Microprocessors and Microsystems 89 (2022) 104441

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

Review of ASIC accelerators for deep neural network

Raju Machupalli, Masum Hossain, Mrinal Mandal *

Department of Electrical and Computer Engineering, University of Alberta, Edmonton, Canada

ARTICLE INFO

Keywords:
Deep neural network
Hardware accelerator
Neural processor
Domain specific accelerator
ASIC

ABSTRACT

Deep neural networks (DNNs) have become an essential tool in artificial intelligence, with a wide range of applications such as computer vision, medical diagnosis, security, robotics, and autonomous vehicle. The DNNs deliver the state-of-the-art performance in many applications. The complexity of the DNN models generally increases with application complexity and deployment of complex DNN models requires high computational power. General-purpose processors are unable to process complex DNNs within the required throughput, latency, and power budget. Therefore, domain-specific hardware accelerators are required to provide high computational resources with superior energy efficiency and throughput within a small chip area. In this paper, existing DNN hardware accelerators are reviewed and classified based on the optimization techniques used in their implementations. Each optimization technique generally improves one or more specific performance parameter(s). For example, the hardware optimized for sparse DNNs may provide poor performance for dense DNNs in terms of power and throughput. Therefore, understanding the tradeoff between different hardware accelerators helps to identify the best accelerator model for application deployment. We identify three major areas, ALU, dataflow, and sparsity, in hardware architectures having the potential to improve the overall performance of an accelerator. Existing hardware accelerators for inference are broadly classified into these three categories. As there is no standard model or performance metrics to evaluate the efficiency of the new DNN hardwares in the literature, the classification model can help to identify appropriate performance parameters and benchmark accelerators.

1. Introduction

Artificial intelligence is the ability of a system to think, learn, and react like humans without explicit programming. The human brain consists of billions of neurons connected in a complex structure with operational efficiency. Similarly, the creation of an intelligent model requires a large number of well-connected computing units (or small building blocks) and enough examples to train the model. Due to the availability of large quantity of data and computing resources in recent time, the creation of intelligent machine is realizable. Machine learning (ML) is a subsection of artificial intelligence in which a mathematical model is trained over many examples to solve a new problem. Deep neural networks (DNNs) are subsections of ML with a deep network structure and shared weights (filters).

The DNNs have been successfully applied to many problems, such as computer vision [1], robotics [2], security [3], medical diagnosis [4], and self-driving car [5]. Most DNNs are based on the convolutional neural networks (CNN) where output feature maps are typically generated by convolving input feature maps with 3D filters. Recent DNN models have been shown to surpass human performance in some applications. The performance improvements typically come with the increased complexity of the DNNs. As seen in Table 1, the classification of a small size image (e.g., 227 × 227 pixels) requires billions of arithmetic operations (i.e., multiplications and additions). The MCN-MobileNet has 4.19 million parameters (weights) and requires 0.58 billion operations to classify an image. A large size VGG-19 DNN model requires about 20 billion operations per classification. The general-purpose processors, like CPU, are unable to provide such huge computing power with required latency. To deploy the DNNs in real-time applications, the embedded processors must have high throughput and low power consumption. Therefore, the demand for domain-specific accelerators is increasing in recent times as these accelerators can provide superior performance at higher energy efficiency.

There are two main phases in DNNs: training and prediction (or inference). In the training mode, an example input with a known outcome is applied to the model to learn its internal parameters. In the prediction (or inference) mode, the possible outcome is calculated based on the input test data. Training typically requires high precision

* Corresponding author.
E-mail address: mmandal@ualberta.ca (M. Mandal).

https://doi.org/10.1016/j.micpro.2022.104441
Received 30 October 2020; Received in revised form 2 November 2021; Accepted 27 November 2021
Available online 12 January 2022

Table 1
Parameter's size and number of operations required for different DNN models [6].

Model	Input size	Parameters size (millions)	Operations (GOPs)
AlexNet	227 × 227	61.07	0.73
Squeezenet	224 × 224	1.31	0.84
VGG-16	224 × 224	138.41	16
VGG-19	224 × 224	143.65	20
GoogleNet	224 × 224	13.36	2
Resnet-18	224 × 224	11.79	2
Resnet-152	224 × 224	60.29	11
Inception-V3	299 × 299	23.85	6
Densenet-201	224 × 224	20.18	4
MCN-mobileNet	224 × 224	4.19	0.58

numerical representation while low precision representation may be enough for inference [7–10]. Generally, training is done using high power GPUs and data centers. The precision and size of the trained DNN models can be reduced significantly with negligible (<1%) change in the accuracy for inference [11]. In real-time deployment, the trained DNNs need to operate in inference mode only if there is no change in application requirements. Therefore, hardware accelerators for inference mode are more important than for training. Therefore, this paper is mainly focused on the inference mode, and all discussions are subject to inference mode of operation.

Globally, a large number of researchers, in both academia and industry are working towards developing optimized hardware for DNNs inference. DNN accelerators are developed using FPGAs, GPUs, and ASICs. GPUs come with a massively parallel compute units and process DNN computations in parallel. GPUs are power hungry, and this limits their applications in embedded systems. FPGAs have high performance per watt and can be configurable in the fields. FPGAs are often used to prototype and validate the design. ASICs are custom designed for specific application with optimum speed and power consumption. ASIC has more applications at embedded devices. ASIC implementation takes longest development cycle compared to GPUs and FPGAs and have no flexibility after design. Talib et al. [76] reviewed several hardware accelerators for machine learning using FPGA, GPU and ASIC platforms and discussed the advantages of each platform over other platforms. Guo et al. [60] surveyed several FPGA-based neural network accelerator designs and summarized the methods used for the design automation. The FPGA allows less control and flexibility over the multiplication and accumulation (MAC) unit design, which typically limits the exploration of the MAC variants. Li et al. [61] presented an overview of the GPU, FPGA and ASIC based accelerators and a detailed explanation of the DianNao [30] family of accelerators. This has motivated significant progress in the ASIC accelerators after the [61] survey. Camus et al. [12] analyzed the precision scalable MAC units from different accelerators and discussed their benefits in different scenarios. Although, the MAC unit is an important block in the DNN accelerator design to improve the performance, the MAC alone cannot define the overall performance. Hence, along with the MAC unit some other factors in the accelerators are to be analyzed. The MAC utilization depends on the data flow and on-chip memory. An efficient architecture should have 100% MAC utilization. Reuther et al. [13] discussed existing ML accelerators based on peak performance vs. power scatter plot. The accelerators are broadly categorized into six types based on the region in the plot. The factors causing variation in the performance of different accelerators are not well explained in [13].

Du et al. [59] presented an overview of self-aware neural network systems, where a system can predict and adapt dynamics in network parameters such as precision, sparsity and network structure based on the input data. The self-aware techniques can significantly improve the accelerators throughput and energy efficiency, but the accelerators should have some flexibility. For example, a DNN with a variable precision requirement at different layers need a variable precision MAC to adapt and save energy. The survey did not include much information on the implementation techniques to incorporate the flexibility in accelerator implementation and how it affects the overall performance. Sze et al. [7] provided an overview of the DNN development platforms, optimization algorithm, accelerator implementations and benchmarks. The paper has detailed explanation of three different dataflow methods but does not include all recent advances in the arithmetic logic unit and sparsity exploration. Chen et al. [35] reviewed several recent DNN accelerators based on their application and technologies used (e.g., ReRAM, Hybrid Memory Cube). Most surveys provide the architectural and performance improvements of existing DNN accelerators, but it would be helpful to analyze the architectures in a generalized framework.

The existing literatures classify the different DNN accelerators based on their implementation techniques or applications. For example, the accelerators in [35] are reviewed based on architectures (e.g., stand-alone, or co-processor-based) or technologies used (e.g., *Re*-RAM, HMC). Similarly, the accelerators in [13] are classified based on the peak power versus performance tradeoff. In the DNN literature, we identify three major areas for improvements in the DNN architecture: Arithmetic logic unit, dataflow, and sparsity. In this paper, we present a comprehensive review of the ASIC accelerators for the DNN architectures. The state-of-the-art accelerators are classified into three broad categories (i.e., ALU, data flow, and sparsity based) based on their architectural differences. This broad classification can provide more insights to develop generic DNN architectures. Additionally, we have added a fourth section that captures a recent trend of analog-digital hybrid digital implementation for faster computation.

The organization of the paper is as follows. Section II presents the background information and performance criteria of hardware architectures. Section III presents a comprehensive review of the DNN hardware architectures and their classification. Section IV presents evaluation methods and observations in existing accelerators, followed by the conclusions in Section V.

2. Background

The superior performance of DNNs generally comes at the cost of high computations. For example, AlexNet [1] which won the ImageNet challenge [14] in 2012, has 61 M parameters and requires 727 M MAC operations per image classification. Large DNNs may require billions of MAC operations per inference as shown in Table 1. Performing large number of operations sequentially affect the throughput. Existing general-purpose processors (GPPs) may be unable to provide the required computational power and throughput within a low power budget. The GPUs can provide high computational speed but consumes a large amount of power. GPUs can therefore be used at servers where the computational speed is more important than the power requirement. Domain-specific accelerators (e.g., ASICs) are known to provide high energy efficiency (around 1∼10 TOPs/W). The FPGAs have less energy efficiency but have the advantage of reconfigurability.

Real-time deployment of DNN is constrained by energy efficiency and throughput of embedded processors to maximize the battery life. For example, a typical mobile phone has a 2–3 Ah (5 V) battery life (i.e., 15 Wh) and the DNN processing power should be only a fraction of the maximum available power. For real-time data processing, the processor should have the throughput equal to the data collection frame rate (e.g., camera frame rate). Fortunately, there is no interdependency among outputs (i.e., in the same layer output or feature map) in a DNN layer. Therefore, parallel implementation of large MAC units can increase the throughput. An example of the DNN accelerator implementation based on Parallel MAC units is shown in Fig. 1. In general, the size of the accelerator in silicon and power requirement are directly proportional to the number of MAC units (working in parallel) and on chip memory. Note that the DNN specific accelerators will have an array of processing elements (PEs) connected to its neighbors. Each PE contains one to

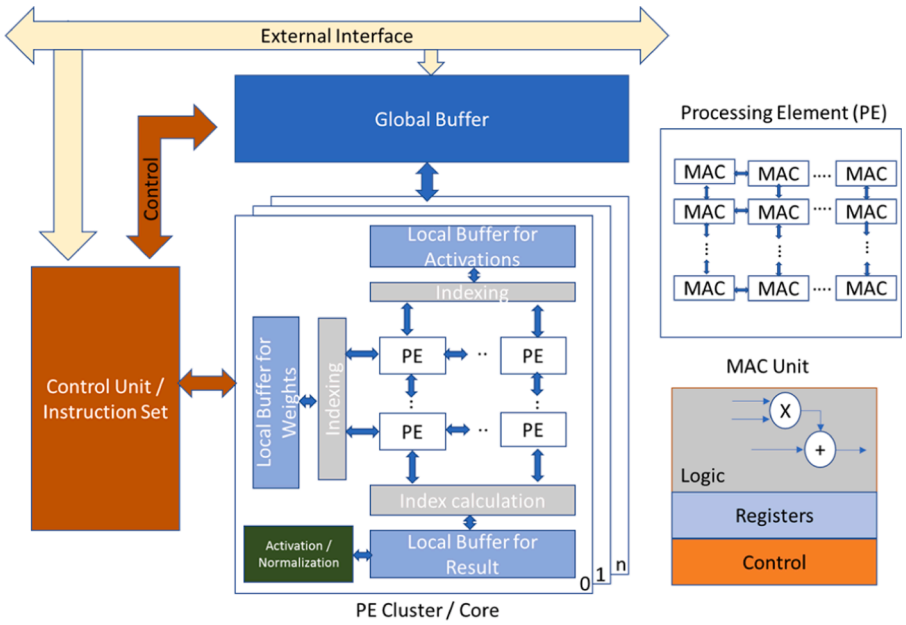


Fig. 1. Block diagram of a generic DNN architecture.

several MAC units connected in such a way that matrix multiplication can be performed with a single instruction. The MAC unit contains a control unit to configure the operation to multiplication or addition or both and register files to store the local parameters and intermediate results. To increase the on-chip storage, the global and local buffer memory blocks are implemented along with PEs.

The cost of the individual MAC units can be reduced with lower bit-length/precision of the MAC units. Energy and area consumption of multiply and add circuits for 4 different precisions are shown in Table 2. An 8-bit fixed point (FIX) add circuit occupies 116x less area and consumes 30x less energy (in picojoules) than a 32-bit floating-point (FL) adder. For multiplication, an 8-bit fixed point circuit consumes 18.5x less energy and occupies 27.3x less area than a 32-bit floating-point. Approximately, energy and area of fixed-point circuits scale linearly for add, quadratically for multiply, with the number of bits [15].

Reduction in the MAC precision can save both the computation and storage requirements. Therefore, the impact of low precision on the accuracy of DNN models has been explored in literature, mainly with respect to quantization [9, 11, 17]. In most DNNs, quantization of weights and activations to less than 16-bit integers can still provide accuracy similar to that of a 32-bit floating-point [17–19]. Linear quantization to 8-bit fixed-point numbers benefits the hardware implementation of the MAC unit, as shown in Table 2. Both energy consumption and silicon footprint increase with the increase in precision when changed from fixed point to floating representation. In Binary network [20], weights and activations are quantized to binary values +1 or -1. Binarization of the network will simplify the multiplication into the XOR operation. Ternary network [21] quantizes the parameters to three levels: -1, 0, and +1. But applications of Binary and Ternary networks are limited.

Depending on the application requirements, the arithmetic operations in a DNN network may be implemented using different bit precisions. Also, there exist models whose optimized bit length varies one layer to the next layer. For example, for a 5-layer Convnet (with 3 convolutional and 2 fully connected layers), the optimized bit length requirement for the 5 layers has been found to be 8–7–7–5–5 bits [22]. In other words, there is no standard precision requirement that is optimal for all layers or models. Therefore, a flexible DNN hardware accelerator (or the associated MAC units) should be able to support all possible bit precisions. For lower precision computations, multiple operations can be performed with a single MAC unit by hardware reuse or sub-word parallel processing. With hardware reuse, the overall throughput or peak performance of an accelerator can be improved for lower precision layers or models. For example, the Tesla T4 [23] GPU can be configured to four precisions: 4-bit, 8-bit, FP16/FP32-mixed and FP32. Tesla T4 achieved the highest speed at the lowest precision (4-bit). The throughput increases at the cost of reduced precision. The peak performance is typically expressed in arithmetic operations per sec (OPS), and it primarily depends on the available MAC units. In a variable precision MAC unit, additional control unit is required to configure the MAC unit into multiple sub-MACs or bit length. The overall size of the MAC unit increases with flexibility (in precision). In other words, the MAC density (i.e., MAC units per unit area) decreases with an increase in flexibility [12]. Therefore, there is a tradeoff between MAC's flexibility and density.

