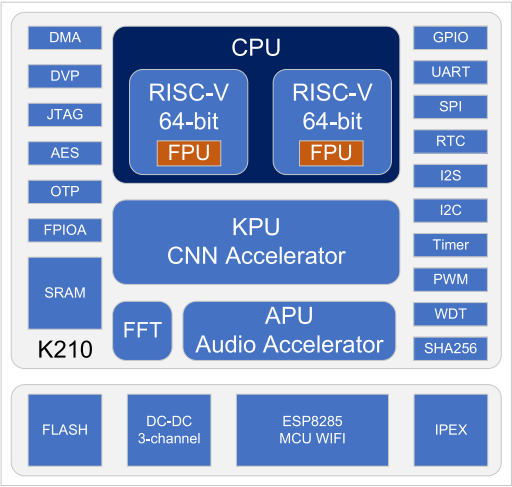


FIGURE 47. Block diagram of KPU and MAIX SoC, adopted from [13].

Sophon's edge developer board, see Fig. 48, is envisioned as a rapid prototype development board for ML applications. It contains a powerful BM1880 capable of efficiently implementing DNN/CNN/RNN/LSTM models using a tailored tensor processing unit. It also features two Arm Cortex-A53 CPUs and a RISC-V CPU. TPU can perform 1 TOPS for 8-bit integer data. This board is mainly used for surveillance cameras, BM1880 [5] is designed using 28 nm process and dissipated 2.5 W. Frameworks such as TensorFlow, PyTorch, ONNX, Caffe, etc. are supported by this board. However, BITMAIN has its own framework called *BITMAIN Neural Network Software Development Kit (BMNNSDK)* [5] and recommends it to achieve high inference throughput and efficiency. BMNET and BMRunTime are included in the BMNNSDK. BMNET is a DNN compiler for TPU processors on edge. It translates CNN-like algorithms into TPU instructions.

Ultra96-V2 [1] and *PYNQ-Z2* [18] are embedded AI boards using FPGAs, see Fig. 49. Ultra96-V2 features a Zynq UltraScale+ MPSoC ZU3EG device. Xilinx's Zynq devices contain both *Processor System (PS)* and *Programmable Logic (PL)*, where PS consists of hardcore processors while PL contains the FPGA. Before the Zynq, processors were connected to an FPGA, which complicated communication

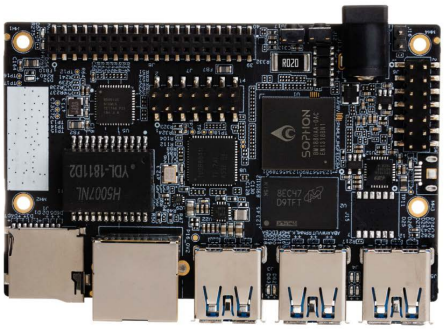


FIGURE 48. Bitmain Sophon(TM) Edge Developer Board, adopted from [8].

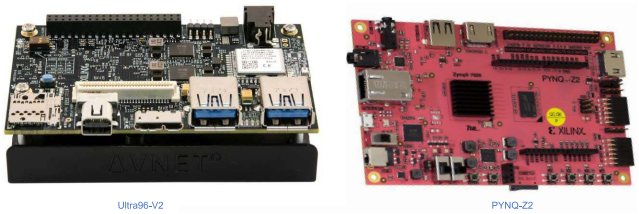


FIGURE 49. Ultra96-V2 [1] and PYNQ-Z2 [18] development boards from Xilinx.

between the PL and the PS. As the latest version of Xilinx's all-programmable System-on-Chip (SoC) families, the Zynq architecture combines a dual-core ARM Cortex-A9 processor with a conventional processor (FPGA). The Advanced eXtensible Interface (AXI) standard is used to connect the various pieces of the Zynq architecture, allowing for high bandwidth and low latency connections. Vivado Design Suite [22] is used to map programs on to Ultra96-V2 board and is widely used in AI and ML projects. For instance, authors in [39] implemented real-time face recognition on Ultra96-V2. Designers may use the Python language and libraries to use Zynq's programmable logic and microprocessors to create more capable and intriguing embedded systems. PYNQ (Python On Zynq) is a Xilinx® open-source project that makes designing embedded systems with Zynq® Systems on Chips simple. PYNQ-Z2 is an FPGA development board based on the ZYNQ XC7Z020 FPGA, which has been meticulously developed to support PYNQ. By combining PL and PS, designers can create more powerful embedded systems using ZYNQ. Furthermore, the SoCs may be programmed in Python, and the code can be developed and tested on the PYNQ-Z2 directly. In the same manner that software libraries are imported and programmed, programmable logic circuits are imported as hardware libraries and programmed through APIs. PYNQ-Z2 board has many interfaces such as user LEDs, push-buttons, switches, MIC input, Ethernet, HDMI Input/Output, MIC Input, Audio Output, Arduino as well as Rasberry Pi interfaces etc. PYNQ takes advantage of the greatest features of both ZYNQ and Python. Machine learning research and prototyping have made extensive use of it. For instance, authors in [207] used this board for implementing CNNs. Xilinx's configurable DPU IP [7] can

also be used together with PYNQ board for creating a network with the desired number of layers, activation functions etc.. Vivado [22], Vitis [21] and Python can be used to work with PYNQ board.

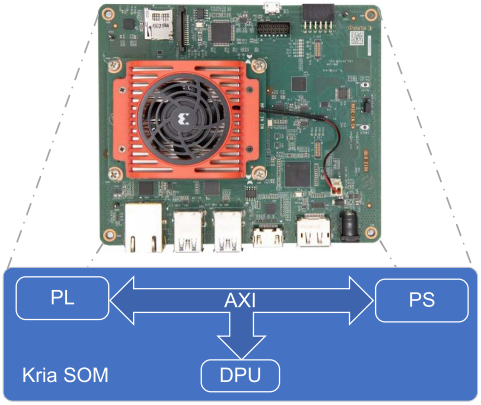


FIGURE 50. Xilinx's Kria KV260 SOM, adopted from [122].

Xilinx's Kria KV260 [23], [122] is an AI starter kit targeted for vision AI applications in smart cities, smart factories, robotics, home automation, etc., see Fig. 50. KV260 includes a Zynq MPSoC, and it supports the Python-based PYNQ framework. The trained models can be implemented in the DPU [7] and are loaded with PYNQ using hardware overlays. In [122], authors have demonstrated pre-trained models based on the MNIST dataset, RESNET based on Caffe framework, and InceptionV1 based on Tensorflow. Furthermore, to exercise the features of KV260, many models from Vitis AI Model Zoo [24] repository are implemented. Traffic detection, lane detection and segmentation algorithms were also implemented and tested in real-time. Silicon Lab has recently introduced BG24/MG24 [19] SoCs with built-in AI accelerators and a new software toolkit. These new devices with optimized hardware and software will help execute AL/ML applications on battery-powered edge devices. The MAX78000 [14] from Maxim Integrated is an AI microcontroller that runs neural networks at extremely low power. It has a hardware-based CNN accelerator, enabling the battery-powered applications to execute AI inferences. AlphaICs' Gluon AI co-processor [9] is optimized for vision AI applications. It comes with an SDK for easy porting of neural networks.

Deep neural networks (DNN) are increasingly being used on IoT-enabled devices like the Raspberry Pi to improve efficiency, security, and privacy. However, the size and complexity of the machine-learning (ML) model that can be deployed in such systems are limited by the available computational and memory resources. The Raspberry Pi is a low-cost, small, and portable computer board with built-in software that allows users to create scripts or programs in Python [229]. There are two main limitations to utilizing a Raspberry Pi for deep learning: 1) the small amount of memory available and 2) the slow processing speed. These limitations severely hamper the implementation of more complex neural networks. There are two ways to deploy deep learning at IoT end devices. 1) Deploy feature vector and model architecture on the Server machine and call with API using Web service to IoT. 2) Deploy feature vector and model architecture on resource-constraint platforms like Raspberry Pi, also called on-device computing. The first method has network latency issues, security risks, and high communication costs. The second method has difficulty in implementing large DNN models due to the limited memory and computational resources of IoT-enabled devices like Raspberry Pi. Furthermore, devices with limited resources, such as the Raspberry Pi, are only used for DNN inference. The trained DNN model can be transferred to the Raspberry Pi through network connectivity. However, network connectivity can introduce delays, data loss, and other security concerns, limiting DNN deployment on the Raspberry Pi [41]. Bhosale et al. [42] proposed Deep Convolutional Neural Network (DCNN) for Covid-19 classification. In this work, the DCNN architecture is deployed on the cloud and uses radiology x-ray images for classification. On the other hand, the authors in [41] proposed a lightweight Deep Learning model (LDC-Net) for Covid-19 classification with lung disease. In this work, LDC-Net was trained on High-Performance Computing (HPC). Furthermore, the trained LDC-Net and weights have been deployed in an IoT-enabled Raspberry Pi with network connectivity for Covid-19 classification.

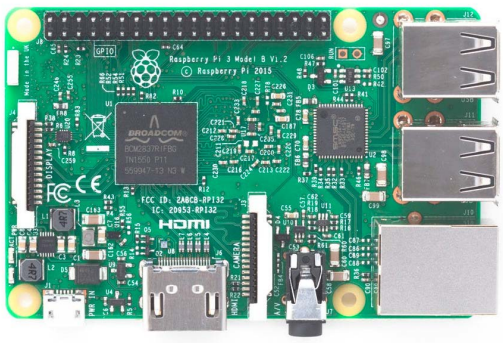


FIGURE 51. Raspberry Pi computer [230].

An Arm processor is a general-purpose processor that belongs to the family of CPUs and uses Reduced Instruction Set Computer (RISC) architecture. Because of their efficiency and flexibility, Arm processors are used in many electronic products, including smartphones, tablets, and wearables. Arm's new portfolio of hardware solutions is now aimed toward Machine Learning (ML) and Deep Neural Network (DNN) applications. In recent times, ARM-based processors have been developed for the acceleration of machine learning applications from various manufacturers viz., Marvell (ThunderX2), Fujitsu (A64FX), Huawei (Kunpeng 920), and Ampere (eMAG). With the help of its recently released *Neural Processing Units (NPUs)*, Arm processors bring machine learning to low-end edge devices.

