POLITECNICO DI TORINO
Repository ISTITUZIONALE

An Updated Survey of Efficient Hardware Architectures for Accelerating Deep Convolutional Neural Networks

Original
An Updated Survey of Efficient Hardware Architectures for Accelerating Deep Convolutional Neural Networks / Capra, M., Bussolino, B., Marchisio, A., Shafique, M., Masera, G., Martina, M.. - In: FUTURE INTERNET. - ISSN 1999-5903. - ELETTRONICO. - 12:7(2020), pp. 113-134. [10.3390/fi12070113]

Availability:
This version is available at: 11583/2839398 since: 2020-07-10T12:50:58Z

Publisher:
MDPI

Published
DOI:10.3390/fi12070113

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

13 October 2026

future internet

MDPI

Review

An Updated Survey of Efficient Hardware Architectures for Accelerating Deep Convolutional Neural Networks

Maurizio Capra [1,*], Beatrice Bussolino [1,*], Alberto Marchisio [2], Muhammad Shafique [2], Guido Masera [1] and Maurizio Martina [1,*]

[1] Department of Electrical, Electronics and Telecommunication Engineering, Politecnico di Torino, 10129 Torino, Italy; guido.masera@polito.it (G.M.)

[2] Embedded Computing Systems, Institute of Computer Engineering, Technische Universität Wien (TU Wien), 1040 Vienna, Austria; alberto.marchisio@tuwien.ac.at (A.M.); muhammad.shafique@tuwien.ac.at (M.S.)

* Correspondence: maurizio.capra@polito.it (M.C.); beatrice.bussolino@polito.it (B.B.); maurizio.martina@polito.it (M.M.)

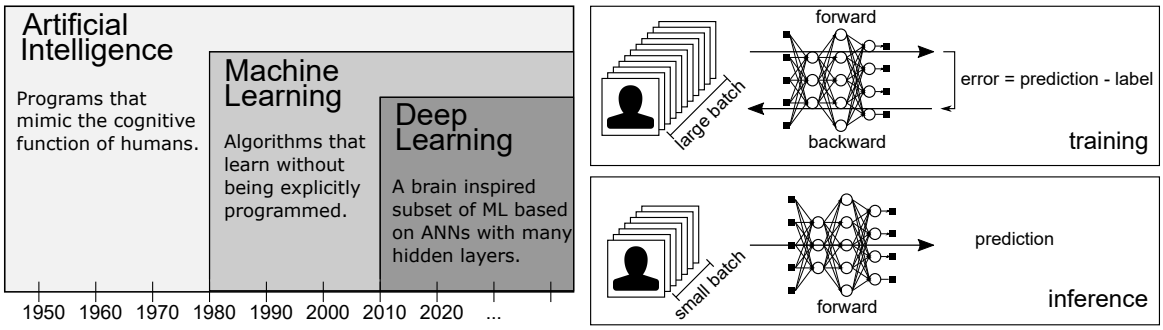
Received: 4 June 2020; Accepted: 2 July 2020; Published: 7 July 2020

Abstract: Deep Neural Networks (DNNs) are nowadays a common practice in most of the Artificial Intelligence (AI) applications. Their ability to go beyond human precision has made these networks a milestone in the history of AI. However, while on the one hand they present cutting edge performance, on the other hand they require enormous computing power. For this reason, numerous optimization techniques at the hardware and software level, and specialized architectures, have been developed to process these models with high performance and power/energy efficiency without affecting their accuracy. In the past, multiple surveys have been reported to provide an overview of different architectures and optimization techniques for efficient execution of Deep Learning (DL) algorithms. This work aims at providing an up-to-date survey, especially covering the prominent works from the last 3 years of the hardware architectures research for DNNs. In this paper, the reader will first understand what a hardware accelerator is, and what are its main components, followed by the latest techniques in the field of dataflow, reconfigurability, variable bit-width, and sparsity.

Keywords: machine learning; artificial intelligence; AI; deep learning; deep neural networks; DNNs; convolutional neural networks; CNNs; VLSI; computer architecture; hardware accelerator; data flow; optimization; efficiency; performance; power consumption; energy; area; latency

1. Introduction

In the era of Big Data and Internet-of-Things (IoT), Artificial Intelligence (AI) has found an ideal environment and a continuous stream of data from which to learn and thrive. In recent years, the research and development in AI, and more specifically its subset Machine Learning (ML), has exponentially increased, spreading in several discipline fields, and covering many applications. The ML consists of several algorithms and paradigms, in which the most impactful ones are the brain-inspired techniques. Among these, one that is based on Artificial Neural Networks (ANNs) has overcome the human accuracy, namely the Deep Learning (DL) [1], as shown in Figure 1a. Since its recent appearance, the DL showed many advantages over previous techniques, on the ability to work directly on raw data in large quantities combined with a very deep architecture. While the lack of preprocessing makes the process more streamlined, the alternation of a large number of layers increases the accuracy making the DL the technique par excellence.



a) The evolution of Artificial Intelligence. b) Inference vs. Training.

Figure 1. (**a**) How artificial intelligence has evolved over the years. (**b**) The processes of training and inference in comparison.

These networks are called Deep Neural Networks (DNNs) and cover a wide range of applications, for instance, business and finance [2–4], healthcare such as cancer detection [5–7], up to robotics [8,9], and computer vision [10–12].

As mentioned above, these networks are very complex and computation/memory-hungry. Therefore it is necessary to provide suitable/specialized hardware platforms for the execution of such algorithms over a consistent data stream. The layers of DNNs can reach a considerable size of up to hundreds of thousands of activations; consequently, the multiplication matrix vectors of an entire network can reach up to require a few billion multiply-and-accumulate (MAC) operations. As shown in Figure 1b, neural networks can pose processing requirements in two different ways, i.e., training during the design phase, and inference during the deployment phase. The inference is run in real-world applications, after the neural network has been trained, and is used to classify or derive predictions from the given inputs in real-world scenarios. While in the inference, the network only experiences the forward-pass, during the training, it experiences both the forward-pass and the backward-pass. During the latter, the prediction is compared with the label, and the error is used to update the weights through the backpropagation process. As a consequence, the training requires a much more extensive computational effort compared to that for the inference. The recent trend in these years have seen DL applications moving towards mobile platforms such as IoT/Edge nodes and smart cyber-physical systems (CPS) devices [13–15]. Often these devices have stringent constraints in terms of latency, power, and energy, for instance, due to their real-time and battery-powered nature. Moreover, moving the computation from the cloud to the edge reduces the privacy and security threats to which various DL systems are subjected, hence increasing the need for embedded DL [16,17]. In short, there is a growing demand for specialized hardware accelerators with optimized memory hierarchies that can meet the enormous compute and memory requirements of different types of complex DNNs, while maintaining a reduced power and energy envelope.

Over the past decade, several architectures have been proposed for the acceleration of DL algorithms. Many papers and surveys on this topic have been produced [18–21]. However, due to rapid developments of DNN hardware, these surveys have either become obsolete or do not represent the emerging trends. Towards this, this paper aims to provide an up-to-date survey covering the state-of-the-art of the last 3 years. The work is therefore not intended as a substitute for existing surveys, but rather as an integral part that can be seen as a continuation of existing surveys. In the following sections, we will deal with the latest architectures with the main focus on new types of dataflow, reconfigurable architectures, variable precision, and sparsity. The reconfigurable architecture, sometimes coupled with adaptable bitwidth, is a flexible solution to accommodate different types of networks and is likely to become the standard in the future. In fact, researchers have shown that networks can be compressed [22,23] and represented on a number of bits that are being reduced over time as techniques are refined. Finally, the sparsity is a technique actively used to eliminate

unnecessary operations and to further lower the power envelope of the accelerators, as well as making them faster and more effective. This also instantiates the need for sparse DNN accelerators like [24–26].

This paper is organized as follows: Section 2 discusses the background related to DNNs, describing the different models and their key features. Section 3 analyzes dataflows designed for energy-efficient architectures. The discussion goes from temporal and spatial architectures to other important themes such as sparsity, variable bit-width precision and reconfigurable architectures. Section 4 shows the typical memory hierarchy used in accelerators and the methodologies to reduce the power consumed by it. Finally, the paper provides the key takeaways in the conclusion. The acronyms used in this paper are reported in Table A1.

2. Background: Deep Neural Networks

An Artificial Neural Network (ANN), henceforth called Neural Network (NN), is a mathematical model inspired by the biological neural networks. However, the NN model is too simple to replicate the behavior of its biological counterpart, faithfully. An NN is formed by interconnected nodes, as in a graph, that are organized in layers (see Figure 2). A layer of input nodes receives signals from the outside, which are then processed by some intermediate layers called hidden layers. The result is finally obtained by the last layer, also called output layer. An NN with more than three hidden layers is defined in literature as a Deep Neural Network (DNN) [27]. In short, the DNNs/NNs are mainly black-box model representation of a given function. That is, the model and its parameters are learned by finding the transfer function (composed of multiple layers of neurons, weights, and activation functions) from input to output through an extensive training process.

The graphs of NNs are direct, i.e., the connections are oriented, and can be acyclic or cyclic. If the NN is acyclic, it is called feedforward, and the output depends only on the current input. If instead, the NN is cyclic, it is defined recurrent, and the output also depends on the previous inputs. Recurrent NNs are, therefore, models with state/memory.

The nodes of NNs are the neurons, graphically and mathematically described in Figure 3 and Equation (1). A neuron receives n inputs $(x_1, x_2, \ldots, x_n)$ and returns a scalar output y. For a given neuron, the inputs are multiplied with the weights $(w_1, w_2, \ldots, w_n)$ and summed together with a bias term b. A non-linear function $\sigma(\cdot)$, called activation function, is then applied to determine the output of the neuron. Common activation functions are Rectified Linear Unit (ReLU), Sigmoid or Hyperbolic tangent.

$$y(\mathbf{x}) = \sigma\left(\sum_{n=0}^{N-1} \mathbf{x}[n]\mathbf{w}[n] + b\right) \tag{1}$$

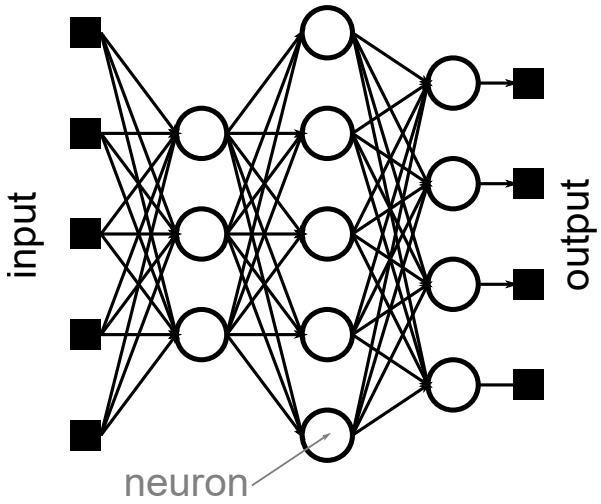


Figure 2. An abstract example of a neural network.