Having a large array of MAC units with a variable bit precision can fulfill the DNN processing requirement in terms of computations. But just having a large array of MAC units does not improve the throughput. To provide operands to all MAC units in a large array, higher memory bandwidth (BW) is required. After a certain point of arithmetic intensity, the memory bandwidth of an accelerator will determine the overall throughput. Fig. 2 shows the estimated roofline model for DNN inference on four different embedded platforms. Arithmetic intensity (AMI) also commonly referred to as the operational intensity or compute-to-communication ratio is expressed as the number of arithmetic operations performed per byte of off-chip memory traffic (expressed in operations/byte). The arithmetic performance of the hardware depends on the AMI as well as the data access rate from the external memory. In other words, the arithmetic performance can be expressed as follows:

$$ArithmeticPerformance = \min(PP, AMI \times BW) \quad (1)$$

TABLE 2
Resource consumption of MAC units at different precisions [16].

operation/precision	Energy (*pJ*)		Area (μm^2)	
	MUL	ADD	MUL	ADD
8-bit FIX	0.2	0.03	282	36
32-bit FIX	3	0.1	3497	137
16-bit FL	1.1	0.4	1640	1360
32-bit FL	3.7	0.90	7700	4184

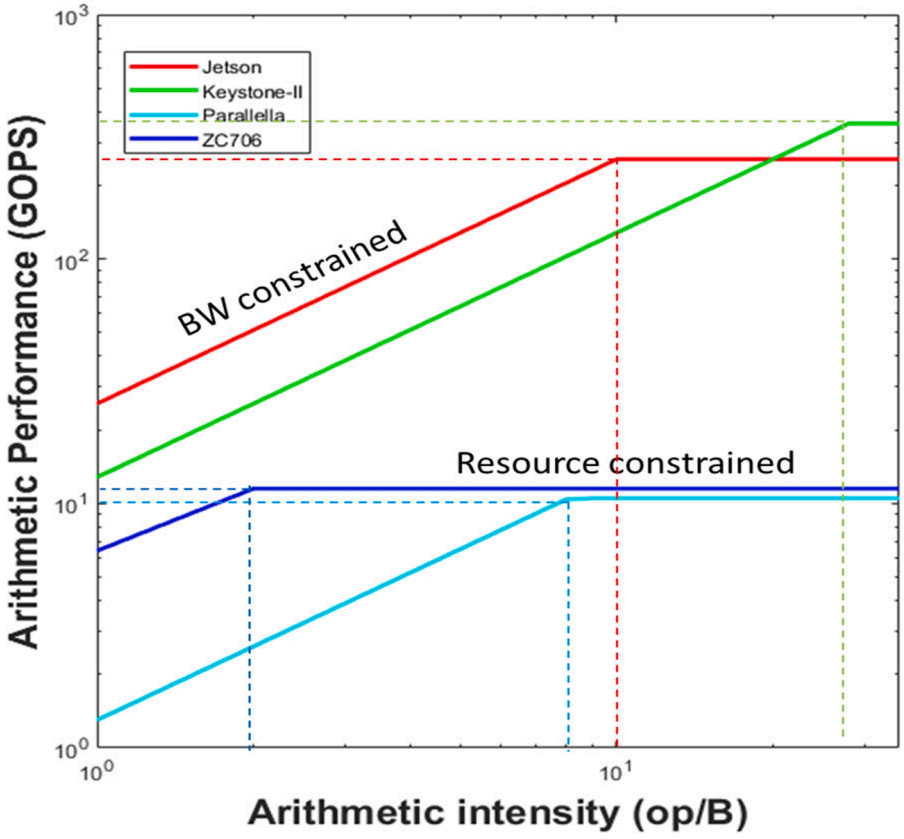


Fig. 2. Comparison of the roofline models for DNN inference [24].

where *BW* is the memory bandwidth and *PP* is the peak performance. It is observed in Fig. 2 that the arithmetic performance increases initially with an increase in the AMI until peak performance (PP) is reached. After achieving the PP, any further increase in the AMI does not increase the performance. It is observed that the arithmetic performance is memory-bound when the AMI is to the left of the break point and is compute-bound when the AMI is to the right [24].

The Arithmetic performance of the hardware typically depends on the PP, AMI, and memory bandwidth. Resources available on the chip defines the PP of the hardware. Arithmetic intensity depends on the dataflow structure implemented and available on-chip memory. Note that an external memory operation is energy and time-consuming. Hence, the hardware should run at a minimum bandwidth to save energy. With minimum bandwidth, the arithmetic performance of hardware can be increased with increased AMI. As seen in Fig. 2, the arithmetic performance increases with an increase in the AMI in the linear region of the curves (as the PP and BW are constant). The AMI can vary through the data flow structure. Therefore, the dataflow structure should be optimized to achieve higher arithmetic performance for a given available bandwidth.

To avoid the data read/write each time (in order to speed up the computation and reduce energy consumption, and to increase the AMI), the read data must be used as much as possible within the chip before writing it back to the memory. Fortunately, the convolution layers in DNNs have this data reuse options. For example, a single filter is reused to calculate all pixels in an output feature map. Therefore, reading coefficients of a filter once is enough. But keeping all filter coefficients at each MAC unit is a resource (i.e., memory) consuming option. To reduce the overall energy cost of data movement, several levels of memory (e.g., global buffer, local buffer, registers) can be implemented in hardware. A rough estimation of the available memory size, latency, and energy consumption per operations at different levels are shown in Table 3 [7].

Table 3
Memory hierarchy in a general accelerator, and its approximate performances [7].

Memory level	Access time (approx. cycles)	Available capacity	Energy consumption (normalized)
Registers	1	>0.5 KBs	1x (Refence)
PEs cache	2–4	~1–10 KBs	2x
Local buffer	10	~100 KBs	4x
Global buffer	40	~10 MBs	6x
Main memory	200	In GBs	200x

The global buffer (with a size of hundreds of kilobytes) connects to DRAM, and local buffer dedicated to a few processing elements (PE). Read/write data from Global buffer to a MAC consumes around six times more energy and 40x latency than read/write from register files. Register files (RF) corresponding to a MAC unit of a PE are connected to a local buffer and consumes the least amount of energy to read/write the data. The advantage of the local buffer is limited by its available size. The energy consumption and access time increase from low-level memory (Registers) to high-level memory (Global buffer).

In a DNN, the output of a convolution or fully connected layer goes through an activation function. The Rectified Linear Unit (ReLU) is a nonlinear activation function widely used in DNNs (Sigmoid is another widely used activation function), which maps the output value of a feature map as follows.

$$y = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases} \tag{2}$$

where x is the input and y is the output of the activation function. It is observed that the negative output values are truncated to zero by the

activation function. This truncation can make the output values sparse. It has been shown that the AlexNet has a sparsity between 19% to 63%, where the sparsity is defined as the percentage of the data (e.g., feature maps, filter coefficients) that are zero. The sparsity in a DNN has been exploited by several researchers to increase the throughput and reduce power consumption.

The DNN model size (i.e., the number of the DNN weights) can be reduced through pruning without affecting the model accuracy. The pruning eliminates insignificant connections or weights (i.e., making the insignificant weights zero) in a DNN. Note that multiplication with a very small value operand results in insignificant value that are not likely to alter the outcome. This observation makes the case for opportunistic energy savings by eliminating insignificant multiplications. The DNN architectures can therefore be designed to skip multiplications with zeros, which is known as zero skipping.

If a hardware can skip zero multiplication, sparsity in data and zero weights cumulatively reduce the computing power requirement. Higher speed can also be achieved by exploring sparsity in data. To exploit the sparsity further, the storage requirement can be reduced by encoding the sparse data. The compression techniques may vary from simple run-length coding to compressed sparse column (CSC) or compressed sparse row (CSR) [25]. Compression techniques however need additional encode and decode modules in the hardware.

Based on the above discussion, it can be inferred that an efficient hardware accelerator must be optimized for low precision, best data flow, and be flexible for varying precision and sparse models. As can be expected, there is a tradeoff between flexibility and optimized architectures. An architecture optimized for sparse models will affect the throughput on dense models. Accelerators optimized for the convolutional layer may not perform well on a fully connected layer due to the data reusability. In a convolutional layer, weights are reused but in fully connected layers input features are reusable for optimal performance. Overall, efficient hardware for DNNs should have scalable precision to support different DNN models, optimized data flow structure to increase the arithmetic intensity, and should utilize the sparsity.

3. Hardware classification

The deployment of DNNs in real-time applications requires low power and high throughput DNN accelerators. Many efficient DNN accelerator architectures have been proposed over the last decade to reflect versatile effort to improve the overall performance of the DNNs. Domain-specific accelerators will always have a scope to improve the overall performance by customizing architectures towards a specific application. Even the accuracy requirement of the same application can make a difference in the DNN complexity. A generalized DNN accelerator architecture should have the flexibility to work on different models at the optimum performance.

The DNN architectures can be broadly divided into three categories based on the area in which the architecture has been primarily optimized. These three areas are Arithmetic logical unit (ALU), Dataflow, and Sparsity. In the ALU category, the basic building block, i.e., the MAC units (or an array of MAC units) are modified such that the accelerator can have large computing resources and flexibility to achieve the optimal performance with variable bit precision. In the Dataflow category, the parameters (e.g., weights, activations, partial sums) are managed such that the overall (intra chip) data movement energy is reduced, and high arithmetic intensity (Ops/Byte) can be achieved. In the Sparsity category, the unstructured sparse data is managed such that the matrix multiplication units (e.g., a 2-D array of MAC units) can avoid the zero multiplications effectively. A comprehensive review of the DNN architectures based on these three criteria is presented in the following.

3.1. ALU based accelerators

Computation hungry DNN algorithms require a huge amount of computing hardware resources. Large arrays of PEs are typically implemented in parallel to improve the computational power of a processor. Graphical Processor Units (GPUs) have thousands of PEs in parallel. Hence, the GPUs are widely used as accelerators for DNNs. The GPUs can provide the throughput requirement but consumes high energy. The energy consumption of a MAC unit can be reduced by decreasing the bit length. Therefore, low precision DNN accelerator architectures have been proposed for DNN inference.

Chen et al. [26] proposed an architecture, known as DianNao architecture, with Neuron flow unit (NFU) as the basic arithmetic building block. An NFU has 16 neurons, with each neuron having sixteen 16-bit fixed-point multipliers in stage 1 and 15 adders in a tree structure at stage 2 to add the multiplication results. Stage 3 has an activation layer. DianNao has three memory blocks input buffer, output buffer and synapse buffer to store inputs, outputs, and weights respectively. Based on the DianNao architecture, a series of accelerators DaDianNao [27], ShiDianNao [28], PuDianNao [29] have been proposed by improving the NFU unit as well as dataflow. The DianNao family can provide 450x speedup and 150x reduction in energy with 64-chips over a GPU [30]. Although, the Diannao family provides a good speed-up, it does not support variable precision. Running a 4 or 8-bit DNN Model will consume energy as high as the 16-bit model.

To save the energy at lower precision, the Dynamic Voltage, Accuracy and Frequency Scaling (DVAFS) MAC based CNN architecture (ENVISION) has been proposed in [31]. In DVAFS, all run-time adaptable parameters influencing power consumption: activity (α), frequency (f) and voltage (V) are scalable. The dynamic power consumption at constant throughput is given by [31]

$$P_{DVAFS} = \frac{\alpha}{k_1} C \frac{f}{N} \left(\frac{\alpha}{k_2}\right)^2 \tag{3}$$

where k_1, k_2 and N are scaling factors of switching activity, voltage, and level of parallelism, respectively. For lower precision, the switching activity can be reduced by masking lower LSBs at the inputs of the MAC units. For example, as shown in Fig. 3, the configuration of 8b-MAC to 4b-MAC leaves a portion of the MACs unused. The unused region can be masked to reduce the switching activity. The reduced precision MAC (4b or 2b) will have shorter critical path than the full precision MAC (8b). The shorter critical path can help to increase the operation frequency or to reduce the input voltage for energy efficiency. With sub-word parallel processing, one MAC unit at full precision (8b) can be configured to more than one MAC units of lower precision. As seen in Fig. 3, one 8b-MAC can be configured to two 4b-MACs or four 2b-MACs. At constant throughput, the sub-word parallel processing helps to reduce the operating frequency (1 MAC/clock at 8-b precision, 2 MACs/clock at 4-b precision and 4 MACs/clock at 2-b precision). The reduced switching activity, frequency and voltages have been explored to increase the overall energy efficiency in the DVAFS. The energy efficiency is further improved by modulating the body bias (BB) in an FDSOI technology [31]. The body bias permits tuning of the dynamic vs. leakage power balance while considering the computational precision. On average 0.26–10 TOPS/W peak efficiency is reported (implemented in 28 nm FDSOI technology). Note that processing at the full precision (i.e., 8 bit) with DVAFS comes at a slightly higher energy and area penalty (compared to 8-bit standard precision) due to additional control circuitry for configuration and larger register.

Shin et al. [32] proposed a Deep Neural Processing Unit (DNPU) architecture for general DNN models using reconfigurable MAC with sub-word parallel processing (SWP) approach on one operand. In SWP, parts of bits processed separately using the lower precision MACs and results are combined to get full results, as shown in Fig. 4. In Fig. 4(a), both activation (A) and weight (W) have 8-bit precisions and W is represented as two 4-bit subwords. The SWP architecture generates 16-bit multiplication output by combining the two sub results. In Fig. 4(b), A has 8-bit precision, but W represents two independent 4b words, and the

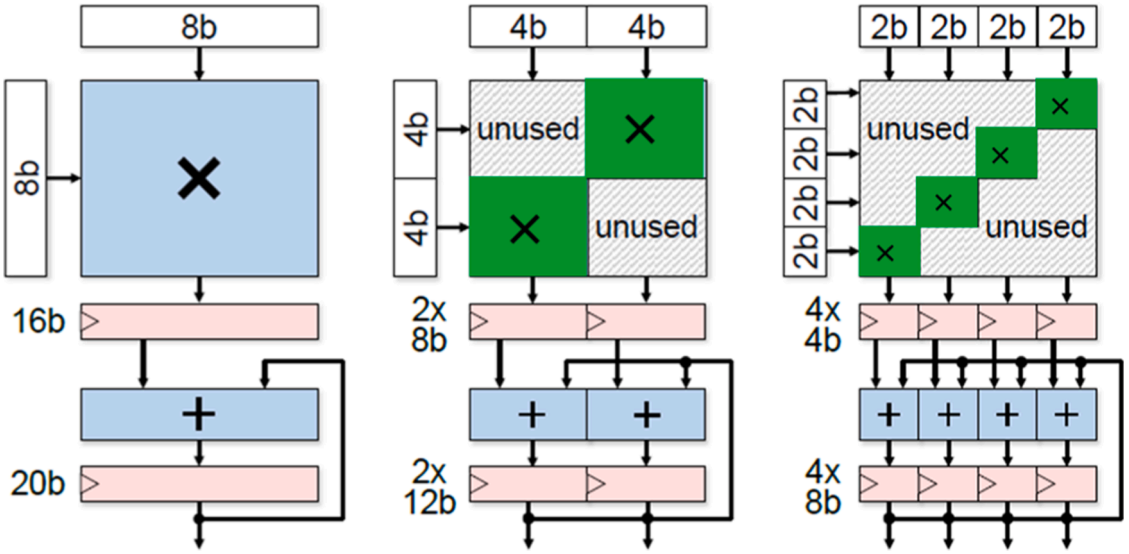


Fig. 3. Implementation of variable symmetric precision (8bx8b, 4bx4b, 2bx2b) MAC unit using the DVAFS architecture. In Green regions DVAFS techniques are used. Average throughput is one multiplication and accumulation (one 8b or two 4b or four 2b) per cycle [12].