Arm ML processor uses the Neural Network (NN) software development kit provided by the company to interface the ML software and corresponding hardware [214]. The Arm-based ML accelerator consists of a number of computing engines up to 16, each of which includes a programmable layer engine and a MAC convolution engine, see Fig. 52. Each computing engine has its own local memory to process the ML models. Starting with weights applied to incoming data, processing via the MAC convolution engine, and finally, results processed by the *Programmable Layer Engine (PLE)*, the flow is typical of DNN implementations. There are 128 multiple-accumulate (MAC) units in the MAC convolution engine. MAC convolution engine receives the input data from the input feature map read block, weights from the weight decoder, and performs the required MAC operation. The result of the convolution is processed by the PLE, which is a vectorized microcontroller. It is more akin to a RISC platform designed to wrap up the processing of a layer for a piece of a DNN model with several layers. The PLE is in charge of tasks like pooling and activation. The throughput of the proposed ML processor is 4.6 TOPS. The proposed design is implemented using 7 nm chip technology, and it is scalable, and can achieve the throughput of 150 TOPS for high-end applications.

The Arm AI platform, also known as Project Trillium, is a heterogeneous compute platform that includes Arm Cortex CPUs, Ethos NPUs, Mali GPUs, and microNPUs to accelerate the ML algorithms [142]. Arm supports various ML frameworks such as TensorFlow Lite, Caffe, PyTorch, etc. and accelerates the ML applications using software libraries including arm NN, arm COMPUTE LIBRARY, and Common Microcontroller Software Interface Standard-NN (CMSIS-NN). The hardware products such as Arm Cortex CPUs, Ethos NPUs, Mali GPUs, microNPUs, FPGAs, DSPs, etc. ARM's new Cortex-A55/A75 and Mali-G72 combination targets machine learning on edge computing devices.

Arm has developed its Ethos series of ML processors for machine learning applications. Ethos series is classified into N-series and U-series [25]. Ethos N-series was introduced in October 2019, containing NPUs identical to the Cortex family. Ethos U-series was introduced in early 2020, and it contains microNPUs. MicroNPUs are paired with the CPU, like the Cortex-M55, to process the ML algorithms. Ethos-U55 achieves a throughput of 0.5 TOPS, containing 32 to 256 8-bit MAC units [143]. Ethos-U55 supports 8-bit and 16-bit integer data types. Ethos-U65 achieves a throughput of 1 TOPS, containing 256 to 312 8-bit MAC units. Ethos-N57 achieves a throughput of 2 TOPS, containing 1024 8-bit MAC units. Ethos-N77 is a highly efficient ML inference processor that achieves a throughput of 5 TOPS and is best suitable for mobile devices. Ethos-N77 ML processors can be used for facial or object recognition applications. Ethos-N78 is a scalable and efficient ML inference processor that achieves a throughput of 1 to 10 TOPS [144]. Arm's Cortex-M55 and the Ethos-U55 can be used as an AI accelerator in edge computing devices [89]. This combination achieves a 32× improvement in ML processing compared to the base Cortex-M55 core. Furthermore, TinyML [20] advancements have made it possible to use ML models on the microcontroller hardware found in our household appliances, including printers, TVs, smartwatches, and pacemakers, which can now carry out tasks that were previously only capable of being done by computers and smartphones. The machine learning and embedded ultra-low power systems communities have joined forces to create TinyML foundation. This joint effort has paved the way for innovative and captivating alternative uses of on-device machine learning. TinyML supports various frameworks, including TensorFlow Lite Micro (TFLM), TensorFlow-Native, Embedded Learning Library (ELL), Graph Lowering (GLOW), etc. Google developed an open-source framework called CFU Playground [174] for TinyML acceleration on FPGA. CFU playground toolchain combines open-source software (TensorFlow), RTL generators (LiteX, Migen, etc.), and FPGA tools for synthesis (yosys), place, and route (vpr). The CFU playground framework makes it possible to investigate custom architectures for the acceleration of Tiny ML for embedded ML systems. TinyML is used in many applications, including medical face mask detection [156], eating detection [166], Li-Ion batteries parameter estimation [69], etc. The most in-demand research areas among the TinyML community include sound recognition, computer vision, and the development of low-power accurate ML models. More research is needed to fully comprehend the advantages and drawbacks of the topics under discussion, even if many applications have demonstrated TinyML's promise. Some key issues that require further research in this area include developing benchmarks, memory constraints, energy, processor capacity, cost reduction, etc.

VIII. COMPARISON BETWEEN VARIOUS HARDWARE ARCHITECTURES FOR DNN ACCELERATION

The performance of the various hardware accelerators for the DNN acceleration depends on the target application. However, researchers defined some standard metrics, namely, area, power, and throughput, to measure the performance of the hardware accelerators for the development and deployment of DNNs. Here, the area is nothing but the portion of silicon required for the DNN acceleration, which is generally represented in squared millimeters or squared micrometers. The area depends on the size of the on-chip memory and the technology used during the hardware synthesis process. Power is nothing but the amount of power consumed by the specific hardware during the DNN acceleration. The power consumption mainly depends on off-chip and on-chip memories. Throughput is used to measure the productivity of the hardware accelerator. The comparison between the various hardware accelerator architectures for DNN acceleration is shown in Table 7. Due to a lack of data on their footprint, power consumption, and throughput, CGRA-based accelerators are not represented in Table 7.

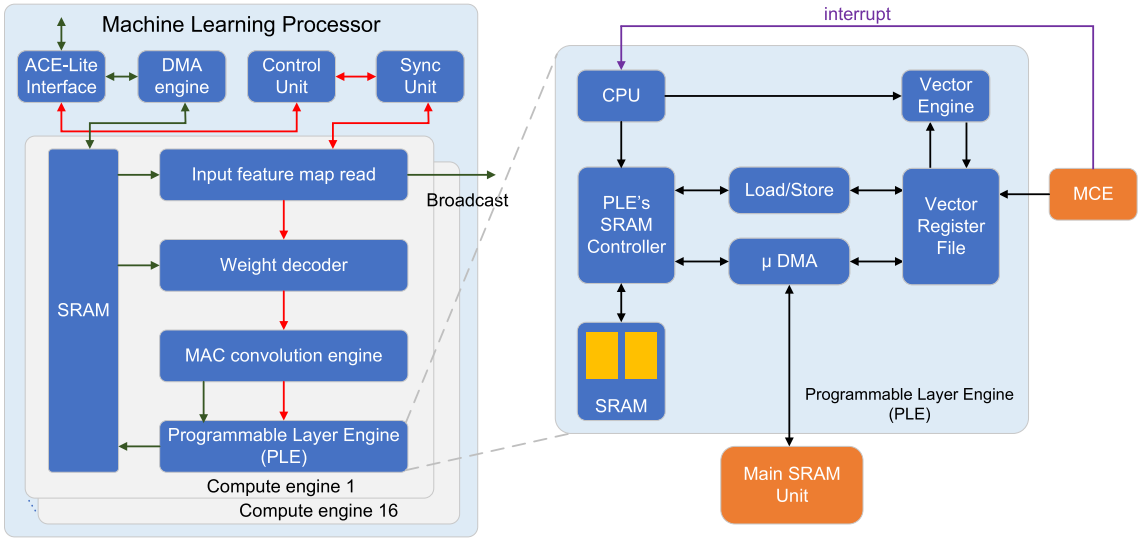


FIGURE 52. Arm based ML processor [214].

As expected, temporal or general purpose architectures such as CPU and GPU have greater power consumption and area than special purpose architectures such as FPGA and ASIC because they are not tailored for a particular application. The essential hardware metrics like power, area, technology, and throughput are reported for each hardware architecture. In Table 8, we have compared the few embedded development boards discussed above concerning general-purpose CPUs/GPUs, specialized co-processors they contain, performance, power, SDKs, and supported ML frameworks.

IX. FUTURE DIRECTIONS

In the future, hardware AI acceleration is set to become ubiquitous. In recent processors, some AI accelerator hardware becoming a standard feature, indicating that AI acceleration is an essential general-purpose task. This paper reviewed several FPGA-based, ASIC-based, GPU-based, CGRA-based, and edge AI hardware accelerators. However, looking at the industry trends and startups in this space indicates that we are still in the early stage of the AI revolution. Many more energy-efficient architectures will emerge in the future. In particular, architectures with transprecision or approximate computing, high-bandwidth memories, and emerging non-volatile memories such as MRAM and ReRAM may appear in the market. Evolving architectures involving the Tsetlin machine are another promising future research direction.

Emerging technologies such as nanomaterials, optical computing, and DNA computing may accelerate DNNs in the near future. Carbon nanomaterials, such as carbon nanotubes (CNTs) and graphene, are particularly intriguing due to their rapid electron transport [58]. CNT and graphene have desirable switching and optical properties, making them well-suited to electronic and optical architectures [189]. New chip architectures become possible with the help of CNTs and other nanomaterials. Researchers at MIT and Stanford have developed a new 3-D architecture based on a network of millions of carbon nanotubes [191]. Computations in optical computing technology can happen at the speed of light, much faster than conventional electron-driven chips. MIT is driving research in advanced optical materials, switches, lasers, and nano-optics [106] to advance optical computing. We may expect a greater deployment of optical chips in the future. DNA computing is a type of parallel computing in which many different DNA molecules are used to test many possibilities simultaneously [138]. The major advantage of DNA is its potential for memory storage. A single gram of DNA can store 215 petabytes (215 million gigabytes) [6]. Although DNA information storage has enormous application potential, many issues, such as the high cost of writing and reading information and techniques to erase and rewrite the information in DNA that is still unknown, must be addressed before its widespread use [75].

In FPGA-based architectures, following future directions seems to be promising. The combination of FPGAs and cloud computing opens new avenues for developing deep learning applications. The FPGA cloud service is still in its early stages. Many imperfections must be investigated, such as the virtualization of FPGA hardware resources, task migration, etc. Most current research focuses on lowering the bandwidth requirements for off-chip memory access. The performance of multiple FPGA chips combined is favorable. However, dealing with processing scheduling and chip allocation remains a significant challenge. Future research could focus on the development of in-memory-computing processors. Moreover, further improvements are required in the computation of the activation functions used in DNNs. Because most studies focus on loop optimization, only a few researchers are currently working on activation function optimization. There will be frameworks to integrate existing or new architectures, which will help quickly deploy

TABLE 7. Comparison among accelerators implemented on different hardware platforms.