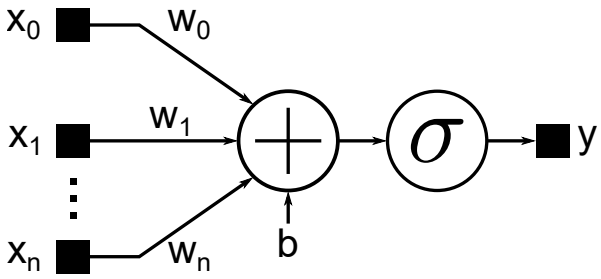


Figure 3. A graphical model of the neuron.

2.1. Training and Inference

NNs learn to achieve the desired results by modifying their internal parameters, i.e., weights and biases. The phase in which the network learns is called training. Once the network has been trained, it can be used to solve unknown problems during the inference phase when deployed in real-world.

One of the most used learning paradigms is supervised learning, thanks to the large amount of (labeled) data that has become available in the so-called big-data era. Supervised learning requires labeled data, i.e., input-output pairs, where the output is the result that the network should obtain from the related input. Supervised learning consists of three steps repeated until convergence:

1. *Forward pass*: the input is fed into the network that produces an output.
2. *Backward pass*: a loss L is computed comparing the produced output and the desired output. The loss L is then used for the backpropagation algorithm [28], that applying the chain rule of calculus computes the gradient $\frac{\partial L}{\partial w}$ for each weight (and bias) of the network.
3. *Parameters update*: each weight and bias is updated by an amount proportional to its gradient. All the gradients can be multiplied by the same factor, defined learning rate, or more complex optimization algorithms can be used, such as Gradient Descent with Momentum [29] or Adam [30].

Other learning paradigms are unsupervised learning and reinforcement learning. Unsupervised learning works with unlabeled data and consists of finding common patterns and structures that data may have in common. Reinforcement learning involves the network (agent) interacting in an environment. An interpreter assesses the correctness of the interactions and returns a reward or punishment to the agent, who aims to maximize the reward.

2.2. Layers

As described in the previous paragraphs, neurons are organized in layers that can have different shapes and characteristics. This section presents a short overview of different layers that are most commonly used in NNs.

Fully Connected (FC) Layers. In FC layers, the neurons are arranged in the shape of a vector (see Figure 4). Considering a layer with C_o neurons and C_i inputs, each neuron c_o receives all the C_i inputs (Equation (2)). Therefore, each neuron has C_i weights and the total number of weights of the layer is $C_i \times C_o$.

$$\mathbf{O}[c_o] = \sum_{c_i=0}^{C_i-1} \mathbf{W}[c_o, c_i]\mathbf{I}[c_i] + \mathbf{b}[c_o] \qquad 0 \leq c_o < C_o, \;\; 0 \leq c_i < C_i \tag{2}$$

The number of inputs and outputs of an FC layer can be high. Consequently, also the weight matrix can have a significant size, making it a critical element, especially on hardware platforms with limited memory. However, it is not always necessary for each neuron to analyze the totality of the inputs, and convolutional layers have been introduced to solve this problem.

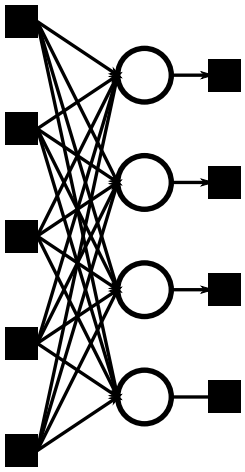


Figure 4. A fully connected layer with $C_i = 5$ and $C_o = 4$.

Convolutional (Conv) Layers. The inputs and outputs of a Conv layer are organized in 2D grids, defined as feature maps (FM). Multiple feature maps can form each input/output: the number of input/output feature maps is referred to as the number of input/output channels. The neurons of the Conv layer, rather than analyzing the whole input, receive only a sub-grid of dimension $H_k \times W_k$ ($C_i \times H_k \times W_k$ if there are multiple input channels). Horizontally adjacent neurons process grids of adjacent inputs separated by S positions, where S is a parameter known as stride. As shown in Figure 5, neurons that produce values belonging to the same output feature map (OFM) usually share the same kernel of weights. Therefore, each neuron of OFM c_o has a kernel of $C_i \times H_k \times W_k$ weights, and the total number of weights is $C_o \times C_i \times H_k \times W_k$. Equation (3) describes the operations performed in the Conv layer.

$$\begin{gathered}\mathbf{Ofm}[c_o, h_o, w_o] = \sum_{c_i=0}^{C_i-1} \sum_{h_k=0}^{H_k-1} \sum_{w_k=0}^{W_k-1} \mathbf{W}[c_i, c_o, h_k, w_k] \mathbf{Ifm}[c_i, Sh_o + h_k, Sw_o + w_k] + \mathbf{b}[c_o] \\ 0 \leq c_o < C_o, \;\; 0 \leq h_o < H_o, \;\; 0 \leq w_o < W_o \\ 0 \leq h_k < H_k, \;\; 0 \leq w_k < W_k\end{gathered} \tag{3}$$

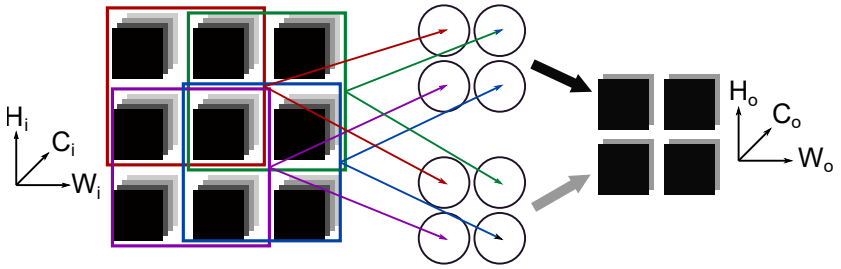


Figure 5. A convolutional layer with $C_i = 4$, $H_i = W_i = 3$, $C_o = H_o = W_o = 2$, $H_k = W_k = 2$.

Normalization Layers. Batch Normalization (BN) layers are often inserted at various points in the neural networks after Conv or FC layers. As can be seen from Equation (4) describing the BN layers, the values are processed so that their mean is zero, and the variance is 1. γ and β are two trainable parameters inserted to integrate normalization in the training phase.

$$y = \frac{x - E[x]}{Var[x] + \epsilon} \cdot \gamma + \beta \tag{4}$$

The BN layers have two primary purposes. They contribute to accelerating the convergence of the training phase. Since the values always maintain a constant distribution, the network does not

have to adapt to different ranges at each training step. Moreover, they avoid value saturation of the values inside the network. Since saturating non-linear functions, such as Sigmoid, Tanh, or Softmax, are often used, having values with zero mean and variance 1 prevents too many values from being saturated, which would cause a considerable loss of information and slow down the training process. Pooling Layers. The main purpose of the pooling layers is to shrink the size of the feature maps within the network, to decrease the number of parameters and the number of operations to be performed. In the pooling layers, different sub-grids of the input feature maps are selected. For each sub-grid, a single value is calculated, which is a statistical metric of the group, e.g., the maximum value (MaxPooling) or the average value (AvgPooling). The sub-grids are usually selected of equal size, adjacent and non-overlapping.

2.3. DNN Models

Since their origin, DNNs have been developed in a large number and a very diverse types of models in order to achieve high accuracy. The first DNN model to become famous was LeNet [28], a Convolutional Neural Network (CNN) that was used to recognize handwritten digits. The real boom for CNNs, which are the most widely used for object detection and recognition, came in 2012 when AlexNet [31] won the ILSVRC competition [32] by outperforming the earlier methods. Since then, also thanks to the increased availability of computational hardware and memory resources, the DNN models have become more and more complex and precise. Table 1 outlines a timeline of the models that have become more popular, describing the innovations introduced compared to previous models.

Table 1. Comparison of the most popular models in the history of DNNs.

Model		Year	Contribution	# Param	Depth	Top-5 Acc ImageNet (%)
LeNet [28]		1998	First popular CNN	60 k	5	-
AlexNet [31]		2012	- First CNN to win ILSVRC - ReLUintroduction	60 M	8	79.06
VGG16 [33]		2014	Smaller kernel sizes	138 M	16	90.37
GoogLeNet [34]		2015	Inception block	4 M	22	87.52
Inception	v3 [35] v4 [36]	2015 2016	Factorized convolutions Simplified inception blocks	24 M 43 M	48 77	93.59 95.30
ResNet [37]	50 152	2016	- Skip connections - Residual learning	26 M 60 M	50 152	92.93 93.98
Xception [38]		2017	Depthwise and pointwise convolutions	23 M	38	94.50
ResNetXt-101_64x4d [39]		2017	Grouped convolution	83 M	101	94.70
DenseNet161 [40]		2017	- Regular structure - Information flow across layers	28 M	161	93.60
SeNet154 [41]		2018	Exploit dependencies between feature maps	115 M	154	95.53
NasNet-A [42]		2018	- Neural Architecture Search - Transfer learning	89 M	29	96.16
BERT-L [43]		2019	Transformer network for Natural Language Processing (NLP)	332 M	24	-
Megatron [44]		2019	Model parallel transformer for NLP	3.9 B	48	-

3. Energy-Efficient Architectures

The panorama of hardware solutions for the development and deployment of DNNs is vast, ranging from general-purpose solutions (CPUs and GPUs) and programmable solutions (FPGAs), to special-purpose ASICs. It is not straightforward to define which of these is the best solution, as it is strictly dependent on the application and the corresponding design constraints including even the time-to-market. For example, an edge IoT chip will require a small area, energy-efficient solution, while a cloud computing server for ML will demand a lot of flexibility. Moreover, a time-to-market may impose usage of GPUs or embedded GPUs as the only feasible platform option.

In the following subsections, the pros and cons of each solution will be discussed in detail, making a clear distinction between temporal and spatial architectures (see Figure 6). These two architectures have a similar computational structure, consisting of a set of multiple processing units. However, while the units of the spatial architectures can have internal control, in temporal architectures, the control is centralized. An analogous distinction can be made for the memory: in spatial architectures, the units can have a register file (RF) to store data, while in temporal architectures, the units have no memory capacity. Moreover, in spatial architectures, the units can also be interconnected to exchange data. Summarizing, the computational units of temporal architectures are typically ALUs, while that of spatial architectures are complex Processing Elements (PEs) that can potentially support articulated data movement patterns.