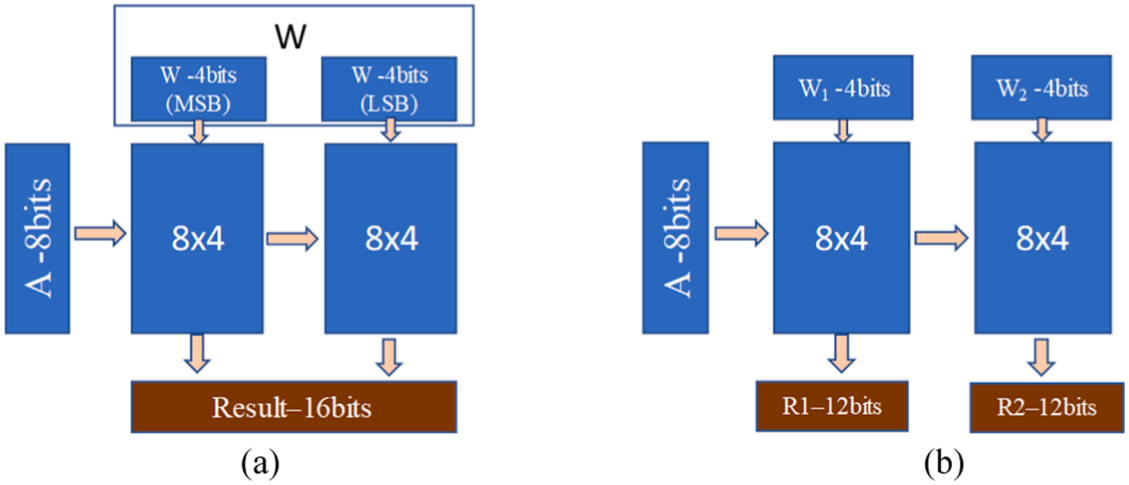


Fig. 4. Sub-word parallel (SWP) architecture, (a) use of two 8bx4b MAC units to perform one 8bx8b operation, (b) Two 8bx4b MAC operations implemented in parallel.

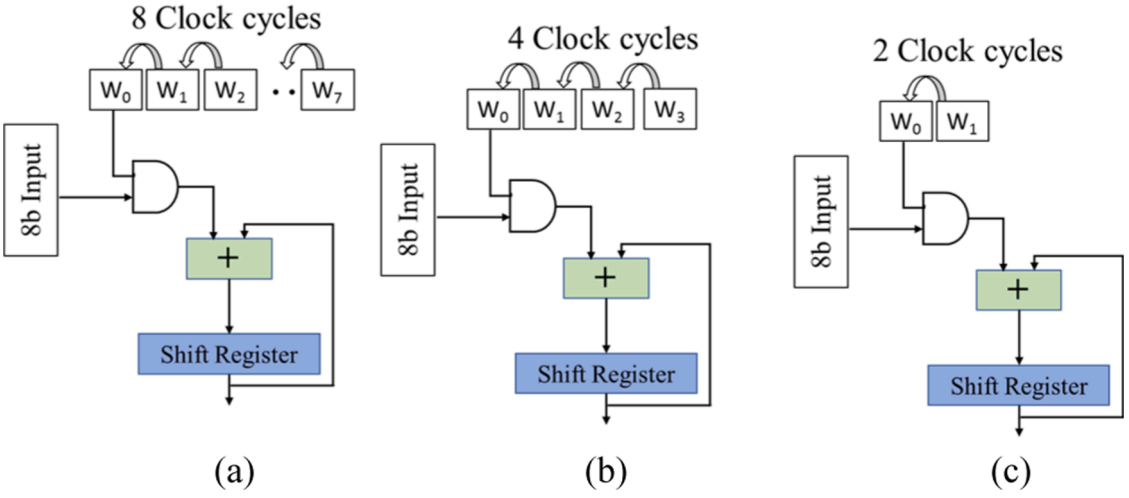


Fig. 5. Bit-serial MAC configured as (a) 8bx8b MAC unit, (b) 8bx4b MAC unit, and (c) 8bx2b MAC unit (Weight only scaling).

SWP generates two 12-bit multiplication outputs. In other words, the DNPU architecture allows the fixed precision on one operand (A) and the variable precision on the other (W). The DNPU reported 8.1 TOPS/W energy efficiency (with 4-bit precision) on 65 nm CMOS technology. Although, the DNPU architecture exposed the SWP for only one operand, the SWP can be exposed in both operands using a DVAFS like architecture shown in Fig. 3.

Lee et al. [33] proposed the Unified Neural Processing Unit (UNPU) architecture using a bit-serial MAC unit. Schematic of a Weight only bit-serial MAC unit is shown in Fig. 5. The Bit serial MAC requires just an adder and a shift register and does not require multiplication. In each clock cycle, one bit of weight (LSB bit first) is supplied, and activation is added to shifted value of the previous cycle partial product. The number of cycles required to finish a MAC operation depends on the weight precision. For an 8-bit precision weight value, eight clock cycles are required to perform the MAC operation as shown in Fig. 5(a). Four and two clock cycles for 4-bit and 2-bit weights respectively as shown in Fig. 5(b) and (c). The architecture supports any weight bit precision from 1b to 16b and reported 1.43× higher power efficiency for a convolutional layer at 4b weight compared to the DNPU.

Alternatively, to reduce the power and area consumption of multipliers, approximate multiplier or logarithmic multiplier have been proposed. Note that the neural networks and their associated applications are known for exhibiting intrinsic resilience to errors, which makes them appropriate candidates for approximate computations. A review on effect of approximate multipliers on the DNN performance can be found in [71]. Ansari et al. [70] proposed an improved logarithmic multiplier (ILM) that rounds both inputs to their nearest powers of two by using a nearest-one detector (NOD) circuit. The MNIST and CIFAR-10 dataset classification using ILM showed up to 21.85% reduction in energy consumption and 1.4% improvement in classification accuracy.

Note that the MAC optimization presented above is primarily based on binary number system. A few accelerators have been proposed based on non-conventional number systems, e.g., the residual number system (RNS) and Posit numbers. Posit numbers have better dynamic range and are suitable to represent weights in DNN with lower bit precision. Carmichael et al. [73] proposed Deep positron, and DNN architecture based on posit number system and evaluated its robustness at low precision (<8-bits). The residual number system is represented by k integers $\{m_1, m_2, \ldots m_k\}$, called moduli which should be relatively prime by each other. In the RNS, an integer value, X is represented with residues $\{r_1, r_2, \ldots r_k\}$ where $r_i = |X|_{mi}$. Any arithmetic operation in the RNS is equal to the same operation on residues. For example, for two numbers (in RNS) $x_1 = \{a_1, a_2, a_3\}$ and $x_2 = \{b_1, b_2, b_3\}$, $x_1 + x_2$ can be calculated as $\{a_1 + b_1, a_2 + b_2, a_3 + b_3\}$. In RNS, any arithmetic operation can be break down to same operation on residues which are represented with lower precision than actual binary number. It reduces the bit precision requirement at the cost of increased number of computations. In digital domain, the RNS can improve the speed and reduce the energy in high precision computations. Olsen et al. [75] implemented RNS based matrix multiplication to accelerate neural network processing on FPGA and achieved 7–9x speed compared to the 32-bit fixed-point implementation. The reduction in the precision requirement is very helpful in analog domain implementation where higher precision MACs have some limitations with their non-linear and hysteretic behavior. Samimi et al. [72] proposed RESnet accelerator in analog domain with RNS. The RNS-based RESnet consumes 145.5× less energy and obtains 35.4× speedup as compared to NVIDIA GPU GTX 1080. Accelerators with emerging technologies are discussed further in section III D.

3.2. Dataflow accelerator

The focus of the data flow accelerators is on data management to reduce the off-chip memory read-write. Spatial and Temporal architectures are well studied for data reusability. Efficiency of dataflow accelerators can be measured with arithmetic intensity, number of operations performed per byte of off-chip memory read. The dataflow can be optimized by reusing the parameters in different layers wherever possible. For example, in a convolutional layer, both weights and activations can be reused. Each neuron has unique weights in a fully connected layer, and hence weights cannot be reused but input data (i.e., feature maps) can be reused. The reusable parameters are stored in local registers so that data movement between a MAC and higher-level memory can be reduced.

For a MAC unit, three memory reads (i.e., weight, activation, and partial sum), and one memory write (i.e., updated partial sum) are required. One of the parameters (e.g., weight) can be stored locally in a register file and can be reused for the next few calculations. The parameters stored differ from architecture to architecture based on the data flow structure implemented. There are 4 major types of data flow structures to manage the input/output data of a MAC in a DNN: No local reuse (NLR), Weight stationary (WS), Output stationary (OS), and Row stationary (RS). In NLR, all memory operations are performed directly from the main memory (e.g., DRAM). In WS, the weights are stored in the RF (i.e., local memory). In OS, the partial sum outputs are stored in the RF to reduce read and write operations. In RS, a row of filter weights is stored in the RF.

Google has developed the Tensor Processing unit (TPU) accelerator for efficient implementation of machine learning techniques. The TPU architecture [34] has a systolic array of 256 × 256 MAC units as a matrix multiplication unit. The implemented systolic array structure is basically a 2D single instruction multiple data (SIMD) architecture with specialized weight-stationary dataflow [35]. The block diagram of TPU is shown in Fig. 6. The weights can be fetched directly from DRAM and stored in the weight FIFO (First-In-First-Out) register. Input activations from the external memory or previous layer results are stored in the unified local buffer. Systolic data setup block is used to rearrange the input data such that convolution can be performed on a matrix multiply unit. The first version of TPU, known as TPU1, focused on the inference tasks, and has been deployed in Google's datacenter since 2015. TPU2, also known as Cloud TPU, has been used for both training and inference in the datacenter. TPU2 also adopted a systolic array and introduced vector-processing units.

The SCNN (sparse CNN) accelerator proposed by Parashar et al. [36] uses a dot product dataflow termed as PlanarTiled-InputStationary-CartesianProduct (PT-IS-CP). The Cartesian Product (CP) term indicates the implementation of MAC units in a PE such that a full Cartesian Product of weights and activations ($W \times A$) are calculated. The CP implementation maximizes the spatial reuse. The Input stationary (IS) term indicates that activations are reused at the PEs by storing it in a local memory. The Planer Tile defines distribution of data across PEs. In SCNN, activations and weights are partitioned into smaller tiles and distributed across the PEs.

In the output stationary (OS) dataflow, the partial sums are stored in the local register files. The OS works well with the fully connected layers as each neuron output depends on all input activtions. Instead of multiplying all inputs with the corresponding weights (which may be a few hundred), in each clock cycle, a few inputs (e.g., K) are multiplied with weights and the partial sum is stored locally. The entire operations will require N/K clock cycles where N is the number of inputs. ShiDianNao [28], an example of OS dataflow, was implemented for K=16.

Chen et al. [37] proposed a row stationary (RS) dataflow-based accelerator called *Eyeriss* that minimizes the data movement energy on a spatial architecture. Note that in the RS dataflow, a row of operands (i.e., input, weights and partial sums corresponding to a PE) are stored in the RF. A schematic of row stationary dataflow in *Eyeriss* is shown in Fig. 7. Inputs are reused across the PEs connected in diagonally. The partial sums are accumulated in vertical direction. Each PE has local registers to store at least one row of weights and activations, one MAC unit and controller. The controller is responsible for the temporal reuse of MAC units to perform 1-D convolution. Implementation of 1-D convolution using the RS dataflow in a PE is shown in Fig. 8. Large

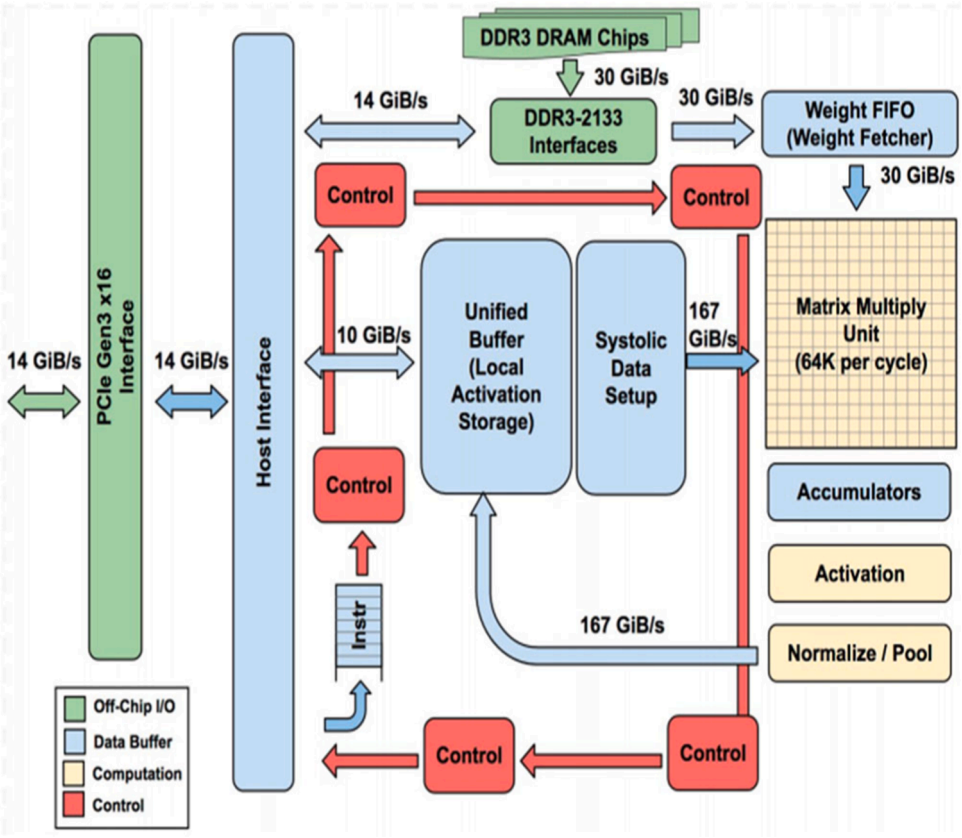


Fig. 6. Block diagram of a tensor processing unit (TPU) [34].