Accelerator	Year	Platform	Area (mm^2)	Power (W)	Throughput (GOPS)
CNP [85]	2009	FPGA	N/A	15	N/A
Parallel coprocessor for CNN [182]	2009	FPGA	N/A	11	6.74
MAPLE [47]	2010	FPGA	N/A	N/A	7
DC-CNN [51]	2010	FPGA	N/A	14	16
NeuFlow [84]	2011	FPGA	N/A	10	147
NeuFlow [170]	2012	ASIC	12.5	0.6	320
Memory- Centric Accelerator [169]	2013	FPGA	N/A	N/A	17
nn-X [91]	2014	FPGA	N/A	8	23.18
DianNao [53]	2014	ASIC	3.02	0.485	452
DaDianNao [54]	2014	ASIC	**0.78**	15.97	5580
Origami [50]	2015	ASIC	3.09	0.744	274
PuDianNao [141]	2015	ASIC	3.51	0.596	1056
ShiDianNao [78]	2015	ASIC	4.86	0.32	194
Roofline based Accelerator [222]	2015	FPGA	N/A	18.61	61.62
Embedded FPGA Accelerator [177]	2016	FPGA	N/A	9.63	136.97
fpgaConvNet [206]	2016	FPGA	N/A	N/A	12.73
DeepBurning [211]	2016	FPGA	N/A	N/A	73
SCNN [167]	2017	ASIC	7.9	N/A	2000
FlexFlow [145]	2017	ASIC	3.89	6.8	420
Nvidia V100 [17]	2017	GPU	815	250	**15700**
Eyeriss [56]	2017	ASIC	12.25	0.278	N/A
TPU [120]	2017	ASIC	<331	N/A	N/A
DLAU [208]	2017	FPGA	N/A	**0.234**	N/A
ESE [102]	2017	FPGA	N/A	41	282
FP-DNN [95]	2017	FPGA	N/A	25	364.4
FINN [204]	2017	FPGA	N/A	11.7	2465.5
Angle-Eye [96]	2018	FPGA	N/A	3.5	137
Bit Fusion [188]	2018	ASIC	5.87	0.895	N/A
MAERI [131]	2018	ASIC	6	N/A	N/A
DNPU [190]	2018	ASIC	16	0.279	300
UNPU [133]	2018	ASIC	16	0.297	345.6
Jetson AGX Xavier [28]	2018	GPU	N/A	10	32
Tesla T4 [31]	2018	GPU	545	70	130
Accelerator for SSDLiteM2 [82]	2018	FPGA	N/A	9.9	N/A
Intel Xeon Platinum 9282	2019	CPU	N/A	400	3200
AMD Ryzen Threadripper 3970x	2019	CPU	N/A	280	1859
LNPU [135]	2019	ASIC	16	0.367	>300
SNAP [225]	2019	ASIC	2.4	**0.16-0.36**	N/A
Eyeriss2 [57]	2019	ASIC	N/A	N/A	153.6
BFP arithmetic-based Accelerator [140]	2019	FPGA	N/A	9.18	760.83
Tera-OPS streaming Accelerator [161]	2019	FPGA	N/A	18.29	1877
Caffeine [223]	2019	FPGA	N/A	26	354
SIGMA [176]	2020	ASIC	65.1	22.3	10800
Nvidia A100 [16]	2020	GPU	826	400	**19500**
CNN2Gate [90]	2020	FPGA	N/A	N/A	80.04
Jetson Xavier NX [26]	2021	GPU	N/A	10-20	14-21
GAMMA [224]	2021	ASIC	30.6	N/A	N/A
Accelerator for space DNN [216]	2021	FPGA	N/A	3.82	1.34
NPE [125]	2021	FPGA	N/A	20	135.14
LP-CNN [81]	2021	FPGA	N/A	3.92	129.2
Reconfigurable YOLOv3 Accelerator [209]	2021	FPGA	N/A	25	N/A
Ad-MobileNet [45]	2021	FPGA	N/A	3.25	N/A
Energy-efficient CNN Accelerator [116]	2021	FPGA	N/A	0.628	N/A
Dynamically Reconfigurable Architecture [117]	2022	FPGA	N/A	2.039	40.71
RCNN Accelerator [92]	2022	FPGA	N/A	1.15	N/A

TABLE 8. Comparison of various embedded edge AI development boards.

	Coral Dev Board	Jetson Nano	BeagleBone	Myriad X	Maixduino	Sophon Edge
General Purpose Processors	NXP i.MX 8M SoC (quad Cortex-A53, Cortex-4F), GC7000 GPU	Quad-core ARM A57, 128-core Maxwell	Dual Cortex-A15, Dual SGX544, Dual PRU-ICSS	Dual LEON4, 16 SHAVE (Vector Processing Unit)	Dual RISC-V Core 64bit, with FPU	Dual Cortex A53. Single RISC-V
AI Co-Processor	Google Edge TPU	128-core NVIDIA GPU	Dual C66x DSP, Quad EVE	Neural Computing Engine	KPU/NPU	BM1880, TPU
Performance	**4 TOPS**	472 GFLOPS	~100GOPS	1 TOPS	0.25 -0.5 TOPS	1 TOPS
Power	2 TOPS per watt	5-10W	5-10W	1 W	**0.3 W**	2.5W
SDK	Edge TPU AI	JetPack	TI Deep Learning	Myriad Development Kit	Maixduino SDK	BITMAIN Neural Network SDK
Supported Frameworks	TensorFLow Lite	TensorFlow, PyTorch, Caffe	Caffe, TensorFlow	Caffe, TensorFlow, MXNet	TensorFlowLite	Caffe, TensorFlow, PyTorch, MXnet

applications. Most importantly, FPGA-based accelerator research will be towards training and not inference.

In ASIC-based hardware accelerators, following future research trends are suggested. TPU is already a standard in the field of deep learning. More capable replacements are likely to emerge in the coming years. There will be entirely new architectures to target low-latency and low-power applications. Most current studies assume a trained DNN and focus on increasing the speed of its inference. There have been only a few studies on accelerator design for DNN training. Therefore, there will be more emphasis on developing ASIC-based DNN training accelerators in the future. More research and breakthroughs in CPU-GPU heterogeneous architectures are required for more efficient DNN implementations. Special-purpose or data center system-on-chips (SoCs) with embedded FPGA or GPU-based machine learning accelerators appear to be gaining traction. In CGRA-based accelerators, architectures driven by programming might be interesting. Other directions include introducing process-in-memory into CGRA architectures to address the data movement bottleneck. Further improvements are needed for the architectures that support dynamic configuration, as it is an important step towards the widespread use of CGRAs.

The following trends may be observed in the future development of Edge AI accelerators. Edge AI operates in a heterogeneous environment where the data at the edge and the preprocessing techniques required for each sensor vary greatly between applications. Therefore, more customized, powerful, and energy-efficient chips for specific edge ML applications will be developed. Multimodal deep learning is a major development that pulls data from multiple sources to extract more granular features. Using these multimodal techniques, the car's make and model can be pinpointed instead of just recognizing a car. Other potential research directions include using distributed ML algorithms to speed up ML algorithm training and reduce the amount of memory required for processing. ML applications at the edge require high accuracy. Therefore, methods for implementing cutting-edge models at the edge while maintaining accuracy based on deep learning model pruning and quantization are among the new research directions. We will also see the development of customized and general SDK frameworks targeting specific or multiple edge accelerators for easy deployment of neural network applications.

X. CONCLUSION

Deep Neural Networks have recently gained popularity in a variety of applications. They are, however, computationally demanding, making them difficult to handle by general-purpose architectures. In this context, a detailed review of recent advances in DNN acceleration on specialized hardware architectures such as FPGA, ASIC, GPU, and CGRA is presented. Furthermore, embedded AI accelerators for the edge environment have been thoroughly discussed. The review begins with a detailed background of DNNs, focusing on their key operations and applications. CNNs, which have a wide range of applications, have also been included in the review. To improve the performance of the hardware accelerator, we discussed various computing architectures, such as temporal and spatial architectures, as well as different dataflow patterns. The review focused on recent advancements in the acceleration of DNNs on FPGA, ASIC, GPU, CGRA, and Embedded AI accelerators. The review divided the FPGA-based accelerators into three categories and briefly discussed their key features, including the frameworks available for each. Similarly, ASIC-based accelerators are classified, and the review summarizes the accelerators available in the literature based on area, power dissipation, throughput, resource utilization, and so on. A comprehensive review of Nvidia's GPU-based accelerators was also presented. Furthermore, the review compared the various popular FPGA/ASIC/GPU-based accelerators. It has been observed that temporal architectures, such as CPU and GPU, dissipate more power than spatial architectures, such as FPGA and ASIC; however, they have higher throughput than FPGA and ASIC. As a result, it is difficult to say that one architecture is superior to another because it depends on the target application and requirements. Furthermore, the survey presented and compared recent research contributions in Arm-based machine learning processors and a few embedded AI hardware accelerators in terms of their cores, performance, power, availability of Software Development Kits (SDKs), and supported frameworks. Finally, the review suggests future research directions for DNN acceleration using various hardware architectures, including FPGA, ASIC, GPU, CGRA, and Edge AI accelerators.

REFERENCES

[1] *ULTRA96-V2*. Accessed: Jan. 7, 2022. [Online]. Available: https://www.avnet.com/opasdata/d120001/medias/docus/198/5365-pb-ultra96-v2-v10b.pdf

[2] *Accelerate Fast Math With Intel Oneapi Math Kernel Library*. Accessed: Jun. 5, 2021. [Online]. Available: https://software.intel.com/content/www/us/en/develop/tools/oneapi/components/onemkl.html#gs.3595x9

[3] *Advanced AI Embedded Systems: NVIDIA Jetson: The AI Platform for Autonomous Machines*. Accessed: Aug. 2, 2022. [Online]. Available: https://www.nvidia.com/en-in/autonomous-machines/embedded-systems/

[4] *BeagleBone AI: Fast Track to Embedded Artificial Intelligence*. [Online]. Available: https://beagleboard.org/AI Accessed: Jan. 2, 2022.

[5] *BitMain Neural Network SDK: Introduction*. Accessed: Jan. 2, 2022. [Online]. Available: https://sophon-edge.gitbook.io/project/

[6] R. F. Service, "DNA could store all of the world's data in one room," *Science*, Mar. 2017. [Online]. Available: https://www.science.org/content/article/dna-could-store-all-worlds-data-one-room

[7] *DPU for Convolutional Neural Network*. Accessed: May 1, 2022. [Online]. Available: https://www.xilinx.com/products/intellectual-property/dpu.html

[8] *Edge TPU Developer Board*. [Online]. Available: https://www.sophon.ai/product/introduce/edb.html Accessed: Jan. 2, 2022.