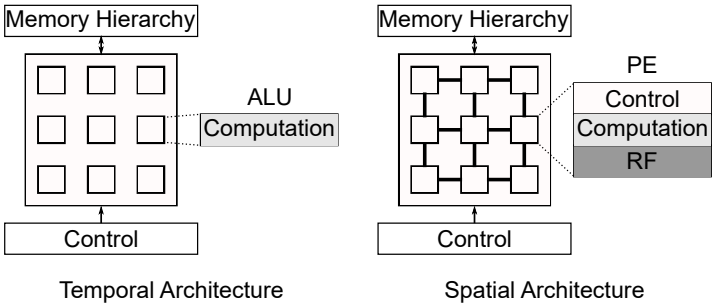


Figure 6. Comparison between the temporal and spatial architectures.

3.1. Temporal Architectures

CPUs and GPUs belong to the category of spatial architectures. Vector CPUs have multiple ALUs that can process multiple data in parallel. Most of them adopt the Single-Instruction Multiple-Data (SIMD) execution model, which applies a single instruction to different data simultaneously. Similarly, GPUs are formed by many processing cores, and they use the Single-Instruction Multiple-Threads (SIMT) execution model. CPUs and GPUs are general-purpose chips that must be able to support an extensive range of applications. For this reason, it is infrequent to find hardware optimizations specific for ML and DNNs. An approach commonly adopted is the attempt to better adapting the application to the chosen hardware platform. For example, the convolutional layer, using sub-grids of the original feature maps, requires discontinuous accesses to the memory. In [45], it is shown how to optimize the storage of feature maps to decrease the number of discontinuous accesses to the memory. Since the libraries for Basic Linear Algebra Subroutines (BLAS) are highly optimized, it is also possible to perform convolution lowering [46,47] to transform the convolution into a General Matrix Multiplication (GeMM), or to move in the frequency domain through Fast Fourier Transform (FFT) [48] and perform a point-wise multiplication of matrices.

Among the different available technologies, CPU cores are the least used for DNNs inference and training. CPUs have the advantage of being easily programmable to perform any kind of task. Still, their throughput is limited by the small number of cores and, therefore, by the small number of operations executable in parallel. Figure 7 compares the number of cores of CPUs and GPUs. The Intel Xeon Platinum 9222, a high-end processor used in servers with price over USD 10,000, has a number of

floating-point operations per second per Watt ($FLOPS/W$) similar to the $FLOPS/W$ of the 2014 Nvidia GT 740 GPU with price below USD 100 ($\sim 12 GFLOPS/W$). High-end GPUs, with $FLOPS/W$ in the order of TERAs, significantly surpass any CPU. However, some attempts have been recently made to accelerate DNNs deployment (inference in particular) on CPUs. At instruction level, Intel introduced DL Boost, a set of features that include AVX-512 Vector Neural Network Instructions (AVX-512-VNNI), part of AVX-512 Instructions [49], to accelerate CNNs algorithms, and Brain floating-point format (bfloat16) [50]. Brain floating-point format is a 16-bit format that uses a floating radix point and has a dynamic range similar to that of the 32-bit IEEE 754 single-precision floating-point format. bfloat16 is also supported by ARMv8.6-A and is included in AMD's ROCm libraries. For what concerns the ML libraries, Intel has created BigDL [51], a distributed deep learning library for DNNs algorithms acceleration on CPU clusters. There is also an Intel distribution of Caffe [52], a popular deep learning framework, targeting Intel Xeon processors.

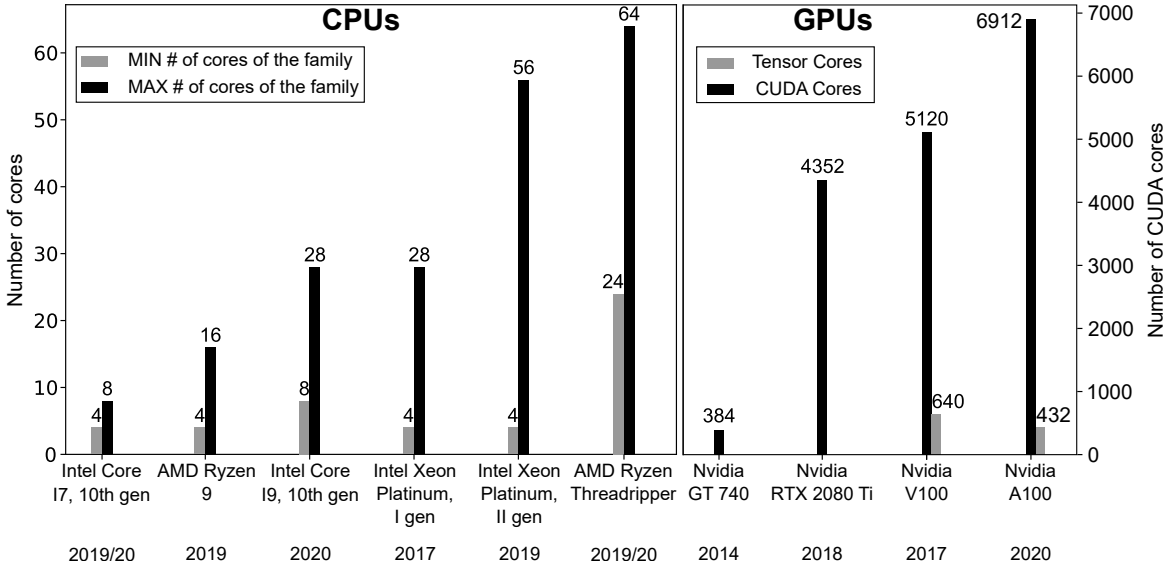


Figure 7. Comparison of the number of CPU cores and GPUs. CPUs and GPUs models have been selected for different targets, e.g., personal computers or servers, and different price ranges. For the CPUs, the gray and black lines correspond to the minimum and the maximum cores of a family, respectively. For the GPUs, the black lines represent the number of CUDA cores, and the gray line represents the Tensor cores present in the Nvidia Tesla V100 only.

GPUs are the current workhorses for DNNs' inference and especially training. They contain up to thousands of cores (see Figure 7) to work efficiently on highly-parallel algorithms. Matrix multiplications, the core operations of DNNs, belong to this class of parallel algorithms. Among the GPUs' producers, Nvidia can be considered the winner of the AI challenge. In fact, the most popular DL frameworks, such as TensorFlow [53], PyTorch [54], or Caffe [52], support execution on Nvidia GPUs through the Nvidia cuDNN library [55], a GPU-accelerated library of primitives for DNNs with highly-optimized implementations of standard layers. DL frameworks allow to describe very complex neural networks in a few lines of code and run them on GPUs without needing to know GPU programming. cuDNN is part of CUDA-X AI [56], a collection of Nvidia's GPU acceleration libraries that accelerate DL and ML.

At the hardware level, Nvidia has combined Tensor cores [57] with traditional CUDA cores in some of its platforms. Tensor cores are a new structure designed to accelerate large matrix operations and perform mixed-precision Matrix Multiply-and-Accumulate (MMAC) calculations in a single operation. The recently announced Nvidia Ampere A100 supports a new numerical format called Tensor Format (TF32) that has the range of 32-bit floating point (FP32) numbers and the precision of 16-bit floating point numbers, using a 19-bit representation. TF32 format on A100 architecture

provides a 10x performance increase compared to FP32 format on V100 architecture [58]. Moreover, the Tensor cores in A100 architecture are optimized to exploit sparsity for an additional boost (2x) of the performances (see Section 3.2.1 for an insight of sparsity in NNs).

3.2. Spatial Architectures: Fpgas and Asics

DNNs accelerators implemented on FPGAs (Field-Programmable-Gate-Arrays) and ASICs (Application-Specific-Integrated-Circuit) usually fall into the category of spatial architectures. FPGAs and ASICs differ substantially. The primary purpose of FPGAs is programmability to implement any possible design. They are relatively cost-effective with short time-to-market, and the design flow is simple. However, FPGAs can not be optimized for the various requirements of different applications, are less energy-efficient, and have lower performances than ASICs. On the contrary, ASICs need to be designed and produced for a specific application that cannot be changed over time. The design flow is consequently more complex, and the production cost is higher, but the resulting chip is highly-optimized and energy-efficient. In this section, however, no distinction will be made between ASIC and FPGA based implementations. FPGAs are, in fact, often used to prototype what will then be developed on ASICs.

A hardware accelerator for DNNs (implemented on ASIC or FPGA) typically consists of an array of PEs for computation (see Figure 8). The PEs are interconnected by a Network-on-Chip (NoC) designed to achieve the desired data movement scheme. The three levels of the memory hierarchy are the Register Files (RFs) in the PEs, that store data for inter-PE movements or accumulations, the Global Buffers (GBs), that stores enough values to feed the PEs, and the off-chip memory, usually a DRAM. As seen in the Background Section, the operations in DNNs are mostly simple Multiply-and-Accumulate (MAC) but need to be performed on a considerable amount of data. The real bottleneck in DDNs computation is memory accesses. Therefore, one of the key design issues for memory hierarchy is to reduce the DRAM accesses, since they have a high latency and energy cost. The reuse of the data stored in smaller, faster, and low-energy memories (GLB and RFs) is favored.

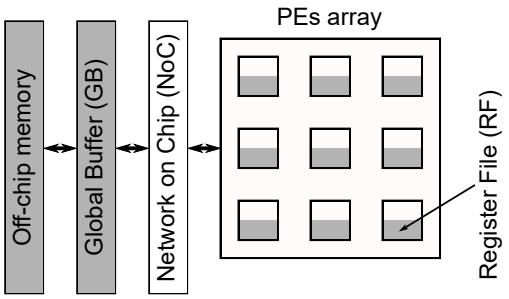


Figure 8. Typical design of an hardware accelerator for DNNs.

Given the memory problem, during the development of the first accelerators for DNNs, the focus was placed on investigating efficient dataflows, i.e., spatial and temporal mapping of operations on PEs, which would reduce the number of off-chip memory accesses by reusing the data already stored in GB and RF, as much as possible. Moreover, operations and data movement must be orchestrated to have good throughput performance as well. From these studies, various accelerators were born that exploit the different possibilities of data reuse offered by DNNs, CNNs in particular. Three kinds of data reuse are identified in Conv layers:

- Weight reuse: a kernel of weights is reused HoxWo times for each sub-grid of the input feature maps;
- Input reuse: the input feature maps are reused Co times to compute each output feature map.
- Convolutional reuse: when the weight kernel slides through the input feature maps, the sub-grids used for computation usually overlap. The input values that fall in the overlapping region are reused to compute two or more output values.