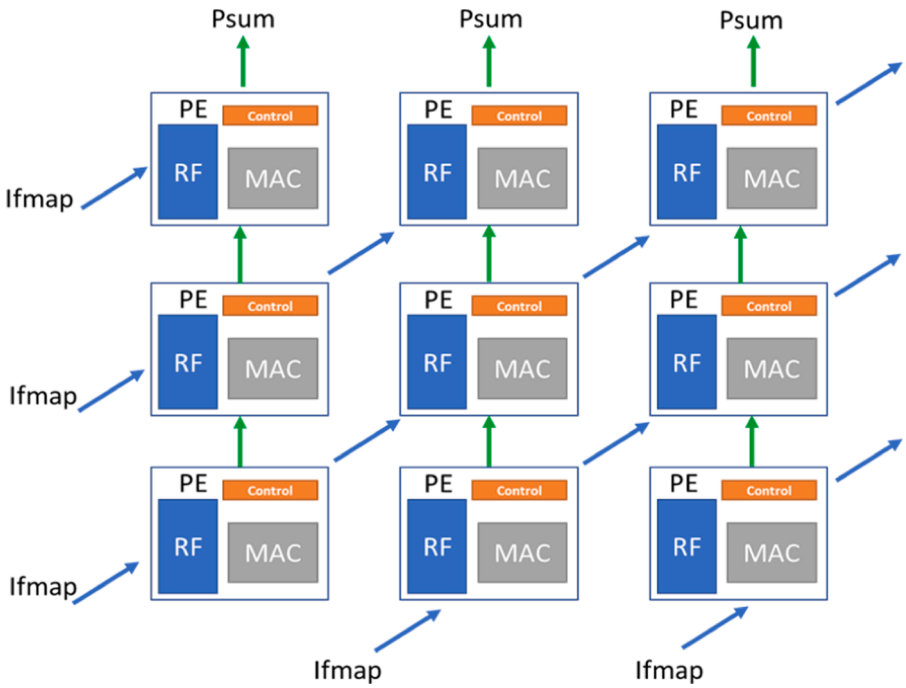


Fig. 7. Schematic of row stationary dataflow.

portion of RF is allocated to the weights. A row of input vector is reused to calculate the partial sums of multiple output feature maps. Fig. 8 shows how the same PE can be used to calculate multiple output features by reusing the input data. It has been shown that the RS dataflow is more energy efficient than the existing dataflows [37] in both convolutional (1.4–2.5×) and fully connected layers (at least 1.3× for batch size>16). To support a wide variety of DNN models and further increase in the resource utilization, an improved version of Eyeriss is proposed in [38] called Eyeriss *V2*. The Eyeriss *V2* introduces a highly flexible on-chip network, called hierarchical mesh, which can adapt to different

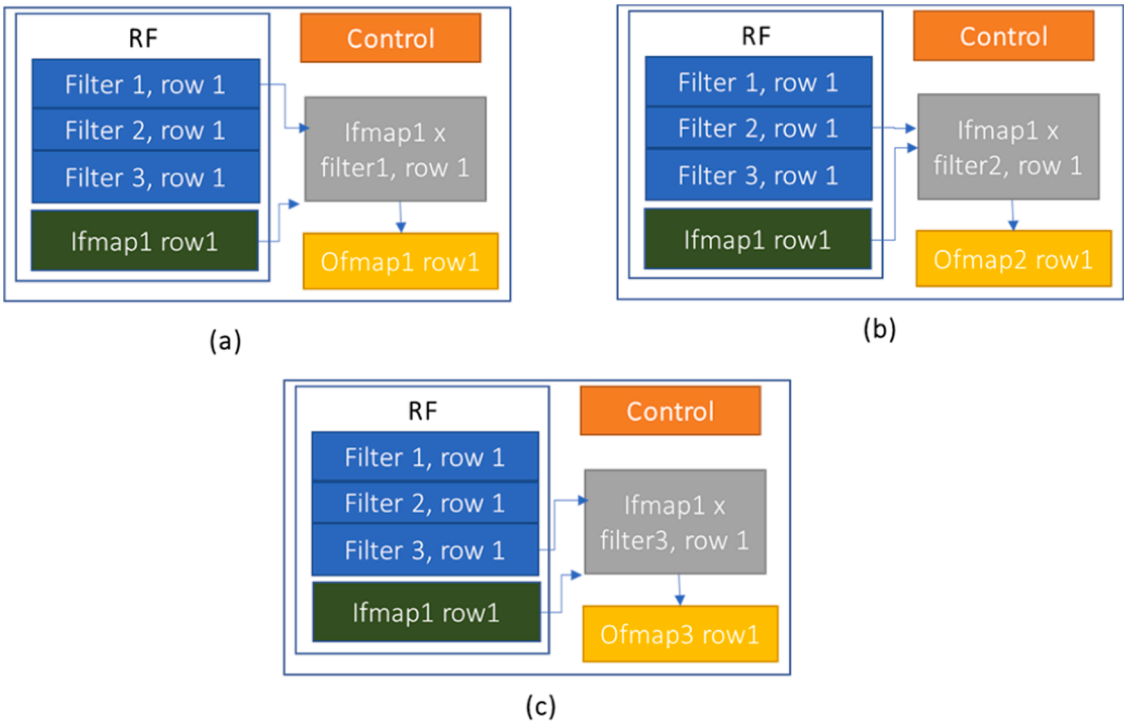


Fig. 8. Implementation of row stationary dataflow on Eyeriss architecture. (a) 1-D convolution between first row of filter 1(Filter1, row1) and input feature map 1 (Ifmap1). (b) 1-D convolution between first row of filter 2 (Filter2, row1) and input feature map 1 (Ifmap1). (c) 1-D convolution between first row of filter 3(Filter3, row1) and input feature map 1 (Ifmap1).

amounts of data reuse and bandwidth requirements of different data types. Eyeriss *V2* reports 12.6× faster and 2.5× more energy efficiency than Eyeriss running the MobileNet. Venkatesan et al. [66] proposed multi-level weight-output stationary dataflows: Weight Stationary–Local Output Stationary (WS-LOS) and Output Stationary–Local Weight Stationary (OS-LWS). The advantages of these dataflows over the IS, WS and OS dataflows are also discussed. An automated framework, MAGNet, to generate an accelerator for a neural network has been proposed in [66]. Using this framework, an accelerator can achieve up to 40 fJ/op and 2.8TOPS/mm^2 in a 16 nm FinFET technology.

In most of the DNN accelerators, the layers are iteratively processed. However, by processing each layer to completion, the accelerator must use off-chip memory to store intermediate data between layers as the intermediate data is too large to fit on chip. Alwani et al. [39] explored the dataflow across layers and proposed the Fused-layer CNN accelerator. In Fused-layer accelerator, neurons in multiple layers which depend on generated intermediate data are processed once. It increases the data reuse across the layers. The data dependency between two layers can be seen in Fig. 9. Layer 1 output features (Tile 1′ and 2′) can be further processed to generate the layer 2 outputs, which avoids the storage requirement and memory read-write operations for layer 1 output features (Tile 1′ and 2′). For example, Tile 1 input data processed through layer 1 generates Tile 1′ data. Instead of storing the Tile 1′ data in global or external memory, the layer 2 computations can be performed to generate the green pixels (layer 2 output). To generate the red pixels at layer 2, only small amount of data needs to be read from the higher-level memory. The overlapped data can be reused by storing in the local memories. Fused-layer method avoids the storage requirement of intermediate results (layer 1 outputs) externally. If multiple processors run in parallel, the intermediate results can be reused across the

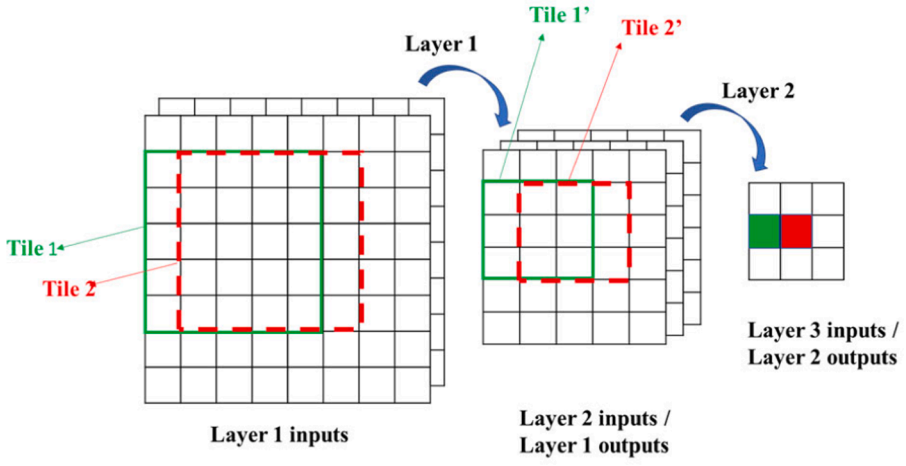


Fig. 9. Example of fusing two convolutional layers.

processors without read/write to external memory. Based on this principle, Shao et al. [64] proposed the SIMBA accelerator based on a Multi-chip-module (MCM). In the MCM, small chiplets (i.e., small chips) are integrated at the package level. Each chiplet has 4 × 4 PE array with weight stationary dataflow. The SIMBA integrates 36-chiplets, each with 4 TOPS peak performance, to achieve up to 128 TOPS peak and 6.1 TOPS/W [64].

3.3. Sparsity based accelerators

The computational and memory requirement of a DNN model can be reduced through pruning without significant loss of accuracy. In pruning, at the time of training, any insignificant weights are set to zero. The pruned weights (or zeros) can be in regular structure or random. In regular structure pruning, also call structured sparsity, a neuron will be removed (i.e., all the weights connected to the neuron are set to zero). The pruning in structure sparsity can be at the level of neuron, filter, or channel of filter. In unstructured pruning, all the insignificant weights, which are random across the weight tensor are set to zero. The unstructured pruning is simple, it can be done just by adding a regularization to the training algorithm. But, due to unpredictable zero patterns in the unstructured sparse model, it requires complicated hardware design to compress the non-zero weights and skip zero multiplications. Over the time, researchers found complex algorithms for structured pruning where a complete neuron, filter or channel of filters are removed. The architectures for structured sparsity are comparatively simpler.

Albericio et al. [40] proposed the Cnvlutin architecture to exploit the sparsity in feature maps. Computation with zeros in the inputs are eliminated by indexing the input data. Non-zero input data along with index value are supplied to compute unit. Based on index value, the compute unit selects the corresponding weight from filters and performs multiplication. The controller fills the index buffer on the fly such that it does not consume extra clock cycle. To further increase the acceleration, Cnvlutin prunes near-zero outputs during inference to increase the sparsity of the next layer's input data. Experiments with several CNNs, including AlexNet, GoogleNet, and VGG-19, showed 1.2–1.6× throughput increases over DaDianNao [27] without any loss in accuracy on ImageNet data. The Cnvlutin reported an area overhead of 4.5% over DaDianNao. Judd et al. [41] proposed Cnvlutin-2 architecture, an extension of Cnvlutin by exploring both input and weight sparsity. Cnvlutin-2 is further optimized to reduce the memory bandwidth.

Eyeriss [42] also explored the sparsity in inputs to reduce the energy consumption. MAC units corresponding to the zero inputs are inactivated by gating method (disable). The gating method saves the energy but does not increase the throughput. Eyeriss *V2* [38] can process the sparse data directly in the compressed format for both the weights and activations, and therefore is able to improve both processing speed and energy efficiency with sparse models.

Han et al. [25] deep compressed the model by pruning the redundant connections and by enabling multiple connections sharing the same weight. Deep compression uses threshold-based pruning, quantization and Huffman coding techniques to reduce the overall size of the model to fit on the chip memory. Han et al. [43] proposed an energy efficient inference engine (EIE) to accelerate deep compressed model's inference. To exploit the sparsity and reduce the memory bandwidth, the data is compressed using a variation of the compressed sparse column (CSC) format. For each column M_j of matrix M, a vector $\boldsymbol{v}$ that contains the non-zero weights, and another equal length vector $\boldsymbol{z}$ that encodes the number of zeros before the corresponding entry in $\boldsymbol{v}$, are stored. Each entry of $\boldsymbol{v}$ and $\boldsymbol{z}$ is represented by a four-bit value. If more than 15 zeros appear before a non-zero entry, then a zero is added in vector v. For example, the following column [0, 0, 1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3] is encoded as $v = [1, 2, 0, 3]$, $z = [2, 0, 15, 2]$. Weight matrix distributed across the PEs and stored in a compressed format. The EIE performs the *sparse matrix* × *sparse vector* operation by scanning vector a (activations) to find its next non-zero value a_j and broadcasting a_j along with its index j to all PEs. Each PE then multiplies a_j by the non-zero elements in column W_j. Compared with DaDianNao, the EIE has 2.9x, 19x and 3x better throughput, energy efficiency and less area, respectively [43].

Parashar et al. [36] proposed the SCNN accelerator for compressed-sparse convolutional neural network. Weights and activations are compressed with variants of the CSR methods used in [43]. For example, as shown in Fig. 10, filter of 3 × 3 are compressed into data vector (row wise), containing non-zero filter values and index vector. In the index vector, the first value represents the number of non-zero elements in the data vector followed by the number of zeros before each value in the corresponding data vector. Multiplication between compressed weights and activations are performed like dense matrix multiplication. The output activation's index is calculated based on the input's and weight's index at accumulation buffers using cross bar connections. The SCNN accelerates a CNN by 2.7x, while still being 2.3x more energy efficient (compared to the uncompressed network).

With zero skipping implementation for sparse models, small to large percentage of MAC units may be end up in the inactive state to synchronize with other PEs. Zhang et al. [45] used parallel associative search to maximize the even distribution of data across the MAC units and implemented in SNAP accelerator. The SNAP maintains an average 75% hardware utilization. Similarly, Lee et al. [57] proposed the LNPU architecture for sparse DNN model learning. The LNPU has input load buffer module which distributes the workload evenly to the PEs accounting for irregular sparsity. The overall MAC utilization increased in the LNPU. Lin et al. [58] proposed a Dual-core deep learning accelerator based on compression, zero skipping and Fused-layer techniques.

Zhang et al. [44] proposed the Cambricon-X architecture to exploit the sparsity in filter weights by adding a buffer control module. The buffer control module includes an indexing module that selects and transfers the useful input neurons (neurons corresponding to non-zero weights in the filter) to PE. A PE stores the compressed filter weights locally and performs the computation asynchronously. Cambricon-X reported 7.23x speedup and 6.43x energy saving against the DianNao accelerator.

The architectures presented above are unstructured sparsity based but the unstructured sparsity in weights need complex decode module to decompress the weights and calculate the respective activation indices. Based on this observation, Zhou et al. [65] showed that pruning block of weights in a DNN model reduces the irregularity in weight sparsity. Zhou et al. [65] proposed the *Cambricon-S* accelerator that uses structured sparsity in weights and encoded to achieve higher compression ratio. *Cambricon-S* reported 1.71× speed and 1.37× energy efficiency compared to the *Cambricon-X*.

3.4. Hybrid implementation techniques

With increasing complexity in the neural network architectures, the required computing power far exceeds what is achievable with today's technology [67]. Hence alternative technologies like analog computation, photonic and quantum computing are being explored. The new technologies are mostly applied at the ALU level in DNN accelerators to improve the speed and energy efficiency. Therefore, this section (i.e., Section D) can be seen as an extension to the ALU based accelerator classification (section III A).