[9] *Gluon AI Co-Processor*. Accessed: Jan. 10, 2022. [Online]. Available: https://alphaics.ai/products/gluon-ai-accelerator/

[10] Intel Movidius Myriad X Vision Processing Unit. Accessed: Apr. 2, 2022. [Online]. Available: https://www.intel.com/content/www/us/en/products/details/processors/movidius-vpu/movidius-myriad-x.html

[11] *Jetson Nano Developer Kit*. Accessed: Aug. 2, 2022. [Online]. Available: https://www.nvidia.com/en-in/autonomous-machines/embedded-systems/jetson-nano-developer-kit/

[12] *Kendryte K210*. Accessed: Sep. 2, 2022. [Online]. Available: https://canaan.io/product/kendryteai

[13] *Maixduino*. Accessed: Sep. 2, 2022. [Online]. Available: https://www.seeedstudio.com/Sipeed-Maixduino-Kit-for-RISC-V-AI-IoT-p-4047.html

[14] *MAX78000—Artificial Intelligence Microcontroller with Ultra-Low-Power Convolutional Neural Network Accelerator*. Accessed: Jan. 10, 2022. [Online]. Available: https://www.maximintegrated.com/en/products/microcontrollers/MAX78000.html

[15] *Myriad 2 MA2x5x Vision Processor: Transforming Devices Through Ultra Low-Power Machine Vision—Google Search*. Accessed: Apr. 2, 2022. [Online]. Available: https://www.intel.com/content/www/us/en/products/details/processors/movidius-vpu/movidius-myriad-x.html,www.movidius.com

[16] (2020). *Nvidia A100 Tensor Core GPU Architecture*. Accessed: Jun. 13, 2021. [Online]. Available: https://www.nvidia.com/content/dam/en-zz/solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf

[17] (2017). *Nvidia Tesla V100 GPU Architecture*. Accessed: Jun. 13, 2021. [Online]. Available: https://images.nvidia.com/content/technologies/volta/pdf/437317-volta-v100-ds-nv-us-web.pdf

[18] *PYNQ-Z2*. Accessed: Jan. 7, 2022. [Online]. Available: http://www.pynq.io/board.html

[19] *Silicon Labs BG24 and MG24 SoCs*. Accessed: Jan. 10, 2022. [Online]. Available: https://www.silabs.com/wireless/zigbee/efr32mg24-series-2-socs

[20] *TinyML Foundation*. Accessed: Sep. 2, 2022. [Online]. Available: https://www.tinyml.org/

[21] *Vitis Unified Software Platform*. Accessed: Oct. 1, 2022. [Online]. Available: https://www.xilinx.com/products/design-tools/vitis/vitis-platform.html

[22] *Vivado*. Accessed: Jan. 15, 2022. [Online]. Available: https://www.xilinx.com/products/design-tools/vivado.html

[23] Xilinx Kria—Adaptive System-on-Module. Accessed: Oct. 1, 2022. [Online]. Available: https://www.xilinx.com/products/som/kria.html

[24] *Xilinx Vitis AI Model Zoo*. Accessed: Oct. 1, 2022. [Online]. Available: https://github.com/Xilinx/AI-Model-Zoo

[25] (Jul. 2021). *Ethos—ARM—WikiChip*. Accessed: Aug. 7, 2021. [Online]. Available: https://en.wikichip.org/wiki/arm_holdings/ethos

[26] (Aug. 2021). *Jetson Xavier NX*. Accessed: Jul. 15, 2022. [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-xavier-nx/

[27] (2022). *Coral Products*. [Online]. Available: https://coral.ai/products/

[28] (Jul. 2022). *Deploy AI-Powered Autonomous Machines at Scale*. Accessed: Jul. 15, 2022. [Online]. Available: https://www.nvidia.com/en-gb/autonomous-machines/embedded-systems/jetson-agx-xavier/

[29] (2022). *Edge TPU Compiler*. [Online]. Available: https://coral.ai/docs/edgetpu/compiler/#system-requirements

[30] *High-Level Synthesis & Verification*. Accessed: Jul. 2022. [Online]. Available: https://eda.sw.siemens.com/en-US/ic/ic-design/high-level-synthesis-and-verification-platform/

[31] *NVIDIA Tesla T4 Specs*. Accessed: Jun. 17, 2022. [Online]. Available: https://www.nvidia.com/en-us/data-center/tesla-t4/

[32] *Vitis AI*. Accessed: Jun. 17, 2022. [Online]. Available: https://www.xilinx.com/products/design-tools/vitis/vitis-ai.html

[33] M. Alwani, H. Chen, M. Ferdman, and P. Milder, "Fused-layer CNN accelerators," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2016, pp. 1–12.

[34] D. Amodei et al., "Deep speech 2: End-to-end speech recognition in English and Mandarin," in *Proc. 33rd Int. Conf. Mach. Learn.*, vol. 48, M. F. Balcan and K. Q. Weinberger, Eds. New York, NY, USA: arXiv, Jun. 2016, pp. 173–182.

[35] A. Argal, S. Gupta, A. Modi, P. Pandey, S. Shim, and C. Choo, "Intelligent travel chatbot for predictive recommendation in echo platform," in *Proc. IEEE 8th Annu. Comput. Commun. Workshop Conf. (CCWC)*, Jan. 2018, pp. 176–183.

[36] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," 2015, *arXiv:1409.0473*.

[37] Y. Bengio, "Learning deep architectures for AI," *Found. Trends Mach. Learn.*, vol. 2, no. 1, pp. 1–127, 2009.

[38] K. Benkrid and S. Belkacemi, "Design and implementation of a 2D convolution core for video applications on FPGAs," in *Proc. 3rd Int. Workshop Digit. Comput. Video (DCV)*, 2002, pp. 85–92.

[39] M. Bergeron. *Real-Time Face Recognition on Ultra96-V2*. Accessed: Feb. 1, 2022. [Online]. Available: https://www.hackster.io/AlbertaBeef/real-time-face-recognition-on-ultra96-v2-94de9b

[40] Y. H. Bhosale and K. S. Patnaik, "Application of deep learning techniques in diagnosis of COVID-19 (Coronavirus): A systematic review," *Neural Process. Lett.*, pp. 1–53, Sep. 2022.

[41] Y. H. Bhosale and K. Sridhar Patnaik, "IoT deployable lightweight deep learning application for COVID-19 detection with lung diseases using RaspberryPi," in *Proc. Int. Conf. IoT Blockchain Technol. (ICIBT)*, May 2022, pp. 1–6.

[42] Y. H. Bhosale, S. Zanwar, Z. Ahmed, M. Nakrani, D. Bhuyar, and U. Shinde, "Deep convolutional neural network based COVID-19 classification from radiology X-ray images for IoT enabled devices," in *Proc. 8th Int. Conf. Adv. Comput. Commun. Syst. (ICACCS)*, Mar. 2022, pp. 1398–1402.

[43] L. Bishnoi and S. N. Singh, "Artificial intelligence techniques used in medical sciences: A review," in *Proc. 8th Int. Conf. Cloud Comput., Data Sci. Eng. (Confluence)*, Jan. 2018, pp. 1–8.

[44] A. G. Blaiech, K. Ben Khalifa, C. Valderrama, M. A. Fernandes, and M. H. Bedoui, "A survey and taxonomy of FPGA-based deep learning accelerators," *J. Syst. Archit.*, vol. 98, pp. 331–345, Sep. 2019.

[45] S. Bouguezzi, H. B. Fredj, T. Belabed, C. Valderrama, H. Faiedh, and C. Souani, "An efficient FPGA-based convolutional neural network for classification: Ad-MobileNet," *Electronics*, vol. 10, no. 18, p. 2272, Sep. 2021.

[46] A. Boutros, S. Yazdanshenas, and V. Betz, "You cannot improve what you do not measure: FPGA vs. ASIC efficiency gaps for convolutional neural network inference," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 11, no. 3, pp. 1–23, Sep. 2018.

[47] S. Cadambi, A. Majumdar, M. Becchi, S. Chakradhar, and H. P. Graf, "A programmable parallel accelerator for learning and classification," in *Proc. 19th Int. Conf. Parallel Archit. Compilation Techn. (PACT)*, 2010, pp. 273–283.

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

[49] F. Cardells-Tormo, P.-L. Molinet, J. Sempere-Agullo, L. Baldez, and M. Bautista-Palacios, "Area-efficient 2D shift-variant convolvers for FPGA-based digital image processing," in *Proc. Int. Conf. Field Program. Log. Appl.*, 2005, pp. 578–581.

[50] L. Cavigelli, D. Gschwend, C. Mayer, S. Willi, B. Muheim, and L. Benini, "Origami: A convolutional network accelerator," in *Proc. 25th, Ed., Great Lakes Symp. (VLSI)*, New York, NY, USA, May 2015, pp. 199–204.

[51] S. Chakradhar, M. Sankaradas, V. Jakkula, and S. Cadambi, "A dynamically configurable coprocessor for convolutional neural networks," in *Proc. 37th Annu. Int. Symp. Comput. Archit. (ISCA)*, New York, NY, USA, 2010, pp. 247–257.

[52] J.-W. Chang and S.-J. Kang, "Optimizing FPGA-based convolutional neural networks accelerator for image super-resolution," in *Proc. 23rd Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2018, pp. 343–348.

[53] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," in *Proc. 19th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, vol. 14, New York, NY, USA, 2014, pp. 269–284.

[54] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam, "DaDianNao: A machine-learning supercomputer," in *Proc. 47th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Dec. 2014, pp. 609–622.

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

[56] 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.

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

[58] Z. Chen, H.-S. Philip Wong, S. Mitra, A. Bol, L. Peng, G. Hills, and N. Thissen, "Carbon nanotubes for high-performance logic," *MRS Bull.*, vol. 39, no. 8, pp. 719–726, Aug. 2014.

[59] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "CuDNN: Efficient primitives for deep learning," 2014, *arXiv:1410.0759*.

[60] D. Chicco, P. Sadowski, and P. Baldi, "Deep autoencoder neural networks for gene ontology annotation predictions," in *Proc. 5th ACM Conf. Bioinf., Comput. Biol., Health Informat.*, New York, NY, USA, Sep. 2014, pp. 533–540.

[61] P.-S. Chiu, J.-W. Chang, M.-C. Lee, C.-H. Chen, and D.-S. Lee, "Enabling intelligent environment by the design of emotionally aware virtual assistant: A case of smart campus," *IEEE Access*, vol. 8, pp. 62032–62041, 2020.

[62] Y.-k. Choi, K. You, J. Choi, and W. Sung, "A real-time FPGA-based 20 000-word speech recognizer with optimized DRAM access," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 57, no. 8, pp. 2119–2131, Aug. 2010.

[63] J. Choquette, W. Gandhi, O. Giroux, N. Stam, and R. Krashinsky, "NVIDIA A 100 tensor core GPU: Performance and innovation," *IEEE Micro*, vol. 41, no. 2, pp. 29–35, Mar. 2021.

[64] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (ELUs)," 2015, *arXiv:1511.07289*.