For what concerns FC layers, there is an input reuse opportunity since all input values are reused to calculate the output of each neuron.

The first accelerators developed were traditionally classified according to the type of data reuse used. In particular, accelerators are classified as follows:

- Weight Stationary (WS): the weights are stored in the RFs of the PEs and kept fixed, while the inputs are distributed coordinated with the movement of the partial sums between the PEs to obtain the correct results. These accelerators exploit weight reuse and convolutional reuse. Popular WS accelerators are [59,60].
- Output Stationary (OS): each partial sum is kept fixed in a PE, and accumulation is performed until the final sum is reached, while the weights and inputs are distributed in various ways to the PEs. These accelerators can exploit convolutional reuse. Popular OS accelerators are [61,62].
- Row Stationary (RS): this dataflow jointly maximizes the reuse of all data, i.e., inputs, weights, and partial sums. In this dataflow, the operations of a row of the convolution are mapped to the same PE. The weights are kept stationary in the PEs. For instance, Eyeriss [63] is an RS accelerator.
- No Local Reuse (NLR): This dataflow reduces the accelerator area by eliminating the RFs from the PEs and reading data only from GBs. There is no data reuse. DianNao [64] is an NLR accelerator.

In recent years, to further improve the performance and energy efficiency of accelerators, attention has been focused on new strategies. In this survey, those that have led to better results and on which the research has invested more will be discussed, namely sparsity exploitation, variable bitwidth accelerators, and reconfigurable accelerators.

3.2.1. Accelerators with Sparsity Exploitation

The exploitation of sparsity involves taking advantage of the high number of zeros present in the matrices of weights and activations, which are therefore scattered matrices. Sparsity is mainly due to two factors: given the redundancy of the weights in an NN, it is usually possible to prune, i.e., put many values to zero [65–67]. The frequent use of the ReLU function as the activation function resets all the negative values in the matrices of the activations to zero. The number of non-zero values can be reduced to 20–80% and 50–70% for weights and activations, respectively. This factor can be used to avoid multiplication, as the result of zero multiplication is known in advance, and to compress the data when stored in memory. Among the best-known sparsity compression methods, the most used are: Compressed Sparse Row (CSR), Compressed Sparse Column (CSC), Compressed Image Size (CIS), and Run Length Coding (RLC), as depicted in Figure 9. These techniques provide effective and minimal overhead when implemented in hardware. CSR and CSC compress the sparse matrix into three different arrays. The first one represents the non-zero values, the second contains the column index and row index respectively for CSR and CSC, while the third shows the number of non-null elements in the matrix. CIS is composed of an array of non-zero values and a matrix of the same size as the original one. This matrix represents the position of the values contained in the array. This representation is the most hardware friendly since it often does not require any decompression mechanism. Finally, RLC compresses the original data indicating for each value how many times it is repeated.

Accelerators that exploit sparsity have different architectures that allow adapting the computation to the sparse matrices (see Table 2 for a comparison). Cnvlutin [68] uses the CSR scheme to compress the activations but does not consider the sparsity of the weights. In Cambricon-X [69] the PEs store the compressed weights for asynchronous computation, but do not exploit activations sparsity. SCNN [24] uses the CSC scheme for both weights and activations. The values are delivered to an array of multipliers, and the resulting scattered products are summed using a dedicated interconnection mesh. Sparten [70] is based on SCNN architecture, but it improves the distribution of the operations to the multipliers to reduce the overhead. EIE [71] compresses the weights with the CSC scheme and has zero-skipping ability for null activations. Moreover, high energy savings are obtained by avoiding

the use of DRAM. Similarly, NullHop [26] applies the CIS scheme to the weights and skips the null activations. ZeNA [72] is the first zero-aware accelerator that can skip the operations with null results induced by both null weights and activations. SqueezeFlow [25] has a mathematical approach to the sparsity problem and introduces the concise convolution rules to avoid the operations with a null result. The RLC scheme is applied to the weights. SqueezeFlow supports sparse and dense models as well. Eyeriss v2 [73] also supports both sparse and dense models. It utilizes the CSC scheme to weights and activations, which are kept compressed both in the memory and computation domain. For higher flexibility, a hierarchical mesh is used for the PEs interconnections. Unique Weight CNN (UCNN) accelerator [74] proposes a generalization of the sparsity problem. Rather than exploiting only the repetition of weights with zero value, it uses the repetition of weights with any value by reusing CNN sub-computations and reducing the model size in memory.

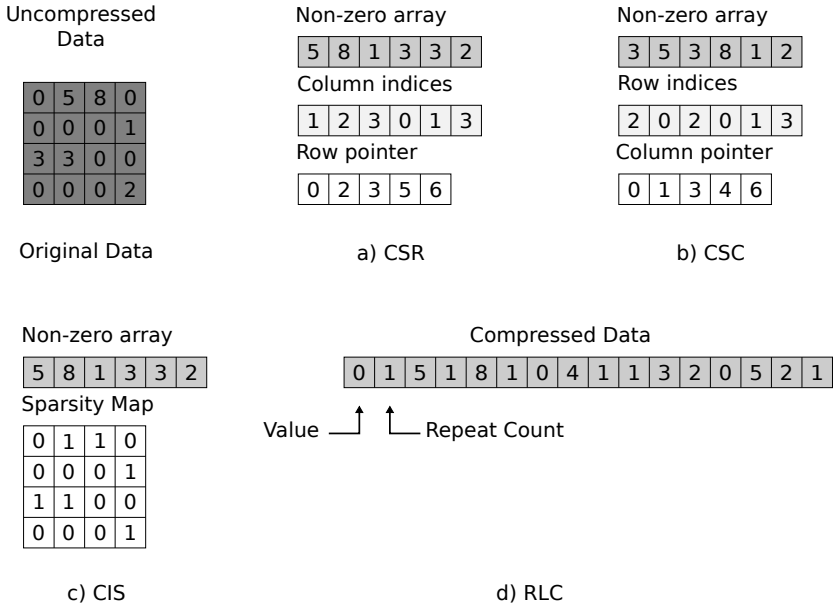


Figure 9. Comparison among different compression techniques: (**a**) Compressed Sparse Row (CSR), (**b**) Compressed Sparse Column (CSC), (**c**) Compressed Image Size (CIS), and (**d**) Run Length Coding (RLC).

Table 2. Comparison of accelerators that exploit sparsity.

Accelerator	Ref.	Contribution	Target	Year
Cnvlutin	[68]	CSR for activations	ASIC	2016
Cambricon-x	[69]	CIS for the weights	ASIC	2016
SCNN	[24]	CSC for weights and activations	ASIC	2017
Sparten	[70]	Improvement of SCNN	ASIC	2019
EIE	[71]	CSC for the weights, zero-skip for activations	ASIC	2016
NullHop	[26]	CIS for the weights, zero-skip for activations	FPGA	2018
ZeNA	[72]	Zero-skip of weights and activations	ASIC	2017
SqueezeFlow	[25]	RLC for the weights, concise convolution rules	ASIC	2019
Eyeriss v2	[73]	CSC for weights and activations, data are kept compressed during computation	ASIC	2019
UCNN	[74]	Generalizes sparsity to non-null weights	ASIC	2018

3.2.2. Variable Bitwidth Accelerators

A DNN can have over a hundred million parameters. Therefore, the memory required to store the data during computation is huge. One of the main techniques used to reduce the memory constraints is bitwidth reduction. Rather than expressing the numbers in IEEE 754 32-bit floating-point format, it is possible to represent them in fixed-point format [75], reducing the bitwidth as much as possible without affecting the accuracy significantly. This strategy allows not only to reduce the occupied memory but also to decrease the power consumption associated with computation [76,77]. It has been demonstrated that most DNNs can be inferred using 8-bit fixed-point values without accuracy degradation [78,79]. However, several studies [80,81] have shown that each layer of a DNN has a different impact on the accuracy, and it is, therefore, possible to use a fine-grained quantization where the bitwidth of the weights and activations is different in each layer.

To exploit not only the memory gain deriving from the bitwidth reduction but also the lower power consumption, hardware accelerators with flexible-bitwidth arithmetic have been developed (see Table 3 for a comparison). Stripes [82] implements variable bitwidth with bit-serial computation. The latency increases linearly with the bitwidth, but this increment can be compensated by heavier exploitation of the inherent parallelism of DNNs. Besides, multipliers, which are one of the most considerable sources of energy consumption excluding memory accesses, are no longer necessary. The bit-serial computation engine consists, in fact, of AND gates and adders only. Stripes is a hybrid architecture that fixes the bitwidth of the weights and provides flexibility for the activations. UNPU [83] has a very similar bit-serial computation engine, but the bitwidth of the activations is fixed to 16-bit while the weights have 1-bit to 16-bit flexibility. Loom [84] architecture is fully temporal since both weights and activations have variable bitwidth and are processed serially. To achieve this flexibility, Loom adopts bit-serial multiplication, that, however, requires the transposition of the inputs. For a more efficient implementation, Loom transposes the outputs rather than the inputs, but the overhead is not negligible. Bit Fusion [85] implements the flexible bitwidth spatially, with an array of PEs that are combined differently depending on the required bitwidth. In detail, the overall computation is partitioned in 2-bit $\times$ 2-bit multiplications, followed by shifted additions. BitBlade [86] is based on the Bit Fusion accelerator, but it further optimizes the architecture eliminating the shift-add logic using bitwise summation.

Table 3. Comparison of accelerators that support variable bitwidth operations.

Accelerator	Ref.	Weight Bits	Activation Bits	Features	Target	Year
Stripes	[82]	16-bit	1-bit to 16-bit	Bit-serial	ASIC	2016
UNPU	[83]	1-bit to 16-bit	16-bit	Bit-serial	ASIC	2019
Loom	[84]	1-bit to 16-bit	1-bit to 16-bit	Bit-serial	ASIC	2018
Bit Fusion	[85]	1,2,4,8,16-bit	1,2,4,8,16-bit	Spatial combination	ASIC	2018
BitBlade	[86]	1,2,4,8,16-bit	1,2,4,8,16-bit	Spatial combination	ASIC	2019

3.2.3. Reconfigurable Accelerators

Given the increasing interest in deep learning, a wide variety of models with very different features and layers have emerged, as seen in Section 2.3. However, the majority of ASIC/FPGA accelerators for DNNs are designed and optimized to support only one type of dataflow. It can be complex to map different layers on these accelerators equally efficiently. To allow for more widespread and mass deployment of ASIC/FPGA accelerators, flexible and easily reconfigurable designs are required to support different types of layers and models.