In ML hardware implementation, the processor-memory bandwidth is often the main bottleneck that limits the achievable energy efficiency. Due to the interconnect loss and signal integrity issues, the data transfer is not as efficient as the data processing. Note that the technologies are optimized for either data processing (Processor technology) or storage (memory Technology). Therefore, DRAM ICs are used for storage and processor ICs are used for processing. Bringing them closer through advanced packaging can reduce energy penalty due to the data movement. But Processing near memory or Process in memory (PIM) can

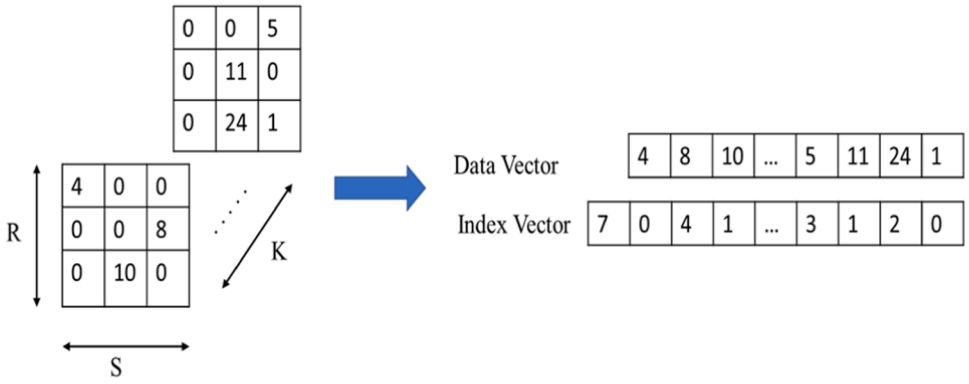


Fig. 10. Weight compression in the SCNN accelerator.

reduce the data movement. The hybrid memory cube (HMC) is a technology that lets vertical integration of DRAM memories on logic circuits and enables near data processing. Neurocube [62] and Tetris [63] are two DNN accelerators based on HMC. Given that the DRAM ICs are optimized for the data storage they are few generations behind the logic CMOS devices in terms of computational efficiency. Therefore, analog computation can be an attractive alternative to conventional digital computation. For instance, the multiplication can be directly integrated into the bit-cells of an SRAM array [7].

In recent years, memristors (or programmable resistive elements), show promising performance improvements. In memristors, weight values are stored as resistor's conductance and multiplication is performed based on Ohm's law

$$i = G \times V \tag{4}$$

where V is the input voltage, G is the resistor's conductance, and i is the output current equivalent to the multiplication result. Fig. 11 shows a schematic of the memristor crossbar in which currents in a column are added together. Using the Kirchhoff's current law, the resulting current (I) can be expressed as follows.

$$I_j = \sum_k i_{k,j} \tag{5}$$

Substituting the i value from Eq. (4), we obtain

$$I_j = \sum_k G_{j,k} \times V_k. \tag{6}$$

The output current I_j in Eq. (6) is equivalent to a neuron output in the neural network. Therefore, the memristor crossbar can be used to implement neural networks. Fig. 11 shows memristor implementation of a vector matrix multiplication, $(V \times W)$. The digital input X is converted into an analog input V using a DAC converter. The weight values W are programmed as resistor's conductance. The output currents are

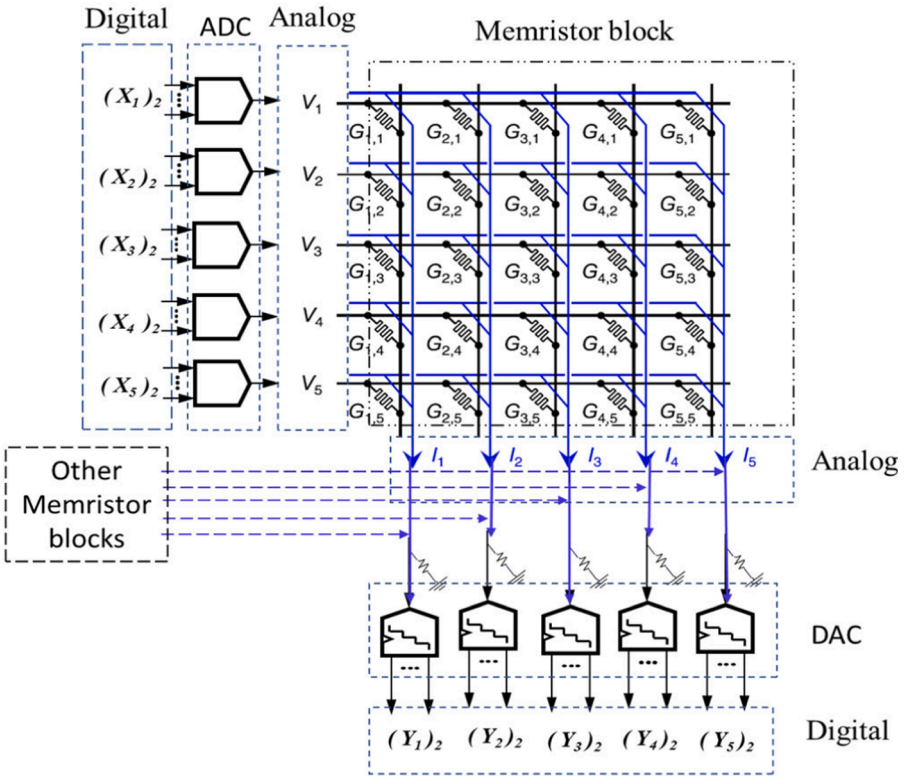


Fig. 11. Resistive memory crossbar implementing vector-matrix multiplication $Y = X * G$. V denotes input voltage vector (analog equivalent of X); G denotes conductance of memory equivalent to weights, and I denote the resultant output currents (analog equivalent of Y). DAC: Digital to Analog conversion block, ADC: Analog to Digital conversion block.

converted back to digital domain using ADC converters. The resistive crossbar implementation can reduce the data movement energy.

The resistive memory can be implemented using different technologies such as Resistive RAM (ReRAM), Phase Charged Memory (PCM), floating-gate charged-trap memory, SpinTransfer Torque Magnetic Random-Access Memory (STTMRAM), and Ferroelectric Field-Effect Transistor (FeFET) [46]. The ReRAM is a popular technology used in resistive crossbar array implementation for neural networks. The two major limitations of this technology are small tunable conductance range and parasitic voltage drop across the array. But more importantly, their non-linear and hysteretic behavior limits their usage for applications where higher precision would be needed. Ultimately, to interface with the digital part of the CNN, we need a DAC to convert the digital input to analog voltage and an ADC to convert the summation output voltage/current to digital. Accuracy of such MAC units are limited by the ADC and DAC resolutions as well as other circuit noises. The conductance range and noise levels define the weight precision, and eight-bit weight precision remains at the upper limit using a single non-volatile memory device [46]. The low precision parameters can still produce similar accuracy in inference, but generally not sufficient for training. The conductance is always positive, and hence only positive weights can be implemented. For negative valued weights, w, two weights w_1 and w_2 whose difference equal to w ($w = w_1 - w_2$) are implemented and resulting output currents are subtracted. A few proposed analog NN architectures are PRIME [47], ISAAC [48], Memristive Boltzmann machine [49], Newton [50], PUMA [68] and mCNN [51].

In some mixed-signal accelerators, the computational units are partially implemented in the analog domain. Cao et al. [52] proposed a hybrid-digital-mixed-signal computing platform using Time-Division Mixed Signal (TD-MS) multiplier. It uses 5b TD-MS multiplier and extends to higher precision (6 to 8- bits) using shift and add. Bankman et al. [53] proposed a mixed signal binary CNN processor which performs multiplication in digital domain and summation using switched capacitor neuron. The weights and input data are represented in binary form hence multiplication in digital domain is efficient. Detailed reviews on analog neural network accelerators can be found in [69, 74].

Similar to the memristors, the digital data in these analog accelerators has to be converted into analog using DAC before processing in analog domain. After processing, the result has to be converted back to digital domain using an ADC. The DAC and ADC converters consume more energy with increase in precision. For higher precision data, energy consumed by converters can nullify the advantage gained with analog computations and the overall performance may degrade compared to the digital domain. The MAC unit does not always need the ADC or DAC elements, but in most cases the non-idealities of the analog MAC require digital calibration and correction that mandates ADC and DACs.

Similar to analog accelerators, photonic accelerators are also being explored to enable faster computation with improved energy efficiency [67]. Detail discussion on such solutions is beyond the scope of this manuscript. However, these solutions also face the same resolution challenge as other analog solutions that limits their usage to certain applications.

4. Evaluation

In this section, we present the performance evaluation of a few selected architectures. Most existing research works use measures such as chip area, throughput, latency and power efficiency for performance comparison. An accelerator proposed for a specific DNN model (e.g., sparsity, kernel size) may not translate its benefits to other DNN models. For example, the performance of sparsity-based accelerators significantly degrades on the denser models due to the presence of additional encode and decode modules. Similarly, the weight stationary data flow typically performs better on the convolutional layers compared to the fully connected layers because of weight reusability in the convolutional layers. Similar performance trend is observed in the variable precision accelerators (ALU-based) where both the latency and the power consumption increase compared to the fixed precision accelerators on a DNN model running at full precision. Therefore, it is important to understand the advantage and drawbacks of each method of accelerator implementation (e.g., ALU-based, RS, WS, OS and sparsity-based accelerators).

Parashar et al. [54] proposed a software framework, known as *Timeloop*, to estimate the energy efficiency of an accelerator architecture on different workloads without physical implementation. It is claimed that the *Timeloop* framework can give over 95% accuracy compared to the actual physical implementation of hardware. Therefore, *Timeloop* framework is used to measure the performance of few architectures. Before considering *Timeloop*, the framework performance on Eyeriss architecture with AlexNet layer 1 workload is verified with manual calculations, the difference is within 5%. Manual calculation uses similar method proposed by Yang [55]. The workload (AlexNet layer 1) is mapped manually on Eyeriss architecture. The parameters (inputs, weights, and partial sums) are stored across the memory hierarchy (DRAM, global buffer, and RF files) such that minimum data read-write operations (or maximum data reuse) are performed. The final output is written back to DRAM. The number of memories read or write operations of each parameter at all levels of memory hierarchy are counted. In the calculation, we consider that the 16-bit MAC consumes 2.20pJ per operation (obtained from the *Timeloop* software). Note that the energy consumed for read/write operation at different level of memory is calculated based on 45 nm CMOS technology [54]. Energy required to read data from RF is assumed to be equal to one MAC operation. The manual calculations require 840 μJ to process AlexNet layer 1 on Eyeriss, and the *Timeloop* reports 866 μJ. The advantage of using the *Timeloop* framework is the optimal mapping of workload on an architecture. Therefore, we will be using the *Timeloop* framework to evaluate the performance.

In a DNN model, the size of parameters varies from layer to layer. Let I, W and O denote the size of the inputs, weights and outputs of a convolution layer. In general, $O > I, W$ in the initial few layers. This is because a large number of feature maps (typically known as depth of the layer) are generated at the initial layers. In the later layers, the size of output features O is reduced. Hence, in the last layers, $W >> O$ in general.

The size of parameters can affect the performance of an architecture. Therefore, five different convolutional workloads, which can generalize to a broad range of workloads (with different filter size, convolution stride, etc.) are considered for evaluation in this section. The configuration of these five workloads is shown in Table 4. The workload, calculated as number of computations in a layer, increases from CONV1 to CONV5.

Using the *Timeloop* framework, the energy performance of three different architectures on the five workloads are calculated. The three

Table 4
Example of five workload configurations in terms of Input (I), Output(O) and Weight(W) sizes. TOTAL-PARAM: Total number of Parameters, I+W+O, (in millions). TOTAL-COMPUT: Total number of computations (in millions).

Parameters	CONV1	CONV2	CONV3	conv4	Conv5
Input (I)	225 × 225 × 3	227 × 227 × 3	64 × 64 × 128	17 × 17 × 256	33 × 33 × 96
Weights (W)	5 × 5 × 3 × 96	11 × 11 × 3 × 96	1 × 1 × 128 × 256	3 × 3 × 256 × 384	3 × 3 × 96 × 256
Output (O)	111 × 111 × 96	55 × 55 × 96	64 × 64 × 256	15 × 15 × 384	31 × 31 × 256
Strides	2	4	1	1	1
Total-PARAM	1.34	0.47	1.6	1.04	0.57
TOTAL-comput	88	105	134	199	212

architectures considered are Row Stationary (RS), Weights Stationary (WS) and Output Stationary (OS) architectures with the same number of resources (e.g., MACs and memory) available. The allocated resources are based on the existing hardware accelerator *EYERISS* [42]. The available on chip global buffer is set to 128 KB and 256 (16 × 16) PEs, and the local buffer at each PE is 440 bytes. The local buffer is used to store weights, and partial outputs in the RS, WS and OS architectures. In the RS architectures, the local buffer is partitioned into 3 parts and is used to store inputs, weights and outputs. In WS, the local buffer is used to store only weights whereas OS architectures store only partial outputs. The latency is calculated based on the number of clock cycles required to process. The clock frequency is set to 200 MHz (with 45 nm CMOS technology).

Two performance parameters, latency and energy are calculated for all three architectures on 5-different workloads using the *Timeloop* framework and results are shown in Fig. 12. In the *Timeloop* framework, the mapper (e.g., a compiler) searches for the optimum map of the workload on the architecture. The search algorithm requires the performance criteria to select the best match. We choose latency and energy as the optimization criteria. In Fig. 12(a), the amount of computation increases monotonically from CONV1 to CONV5, but the energy consumption of the architectures does not always increase with computations. This is because data transfer contributes a significant part of the total energy. For example, between CONV4 and CONV5, there is a small reduction in the total energy consumption despite increased computation due to less data transfer. Therefore, energy efficiency of an architecture depends on both computations and parameters' size. From Fig. 12(b) and (c), it can be observed that the RS architecture has the lowest latency (combining all layers) and the WS architecture consumes the least energy.

The dataflow efficiency depends on how much the parameters are reused within the local memory once read from the external memory. In convolution operation, the filter properties (height (R), width (S) and channel (C)) define the data reusability. For example, in a convolution with a 3 × 3 size filter, one input can be reused to calculate nine partial products (with nine weights) corresponding to nine outputs. Therefore, the performance of the RS, WS and OS dataflows are evaluated with filter size as shown in Fig. 13. In this figure, the workload of CONV5 is being varied by changing the filter size from 1 × 1 to 11 × 11.