[65] J. Cloutier, E. Cosatto, S. Pigeon, F. R. Boyer, and P. Y. Simard, "VIP: An FPGA-based processor for image processing and neural networks," in *Proc. 5th Int. Conf. Microelectron. Neural Netw.*, 1996, pp. 330–336.

[66] R. Collobert, K. Kavukcuoglu, and C. Farabet, "Torch7: A MATLAB-like environment for machine learning," in *Proc. NIPS*, 2011, pp. 1–6.

[67] J. Cong and B. Xiao, "Minimizing computation in convolutional neural networks," in *Proc. ICANN*, 2014, pp. 281–290.

[68] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or −1," 2016, *arXiv:1602.02830*.

[69] G. Crocioni, D. Pau, J.-M. Delorme, and G. Gruosso, "Li-ion batteries parameter estimation with tiny neural networks embedded on intelligent IoT microcontrollers," *IEEE Access*, vol. 8, pp. 122135–122146, 2020.

[70] D. Danopoulos, C. Kachris, and D. Soudris, "Acceleration of image classification with Caffe framework using FPGA," in *Proc. 7th Int. Conf. Modern Circuits Syst. Technol. (MOCAST)*, May 2018, pp. 1–4.

[71] L. Deng and D. Yu, "Deep learning: Methods and applications," *Found. Trends Signal Process.*, vol. 7, nos. 3–4, pp. 197–387, Jun. 2014.

[72] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. De Freitas, "Predicting parameters in deep learning," in *Proc. 26th Int. Conf. Neural Inf. Process. Syst.*, vol. 2. Red Hook, NY, USA: Curran Associates, 2013, pp. 2148–2156.

[73] A. Deshpande, *A Beginner's Guide To Understanding Convolutional Neural Networks*. Los Angeles, CA, USA: University of California, 2018.

[74] J. Domke, E. Vatai, A. Drozd, P. ChenT, Y. Oyama, L. Zhang, S. Salaria, D. Mukunoki, A. Podobas, M. WahibT, and S. Matsuoka, "Matrix engines for high performance computing: A paragon of performance or grasping at straws?" in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*. Los Alamitos, CA, USA: IEEE Computer Society, May 2021, pp. 1056–1065.

[75] Y. Dong, F. Sun, Z. Ping, Q. Ouyang, and L. Qian, "DNA storage: Research landscape and future prospects," *Nat. Sci. Rev.*, vol. 7, no. 6, pp. 1092–1107, Jun. 2020.

[76] L. Du and Y. Du, "Hardware accelerator design for machine learning," in *Machine Learning*, H. Farhadi, Ed. Rijeka, Croatia: IntechOpen, 2018, ch. 1.

[77] L. Du, Y. Du, Y. Li, J. Su, Y.-C. Kuan, C.-C. Liu, and M.-C. F. Chang, "A reconfigurable streaming deep convolutional neural network accelerator for Internet of Things," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 65, no. 1, pp. 198–208, Jan. 2018.

[78] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "Shidiannao: Shifting vision processing closer to the sensor," *SIGARCH Comput. Archit. News*, vol. 43, no. 3S, pp. 92–104, Jun. 2015.

[79] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "ShiDianNao: Shifting vision processing closer to the sensor," in *Proc. 42nd Annu. Int. Symp. Comput. Archit.*, New York, NY, USA, Jun. 2015, pp. 92–104.

[80] C. Dubout and F. Fleuret, "Exact acceleration of linear object detectors," in *Proc. 12th Eur. Conf. Comput. Vis.* Berlin, Germany: Springer-Verlag, 2012, pp. 301–311.

[81] A. J. A. El-Maksoud, M. Ebbed, A. H. Khalil, and H. Mostafa, "Power efficient design of high-performance convolutional neural networks hardware accelerator on FPGA: A case study with GoogLeNet," *IEEE Access*, vol. 9, pp. 151897–151911, 2021.

[82] H. Fan, S. Liu, M. Ferianc, H.-C. Ng, Z. Que, S. Liu, X. Niu, and W. Luk, "A real-time object detection accelerator with compressed SSDLite on FPGA," in *Proc. Int. Conf. Field-Programmable Technol. (FPT)*, Dec. 2018, pp. 14–21.

[83] X. Fan, D. Wu, W. Cao, W. Luk, and L. Wang, "Stream processing dual-track CGRA for object inference," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 6, pp. 1098–1111, Jun. 2018.

[84] C. Farabet, B. Martini, B. Corda, P. Akselrod, E. Culurciello, and Y. Lecun, "NeuFlow: A runtime-reconfigurable dataflow processor for vision," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jun. 2011, pp. 109–116.

[85] C. Farabet, C. Poulet, J. Han, and Y. LeCun, "CNP: An FPGA-based processor for convolutional networks," in *Proc. 19th Int. Conf. Field Program. Log. Appl.*, 2009, pp. 32–37.

[86] X. Feng, H. Zhang, Y. Ren, P. Shang, Y. Zhu, Y. Liang, R. Guan, and D. Xu, "The deep learning—Based recommender system 'pubmende' for choosing a biomedical publication venue: Development and validation study," *J. Med. Internet Res.*, vol. 21, no. 5, May 2019, Art. no. e12957.

[87] K. Fukushima, "Neocognitron: A hierarchical neural network capable of visual pattern recognition," *Neural Netw.*, vol. 1, no. 2, pp. 119–130, 1988.

[88] A. Gainaru, E. Slusanschi, and S. Trausan-Matu, "Mapping data mining algorithms on a GPU architecture: A study," in *Proc. Found. Intell. Syst. 19th Int. Symp., (ISMIS)*, in Lecture Notes in Computer Science, vol. 6804. M. Kryszkiewicz, H. Rybinski, A. Skowron, and Z. W. Ras, Eds. Warsaw, Poland: Springer, Jun. 2011, pp. 102–112.

[89] C. Gartenberg, "ARM's new edge AI chips promise IoT devices that won't need the cloud," Verge, Washington, DC, USA, Tech. Rep., Feb. 2020. [Online]. Available: https://www.theverge.com/2020/2/10/21130800/arm-new-edge-ai-chips-processing-npu-cortex-m55-u55-iot

[90] A. Ghaffari and Y. Savaria, "CNN2Gate: An implementation of convolutional neural networks inference on FPGAs with automated design space exploration," *Electronics*, vol. 9, no. 12, p. 2200, Dec. 2020.

[91] V. Gokhale, J. Jin, A. Dundar, B. Martini, and E. Culurciello, "A 240 G-ops/s mobile coprocessor for deep neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops*, Jun. 2014, pp. 696–701.

[92] K. M. V. Gowda, S. Madhavan, S. Rinaldi, P. B. Divakarachari, and A. Atmakur, "FPGA-based reconfigurable convolutional neural network accelerator using sparse and convolutional optimization," *Electronics*, vol. 11, no. 10, p. 1653, May 2022.

[93] H. Graf, S. Cadambi, V. Jakkula, M. Sankaradass, E. Cosatto, S. Chakradhar, and I. Dourdanovic, "A massively parallel digital learning processor," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 21, D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, Eds. Red Hook, NY, USA: Curran Associates, 2009, pp. 1–8.

[94] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu, "A survey of deep learning techniques for autonomous driving," 2019, *arXiv:1910.07738*.

[95] Y. Guan, H. Liang, N. Xu, W. Wang, S. Shi, X. Chen, G. Sun, W. Zhang, and J. Cong, "FP-DNN: An automated framework for mapping deep neural networks onto FPGAs with RTL-HLS hybrid templates," in *Proc. IEEE 25th Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, Apr. 2017, pp. 152–159.

[96] K. Guo, L. Sui, J. Qiu, J. Yu, J. Wang, S. Yao, S. Han, Y. Wang, and H. Yang, "Angel-Eye: A complete design flow for mapping CNN onto embedded FPGA," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 1, pp. 35–47, Jan. 2018.

[97] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, "[DL] A survey of FPGA-based neural network inference accelerators," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 12, no. 1, pp. 1–26, 2019.

[98] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *Proc. 32nd Int. Conf. Mach. Learn. (ICML)*, vol. 37, 2015, pp. 1737–1746.

[99] F. G. Gustavson, "Two fast algorithms for sparse matrices: Multiplication and permuted transposition," *ACM Trans. Math. Softw.*, vol. 4, no. 3, pp. 250–269, Sep. 1978.

[100] A. Guzhva, S. Dolenko, and I. Persiantsev, "Multifold acceleration of neural network computations using gpu," in *Proc. 19th Int. Conf. Artif. Neural Networks, I*. Berlin, Germany: Springer-Verlag, 2009, pp. 373–380.

[101] R. Hadsell, P. Sermanet, J. Ben, A. Erkan, M. Scoffier, K. Kavukcuoglu, U. Müller, and Y. LeCun, "Learning long-range vision for autonomous off-road driving," *J. Field Robot.*, vol. 26, no. 2, pp. 120–144, Feb. 2009.

[102] S. Han, J. Kang, H. Mao, Y. Hu, X. Li, Y. Li, D. Xie, H. Luo, S. Yao, Y. Wang, H. Yang, and W. J. Dally, "ESE: Efficient speech recognition engine with sparse LSTM on FPGA," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2017, pp. 75–84.

[103] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "EIE: Efficient inference engine on compressed deep neural network," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 243–254.

[104] F. Hannig, V. Lari, S. Boppu, A. Tanase, and O. Reiche, "Invasive tightly-coupled processor arrays: A domain-specific architecture/compiler co-design approach," *ACM Trans. Embedded Comput. Syst.*, vol. 13, no. 4S, pp. 1–29, Jul. 2014.

[105] C. Hao, A. Sarwari, Z. Jin, H. Abu-Haimed, D. Sew, Y. Li, X. Liu, B. Wu, D. Fu, J. Gu, and D. Chen, "A hybrid GPU + FPGA system design for autonomous driving cars," in *Proc. IEEE Int. Workshop Signal Process. Syst. (SiPS)*, Oct. 2019, pp. 121–126.

[106] L. Hardesty, "Researchers build an all-optical transistor," Massachusetts Inst. Technol., Cambridge, MA, USA, Tech. Rep., 2013. [Online]. Available: https://news.mit.edu/2013/computing-with-light-0704

[107] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 1026–1034.

[108] 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.

[109] D. Hefenbrock, J. Oberg, N. T. N. Thanh, R. Kastner, and S. B. Baden, "Accelerating Viola–Jones face detection to FPGA-level using GPUs," in *Proc. 18th IEEE Annu. Int. Symp. Field-Program. Custom Comput. Mach.*, May 2010, pp. 11–18.

[110] C. Heidorn, M. Witterauf, F. Hannig, and J. Teich, "Efficient mapping of CNNs onto tightly coupled processor arrays," *J. Comput.*, vol. 14, no. 8, pp. 541–556, 2019.