FlexFlow [87] and DNA [88] are two accelerators that support a flexible dataflow to exploit the different types of reuse and parallelism of Conv layers. However, they only target CNNs. On the other hand, MPNA [89] supports dedicated PE array units for Conv and FC layers. In [90], an ASIC

reconfigurable processor targeting hybrid NNs, i.e., networks with different layers, is presented. The PEs are organized into two 16x16 arrays, and they are divided into general PEs and super PEs. The former supports Conv and FC layers, while the latter supports Pooling layers, RNN layers, and non-linear activation functions. Each PE has two 8-bit multipliers that can be used separately or jointly to form a 16-bit multiplier, allowing for a variable bitwidth. The arrays can be arbitrarily partitioned into sub-arrays to process multiple layers or networks in parallel. Project Brainwave [91] is an FPGA platform used in Microsoft servers for real-time AI. The core of Project Brainwave is the NPU, a spatially distributed microarchitecture with up to 96 thousand MACs. The architecture achieves flexibility working with vectors and matrices as data-types, using efficient matrix-vector multipliers and multifunction units that can be programmed to implement a function chosen in a broad set. For easy programming, the architecture has a custom SIMD Instruction Set. MAERI [92] has a set of PEs, each containing a Register File and a multiplier. The reconfigurability is obtained with the interconnections. The activations and weight are delivered to the PEs with a distribution tree fully configurable. Similarly, the outputs of the multipliers are collected by a configurable adder tree. SIGMA [93] introduces the Flexible Dot Product Engine (Flex-DPE), which has a structure similar to MAERI. In the Flex-DPE, the multipliers are in fact arranged in a 1D structure. Thanks to highly flexible distribution and reduction networks, multiple variable-sized dot-products can be performed in parallel. Thanks to the flexible distribution network, SIGMA also supports the acceleration of sparse networks. To maximize energy efficiency, DNPU [94] proposes a heterogeneous architecture with two processors, one optimized for Conv layers and CNNs, the other targeting FC layers and Recurrent Neural Networks. Cerebras has recently announced the Cerebras Wafer Scale Engine (WSE), the largest chip ever built and specialized for DL computing only. The WSE has a huge numbers of flexible cores that support general operations (e.g., arithmetic, logical, load/store operations) but are optimized in particular for tensor operations. The memory, in the order of gygabytes, is on-chip and distributed.

4. Memory

Optimizing the hardware integration of basic operations is certainly promising from an energy point of view, but it should be considered that inefficient memory management could nullify all this for two main reasons. First, each DNN is composed of billions of multiply and accumulate (MAC) operations between activations and weights. Thus each MAC requires three memory accesses: two for the factors and one for the writeback of the product. Second, off-chip DRAM access is three orders of magnitude larger than a simple floating-point adder [71]. Therefore, these two reasons become even more marked considering the current DNN trend, which goes towards increasingly complex models, where the size is scaled up for better accuracy. In this scenario, the reuse of data through an efficient dataflow and conscious use of memory are the basis of the co-design of efficient architectures.

Generally, a hardware accelerator for DNN presents a hierarchical memory structure composed of a few levels as shown in Figure 10. The outer one is typically represented by a DRAM in which all network weights and activations of the current layer are saved. Part of these data is periodically moved to a lower level, close to processing elements. This intermediate layer consists of three SRAM buffers: one for the input activations, one for the weights, and one for the output activations. Typically, the activation and weight buffers are separated since several different bit widths could be used. The lowest level instead, is represented by elements of local memory as registers located within the PEs. These are responsible for the data reuse chosen by the dataflow policy. Moreover, additional memory elements can be inserted depending on the specific design and data flow. Exploiting the memory hierarchy, there is no direct communication between the accelerator and the CPU. The CPU loads the data into the DRAM and, when present, programs the register file. Each location of the register file corresponds to a specific parameter of a DNN layer, i.e., input size, output size, number of filters, filter size. The control unit that drives the accelerator, relies on the data contained in the register file to organize the loops related to the dataflow and to generate memory addresses to move the data from the DRAM to the buffers.

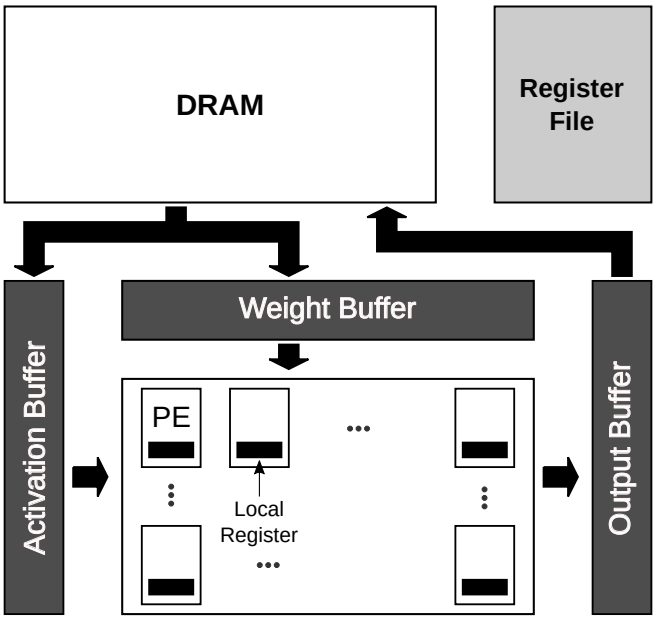


Figure 10. Generic memory hierarchy for a DNN accelerator.

Since the DRAM is the most power-hungry element of the hierarchy, many different techniques have been proposed to reduce the number of DRAM accesses. For example, Stoutchinin et al. [95] proposed an analytical model for the optimization of the memory bandwidth in CNN loop-nest. They showed that with minor interface changes, it is possible to reduce the memory bandwidth of a factor 14×. Li et al. [96] instead, proposed an adaptive layer tiling able to minimize the off-chip DRAM accesses called SmartShuttle. This model can exploit different data reuse paradigms, switching from one to the other in order to better fit tiling over several layer sizes. Although the previous works reduce the memory accesses, they do not consider both the latency and energy per access. Putra et al. [97] tried to optimize these two factors for further performance enhancement.

The memory hierarchy described above is a generic structure used by most parts of the accelerators. However, in other circumstances, specific designs, optimized for the target application, have been opted for, where hardware key elements have been removed. This is the case of ShiDianNo [61], where the whole accelerator has been embedded inside a phone camera sensor by eliminating the need for an intermediate DRAM to store the pictures data. The absence of the memory coupled with an efficient data pattern access leads to a 60× energy saving compared to the previous architecture DianNao [64].

It is therefore clear that memory is one of the most sensitive points of the entire architecture and that for a low-power system, its size and bandwidth must be correctly sized. Wei et al. [98] proposed a framework for FPGA able to allocate efficiently the on-chip memory exploiting the layer diversity and the lifespan of the intermediate buffers.

Although sparsity and sparse models were primarily designed to avoid unnecessary operations between null activations and weights, they indirectly reduce both the memory accesses and the memory size. Model pruning, coupled with the Rectified Linear Unit (ReLU), produces respectively null weights and null activation. The sparse matrices can be compressed, requiring less memory. Moreover, removing the useless operations (multiply by zero value) sharply reduces the memory bandwidth required, speeding up the execution of the DNN, as mentioned in Section 3.2.1.

Logic-in-memory (LIM) is another method of reducing or even eliminate access to memory. This technique involves the integration of part of the computational logic directly into the memory to work on the data without having to extract them as in the case proposed by Khwa et al. [99]. However, this approach can be implemented only in some cases (for example binary networks) and is not feasible with complex state-of-the-art networks [100–102].

5. Hardware Metrics and Comparison

Comparisons between different hardware platforms or accelerators are not always straightforward as designers often present performance depending on the target application. For example, a GPU for server applications is difficult to compare with an accelerator based on ASIC or FPGA for embedded applications. In fact, the power envelopes and the amount of data to process will be the opposite. However, there are some standard metrics that researchers rely on to define the performance of the hardware that refers mainly to the area, power consumption, and the number of operations per second.

Area. The area, generally expressed in squared millimeters or squared micrometers, represents the portion of silicon required to contain all the necessary logic. It strictly depends on the technological node used during the hardware synthesis process and the size of the on-chip memory.

Power. The power consumption comes from the device's power envelope and the application for which it was designed. Battery-powered devices, for example, require extremely efficient accelerators that can overcome the pj per MAC barrier. This latter is a metric widely used to express the efficiency of the computational side of architecture. However, power consumption must also include that resulting from both on-chip and off-chip memories as they are the primary source.

Throughput. Throughput defines how often an accelerator can accomplish a complete convolution, or rather a complete inference. Throughput and latency are derived from the device's operating frequency coupled with the memory bandwidth. Usually, this metric is expressed as billions of operations per second (Gop/s) or as billions of Macs per second (GMAC/s). Considering that a MAC consists of two operations (multiplication and sum), the ratio between Gop/s and GMAC/s is 2 to 1.

There are other metrics that best define an accelerator, such as its flexibility and scalability to new network models or variable bitwidth. As mentioned above, accelerators tend to be very application-dependent, so very often, comparisons between them are complicated and should be evaluated based on datasets or common models.

A comparison between many of the aforementioned models is provided in Table 4, where different hardware platforms and different accelerator models are presented. As expected, general purpose architectures have greater area and power consumption than special purpose architectures, as they are not optimized for a specific application. For each architecture, the main hardware metrics discussed above are reported.

Table 4. Comparison between accelerations implemented on different hardware platforms.

Name	Platform	Reference	Technology (nm)	Area (mm^2)	Power (mW)	Gop/s
Cambricon-X	ASIC	[69]	65	6.38	954	544
SCNN	ASIC	[24]	16	7.9	-	2000
EIE	ASIC	[71]	45	40	600	3000
NullHop	ASIC	[26]	28	8.1	155	450
NullHop	FPGA Xilinx Zynq 7100	[26]	28	-	2300	17.2
SqueezeFlow	ASIC	[25]	65	4.80	536	-
UNPU	ASIC	[83]	65	16	297	345–7000
FlexFlow	ASIC	[87]	65	3.89	1000	420
DNA	ASIC	[88]	65	16	479	194
SIGMA	ASIC	[93]	28	65.10	22,300	10,800
DNPU	ASIC	[94]	65	16	279	300–1200
Nvidia V100	GPU	[57]	12	815	250,000	31,400
Nvidia A100	GPU	[58]	7	826	400,000	78,000
Intel Xeon Platinum 9282	CPU	-	14	-	400,000	3200
AMD Ryzen Threadripper 3970x	CPU	-	7	-	280,000	1859

6. Conclusions

The importance and use of Deep Learning have grown dramatically over the past decade. These algorithms have been able to go beyond human accuracy in a short time. Besides, thanks to their versatility, they can be used in many applications, always with cutting edge results. However, their high effectiveness comes from a tremendous algorithmic complexity that results in a need for computational power. In this scenario, researchers have developed hardware platforms for the acceleration of such algorithms. Considering the usual trend of electronics moving towards mobile devices and IoT nodes, these hardware platforms will have to follow a low-power approach with an efficiency-oriented design.