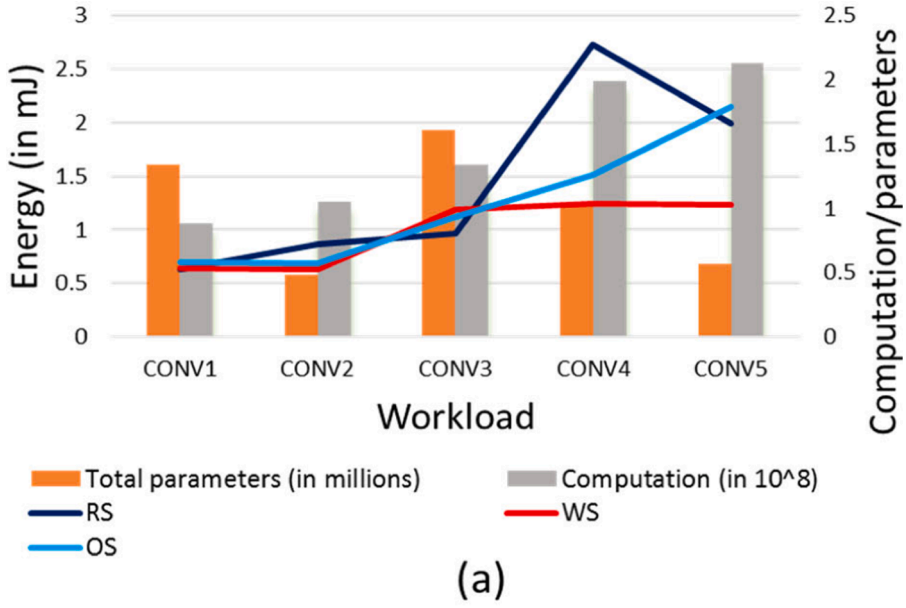
The total computations increase with the filter size and require more energy to process. From Fig. 13, it can be observed that energy consumption increases with filter size for all the architectures. But the energy consumption of the RS and OS architectures increases more compared to that of the WS architecture. To understand the energy variation, we looked at energy consumption of DRAM, global buffer, local buffer access and MAC unit per computation. The MAC unit and local buffer consumes similar energy across the filters. The DRAM and global buffer access energy varies with filter size as shown in Fig. 14. For filter 1 × 1, the WS and RS architectures consume similar amount of energy (as seen in Figs. 13 and 14) because when filter size is 1 × 1, the weight reuse is similar in both the architectures. The input data reusability increases with the filter size but requires more local memory to store the filter weights. In the WS architecture, the local memory is allocated primarily for the filter weights and can store all weights for even large size (11 × 11) filter. Therefore, the DRAM access energy per computation decreases for the WS with increasing filter size as shown in Fig 14. The local memory is primarily allocated for the partial products in the OS architecture and shared with all three parameters in the RS. The OS and RS architectures may not have sufficient space in the local memory for large size filters and hence the increased data reusability with filter size does not significantly affect the DRAM access energy as shown in Fig. 14. The WS architecture requires less DRAM and global memory access energy, which means it maximizes the data reuse within the local memory and requires fewer access to the higher level memory. Therefore, the WS consumes less energy among all architectures with filter size increase as shown in Fig. 13.

The MAC utilization of the RS varies with the filter size as shown in Fig. 14 (see the dotted lines). In RS architecture, the PEs are connected in such a way that the inputs are reused in diagonal PEs and partial sums are accumulated across vertically connected PEs as shown in Fig. 7 (the directions can be configured). When mapping the workload on PE array, a few PEs may end up unallocated. For example, in mapping 3 × 3 filter on four PEs, three rows of filters can be stored in three PEs and accumulate the partial products to get the convolution output. The fourth PE is unused, and it can be used to calculate the next output, but the partial product must be stored in the memory and be read in the next cycle. Additional energy required for the partial product memory read-write can defeat the advantage of using the fourth PE. Therefore, only three PEs are used for calculations and the fourth one left ideally. As only three PEs are effectively utilized, more clock cycles are required to complete the convolution. To fully utilize the PEs in the RS dataflow, the array size should be in multiples of the filter size. In this experiment, the PE array size is 16 × 16 which is not multiples of 3, 5, 7, 11 (i.e., the filter size). Therefore, the MAC utilization of RS dataflow is varying with the filter size (see the dotted green line in Fig. 13). The decrease in MAC utilization increases the latency as shown in Fig. 13. For filter size from 5 to 11, the latency of the RS increases more compared to the WS or OS because of the drop in the MAC utilization. The latency difference between the RS and WS/OS architectures is small (~0.8) at filter size 5 compared to the difference (~1.4) at filter size 3 because of the increase in the MAC utilization for RS.

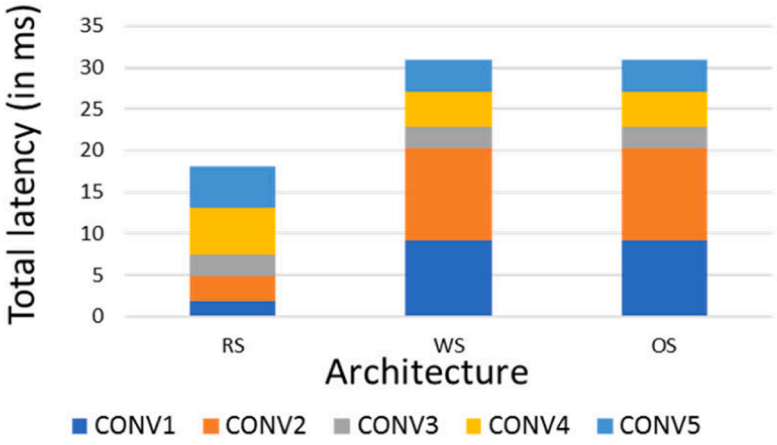
Convolution stride is another parameter which can affect the dataflow. With stride greater than one, the input features may not be reusable in two neighboring output feature (pixels) computations. For example, with stride one, six input pixel values (i_{23}, i_{33}, i_{43}, i_{24}, i_{34}, i_{44}) out of nine can be reused for the next window as shown in Fig. 15(a). In convolution with stride two, only three input pixel values (i_{24}, i_{34}, i_{44}) out of nine can be reused for the next window as shown in Fig. 15(b). On the other hand, in convolution with stride three, there is no common pixel data in the consecutive windows as shown in Fig. 15(c). Therefore, the input data reusability depends on the stride value and there is no reuse of input data in consecutive output feature calculations if the stride value is more than the filter size.

Note that it is important to understand the efficiency of dataflow architectures with stride variation. In this experiment, the stride size is varied from 1 to 4 in the CONV4 workload. The total energy consumption depends on the type of workload and filter size as shown in Figs. 12 (a) and 14. Hence, instead of comparing the total energy, the energy normalized to stride one is compared in respective architectures. The normalized energy here indicates the change in energy due to the change in the stride value. The DRAM access energy increases by about 120% in the WS architecture as stride value changes from 1 to 4 as shown in Fig. 16. The DRAM access energy changes because of the change in the input data reusability and may require reading the full window of input data at each cycle as shown in Fig. 15. In the RS architecture, a row of inputs is stored in the connected PEs and is used in the later computations if not in consecutive computations. Therefore, the DRAM access energy changes more in the WS compared to the marginal change in the RS. The total energy changes with DRAM access energy, more than 20% increase in the WS and marginal change in the RS as shown in Fig. 16. Therefore, for large stride values, the RS has better data reusability compared to the OS and WS architectures.

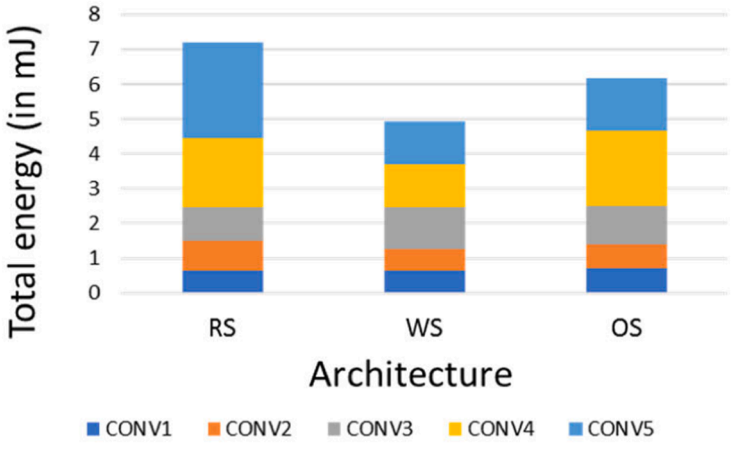
Note that most of the accelerators reviewed in the previous section varies in terms of the available MAC units, memory size, dataflow implemented, and workload used. Therefore, energy and latency parameters are not sufficient to evaluate or compare the existing accelerators. It is difficult and time-consuming to implement all existing accelerators on the *Timeloop* framework, keeping the same amount of resources, to obtain the performance metrics for comparison purpose. In the ALU based accelerators, the MAC implementation enables the performance improvement. Hence, the ALU based accelerators can be evaluated by comparing single MAC units. Camus et al. [12] analyzed



(a)



(b)



(c)

Fig. 12. Performance of three different architectures. (a) energy consumption in different workloads. (b) architecture latency on all workloads. (c) architecture total energy consumption.

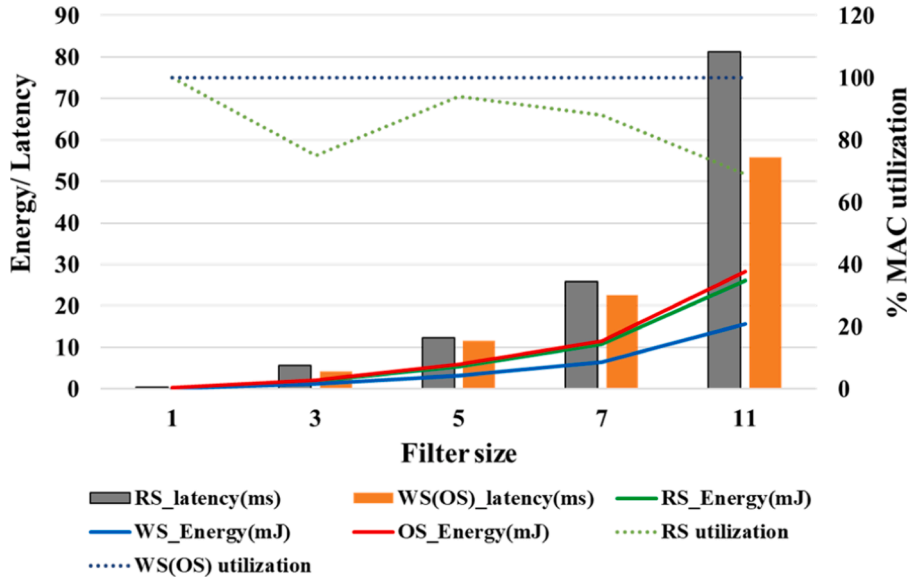


Fig. 13. The RS, WS and OS performance with variation in filter size. Note that the Latency and MAC utilization in the OS and WS are same, and their plots coincide (represented by the dotted blue line).

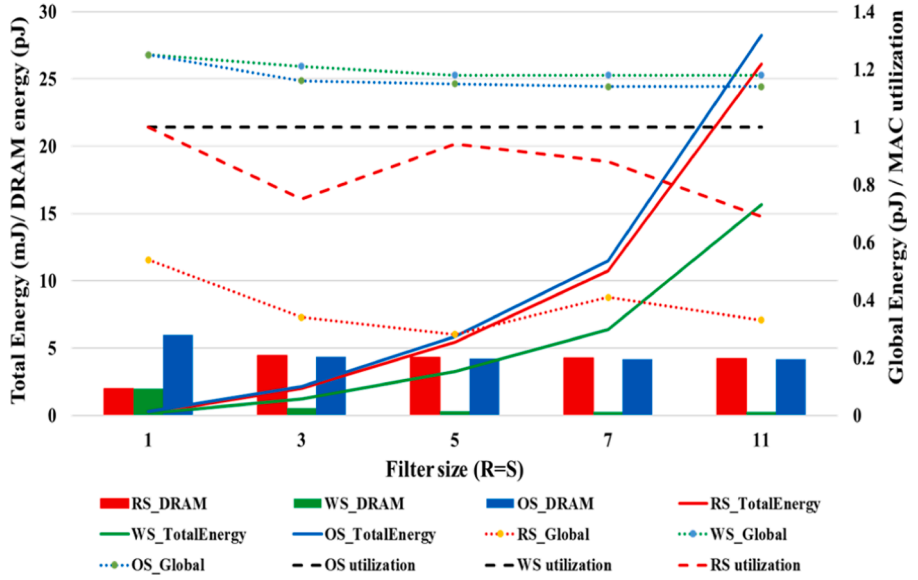


Fig. 14. Energy consumption and MAC utilization in the WS, OS and RS architectures for different filter size. R and S are the number of rows and columns, respectively.

precision scalable MAC units from different accelerators. Similarly, the performance improvement in sparse-based accelerators is defined by the sparse encoder and decoder modules. In order to observe the trend in ALU and sparse based accelerators, power vs speed plot of a few accelerators are shown in Fig. 17. The data for Fig. 17 has been obtained from repository [56]. In Fig. 17, the small size marks indicate the real time performance and the large size marks indicate the peak performance. The ALU based accelerators evaluated for at least two precisions are considered. From Fig. 17, it is observed that the sparsity-based accelerators consume less power (star marks in Fig. 17). The advantage of sparse architectures depends on the amount of sparsity in the input data. For highly sparse data, the additional cost of encoder/decoder can be overcome by the computational advantage (i.e., smaller number of MAC operations after zero skipping). Running a dense model on a sparse accelerator can degrade the performance. Therefore, it is important to evaluate the sparse accelerator with varying sparsity (e.g., from 5% to 90%). The precision can also affect the power and speed of an architecture. Low precision accelerators provide high speed at lower power (see the blue and red color marks in Fig. 16). In sub-word parallel architectures, by running at half precision, the speed can be doubled at the same amount of power [32, 33, 58] or power can be reduced at the same speed [31]. The binary and INT4 precision architectures can achieve high speed at low power but has limited applications.