[111] A. Howard and S. Gupta. (2020). *Introducing the Next Generation of on-Device Vision Models: Mobilenetv3 and Mobilenetedgetpu*. [Online]. Available: https://ai.googleblog.com/2019/11/introducing-next-generation-on-device.html

[112] H. Hu, J. Li, C. Wu, X. Li, and Y. Chen, "Design and implementation of intelligent speech recognition system based on FPGA," *J. Phys., Conf.*, vol. 2171, no. 1, Jan. 2022, Art. no. 012010.

[113] A. S. Hussein, A. Anwar, Y. Fahmy, H. Mostafa, K. N. Salama, and M. Kafafy, "Implementation of a DPU-based intelligent thermal imaging hardware accelerator on FPGA," *Electronics*, vol. 11, no. 1, p. 105, Dec. 2021.

[114] D. Im, D. Han, S. Choi, S. Kang, and H.-J. Yoo, "DT-CNN: Dilated and transposed convolution neural network accelerator for real-time image segmentation on mobile devices," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2019, pp. 1–5.

[115] Texas Instruments. (2015). *Am5729 Sitara Processor*. [Online]. Available: https://www.ti.com/product/AM5729

[116] H. Irmak, N. Alachiotis, and D. Ziener, "An energy-efficient FPGA-based convolutional neural network implementation," in *Proc. 29th Signal Process. Commun. Appl. Conf. (SIU)*, Jun. 2021, pp. 1–4.

[117] H. Irmak, F. Corradi, P. Detterer, N. Alachiotis, and D. Ziener, "A dynamic reconfigurable architecture for hybrid spiking and convolutional FPGA-based neural network designs," *J. Low Power Electron. Appl.*, vol. 11, no. 3, p. 32, Aug. 2021.

[118] S. M. A. H. Jafri, T. N. Gia, S. Dytckov, M. Daneshtalab, A. Hemani, J. Plosila, and H. Tenhunen, "NeuroCGRA: A CGRA with support for neural networks," in *Proc. Int. Conf. High Perform. Comput. Simul. (HPCS)*, Jul. 2014, pp. 506–511.

[119] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. B. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," 2014, *arXiv:1408.5093*.

[120] N. P. Jouppi et al., "In-datacenter performance analysis of a tensor processing unit," in *Proc. 44th Annu. Int. Symp. Comput. Archit.*, 2017, pp. 1–12.

[121] D. Justus, J. Brennan, S. Bonner, and A. S. McGough, "Predicting the computational cost of deep learning models," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2018, pp. 3873–3882.

[122] S. Kalapothas, G. Flamis, and P. Kitsos, "Efficient edge-AI application deployment for FPGAs," *Information*, vol. 13, no. 6, p. 279, May 2022.

[123] A. Karpathy, "Convolutional neural networks for visual recognition," GitHub, Tech. Rep., 2018. [Online]. Available: https://cs231n.github.io/convolutional-networks/

[124] M. Kavitha, R. Srinivasan, and R. Bhuvanya, *Fake News Detection Using Machine Learning Algorithms*. Hoboken, NJ, USA: Wiley, 2022, ch. 10, pp. 181–207.

[125] H. Khan, A. Khan, Z. Khan, L. B. Huang, K. Wang, and L. He, "NPE: An FPGA-based overlay processor for natural language processing," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2021, pp. 1–11.

[126] J.-Y. Kim, "FPGA based neural network accelerators," in *Hardware Accelerator Systems for Artificial Intelligence and Machine Learning* (Advances in Computers), vol. 122, S. Kim and G. C. Deka, Eds. Amsterdam, The Netherlands: Elsevier, 2021, pp. 135–165.

[127] Y. Kim, J. Lee, J.-S. Kim, H. Jei, and H. Roh, "Efficient multi-GPU memory management for deep learning acceleration," in *Proc. IEEE 3rd Int. Workshops Found. Appl. Self Syst. (FASW)*, Sep. 2018, pp. 37–43.

[128] J. P. Klock, J. Correa, M. Bessa, J. Arias-Garcia, F. Barboza, and C. Meinertz, "A new automated energy meter fraud detection system based on artificial intelligence," in *Proc. 11th Brazilian Symp. Comput. Syst. Eng. (SBESC)*, Nov. 2021, pp. 1–8.

[129] A. Kojima and Y. Nose, "Development of an autonomous driving robot car using FPGA," in *Proc. Int. Conf. Field-Programmable Technol. (FPT)*, Dec. 2018, pp. 411–414.

[130] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017.

[131] H. Kwon, A. Samajdar, and T. Krishna, "MAERI: Enabling flexible dataflow mapping over DNN accelerators via reconfigurable interconnects," *ACM Architectural Support Program. Lang. Operating Syst.*, vol. 53, pp. 461–475, Mar. 2018.

[132] A. Lavin and S. Gray, "Fast algorithms for convolutional neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 4013–4021.

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

[134] J. Lee and J. Lee, "NP-CGRA: Extending CGRAs for efficient processing of light-weight deep neural networks," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Feb. 2021, pp. 1408–1413.

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

[136] J. Lee and H.-J. Yoo, "An overview of energy-efficient hardware accelerators for on-device deep-neural-network training," *IEEE Open J. Solid-State Circuits Soc.*, vol. 1, pp. 115–128, 2021.

[137] M. Lee, K. Hwang, J. Park, S. Choi, S. Shin, and W. Sung, "FPGA-based low-power speech recognition with recurrent neural networks," in *Proc. IEEE Int. Workshop Signal Process. Syst. (SiPS)*, Oct. 2016, pp. 230–235.
[138] D. I. Lewin, "DNA computing," *Computing Sci. Eng.*, vol. 4, no. 3, pp. 5–8, May 2002.
[139] B. Li, E. Zhou, B. Huang, J. Duan, Y. Wang, N. Xu, J. Zhang, and H. Yang, "Large scale recurrent neural network on GPU," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2014, pp. 4062–4069.
[140] X. Lian, Z. Liu, Z. Song, J. Dai, W. Zhou, and X. Ji, "High-performance FPGA-based CNN accelerator with block-floating-point arithmetic," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 8, pp. 1874–1885, Aug. 2019.
[141] D. Liu, T. Chen, S. Liu, J. Zhou, S. Zhou, O. Teman, X. Feng, X. Zhou, and Y. Chen, "Pudiannao: A polyvalent machine learning accelerator," in *Proc. 20th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, New York, NY, USA, 2015, pp. 369–381.
[142] "Learn more about the Linaro machine learning initiative," Arm The Architecture for the Digital World, Linaro, Cambridge, U.K., Tech. Rep., Jan. 2019. [Online]. Available: https://www.linaro.org/news/linaro-announces-launch-of-machine-intelligence-initiative/
[143] (Aug. 2021). *Ethos-U55 Arm Developer*. Accessed: Aug. 7, 2021. [Online]. Available: https://developer.arm.com/Processors/Ethos-U55
[144] (Aug. 2021). *High-Performing AI Solutions to Transform our Digital World*. Accessed: Aug. 7, 2021. [Online]. Available: https://www.google.com/search?client=firefox-b-d&q=High-Performing+AI+Solutions+to+Transform+our+Digital+World
[145] W. Lu, G. Yan, J. Li, S. Gong, Y. Han, and X. Li, "FlexFlow: A flexible dataflow accelerator architecture for convolutional neural networks," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2017, pp. 553–564.
[146] T. Luo, S. Liu, L. Li, Y. Wang, S. Zhang, T. Chen, Z. Xu, O. Temam, and Y. Chen, "Dadiannao: A neural network supercomputer," *IEEE Trans. Comput.*, vol. 66, no. 1, pp. 73–88, Jan. 2017.
[147] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," in *Proc. EMNLP*. Lisbon, Portugal: Association for Computational Linguistics, Aug. 2015, pp. 1412–1421.
[148] P. Lv, W. Liu, and J. Li, "A FPGA-based accelerator implementaion for YOLOv2 object detection using Winograd algorithm," in *Proc. 5th Int. Conf. Mech., Control Comput. Eng. (ICMCCE)*, Dec. 2020, pp. 1894–1898.
[149] A. L. Maas, "Rectifier nonlinearities improve neural network acoustic models," Stanford Univ., Stanford, CA, USA, Tech. Rep., 2013.
[150] R. Machupalli, M. Hossain, and M. Mandal, "Review of ASIC accelerators for deep neural network," *Microprocessors Microsyst.*, vol. 89, Mar. 2022, Art. no. 104441.
[151] M. Mathieu, M. Henaff, and Y. LeCun, "Fast training of convolutional networks through FFTs," 2014, *arXiv:1312.5851*.
[152] B. Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins, "Adres: An architecture with tightly coupled VLIW processor and coarse-grained reconfigurable matrix," in *Field Programmable Logic and Application*, P. Y. K. Cheung and G. A. Constantinides, Eds. Berlin, Germany: Springer, 2003, pp. 61–70.
[153] J. Misra and I. Saha, "Artificial neural networks in hardware: A survey of two decades of progress," *Neurocomputing*, vol. 74, nos. 1–3, pp. 239–255, Dec. 2010.
[154] S. Mittal, "A survey of FPGA-based accelerators for convolutional neural networks," *Neural Comput. Appl.*, vol. 32, no. 4, pp. 1109–1139, Feb. 2020.
[155] S. Mittal and J. S. Vetter, "A survey of CPU-GPU heterogeneous computing techniques," *ACM Comput. Surveys*, vol. 47, no. 4, pp. 1–35, Jul. 2015.
[156] P. Mohan, A. J. Paul, and A. Chirania, "A tiny CNN architecture for medical face mask detection for resource-constrained endpoints," in *Innovations in Electrical and Electronic Engineering* (Lecture Notes in Electrical Engineering). Singapore: Springer, 2021, pp. 657–670.
[157] J. J. Moolayil, "A Layman's guide to deep neural networks—Towards data science," *Medium*, May 2020. [Online]. Available: https://towardsdatascience.com/a-laymans-guide-to-deep-neural-networks-ddcea24847fb
[158] D. Moolchandani, A. Kumar, and S. R. Sarangi, "Accelerating CNN inference on ASICs: A survey," *J. Syst. Archit.*, vol. 113, Feb. 2021, Art. no. 101887.
[159] N. Muralimanohar, R. Balasubramonian, and N. Jouppi, "Optimizing NUCA organizations and wiring alternatives for large caches with CACTI 6.0," in *Proc. 40th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Dec. 2007, pp. 3–14.
[160] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *Proc. 27th Int. Conf. Int. Conf. Mach. Learn.* Madison, WI, USA: Omnipress, 2010, pp. 807–814.
[161] D. T. Nguyen, T. N. Nguyen, H. Kim, and H. J. Lee, "A high-throughput and power-efficient FPGA implementation of YOLO CNN for object detection," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 8, pp. 1861–1873, Aug. 2019.
[162] T. Ngyen, S. M. A. H. Jafri, M. Daneshtalab, A. Hemani, S. Dytckov, J. Plosila, and H. Tenhunen, "FIST: A framework to interleave spiking neural networks on CGRAs," in *Proc. 23rd Euromicro Int. Conf. Parallel, Distrib., Network-Based Process.*, Mar. 2015, pp. 751–758.
[163] R. Nikhil, "Bluespec System Verilog: Efficient, correct RTL from high level specifications," in *Proc. 2nd ACM IEEE Int. Conf. Formal Methods Models Co-Design*, Jun. 2004, pp. 69–70.
[164] E. Nurvitadhi, D. Sheffield, J. Sim, A. Mishra, G. Venkatesh, and D. Marr, "Accelerating binarized neural networks: Comparison of FPGA, CPU, GPU, and ASIC," in *Proc. Int. Conf. Field-Program. Technol. (FPT)*, Dec. 2016, pp. 77–84.
[165] E. Nurvitadhi, G. Venkatesh, J. Sim, D. Marr, R. Huang, J. O. G. Hock, Y. T. Liew, K. Srivatsan, D. Moss, S. Subhaschandra, and G. Boudoukh, "Can FPGAs beat GPUs accelerating next-generation deep neural networks?" in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, New York, NY, USA, 2017, pp. 5–14.
[166] M. T. Nyamukuru and K. M. Odame, "Tiny Eats: Eating detection on a microcontroller," in *Proc. IEEE 2nd Workshop Mach. Learn. Edge Sensor Syst. (SenSys-ML)*, Apr. 2020, pp. 19–23.
[167] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally, "SCNN: An accelerator for compressed-sparse convolutional neural networks," in *Proc. 44th Annu. Int. Symp. Comput. Archit.*, New York, NY, USA, Jun. 2017, pp. 27–40.
[168] S.-W. Park, J. Park, K. Bong, D. Shin, J. Lee, S. Choi, and H.-J. Yoo, "An energy-efficient and scalable deep learning/inference processor with tetra-parallel MIMD architecture for big data applications," *IEEE Trans. Biomed. Circuits Syst.*, vol. 9, no. 6, pp. 838–848, Dec. 2015.
[169] M. Peemen, A. A. A. Setio, B. Mesman, and H. Corporaal, "Memory-centric accelerator design for convolutional neural networks," in *Proc. IEEE 31st Int. Conf. Comput. Design (ICCD)*, Oct. 2013, pp. 13–19.
[170] P.-H. Pham, D. Jelaca, C. Farabet, B. Martini, Y. LeCun, and E. Culurciello, "NeuFlow: Dataflow vision processing system-on-a-chip," in *Proc. IEEE 55th Int. Midwest Symp. Circuits Syst. (MWSCAS)*, Aug. 2012, pp. 1044–1047.
[171] M. Pietras, "Hardware conversion of neural networks simulation models for neural processing accelerator implemented as FPGA-based SoC," in *Proc. 24th Int. Conf. Field Program. Log. Appl. (FPL)*, Sep. 2014, pp. 1–4.
[172] T. Posewsky and D. Ziener, "Efficient deep neural network acceleration through FPGA-based batch processing," in *Proc. Int. Conf. ReConFigurable Comput. FPGAs (ReConFig)*, Nov. 2016, pp. 1–8.
[173] T. Posewsky and D. Ziener, "Throughput optimizations for FPGA-based deep neural network inference," *Microprocessors Microsyst.*, vol. 60, pp. 151–161, Jul. 2018.
[174] S. Prakash, T. Callahan, J. Bushagour, C. Banbury, A. V. Green, P. Warden, T. Ansell, and V. J. Reddi, "CFU playground: Full-stack open-source framework for tiny machine learning (tinyML) acceleration on FPGAs," 2022, *arXiv:2201.01863*.
[175] W. Qadeer, R. Hameed, O. Shacham, P. Venkatesan, C. Kozyrakis, and M. Horowitz, "Convolution engine: Balancing efficiency and flexibility in specialized computing," *Commun. ACM*, vol. 58, no. 4, pp. 85–93, Mar. 2015.
[176] E. Qin, A. Samajdar, H. Kwon, V. Nadella, S. Srinivasan, D. Das, B. Kaul, and T. Krishna, "SIGMA: A sparse and irregular GEMM accelerator with flexible interconnects for DNN training," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2020, pp. 58–70.
[177] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song, Y. Wang, and H. Yang, "Going deeper with embedded FPGA platform for convolutional neural network," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, New York, NY, USA, Feb. 2016, pp. 26–35.