Therefore, it is essential to consider critical parts of the system, such as memory and accesses to it, from the initial stages of design. As a result, many dataflows and techniques have been developed to optimize all critical aspects of accelerators. For example, to reduce the impact of memory on power consumption, it is better to use spatial architectures that distribute part of the store elements directly in the processing elements to enable data reuse.

All these paradigms have been refined over the years. With their progress, several surveys have also been produced to collect and explain all the techniques, providing a tutorial for designers. This paper aims to provide an up-to-date survey by mainly focusing on state-of-the-art architectures of the last three years. The contribution of this work is the collection and comparison of the latest architectures that have not been covered in prior surveys.

Appendix A

Table A1. List of Acronyms.

AI	Artificial Intelligence	GPU	Graphic Processing Unit
ALU	Arithmetic Logic Unit	IFM	Input Feature Map
ANN	Artificial Neural Network	IoT	Internet of Things
ASIC	Application Specific Integrated Circuit	LIM	Logic in Memory
BLAS	Basic Linear Algebra Subroutines	MAC	Multiply-and-Accumulate
BN	Batch Normalization	ML	Machine Learning
CIS	Compressed Image Size	MMAC	Matrix Multiply-and-Accumulate
CNN	Convolution Neural Networks	NLP	Natural Language Processing
Conv	Convolutional	NLR	No Local Reuse
CPU	Central Processing Unit	NN	Neural Network
CSC	Compressed Sparse Column	OFM	Output Feature Map
CSR	Compressed Sparse Row	OS	Output Stationary
DL	Deep Learning	PE	Processing Element
DNN	Deep Neural Network	ReLU	Rectified Linear Unit
DRAM	Dynamic Random Access Memory	RF	Register File
FC	Fully Connected	RLC	Run Length Coding
FFT	Fast Fourier Transform	RS	Row Stationary
FM	Feature Map	SIMD	Single-Instruction Multiple-Data
FPGA	Field Programmable Gate Array	SIMT	Single-Instruction Multiple-Threads
GB	Global Buffer	VLSI	Very Large Scale Integration
GeMM	General Matrix Multiplication	WS	Weight Stationary

Author Contributions: Investigation, M.C. and B.B.; resources, M.C. and B.B.; writing–original draft preparation, M.C. and B.B.; writing–review and editing, M.C. and B.B.; visualization, A.M.; supervision, M.S., G.M. and M.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Acknowledgments: This work has been partially supported by the Doctoral College Resilient Embedded Systems which is run jointly by TU Wien's Faculty of Informatics and FH-Technikum Wien.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. LeCun, Y.; Bengio, Y.; Hinton, G. Deep Learning. *Nature* **2015**, *521*, 436–444, doi:10.1038/nature14539.
2. Zanc, R.; Cioara, T.; Anghel, I. Forecasting Financial Markets using Deep Learning. In Proceedings of the 2019 IEEE 15th International Conference on Intelligent Computer Communication and Processing (ICCP), Cluj-Napoca, Romania, 5–7 September 2019; pp. 459–466.
3. Ying, J.J.; Huang, P.; Chang, C.; Yang, D. A preliminary study on deep learning for predicting social insurance payment behavior. In Proceedings of the 2017 IEEE International Conference on Big Data (Big Data), Boston, MA, USA, 11–14 December 2017; pp. 1866–1875.
4. Ha, V.; Lu, D.; Choi, G.S.; Nguyen, H.; Yoon, B. Improving Credit Risk Prediction in Online Peer-to-Peer (P2P) Lending Using Feature selection with Deep learning. In Proceedings of the 2019 21st International Conference on Advanced Communication Technology (ICACT), PyeongChang Kwangwoon_Do, Korea, 17–20 February 2019; pp. 511–515.
5. Arslan, A.K.; Yaşar, Ş.; Çolak, C. An Intelligent System for the Classification of Lung Cancer Based on Deep Learning Strategy. In Proceedings of the 2019 International Artificial Intelligence and Data Processing Symposium (IDAP), Malatya, Turkey, 21–22 September 2019; pp. 1–4.
6. Mohsen, H.; El-Dahshan, E.S.; El-Horbarty, E.S.; M.Salem, A.B. Classification using Deep Learning Neural Networks for Brain Tumors. *Future Comput. Inform. J.* **2018**, *3*, 68–71, doi:10.1016/j.fcij.2017.12.001.
7. Barata, C.; Marques, J.S. Deep Learning For Skin Cancer Diagnosis With Hierarchical Architectures. In Proceedings of the 2019 IEEE 16th International Symposium on Biomedical Imaging (ISBI 2019), Venice, Italy, 8–11 April 2019; pp. 841–845.
8. Grigorescu, S.; Trasnea, B.; Cocias, T.; Macesanu, G. A survey of deep learning techniques for autonomous driving. *J. Field Robot.* **2020**, *37*, 362–386, doi:10.1002/rob.21918.
9. Palossi, D.; Loquercio, A.; Conti, F.; Flamand, E.; Scaramuzza, D.; Benini, L. Ultra Low Power Deep-Learning-powered Autonomous Nano Drones. *arXiv* **2018**, arXiv:1805.01831.
10. Zhang, D.; Liu, S. Top-Down Saliency Object Localization Based on Deep-Learned Features. In Proceedings of the 2018 11th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI), Beijing, China, 13–15 October 2018; pp. 1–9.
11. Minaee, S.; Boykov, Y.; Porikli, F.; Plaza, A.; Kehtarnavaz, N.; Terzopoulos, D. Image Segmentation Using Deep Learning: A Survey. *arXiv* **2020**, arXiv:2001.05566.
12. Kaskavalci, H.C.; Gören, S. A Deep Learning Based Distributed Smart Surveillance Architecture using Edge and Cloud Computing. In Proceedings of the 2019 International Conference on Deep Learning and Machine Learning in Emerging Applications (Deep-ML), Istanbul, Turkey, 26–28 August 2019; pp. 1–6.
13. Capra, M.; Peloso, R.; Masera, G.; Ruo Roch, M.; Martina, M. Edge Computing: A Survey On the Hardware Requirements in the Internet of Things World. *Future Internet* **2019**, *11*, 100, doi:10.3390/fi11040100.
14. Shafique, M.; Theocharides, T.; Bouganis, C.; Hanif, M.A.; Khalid, F.; Hafız, R.; Rehman, S. An overview of next-generation architectures for machine learning: Roadmap, opportunities and challenges in the IoT era. In Proceedings of the 2018 Design, Automation Test in Europe Conference Exhibition (DATE), Dresden, Germany, 19–23 March 2018; pp. 827–832.
15. Marchisio, A.; Hanif, M.A.; Khalid, F.; Plastiras, G.; Kyrkou, C.; Theocharides, T.; Shafique, M. Deep Learning for Edge Computing: Current Trends, Cross-Layer Optimizations, and Open Research Challenges. In Proceedings of the 2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Miami, FL, USA, 15–17 July 2019; pp. 553–559.
16. Zhang, J.J.; Liu, K.; Khalid, F.; Hanif, M.A.; Rehman, S.; Theocharides, T.; Artussi, A.; Shafique, M.; Garg, S. Building Robust Machine Learning Systems: Current Progress, Research Challenges, and Opportunities. In Proceedings of the 56th Annual Design Automation Conference, Las Vegas, NV, USA, 2–6 June 2019.
17. Shafique, M.; Naseer, M.; Theocharides, T.; Kyrkou, C.; Mutlu, O.; Orosa, L.; Choi, J. Robust Machine Learning Systems: Challenges,Current Trends, Perspectives, and the Road Ahead. *IEEE Des. Test* **2020**, *37*, 30–57.
18. Sze, V.; Chen, Y.; Yang, T.; Emer, J.S. Efficient Processing of Deep Neural Networks: A Tutorial and Survey. *Proc. IEEE* **2017**, *105*, 2295–2329, doi:10.1109/JPROC.2017.2761740.