5. Conclusions

Understand the factors affecting the performance on an DNN accelerator are important to develop an energy efficient accelerator. In this study, three major areas ALU, dataflow and sparsity are identified as potential areas to improve the overall performance of a DNN accelerator. The existing architectures are classified into three categories. Advantage and drawback of each category are discussed. Precision

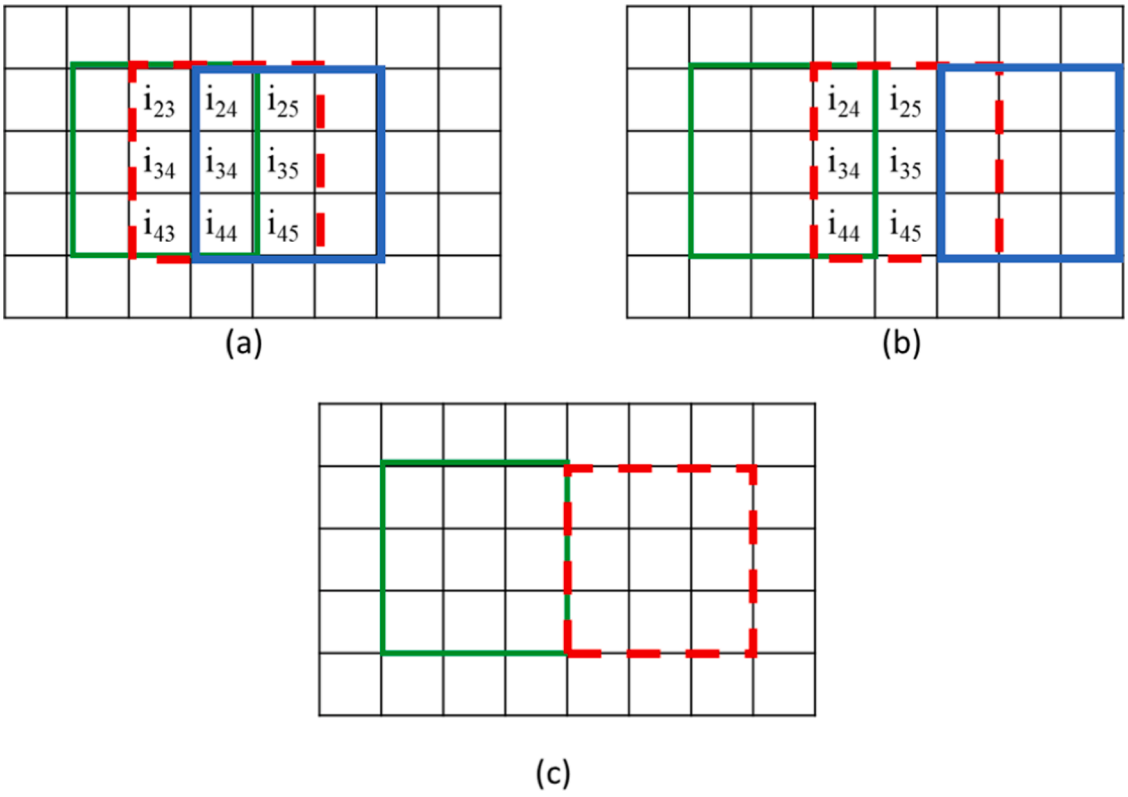


Fig. 15. Impact of the convolution strides on the input data reusability. (a), (b), (c) represent input feature maps with filters (colored boxes) imposed on it to indicate consecutive convolution windows with stride values of 1, 2, and 3, respectively.

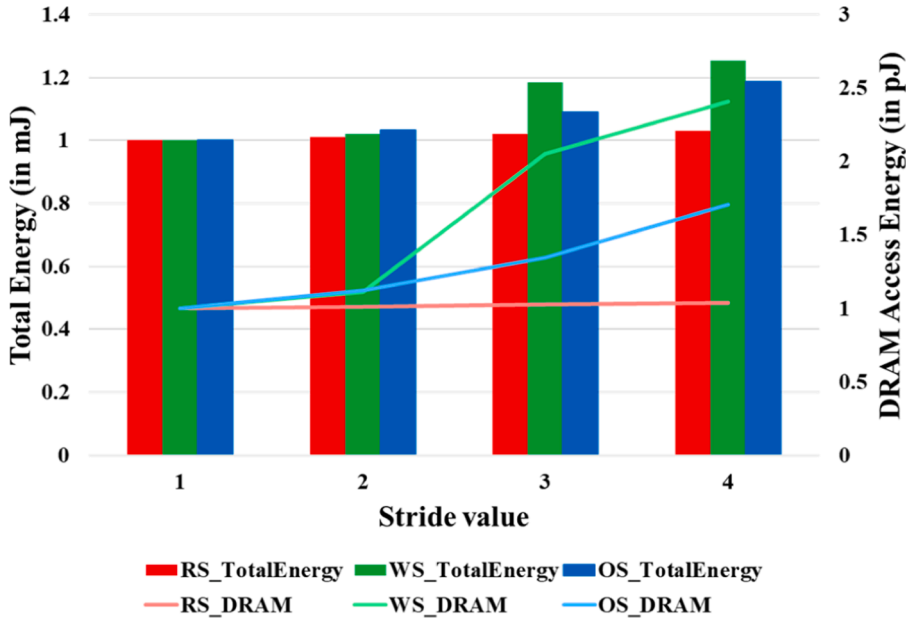


Fig. 16. Energy variation in the RS, WS and OS with stride variation for CONV4 workload.

variable ALU can take advantage of subword parallel processing for low precision DNN models to improve overall throughput or to reduce the power requirement. But precision variable ALU comes with complex configuration circuit. An efficient data flow can improve the arithmetic intensity and memory bandwidth requirement. The dataflow efficiency can vary from layer to layer or with filter size. The sparse models can reduce the power requirement by skipping zero multiplication but increase the latency per MAC operation. Three dataflow architectures are evaluated. The dataflow efficiency depends on the workload. Hence, the dataflow must be chosen based on the accelerator application. The classification discussed in Section III will help the readers in choosing the best technique at different levels in an architecture. An efficient DNN accelerator should have variable precision ALU, flexible dataflow for all types of layers in a DNN model and explore the sparsity with simple control circuitry.

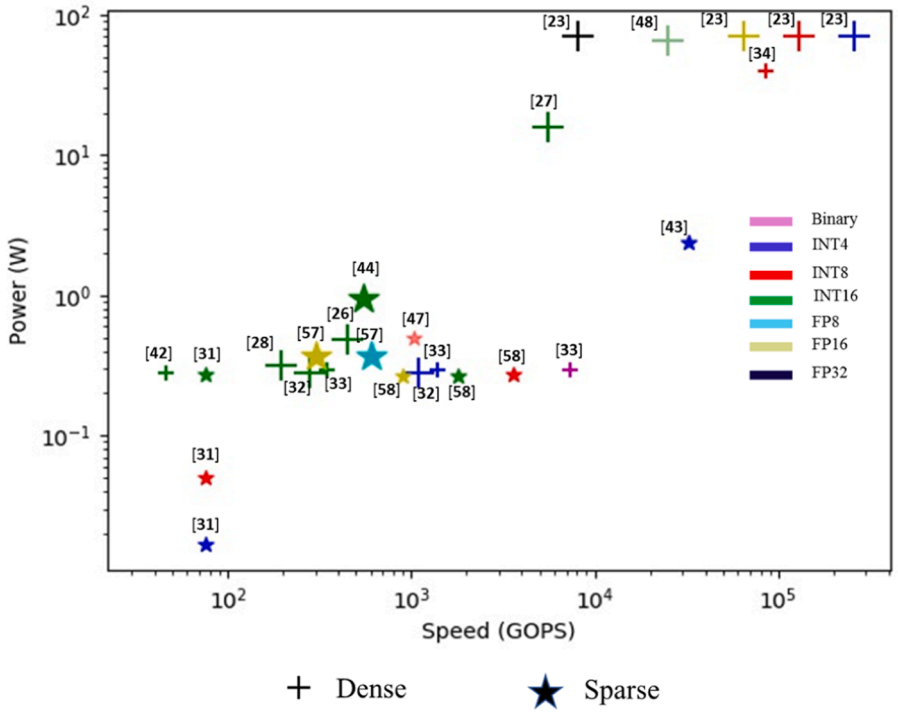


Fig. 17. Performance of DNN architectures with different precision and sparsity levels. The sparsity-based accelerators are denoted with star mark and dense models with plus sign. Small size mark indicates real-time performance and large size mark indicates peak performance.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

We acknowledge the financial support of the Natural Sciences and Engineering Research Council of Canada (NSERC) (Grant number RGPIN-2020-05873).

References

[1] A Krizhevsky, I Sutskever, GE. Hinton, Imagenet classification with deep convolutional neural networks, Adv. Neural Inf. Process. Syst. (2012) 1097–1105, https://doi.org/10.1145/3065386, https://dl.acm.org/doi/.

[2] HA Pierson, MS. Gashler, Deep learning in robotics: a review of recent research, Adv. Robot. 31 (16) (2017) 821–835, https://doi.org/10.1080/01691864.2017.1365009.

[3] DS Berman, AL Buczak, JS Chavis, CL. Corbett, A survey of deep learning methods for cyber security, Information 10 (4) (2019) 122, https://doi.org/10.3390/info10040122.

[4] M Havaei, A Davy, D Warde-Farley, A Biard, A Courville, Y Bengio, C Pal, PM Jodoin, Larochelle H. Brain tumor segmentation with deep neural networks, Med. Image Anal. 35 (2017) 18–31, https://doi.org/10.1016/j.media.2016.05.004.

[5] C Chen, A Seff, A Kornhauser, Xiao J. Deepdriving, Learning affordance for direct perception in autonomous driving, in: In Proceedings of the IEEE International Conference on Computer Vision, 2015, pp. 2722–2730, https://doi.org/10.1109/ICCV.2015.312.

[6] S. Albanie, Convnet Burden, [Online] https://github.com/albanie/convnet-burden , (last access: Oct. 19th, 2020).

[7] V Sze, YH Chen, TJ Yang, JS. Emer, Efficient processing of deep neural networks: a tutorial and survey, in: Proceedings of the IEEE 105, 2017, pp. 2295–2329, https://doi.org/10.1109/JPROC.2017.2761740.

[8] P Colangelo, N Nasiri, E Nurvitadhi, A Mishra, M Margala, K. Nealis, Exploration of low numeric precision deep learning inference using intel® FPGAs, in: 2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), IEEE, 2018, pp. 73–80, https://doi.org/10.1109/FCCM.2018.00020.

[9] S Hashemi, N Anthony, H Tann, RI Bahar, S. Reda, Understanding the impact of precision quantization on the accuracy and energy of neural networks, in: In Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017IEEE, 2017, pp. 1474–1479, https://doi.org/10.23919/DATE.2017.7927224.

[10] C Sakr, Y Kim, N. Shanbhag, Analytical guarantees on numerical precision of deep neural networks, in: Proceedings of the 34th International Conference on Machine Learning 70, 2017, pp. 3007–3016.

[11] P Gysel, M Motamedi, S. Ghiasi, Hardware-oriented approximation of convolutional neural networks, in: Proc. of the 4th International Conference on Learning Representations (ICLR), 2016 (arXiv:1604.03168).

[12] V Camus, L Mei, C Enz, M. Verhelst, Review and benchmarking of precision-scalable multiply-accumulate unit architectures for embedded neural-network processing, IEEE J. Emerg. Sel. Top. Circ. Syst. 9 (4) (2019) 697–711, https://doi.org/10.1109/JETCAS.2019.2950386.

[13] A Reuther, P Michaleas, M Jones, V Gadepally, S Samsi, J. Kepner, Survey and benchmarking of machine learning accelerators, in: In2019 IEEE High Performance Extreme Computing Conference (HPEC), IEEE, 2019, pp. 1–9, https://doi.org/10.1109/HPEC.2019.8916327.

[14] Russakovsky O, Deng J, Su H, Krause J, Satheesh S, Ma S, Huang Z, Karpathy A, Khosla A, Bernstein M, Berg AC. Imagenet large scale visual recognition challenge. International journal of computer vision. 2015;115(3):211-52. https://doi.org/10.1007/s11263-015-0816-y.

[15] Q Chen, C Xin, C Zou, X Wang, B. Wang, A low bit-width parameter representation method for hardware-oriented convolution neural networks, in: IEEE 12th International Conference on ASIC (ASICON), IEEE, 2017, pp. 148–151, https://doi.org/10.1109/ASICON.2017.8252433.

[16] M. Horowitz, 1.1 Computing's energy problem (and what we can do about it), in: In2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC), IEEE, 2014, pp. 10–14, https://doi.org/10.1109/ISSCC.2014.6757323.

[17] I Hubara, M Courbariaux, D Soudry, R El-Yaniv, Y. Bengio, Quantized neural networks: training neural networks with low precision weights and activations, J. Mach. Learn. Res. 18 (1) (2017) 6869–6898.

[18] B Jacob, S Kligys, B Chen, M Zhu, M Tang, A Howard, H Adam, D. Kalenichenko, Quantization and training of neural networks for efficient integer-arithmetic-only inference, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 2704–2713, https://doi.org/10.1109/CVPR.2018.00286.

[19] S Wu, G Li, F Chen, L. Shi, Training and inference with integers in deep neural networks, in: International Conference on Learning Representations, 2018 arXiv:1802.04680.

[20] I Hubara, M Courbariaux, D Soudry, R El-Yaniv, Y. Bengio, Binarized neural networks, Adv. Neural Inf. Process. Syst. (2016) 4107–4115.

[21] Li F, Zhang B, Liu B. Ternary weight networks. arXiv preprint arXiv:1605.04711. 2016 May 16.

[22] P Judd, J Albericio, T Hetherington, TM Aamodt, NE Jerger, A. Moshovos, Proteus: exploiting numerical precision variability in deep neural networks, in: Proceedings of the 2016 International Conference on Supercomputing, 2016, pp. 1–12, https://doi.org/10.1145/2925426.2926294.

[23] NVIDIA T4, Tensor core GPU, (Online) https://www.nvidia.com/en-us/data-center/tesla-t4/(last access: Oct. 19th, 2020).

[24] E Wang, JJ Davis, R Zhao, HC Ng, X Niu, W Luk, PY Cheung, GA. Constantinides, Deep Neural Network Approximation for Custom Hardware: Where We've Been, Where We're Going, ACM Comput. Surv. 52 (2) (2019) 1–39, https://doi.org/10.1145/3309551.

[25] S Han, H Mao, WJ. Dally, Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding, in: Proc. of the 4th International Conference on Learning Representations (ICLR), 2016 (arXiv: 1510.00149).

[26] T Chen, Z Du, N Sun, J Wang, C Wu, Y Chen, O. Temam, Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning, ACM SIGARCH Comput. Architect. News 42 (1) (2014) 269–284, https://doi.org/10.1145/2644865.2541967.

[27] Y Chen, T Luo, S Liu, S Zhang, L He, J Wang, L Li, T Chen, Z Xu, N Sun, O. Temam, Dadiannao: a machine-learning supercomputer, in: 2014 47th Annual IEEE/ACM International Symposium on Microarchitecture, 2014, pp. 609–622, https://doi.org/10.1109/MICRO.2014.58.

[28] Z Du, R Fasthuber, T Chen, P Ienne, L Li, T Luo, X Feng, Y Chen, Temam O. ShiDianNao, Shifting vision processing closer to the sensor, in: Proceedings of the 42nd Annual International Symposium on Computer Architecture, 2015, pp. 92–104, https://doi.org/10.1145/2749469.2750389.

[29] D Liu, T Chen, S Liu, J Zhou, S Zhou, O Teman, X Feng, X Zhou, Y. Chen, Pudiannao: A polyvalent machine learning accelerator, ACM SIGARCH Comput. Architect. News 43 (1) (2015) 369–381, https://doi.org/10.1145/2786763.2694358.

[30] Y Chen, T Chen, Z Xu, N Sun, O. Temam, DianNao family: energy-efficient hardware accelerators for machine learning, Commun. ACM 59 (11) (2016) 105–112, https://doi.org/10.1145/2996864.

[31] B Moons, R Uytterhoeven, W Dehaene, M. Verhelst, 14.5 envision: a 0.26-to-10tops/w subword-parallel dynamic-voltage-accuracy-frequency-scalable convolutional neural network processor in 28nm FDSOI, in: In2017 IEEE International Solid-State Circuits Conference (ISSCC), IEEE, 2017, pp. 246–247, https://doi.org/10.1109/ISSCC.2017.7870353.