[178] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, and J. Kepner, "AI accelerator survey and trends," in *Proc. IEEE High Perform. Extreme Comput. Conf. (HPEC)*, Sep. 2021, pp. 1–9.

[179] M. Rhu, N. Gimelshein, J. Clemons, A. Zulfiqar, and S. W. Keckler, "VDNN: Virtualized deep neural networks for scalable, memory-efficient neural network design," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*. Piscataway, NJ, USA: IEEE Press, Oct. 2016, pp. 1–13.

[180] T. Ridnik, H. Lawen, A. Noy, E. Ben Baruch, G. Sharir, and I. Friedman, "TResNet: High performance GPU-dedicated architecture," 2020, *arXiv:2003.13630*.

[181] S. Saha, "A comprehensive guide to convolutional neural networks—The ELI5 way," Towards Data Sci., Toronto, ON, Canada, Tech. Rep., 2018. [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[182] M. Sankaradas, V. Jakkula, S. Cadambi, S. Chakradhar, I. Durdanovic, E. Cosatto, and H. P. Graf, "A massively parallel coprocessor for convolutional neural networks," in *Proc. 20th IEEE Int. Conf. Appl.-Specific Syst., Archit. Processors*, Jul. 2009, pp. 53–60.

[183] V. Sati, S. M. Sánchez, N. Shoeibi, A. Arora, and J. M. Corchado, "Face detection and recognition, face emotion recognition through NVIDIA Jetson Nano," in *Proc. Int. Symp. Ambient Intell.* Cham, Switzerland: Springer, 2020, pp. 177–185.

[184] S. Saglam, F. Tat, and S. Bayar, "FPGA implementation of CNN algorithm for detecting malaria diseased blood cells," in *Proc. Int. Symp. Adv. Electr. Commun. Technol. (ISAECT)*, Nov. 2019, pp. 1–5.

[185] U. Schmidt and S. Roth, "Shrinkage fields for effective image restoration," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2014, pp. 2774–2781.

[186] D. Selvathi, R. D. Nayagam, D. J. Hemanth, and V. E. Balas, "FPGA implementation of on-chip ANN for breast cancer diagnosis," *Intell. Decis. Technol.*, vol. 10, no. 4, pp. 341–352, Dec. 2016.

[187] H. Sharma, J. Park, D. Mahajan, E. Amaro, J. K. Kim, C. Shao, A. Mishra, and H. Esmaeilzadeh, "From high-level deep neural models to FPGAs," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2016, pp. 1–12.

[188] H. Sharma, J. Park, N. Suda, L. Lai, B. Chau, V. Chandra, and H. Esmaeilzadeh, "Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network," in *Proc. ACM/IEEE 45th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2018, pp. 764–775.

[189] R. Shi, H. Xu, B. Chen, Z. Zhang, and L.-M. Peng, "Scalable fabrication of graphene devices through photolithography," *Appl. Phys. Lett.*, vol. 102, no. 11, Mar. 2013, Art. no. 113102.

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

[191] M. M. Shulaker, G. Hills, R. S. Park, R. T. Howe, K. Saraswat, H.-S. P. Wong, and S. Mitra, "Three-dimensional integration of nanotechnologies for computing and data storage on a single chip," *Nature*, vol. 547, pp. 74–78, Jul. 2017.

[192] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.

[193] G. Smith and F. F. Leymarie, "The machine as artist: An introduction," *Arts*, vol. 6, no. 4, p. 5, Apr. 2017.

[194] S. Srinivas, R. K. Sarvadevabhatla, K. R. Mopuri, and N. Prabhu, "Kruthiventi, and R. V. Babu," A taxonomy of deep convolutional neural nets for computer vision," *Frontiers Robot. AI*, vol. 2, p. 36, Jan. 2016.

[195] V. Sriram, D. Cox, K. H. Tsoi, and W. Luk, "Towards an embedded biologically-inspired machine vision processor," in *Proc. Int. Conf. Field-Programmable Technol.*, Dec. 2010, pp. 273–278.

[196] D. Strigl, K. Kofler, and S. Podlipnig, "Performance and scalability of GPU-based convolutional neural networks," in *Proc. 18th Euromicro Conf. Parallel, Distrib. Network-Based Process.*, Feb. 2010, pp. 317–324.

[197] N. Suda, V. Chandra, G. Dasika, A. Mohanty, Y. Ma, S. Vrudhula, J.-S. Seo, and Y. Cao, "Throughput-optimized OpenCL-based FPGA accelerator for large-scale convolutional neural networks," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, New York, NY, USA, Feb. 2016, pp. 16–25.

[198] M. Svedin, S. W. D. Chien, G. Chikafa, N. Jansson, and A. Podobas, "Benchmarking the NVIDIA GPU lineage: From early K 80 to modern A 100 with asynchronous memory transfers," in *Proc. 11th Int. Symp. Highly Efficient Accel. Reconfigurable Technol.*, Jun. 2021, pp. 1–6.

[199] D.-F. Syu, S.-W. Syu, S.-J. Ruan, Y.-C. Huang, and C.-K. Yang, "FPGA implementation of automatic speech recognition system in a car environment," in *Proc. IEEE 4th Global Conf. Consum. Electron. (GCCE)*, Oct. 2015, pp. 485–486.

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

[201] M. A. Talib, S. Majzoub, Q. Nasir, and D. Jamal, "A systematic literature review on hardware implementation of artificial intelligence algorithms," *J. Supercomput.*, vol. 77, no. 2, pp. 1897–1938, Feb. 2021.

[202] M. Tanomoto, S. Takamaeda-Yamazaki, J. Yao, and Y. Nakashima, "A CGRA-based approach for accelerating convolutional neural networks," in *Proc. IEEE 9th Int. Symp. Embedded Multicore/Many-Core Syst. Chip*, Sep. 2015, pp. 73–80.