19. Deng, B.L.; Li, G.; Han, S.; Shi, L.; Xie, Y. Model Compression and Hardware Acceleration for Neural Networks: A Comprehensive Survey. *Proc. IEEE* **2020**, *108*, 485–532.
20. Schuman, C.D.; Potok, T.E.; Patton, R.M.; Birdwell, J.D.; Dean, M.E.; Rose, G.S.; Plank, J.S. A Survey of Neuromorphic Computing and Neural Networks in Hardware. *arXiv* **2017**, arXiv:1705.06963.
21. Chen, Y.; Xie, Y.; Song, L.; Chen, F.; Tang, T. A Survey of Accelerator Architectures for Deep Neural Networks. *Engineering* **2020**, *6*, 264–274, doi:10.1016/j.eng.2020.01.007.
22. Han, S.; Mao, H.; Dally, W.J. Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding. *arXiv* **2015**, arXiv:1510.00149.
23. Cai, H.; Gan, C.; Han, S. Once for All: Train One Network and Specialize it for Efficient Deployment. *arXiv* **2019**, arXiv:1908.09791.
24. Parashar, A.; Rhu, M.; Mukkara, A.; Puglielli, A.; Venkatesan, R.; Khailany, B.; Emer, J.; Keckler, S.W.; Dally, W.J. SCNN: An accelerator for compressed-sparse convolutional neural networks. In Proceedings of the 2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA), Toronto, ON, Canada, 24–28 June 2017; pp. 27–40; doi:10.1145/3079856.3080254.
25. Li, J.; Jiang, S.; Gong, S.; Wu, J.; Yan, J.; Yan, G.; Li, X. SqueezeFlow: A Sparse CNN Accelerator Exploiting Concise Convolution Rules. *IEEE Trans. Comput.* **2019**, *68*, 1663–1677, doi:10.1109/TC.2019.2924215.
26. Aimar, A.; Mostafa, H.; Calabrese, E.; Rios-Navarro, A.; Tapiador-Morales, R.; Lungu, I.; Milde, M.B.; Corradi, F.; Linares-Barranco, A.; Liu, S.; et al. NullHop: A Flexible Convolutional Neural Network Accelerator Based on Sparse Representations of Feature Maps. *IEEE Trans. Neural Netw. Learn. Syst.* **2019**, *30*, 644–656, doi:10.1109/TNNLS.2018.2852335.
27. Bengio, Y. Learning Deep Architectures for AI. *Found. Trends Mach. Learn.* **2009**, *2*, 1–127, doi:10.1561/2200000006.
28. Lecun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-Based Learning Applied to Document Recognition. *Proc. IEEE* **1998**, *86*, 2278 – 2324, doi:10.1109/5.726791.
29. Qian, N. On the momentum term in gradient descent learning algorithms. *Neural Netw.* **1999**, *12*, 145–151, doi:10.1016/S0893-6080(98)00116-6.
30. Kingma, D.; Ba, J. Adam: A Method for Stochastic Optimization. In Proceedings of the International Conference on Learning Representations, Banff, AB, Canada, 14–16 April 2014.
31. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. ImageNet Classification with Deep Convolutional Neural Networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems—Volume 1*; Curran Associates Inc.: Red Hook, NY, USA, 2012; pp. 1097–1105.
32. Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M.; et al. ImageNet Large Scale Visual Recognition Challenge. *Int. J. Comput. Vis.* **2015**, *115*, 211–252, doi:10.1007/s11263-015-0816-y.
33. Simonyan, K.; Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv* **2014**, arXiv:1409.1556.
34. Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; Rabinovich, A. Going deeper with convolutions. In Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, USA, 7–12 June 2015; pp. 1–9.
35. Szegedy, C.; Vanhoucke, V.; Ioffe, S.; Shlens, J.; Wojna, Z. Rethinking the Inception Architecture for Computer Vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, 27–30 June 2016*; IEEE Computer Society: Washington, DC, USA, 2016; pp. 2818–2826, doi:10.1109/CVPR.2016.308.
36. Szegedy, C.; Ioffe, S.; Vanhoucke, V.; Alemi, A.A. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. In Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, San Francisco, CA, USA, 4–9 February 2017; Singh, S.P., Markovitch, S., Eds.; AAAI Press: Palo Alto, CA, USA, 2017; pp. 4278–4284.
37. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep Residual Learning for Image Recognition. In Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, 27–30 June 2016; pp. 770–778.

38. Chollet, F. Xception: Deep Learning with Depthwise Separable Convolutions. In Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21–26 July 2017; pp. 1800–1807.
39. Xie, S.; Girshick, R.; Dollár, P.; Tu, Z.; He, K. Aggregated Residual Transformations for Deep Neural Networks. In Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21–26 July 2017; pp. 5987–5995, doi:10.1109/CVPR.2017.634.
40. Huang, G.; Liu, Z.; Weinberger, K.Q. Densely Connected Convolutional Networks. In Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Honolulu, HI, USA, 21–26 July 2017; pp. 2261–2269.
41. Hu, J.; Shen, L.; Sun, G. Squeeze-and-Excitation Networks. In Proceedings of the 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–22 June 2018; pp. 7132–7141, doi:10.1109/CVPR.2018.00745.
42. Zoph, B.; Vasudevan, V.; Shlens, J.; Le, Q.V. Learning Transferable Architectures for Scalable Image Recognition. In Proceedings of the 2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, 18–22 June 2018; IEEE Computer Society: Washington, DC, USA, 2018; pp. 8697–8710, doi:10.1109/CVPR.2018.00907.
43. Devlin, J.; Chang, M.; Lee, K.; Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, 2–7 June 2019, Volume 1 (Long and Short Papers)*; Burstein, J., Doran, C., Solorio, T., Eds.; Association for Computational Linguistics: Stroudsburg , PA, USA, 2019; pp. 4171–4186.
44. Shoeybi, M.; Patwary, M.; Puri, R.; LeGresley, P.; Casper, J.; Catanzaro, B. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. *arXiv* **2019**, arXiv:1909.08053.
45. Rodriguez, A.; Segal, E.; Meiri, E.; Fomenko, E.; Kim, Y.; Shen, H.; Ziv, B. Lower Numerical Precision Deep Learning Inference and Training. *Intel White Paper* **2018**, *3*, 1–19.
46. Chellapilla, K.; Puri, S.; Simard, P. High Performance Convolutional Neural Networks for Document Processing. In *Tenth International Workshop on Frontiers in Handwriting Recognition*; Lorette, G., Ed.; Université de Rennes 1, Suvisoft: La Baule, France, 2006. Available online: http://www.suvisoft.com (accessed on 7 June 2020).
47. Vasudevan, A.; Anderson, A.; Gregg, D. Parallel Multi Channel convolution using General Matrix Multiplication. In Proceedings of the 2017 IEEE 28th International Conference on Application-specific Systems, Architectures and Processors (ASAP), Seattle, WA, USA, 10–12 July 2017; pp. 19–24.
48. Mathieu, M.; Henaff, M.; LeCun, Y. Fast Training of Convolutional Networks through FFTs. *arXiv* **2013**, arXiv:1312.5851.
49. James, R. Intel AVX-512 Instructions. 2017. Available online: https://software.intel.com/content/www/cn/zh/develop/articles/intel-avx-512-instructions.html (accessed on 6 June 2020).
50. bfloat16—Hardware Numerics Definition. 2018. Available online: https://software.intel.com/content/www/us/en/develop/download/bfloat16-hardware-numerics-definition.html (accessed on 6 June 2020).
51. Gogar, S.L. BigDL—Scale-out Deep Learning on Apache Spark* Cluster. 2017. Available online: https://software.intel.com/content/www/us/en/develop/articles/bigdl-scale-out-deep-learning-on-apache-spark-cluster.html (accessed on 6 June 2020).
52. Jia, Y.; Shelhamer, E.; Donahue, J.; Karayev, S.; Long, J.; Girshick, R.B.; Guadarrama, S.; Darrell, T. Caffe: Convolutional Architecture for Fast Feature Embedding. In Proceedings of the ACM International Conference on Multimedia, MM'14, Orlando, FL, USA, 3–7 November 2014; Hua, K.A., Rui, Y., Steinmetz, R., Hanjalic, A., Natsev, A., Zhu, W., Eds.; ACM: New York, NY, USA, 2014; pp. 675–678, doi:10.1145/2647868.2654889.
53. Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G.S.; Davis, A.; Dean, J.; Devin, M.; et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015. Available online: tensorflow.org (accessed on 6 June 2020).
54. Paszke, A.; Gross, S.; Chintala, S.; Chanan, G.; Yang, E.; DeVito, Z.; Lin, Z.; Desmaison, A.; Antiga, L.; Lerer, A. Automatic Differentiation in PyTorch. In Proceedings of the NIPS 2017 Workshop on Autodiff, long Beach, CA, 4-9 December 2017.

55. Chetlur, S.; Woolley, C.; Vandermersch, P.; Cohen, J.; Tran, J.; Catanzaro, B.; Shelhamer, E. cuDNN: Efficient Primitives for Deep Learning. *arXiv* **2014**, arXiv:1410.0759.
56. Available online: https://developer.nvidia.com/gpu-accelerated-libraries (accessed on 6 June 2020).
57. NVIDIA TESLA V100 GPU ARCHITECTURE. 2017. Available online: https://images.nvidia.com/content/technologies/volta/pdf/437317-Volta-V100-DS-NV-US-WEB.pdf (accessed on 6 June 2020).
58. NVIDIA A100 Tensor Core GPU Architecture. 2020. Available online: https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/nvidia-ampere-architecture-whitepaper.pdf (accessed on 6 June 2020).
59. Gokhale, V.; Jin, J.; Dundar, A.; Martini, B.; Culurciello, E. A 240 G-ops/s Mobile Coprocessor for Deep Neural Networks. In Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops, Columbus, OH, USA, 23–28 June 2014; pp. 696–701, doi:10.1109/CVPRW.2014.106.
60. Jouppi, N.P.; Young, C.; Patil, N.; Patterson, D.; Agrawal, G.; Bajwa, R.; Bates, S.; Bhatia, S.; Boden, N.; Borchers, A.; et al.. In-Datacenter Performance Analysis of a Tensor Processing Unit. *SIGARCH Comput. Archit. News* **2017**, *45*, 1–12, doi:10.1145/3140659.3080246.
61. Du, Z.; Fasthuber, R.; Chen, T.; Ienne, P.; Li, L.; Luo, T.; Feng, X.; Chen, Y.; Temam, O. ShiDianNao: Shifting vision processing closer to the sensor. In Proceedings of the 2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA), Portland, OR, USA, 13–17 June 2015; pp. 92–104, doi:10.1145/2749469.2750389.
62. Cavigelli, L.; Benini, L. Origami: A 803-GOp/s/W Convolutional Network Accelerator. *IEEE Trans. Circuits Syst. Video Technol.* **2017**, *27*, 2461–2475, doi:10.1109/TCSVT.2016.2592330.
63. Chen, Y.; Krishna, T.; Emer, J.S.; Sze, V. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. *IEEE J. Solid-State Circuits* **2017**, *52*, 127–138, doi:10.1109/JSSC.2016.2616357.
64. Chen, T.; Du, Z.; Sun, N.; Wang, J.; Wu, C.; Chen, Y.; Temam, O. DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning. *Int. Conf. Archit. Support Program. Lang. Oper. Syst.* **2014**, *49*, 269–284, doi:10.1145/2541940.2541967.
65. Han, S.; Pool, J.; Tran, J.; Dally, W.J. Learning Both Weights and Connections for Efficient Neural Networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems—Volume 1*; MIT Press: Cambridge, MA, USA, 2015; pp. 1135–1143.
66. Srinivas, S.; Babu, R.V. Data-free Parameter Pruning for Deep Neural Networks. In *Proceedings of the British Machine Vision Conference 2015, BMVC 2015, Swansea, UK, 7–10 September 2015*; Xie, X., Jones, M.W., Tam, G.K.L., Eds.; BMVA Press, South Road, Durham, 2015; pp. 31.1–31.12, doi:10.5244/C.29.31.
67. Marchisio, A.; Hanif, M.A.; Martina, M.; Shafique, M. PruNet: Class-Blind Pruning Method For Deep Neural Networks. In Proceedings of the 2018 International Joint Conference on Neural Networks (IJCNN), Rio de Janeiro, Brazil, 8–13 July 2018; pp. 1–8.
68. Albericio, J.; Judd, P.; Hetherington, T.; Aamodt, T.; Jerger, N.E.; Moshovos, A. Cnvlutin: Ineffectual-Neuron-Free Deep Neural Network Computing. In Proceedings of the 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA), Seoul, Korea, 18–22 June 2016; pp. 1–13; doi:10.1109/ISCA.2016.11.
69. Zhang, S.; Du, Z.; Zhang, L.; Lan, H.; Liu, S.; Li, L.; Guo, Q.; Chen, T.; Chen, Y. Cambricon-X: An accelerator for sparse neural networks. In Proceedings of the 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Taipei, Taiwan, 15–19 October 2016; pp. 1–12; doi:10.1109/MICRO.2016.7783723.
70. Gondimalla, A.; Chesnut, N.; Thottethodi, M.; Vijaykumar, T.N. SparTen: A Sparse Tensor Accelerator for Convolutional Neural Networks. In Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, Columbus, OH, USA, 12–16 October 2019; Association for Computing Machinery: New York, NY, USA, 2019; pp. 151–165; doi:10.1145/3352460.3358291.
71. Han, S.; Liu, X.; Mao, H.; Pu, J.; Pedram, A.; Horowitz, M.A.; Dally, W.J. EIE: Efficient Inference Engine on Compressed Deep Neural Network. In Proceedings of the 43rd ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2016, Seoul, Korea, 18–22 June 2016; IEEE Computer Society: Washington, DC, USA, 2016; pp. 243–254; doi:10.1109/ISCA.2016.30.
72. Kim, D.; Ahn, J.; Yoo, S. ZeNA: Zero-Aware Neural Network Accelerator. *IEEE Des. Test* **2018**, *35*, 39–46, doi:10.1109/MDAT.2017.2741463.
73. Chen, Y.; Yang, T.; Emer, J.; Sze, V. Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices. *IEEE J. Emerg. Sel. Top. Circuits Syst.* **2019**, *9*, 292–308.