[32] D Shin, J Lee, J Lee, HJ. Yoo, 14.2 DNPU: An 8.1 TOPS/W reconfigurable CNN-RNN processor for general-purpose deep neural networks, in: IEEE International Solid-State Circuits Conference (ISSCC), 2017, pp. 240–241, https://doi.org/10.1109/ISSCC.2017.7870350.

[33] J Lee, C Kim, S Kang, D Shin, S Kim, HJ Yoo, UNPU: A 50.6 TOPS/W unified deep neural network accelerator with 1b-to-16b fully-variable weight bit-precision, in: IEEE International Solid-State Circuits Conference-(ISSCC), 2018, pp. 218–220, https://doi.org/10.1109/ISSCC.2018.8310262.

[34] NP Jouppi, C Young, N Patil, D Patterson, G Agrawal, R Bajwa, S Bates, S Bhatia, N Boden, A Borchers, R. Boyle, In-datacenter performance analysis of a tensor processing unit, in: Proc. of the 44th Annual International Symposium on Computer Architecture, 2017, pp. 1–12, https://doi.org/10.1145/3140659.3080246.

[35] Y Chen, Y Xie, L Song, F Chen, T. Tang, A survey of accelerator architectures for deep neural networks, Engineering 6 (2020) 264–274, https://doi.org/10.1016/j.eng.2020.01.007.

[36] A Parashar, M Rhu, A Mukkara, A Puglielli, R Venkatesan, B Khailany, J Emer, SW Keckler, WJ. Dally, SCNN: an accelerator for compressed-sparse convolutional neural networks, ACM SIGARCH Comput. Architect. News 45 (2) (2017) 27–40, https://doi.org/10.1145/3140659.3080254.

[37] YH Chen, J Emer, V. Sze, Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks, ACM SIGARCH Comput. Architect. News 44 (3) (2016) 367–379, https://doi.org/10.1145/3007787.3001177.

[38] YH Chen, TJ Yang, J Emer, V. Sze, Eyeriss v2: a flexible accelerator for emerging deep neural networks on mobile devices, IEEE J. Emerg. Sel. Top. Circ. Syst. 9 (2) (2019) 292–308, https://doi.org/10.1109/JETCAS.2019.2910232.

[39] M Alwani, H Chen, M Ferdman, P. Milder, Fused-layer CNN accelerators, in: 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2016, pp. 1–12, https://doi.org/10.1109/MICRO.2016.7783725.

[40] J Albericio, P Judd, T Hetherington, T Aamodt, NE Jerger, A. Moshovos, Cnvlutin: Ineffectual-neuron-free deep neural network computing, ACM SIGARCH Comput. Architect. News 44 (3) (2016) 1–3, https://doi.org/10.1145/3007787.3001138.

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

[42] YH Chen, T Krishna, JS Emer, V. Sze, Eyeriss: an energy-efficient reconfigurable accelerator for deep convolutional neural networks, IEEE J. Solid-State Circ. 52 (1) (2016) 127–138, https://doi.org/10.1109/JSSC.2016.2616357.

[43] S Han, X Liu, H Mao, J Pu, A Pedram, MA Horowitz, WJ. Dally, EIE: efficient inference engine on compressed deep neural network, ACM SIGARCH Comput. Architect. News 44 (3) (2016) 243–254, https://doi.org/10.1145/3007787.3001163.

[44] S Zhang, Z Du, L Zhang, H Lan, S Liu, L Li, Q Guo, T Chen, Y. Chen, Cambricon-x: an accelerator for sparse neural networks, in: 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), IEEE, 2016, pp. 1–12, https://doi.org/10.1109/MICRO.2016.7783723.

[45] JF Zhang, CE Lee, C Liu, YS Shao, SW Keckler, Z. Zhang, SNAP: A 1.67—21.55 TOPS/W sparse neural acceleration processor for unstructured sparse deep neural network inference in 16nm CMOS, in: Symposium on VLSI Circuits, 2019, pp. C306–C307, https://doi.org/10.23919/VLSIC.2019.8778193.

[46] TP Xiao, CH Bennett, B Feinberg, S Agarwal, MJ. Marinella, Analog architectures for neural network acceleration based on non-volatile memory, Appl. Phys. Rev. 7 (3) (2020), 031301, https://doi.org/10.1063/1.5143815.

[47] P Chi, S Li, C Xu, T Zhang, J Zhao, Y Liu, Y Wang, Y. Xie, Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory, ACM SIGARCH Comput. Architect. News 44 (3) (2016) 27–39, https://doi.org/10.1109/ISCA.2016.13.

[48] A Shafiee, A Nag, N Muralimanohar, R Balasubramonian, JP Strachan, M Hu, RS Williams, V.ISAAC Srikumar, A convolutional neural network accelerator with in-situ analog arithmetic in crossbars, ACM SIGARCH Comput. Architect. News 44 (3) (2016) 14–26, https://doi.org/10.1109/ISCA.2016.12.

[49] MN Bojnordi, E. Ipek, Memristive boltzmann machine: A hardware accelerator for combinatorial optimization and deep learning, in: In2016 IEEE International Symposium on High Performance Computer Architecture (HPCA, IEEE, 2016, pp. 1–13, https://doi.org/10.1109/HPCA.2016.7446049.

[50] A Nag, R Balasubramonian, V Srikumar, R Walker, A Shafiee, JP Strachan, N. Muralimanohar, Newton: Gravitating towards the physical limits of crossbar acceleration, IEEE Micro 38 (5) (2018) 41–49, https://doi.org/10.1109/MM.2018.053631140.

[51] P Yao, H Wu, B Gao, J Tang, Q Zhang, W Zhang, JJ Yang, H. Qian, Fully hardware-implemented memristor convolutional neural network, Nature 577 (7792) (2020) 641–646, https://doi.org/10.1038/s41586-020-1942-4.

[52] N Cao, M Chang, Raychowdhury A. 14.1 A 65nm 1.1-to-9.1 TOPS/W hybrid-digital-mixed-signal computing platform for accelerating model-based and model-free swarm robotics, in: IEEE International Solid-State Circuits Conference-(ISSCC), 2019, pp. 222–224, https://doi.org/10.1109/ISSCC.2019.8662311.

[53] D Bankman, L Yang, B Moons, M Verhelst, B. Murmann, An Always-On 3.8uJ/86% CIFAR-10 mixed-signal binary CNN processor with all memory on chip in 28-nm CMOS, IEEE J. Solid-State Circ. 54 (1) (2018) 158–172, https://doi.org/10.1109/JSSC.2018.2869150.

[54] A Parashar, P Raina, YS Shao, YH Chen, VA Ying, A Mukkara, R Venkatesan, B Khailany, SW Keckler, J. Emer, Timeloop: a systematic approach to dnn accelerator evaluation, in: IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), IEEE, 2019, pp. 304–315, https://doi.org/10.1109/ISPASS.2019.00042.

[55] TJ Yang, YH Chen, J Emer, V. Sze, A method to estimate the energy consumption of deep neural networks, in: 51st Asilomar Conference on Signals, Systems, and Computers, IEEE, 2017, pp. 1916–1920, https://doi.org/10.1109/ACSSC.2017.8335698.

[56] K. Guo, W. Li, K. Zhong, Z. Zhu, S. Zeng, S. Han, Y. Xie, P. Debacker, M. Verhelst, Y. Wang. Neural Network Accelerator Comparison, [Online] https://nicsefc.ee.tsinghua.edu.cn/projects/neural-network-accelerator/(last access: Oct. 20, 2020).

[57] J Lee, J Lee, D Han, J Lee, G Park, Yoo HJ. 7.7, LNPU: A 25.3 tflops/w sparse deep-neural-network learning processor with fine-grained mixed precision of fp8-fp16, in: IEEE International Solid-State Circuits Conference-(ISSCC), 2019, pp. 142–144, https://doi.org/10.1109/ISSCC.2019.8662302.

[58] CH Lin, CC Cheng, YM Tsai, SJ Hung, YT Kuo, PH Wang, PK Tsung, JY Hsu, WC Lai, CH Liu, SY. Wang, 7.1 A 3.4-to-13.3 TOPS/W 3.6 TOPS dual-core deep-learning accelerator for versatile AI applications in 7nm 5G smartphone SoC, in: In2020 IEEE International Solid-State Circuits Conference-(ISSCC), IEEE, 2020, pp. 134–136, https://doi.org/10.1109/ISSCC19947.2020.9063111.

[59] Z Du, Q Guo, Y Zhao, T Zhi, Y Chen, Z. Xu, Self-aware neural network systems: a survey and new perspective, in: Proceedings of the IEEE 108, 2020, pp. 1047–1067, https://doi.org/10.1109/JPROC.2020.2977722.

[60] K Guo, S Zeng, J Yu, Y Wang, H Yang, [DL] A survey of FPGA-based neural network inference accelerators, ACM Trans. Reconfig. Technol. Syst. 12 (1) (2019) 1–26, https://doi.org/10.1145/3289185.

[61] Z Li, Y Wang, T Zhi, T. Chen, A survey of neural network accelerators, Front. Comput. Sci. 11 (5) (2017) 746–761, https://doi.org/10.1007/s11704-016-6159-1″, https://doi.org/10.1007/s11704-016-6159-1.

[62] D Kim, J Kung, S Chai, S Yalamanchili, S. Mukhopadhyay, Neurocube: a programmable digital neuromorphic architecture with high-density 3D memory, ACM SIGARCH Comput. Architect. News 44 (3) (2016) 380–392, https://doi.org/10.1145/3007787.3001178.

[63] H Lu, X Wei, N Lin, G Yan, X. Li, Tetris: re-architecting convolutional neural network computation for machine learning accelerators, in: In2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD, IEEE, 2018, pp. 1–8, https://doi.org/10.1145/3240765.3240855.

[64] YS Shao, J Clemons, R Venkatesan, B Zimmer, M Fojtik, N Jiang, B Keller, A Klinefelter, N Pinckney, P Raina, SG. Tell, Simba: Scaling deep-learning inference with multi-chip-module-based architecture, in: InProceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, 2019, pp. 14–27, https://doi.org/10.1145/3352460.3358302.

[65] X Zhou, Z Du, Q Guo, S Liu, C Liu, C Wang, X Zhou, L Li, T Chen, Y. Chen, S Cambricon, Addressing irregularity in sparse neural networks through a cooperative software/hardware approach, in: 2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO, IEEE, 2018, pp. 15–28.

[66] R Venkatesan, YS Shao, M Wang, J Clemons, S Dai, M Fojtik, B Keller, A Klinefelter, N Pinckney, P Raina, Y. Zhang, Magnet: a modular accelerator generator for neural networks, in: In2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), IEEE, 2019, pp. 1–8.

[67] TF De Lima, HT Peng, AN Tait, MA Nahmias, HB Miller, BJ Shastri, PR. Prucnal, Machine learning with neuromorphic photonics, J. Lightw. Technol. 37 (5) (2019) 1515–1534.

[68] A Ankit, IE Hajj, SR Chalamalasetti, G Ndu, M Foltin, RS Williams, P Faraboschi, WM Hwu, JP Strachan, K Roy, DS. Milojicic, PUMA: a programmable ultra-efficient

memristor-based accelerator for machine learning inference, in: Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, 2019, pp. 715–731.
[69] TP Xiao, CH Bennett, B Feinberg, S Agarwal, MJ. Marinella, Analog architectures for neural network acceleration based on non-volatile memory, Appl. Phys. Rev. 7 (3) (2020), 031301.
[70] MS Ansari, BF Cockburn, J. Han, An improved logarithmic multiplier for energy-efficient neural computing, IEEE Trans. Comput. (2020).
[71] MS Kim, AA Garcia, H Kim, N. Bagherzadeh, The effects of approximate multiplication on convolutional neural networks, IEEE Trans. Emerg. Top. Comput. (2021).
[72] N Samimi, M Kamal, A Afzali-Kusha, M. Pedram, Res-DNN: A residue number system-based DNN accelerator unit, IEEE Trans. Circ. Syst. Regul. Pap. 67 (2) (2019) 658–671.
[73] Z Carmichael, HF Langroudi, C Khazanov, J Lillie, JL Gustafson, D Kudithipudi, Deep positron: a deep neural network using the posit number system, in: In2019 Design, Automation & Test in Europe Conference & Exhibition (DATE), IEEE, 2019, pp. 1421–1426.
[74] H Tsai, S Ambrogio, P Narayanan, RM Shelby, GW. Burr, Recent progress in analog memory-based accelerators for deep learning, J. Phys. D 51 (28) (2018), 283001.
[75] EB. Olsen, RNS Hardware matrix multiplier for high precision neural network acceleration:" RNS TPU", in: In2018 IEEE International Symposium on Circuits and Systems (ISCAS, IEEE, 2018, pp. 1–5.
[76] MA Talib, S Majzoub, Q Nasir, D. Jamal, A systematic literature review on hardware implementation of artificial intelligence algorithms, J. Supercomput. 77 (2021) 1897–1938.

Raju Machupalli is pursuing his M.Sc. at the University of Alberta, Canada. He completed his master's in Integrated Circuit Technology from University of Hyderabad, India and Bachelors in Electronics and Communications from JNTUH, India in 2017 and 2014, respectively. His areas of interest include VLSI, Embedded Systems, FPGA and ASIC based Hardware accelerators for Machine Learning, Artificial Intelligence, and Image Processing.

Masum Hossain received the B.Sc. degree from the Bangladesh University of Engineering and Technology, Dhaka, Bangladesh, in 2002, the M.Sc. degree from Queen's University, Kingston, ON, Canada, in 2005, and the Ph.D. degree from the University of Toronto, Toronto, ON, in 2010. From 2008 to 2010, he was with Analog and Mixed Signal Division, Gennum Corporation, Burlington, ON, where he was involved in the development of world's highest capacity and most power efficient cross point router solution. He was with Rambus Laboratory, Sunnyvale, CA, USA, as a Senior Member of Technical Staff, where he was involved in advanced equalization and clock recovery techniques for highspeed interfaces. He has spent several years in industrial research. In 2013, he joined the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB, Canada. Dr. Hossain was a recipient of the Best Student Paper Award at the 2008 IEEE Custom Integrated Circuits Conference and the Analog Device's Outstanding Student Designer Award in 2010.

Mrinal Mandal is currently a Full Professor in the Department of Electrical and Computer Engineering and is the Director of the Digital Image Analysis Laboratory, University of Alberta, Edmonton, AB, Canada. He has authored the book *Multimedia Signals and Systems* (Springer) and coauthored the book *Continuous and Discrete Time Signals and Systems* (Cambridge University Press). His current research interests include medical image analysis, machine learning, image and video processing, and VLSI architectures. He has published more than 200 papers in refereed journals and conferences and holds a U. S. patent on wavelet transform architecture. He is currently the Principal Investigator of projects funded by NSERC, Alberta Innovates, and DSP Innovation. Prof. Mandal has been a past recipient of the Canadian Commonwealth Fellowship and Humboldt Research Fellowship (Germany).