[203] Y. Tkachenko, "Autonomous CRM control via CLV approximation with deep reinforcement learning in discrete and continuous action space," 2015, *arXiv:1504.01840*.

[204] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "FINN: A framework for fast, scalable binarized neural network inference," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2017, pp. 65–74.

[205] A. Vasudevan, A. Anderson, and D. Gregg, "Parallel multi channel convolution using general matrix multiplication," in *Proc. IEEE 28th Int. Conf. Appl.-Specific Syst., Archit. Processors (ASAP)*, Jul. 2017, pp. 19–24.

[206] S. I. Venieris and C.-S. Bouganis, "FpgaConvNet: A framework for mapping convolutional neural networks on FPGAs," in *Proc. IEEE 24th Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, May 2016, pp. 40–47.

[207] T. V. Huynh, "FPGA-based acceleration for convolutional neural networks on PYNQ-Z2," *Int. J. Comput. Digit. Syst.*, vol. 11, no. 1, pp. 441–449, Jan. 2022.

[208] C. Wang, L. Gong, Q. Yu, X. Li, Y. Xie, and X. Zhou, "DLAU: A scalable deep learning accelerator unit on FPGA," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 36, no. 3, pp. 513–517, Mar. 2017.

[209] J. Wang and S. Gu, "FPGA implementation of object detection accelerator based on Vitis-AI," in *Proc. 11th Int. Conf. Inf. Sci. Technol. (ICIST)*, May 2021, pp. 571–577.

[210] T. Wang, C. Wang, X. Zhou, and H. Chen, "A survey of FPGA based deep learning accelerators: Challenges and opportunities," 2018, *arXiv:1901.04988*.

[211] Y. Wang, J. Xu, Y. Han, H. Li, and X. Li, "DeepBurning: Automatic generation of FPGA-based learning accelerators for the neural network family," in *Proc. 53rd Annu. Design Autom. Conf.*, Jun. 2016, pp. 1–6.

[212] S. Williams, A. Waterman, and D. Patterson, "Roofline: An insightful visual performance model for multicore architectures," *Commun. ACM*, vol. 52, no. 4, pp. 65–76, 2009.

[213] W. Vanderbauwhede and K. Benkrid, *High-Performance Computing Using FPGAs*. New York, NY, USA: Springer, 2013.

[214] W. G. Wong, "More details emerge about arm's machine learning," Electron. Des. Mag., Hasbrouck Heights, NJ, USA, Tech. Rep., Jun. 2018. [Online]. Available: https://www.electronicdesign.com/industrial-automation/article/21806582/more-details-emerge-about-arms-machine-learning

[215] B. Wu, A. Wan, F. Iandola, P. H. Jin, and K. Keutzer, "SqueezeDet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jul. 2017, pp. 446–454.

[216] H. Xiao, K. Zhao, and G. Liu, "Efficient hardware accelerator for compressed sparse deep neural network," *IEICE Trans. Inf. Syst.*, vol. 104, no. 5, pp. 772–775, May 2021.

[217] S. Xiong, G. Wu, X. Fan, X. Feng, Z. Huang, W. Cao, X. Zhou, S. Ding, J. Yu, L. Wang, and Z. Shi, "MRI-based brain tumor segmentation using FPGA-accelerated neural network," *BMC Bioinf.*, vol. 22, no. 1, pp. 1–15, Dec. 2021.

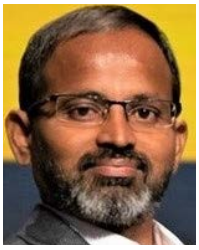
[218] A. Yazdanbakhsh, J. Park, H. Sharma, P. Lotfi-Kamran, and H. Esmaeilzadeh, "Neural acceleration for GPU throughput processors," in *Proc. 48th Int. Symp. Microarchitecture*, Dec. 2015, pp. 482–493.

[219] K. Seshadri, B. Akin, J. Laudon, R. Narayanaswami, and A. Yazdanbakhsh, "An evaluation of edge TPU accelerators for convolutional neural networks," 2021, *arXiv:2102.10423*.

[220] X. Yin, L. Chen, X. Zhang, and Z. Gao, "Object detection implementation and optimization on embedded GPU system," in *Proc. IEEE Int. Symp. Broadband Multimedia Syst. Broadcast. (BMSB)*, Jun. 2018, pp. 1–5.
[221] R. Zanc, T. Cioara, and I. Anghel, "Forecasting financial markets using deep learning," in *Proc. IEEE 15th Int. Conf. Intell. Comput. Commun. Process. (ICCP)*, Sep. 2019, pp. 459–466.
[222] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, Feb. 2015, pp. 161–170.
[223] C. Zhang, G. Sun, Z. Fang, P. Zhou, P. Pan, and J. Cong, "Caffeine: Toward uniformed representation and acceleration for deep convolutional neural networks," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 11, pp. 2072–2085, Nov. 2019.
[224] G. Zhang, N. Attaluri, J. S. Emer, and D. Sánchez, "Gamma: Leveraging Gustavson's algorithm to accelerate sparse matrix multiplication," in *Proc. 26th ACM Int. Conf. Architectural Support Program. Lang. Operating Syst.*, Apr. 2021, pp. 687–701.
[225] J.-F. Zhang, C.-E. Lee, C. Liu, Y. S. Shao, S. W. Keckler, and Z. Zhang, "SNAP: A 1.67—21.55 TOPS/W sparse neural acceleration processor for unstructured sparse deep neural network inference in 16 nm CMOS," in *Proc. Symp. VLSI Circuits*, Jun. 2019, pp. C306–C307.
[226] Z.-Q. Zhao, P. Zheng, S.-T. Xu, and X. Wu, "Object detection with deep learning: A review," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 30, no. 11, pp. 3212–3232, Nov. 2019.
[227] J. Zhu, L. Wang, H. Liu, S. Tian, Q. Deng, and J. Li, "An efficient task assignment framework to accelerate DPU-based convolutional neural network inference on FPGAs," *IEEE Access*, vol. 8, pp. 83224–83237, 2020.
[228] J. Zhu, T. Yang, R. Liu, X. Xu, and X. Zhu, "Image recognition of CT diagnosis for cholangiocarcinoma treatment based on FPGA processor and neural network," *Microprocessors Microsyst.*, vol. 81, Mar. 2021, Art. no. 103645.
[229] S. Monk, *Programming the Raspberry Pi: Getting Started With Python*. New York, NY, USA: McGraw-Hill Education, 2016.
[230] (Nov. 2022). *Photos of the Raspberry Pi Through the Ages: From the Prototype to Pi3 B+*. Accessed: Nov. 14, 2022. [Online]. Available: https://www.zdnet.com/pictures/photos-of-the-raspberry-pi-through-the-ages-from-the-prototype-to-pi-3/

PUDI DHILLESWARARAO (Graduate Student Member, IEEE) received the M.Tech. degree in microelectronics and VLSI from the National Institute of Technology, Durgapur, India, in 2014. He is currently pursuing the Ph.D. degree with the School of Electrical Sciences, Indian Institute of Technology, Bhubaneswar, India. His current research interests include programmable hardware accelerators, compilers, and high-level synthesis.

SRINIVAS BOPPU (Member, IEEE) received the international M.Sc. degree in IC design from Nanyang Technological University, Singapore, and the Technical University of Munich, Germany, and the Ph.D. degree from the Chair for Hardware/Software Co-Design, Department of Computer Science, University of Erlangen–Nuremberg, Germany in 2015. He was a Senior Consultant at Infineon Technologies Munich, Germany. He also worked at Freescale Semiconductors India and ST Microelectronics as a Physical Design Engineer. He has been an Assistant Professor with the School of Electrical Sciences, IIT Bhubaneswar, since October 2017. He has more than 15 years of experience in both academia and industry in the field of the VLSI domain. He received the Full Scholarship from Infineon Technologies Asia Pacific Pte. Ltd., for M.Sc. degree. His research interests include high-level synthesis, programmable hardware accelerators, compilers, scheduling and mapping approaches, low-power VLSI design, SoC design, and design automation of integrated circuits.



M. SABARIMALAI MANIKANDAN (Senior Member, IEEE) received the B.E. degree in electronic and communication engineering from Bharathiar University, Coimbatore, India, the M.E. degree in microwave and optical engineering from Madurai Kamaraj University, Madurai, India, and the Ph.D. degree in cardiovascular signal processing from the Department of Electronics and Communication Engineering, IIT Guwahati, Guwahati, India. He was an Assistant Professor at Amrita Vishwa Vidyapeetham University, Ettimadai, India. He was the Chief Engineer at the Advanced Technology Group, Samsung India Electronic Pvt., Ltd., Noida, India. He was an Assistant Professor at the Biomedical System Laboratory, School of Electrical Sciences, IIT Bhubaneswar, India. He is currently an Associate Professor of electrical engineering with IIT Palakkad. He has published more than 70 research papers in reputed journals and conference proceedings. His research interests include signal and image processing, adaptive machine learning, the Internet of Things, VLSI signal processing, machine learning architectures, application system development: health (human, machine, structural) monitoring systems, audio and speech processing systems for human–machine interactions, biometric and data security for authentication and authorization, environmental monitoring systems for ambient assisted living, UAV-assisted IoT for smart surveillance systems, and context and quality aware pattern learning networks for event recognition. He was a recipient of the 2012 Outstanding Performance Award during his tenure at Samsung India Electronic Pvt., Ltd. He served as a Reviewer for many reputed journals of the IEEE, IET, Springer, Hindawi, PLOS One, Frontiers, and Elsevier.

LINGA REDDY CENKERAMADDI (Senior Member, IEEE) received the master's degree in electrical engineering from the Indian Institute of Technology Delhi (IIT Delhi), New Delhi, India, in 2004, and the Ph.D. degree in electrical engineering from the Norwegian University of Science and Technology (NTNU), Trondheim, Norway, in 2011.

He worked on mixed-signal circuit design at Texas Instruments. He worked on radiation imaging for an atmosphere-space interaction monitor (ASIM mission to the International Space Station) at the University of Bergen, Bergen, Norway, from 2010 to 2012. He is currently the Leader of the Autonomous and Cyber-Physical Systems (ACPS) Research Group and a Professor with the University of Agder, Grimstad, Norway. Several of his master's students received the Best Master Thesis Awards in information and communication technology (ICT). He has coauthored over 120 research publications that have been published in prestigious international journals and standard conferences. His main scientific interests include cyber-physical systems, autonomous systems, and wireless embedded systems.

Dr. Cenkeramaddi is a member of the editorial boards of various international journals and the technical program committees of several IEEE conferences. He is the Principal Investigator and a Co-Principal Investigator of many research grants from the Norwegian Research Council.