74. Hegde, K.; Yu, J.; Agrawal, R.; Yan, M.; Pellauer, M.; Fletcher, C. UCNN: Exploiting Computational Reuse in Deep Neural Networks via Weight Repetition. In Proceedings of the 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA), Angeles, CA, USA, 1–6 June 2018; pp. 674–687.
75. Granas, A.; Dugundji, J. *Fixed Point Theory*; Springer: Berlin/Heidelberg, Germany, 2003.
76. Horowitz, M. 1.1 Computing's energy problem (and what we can do about it). In Proceedings of the 2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC), San Francisco, CA, USA, 9–13 February 2014; pp. 10–14; doi:10.1109/ISSCC.2014.6757323.
77. Hanif, M.A.; Marchisio, A.; Arif, T.; Hafiz, R.; Rehman, S.; Shafique, M. X-DNNs: Systematic Cross-Layer Approximations for Energy-Efficient Deep Neural Networks. *J. Low Power Electron.* **2018**, *14*, 520-534.
78. Gysel, P.; Pimentel, J.; Motamedi, M.; Ghiasi, S. Ristretto: A Framework for Empirical Study of Resource-Efficient Inference in Convolutional Neural Networks. *IEEE Trans. Neural Netw. Learn. Syst.* **2018**, *29*, 5784–5789.
79. Jacob, B.; Kligys, S.; Chen, B.; Zhu, M.; Tang, M.; Howard, A.; Adam, H.; Kalenichenko, D. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 2704–2713; doi:10.1109/CVPR.2018.00286.
80. Sankaradas, M.; Jakkula, V.; Cadambi, S.; Chakradhar, S.; Durdanovic, I.; Cosatto, E.; Graf, H.P. A Massively Parallel Coprocessor for Convolutional Neural Networks. In Proceedings of the 2009 20th IEEE International Conference on Application-specific Systems, Architectures and Processors, Boston, MA, USA, 7–9 July 2009; pp. 53–60; doi:10.1109/ASAP.2009.25.
81. Sakr, C.; Shanbhag, N. Per-Tensor Fixed-Point Quantization of the Back-Propagation Algorithm. *arXiv* **2019**, arXiv:1812.11732.
82. Judd, P.; Albericio, J.; Moshovos, A. Stripes: Bit-Serial Deep Neural Network Computing. *IEEE Comput. Archit. Lett.* **2017**, *16*, 80–83.
83. Lee, J.; Kim, C.; Kang, S.; Shin, D.; Kim, S.; Yoo, H. UNPU: An Energy-Efficient Deep Neural Network Accelerator With Fully Variable Weight Bit Precision. *IEEE J. Solid-State Circuits* **2019**, *54*, 173–185.
84. Sharify, S.; Lascorz, A.D.; Siu, K.; Judd, P.; Moshovos, A. Loom: Exploiting Weight and Activation Precisions to Accelerate Convolutional Neural Networks. In Proceedings of the 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC), San Francisco, CA, USA, 24–28 June 2018; pp. 1–6.
85. Sharma, H.; Park, J.; Suda, N.; Lai, L.; Chau, B.; Chandra, V.; Esmaeilzadeh, H. Bit Fusion: Bit-Level Dynamically Composable Architecture for Accelerating Deep Neural Network. In Proceedings of the 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA), Los Angeles, CA, USA, 1–6 June 2018; pp. 764–775.
86. Ryu, S.; Kim, H.; Yi, W.; Kim, J. BitBlade: Area and Energy-Efficient Precision-Scalable Neural Network Accelerator with Bitwise Summation. In Proceedings of the 2019 56th ACM/IEEE Design Automation Conference (DAC), Las Vegas, NV, USA, 2–6 June 2019; pp. 1–6.
87. Lu, W.; Yan, G.; Li, J.; Gong, S.; Han, Y.; Li, X. FlexFlow: A Flexible Dataflow Accelerator Architecture for Convolutional Neural Networks. In Proceedings of the 2017 IEEE International Symposium on High Performance Computer Architecture (HPCA), Austin, TX, USA, 4–8 February 2017; pp. 553–564.
88. Tu, F.; Yin, S.; Ouyang, P.; Tang, S.; Liu, L.; Wei, S. Deep Convolutional Neural Network Architecture With Reconfigurable Computation Patterns. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2017**, *25*, 2220–2233.
89. Hanif, M.A.; Putra, R.V.W.; Tanvir, M.; Hafiz, R.; Rehman, S.; Shafique, M. MPNA: A Massively-Parallel Neural Array Accelerator with Dataflow Optimization for Convolutional Neural Networks. *arXiv* **2018**, arXiv:1810.12910.
90. Yin, S.; Ouyang, P.; Tang, S.; Tu, F.; Li, X.; Liu, L.; Wei, S. A 1.06-to-5.09 TOPS/W reconfigurable hybrid-neural-network processor for deep learning applications. In Proceedings of the 2017 Symposium on VLSI Circuits, Kyoto, Japan, 5–8 June 2017; pp. C26–C27.
91. Fowers, J.; Ovtcharov, K.; Papamichael, M.; Massengill, T.; Liu, M.; Lo, D.; Alkalay, S.; Haselman, M.; Adams, L.; Ghandi, M.; et al. A Configurable Cloud-Scale DNN Processor for Real-Time AI. In Proceedings of the 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA), Los Angeles, CA, USA, 1–6 June 2018; pp. 1–14.

92. Kwon, H.; Samajdar, A.; Krishna, T. MAERI: Enabling Flexible Dataflow Mapping over DNN Accelerators via Reconfigurable Interconnects. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*; Association for Computing Machinery: New York, NY, USA, 2018; pp. 461–475; doi:10.1145/3173162.3173176.
93. Qin, E.; Samajdar, A.; Kwon, H.; Nadella, V.; Srinivasan, S.; Das, D.; Kaul, B.; Krishna, T. SIGMA: A Sparse and Irregular GEMM Accelerator with Flexible Interconnects for DNN Training. In Proceedings of the 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA), San Diego, CA, USA, 22–26 February 2020; pp. 58–70.
94. Shin, D.; Lee, J.; Lee, J.; Lee, J.; Yoo, H. DNPU: An Energy-Efficient Deep-Learning Processor with Heterogeneous Multi-Core Architecture. *IEEE Micro* **2018**, *38*, 85–93.
95. Stoutchinin, A.; Conti, F.; Benini, L. Optimally Scheduling CNN Convolutions for Efficient Memory Access. *arXiv* **2019**, arXiv:1902.01492.
96. Li, J.; Yan, G.; Lu, W.; Jiang, S.; Gong, S.; Wu, J.; Li, X. SmartShuttle: Optimizing off-chip memory accesses for deep learning accelerators. In Proceedings of the 2018 Design, Automation Test in Europe Conference Exhibition (DATE), Dresden, Germany, 19–23 March 2018; pp. 343–348; doi:10.23919/DATE.2018.8342033.
97. Putra, R.V.W.; Hanif, M.A.; Shafique, M. DRMap: A Generic DRAM Data Mapping Policy for Energy-Efficient Processing of Convolutional Neural Networks. In Proceedings of the 57th Annual Design Automation Conference 2020, San Francisco , CA , USA, 19–23 July 2020.
98. Wei, X.; Liang, Y.; Cong, J. Overcoming Data Transfer Bottlenecks in FPGA-based DNN Accelerators via Layer Conscious Memory Management. In Proceedings of the 2019 56th ACM/IEEE Design Automation Conference (DAC), Las Vegas, NV, USA, 2–6 June 2019; pp. 1–6.
99. Khwa, W.; Chen, J.; Li, J.; Si, X.; Yang, E.; Sun, X.; Liu, R.; Chen, P.; Li, Q.; Yu, S.; et al. A 65 nm 4 Kb algorithm-dependent computing-in-memory SRAM unit-macro with 2.3 ns and 55.8 TOPS/W fully parallel product-sum operation for binary DNN edge processors. In Proceedings of the 2018 IEEE International Solid-State Circuits Conference—(ISSCC), San Francisco, CA, USA, 11–15 February 2018; pp. 496–498; doi:10.1109/ISSCC.2018.8310401.
100. Yu, S.; Chen, P. Emerging Memory Technologies: Recent Trends and Prospects. *IEEE Solid-State Circuits Mag.* **2016**, *8*, 43–56.
101. Lee, H.J.; Kim, C.H.; Kim, S.W. Design of Floating-Point MAC Unit for Computing DNN Applications in PIM. In Proceedings of the 2020 International Conference on Electronics, Information, and Communication (ICEIC), Barcelona, Spain, 19–22 January 2020; pp. 1–7.
102. Schabel, J.; Baker, L.; Dey, S.; Li, W.; Franzon, P.D. Processor-in-memory support for artificial neural networks. In Proceedings of the 2016 IEEE International Conference on Rebooting Computing (ICRC), San Diego, CA, USA, 17–19 October 2016; pp. 1–8.

© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).