electronics

MDPI

Review

Embedded Intelligence on FPGA: Survey, Applications and Challenges

Kah Phooi Seng [1,*], Paik Jen Lee [2,3] and Li Minn Ang [2]

[1] School of Engineering and Information Technology, University of New South Wales (UNSW Adfa), Canberra, ACT 2612, Australia
[2] School of Science, Technology and Engineering, University of the Sunshine Coast, Petrie, QLD 4502, Australia; lee.p.jen@outlook.com (P.J.L.); lang@usc.edu.au (L.M.A.)
[3] National Instruments (M), Penang 11960, Malaysia
* Correspondence: k.seng@unsw.edu.au

Abstract: Embedded intelligence (EI) is an emerging research field and has the objective to incorporate machine learning algorithms and intelligent decision-making capabilities into mobile and embedded devices or systems. There are several challenges to be addressed to realize efficient EI implementations in hardware such as the need for: (1) high computational processing; (2) low power consumption (or high energy efficiency); and (3) scalability to accommodate different network sizes and topologies. In recent years, an emerging hardware technology which has demonstrated strong potential and capabilities for EI implementations is the FPGA (field programmable gate array) technology. This paper presents an overview and review of embedded intelligence on FPGA with a focus on applications, platforms and challenges. There are four main classification and thematic descriptors which are reviewed and discussed in this paper for EI: (1) EI techniques including machine learning and neural networks, deep learning, expert systems, fuzzy intelligence, swarm intelligence, self-organizing map (SOM) and extreme learning; (2) applications for EI including object detection and recognition, indoor localization and surveillance monitoring, and other EI applications; (3) hardware and platforms for EI; and (4) challenges for EI. The paper aims to introduce interested researchers to this area and motivate the development of practical FPGA solutions for EI deployment.

Keywords: embedded intelligence; FPGA; embedded systems; deep learning; neural networks; artificial intelligence

Citation: Seng, K.P.; Lee, P.J.; Ang, L.M. Embedded Intelligence on FPGA: Survey, Applications and Challenges. *Electronics* **2021**, *10*, 895. https://doi.org/10.3390/electronics10080895

Academic Editor: Luis Gomes

Received: 23 December 2020
Accepted: 24 January 2021
Published: 8 April 2021

1. Introduction

In recent years, machine learning [1] and swarm intelligence [2] algorithms have demonstrated successes in many fields and solved practical problems in industry. In particular, deep neural networks (DNNs) such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs) have been shown to be highly successful for diverse applications in healthcare, computer vision, manufacturing processes (e.g., the Industrial Internet of Things (IIoT)) and many other applications requiring complex and intelligent information processing. To increase the value of machine learning to society, it is desirable to embed DNN techniques and algorithms into portable and low-cost devices. Embedded intelligence (EI) is an emerging research field and has the objective to incorporate machine learning algorithms and intelligent decision-making capabilities into mobile and embedded devices or systems.

There are several challenges to be addressed to realize efficient EI implementations in hardware such as the need for: (1) high computational processing; (2) cost efficiency and/or low power consumption; and (3) scalability techniques and architectures to accommodate different networks, sizes and topologies. The challenge for (1) is particularly important as EI algorithms and techniques are becoming increasingly more sophisticated. For example, DNNs which use multilayer network models for the extraction of high-level features

require many network layers and neurons to be computed within the EI devices. The challenge for (2) is the requirement for cost and energy efficiency to meet the demand for cost-effective embedded devices with a limited energy supply such as batteries. The challenges for (3) are to find solutions for EI devices which enable the algorithms and techniques to be implemented within architectures which are flexible and scalable to accommodate the targeted computational requirements and hardware resources which are available for the implementation.

Examples of applications which would benefit from EI are in autonomous systems and edge computing. The authors in [3] discuss the challenges of edge computing for autonomous driving systems to have the capability to deliver high computational power to meet the energy consumption level which is reasonable to be able to guarantee the safety systems on autonomous vehicles. The edge computing systems for autonomous driving need to have the capability to process a large amount of data coming from multiple sensing (e.g., vision and proximity distance sensors) and safety systems in real-time.

In recent years, an emerging hardware technology which has demonstrated strong potential and capabilities for EI implementations is the FPGA (field programmable gate array) technology [4]. A key advantage of FPGA compared with CPU and GPU technology for EI is that the hardware implementation can be customized to meet the specific requirements of the EI algorithm to be implemented. For example, DNNs such as CNNs or RNNs (recurrent neural network) can be implemented in FPGA to only perform the required target logic (e.g., inference) and in hardware. This leads to an efficient hardware implementation which is capable of higher computational processing, higher energy efficiency (e.g., by using fixed-point computations and appropriate quantized memory representations in place of floating-point). FPGA-based systems also have advantages of constant latency due to dedicated hardware compared with CPU and GPU systems. These features for FPGA have drawn many researchers to propose the realization of machine learning and decision-making algorithms and techniques (which traditionally have been realized on centralized computational systems e.g., cloud computing) into embedded devices/systems.

This paper presents an overview and review of embedded intelligence on FPGA with a focus on applications, platforms and challenges. The paper aims to introduce interested researchers to this area and motivate the development of practical FPGA solutions for EI deployment. The remainder of the paper is organized as follows. Section 2 will give an overview of EI research on FPGA. Section 3 discusses various EI techniques on FPGA. In Section 4, different EI applications that uses FPGA as their hardware platform is reviewed. Section 5 describes various hardware platforms of EI such as Intel, Xilinx and other FPGA vendors. Section 6 will discuss about some of the challenges of EI FPGA which need to be addressed and the various efforts made by researchers to overcome them. Section 7 gives some concluding remarks.

2. Overview of Embedded Intelligence (EI) on FPGA

Table 1 gives an overview and classification of EI research on FPGA. The table shows the full spectrum in this research area, and also serves as a concise summary of the scope of the paper. There are four main classification and thematic descriptors which are reviewed and discussed in this paper for EI: (1) EI techniques including machine learning and neural networks, deep learning, expert systems, fuzzy intelligence, swarm intelligence, self-organizing map (SOM) and extreme learning; (2) applications for EI including object detection and recognition, indoor localization and surveillance monitoring, and other EI applications; (3) hardware and platforms for EI; and (4) challenges for EI. The relevant works and references which correspond to the respective classification and thematic descriptors are also listed to facilitate the rapid searching of the reviewed works for readers.

Table 1. Classification and Thematic Descriptors of Embedded Intelligence on FPGA.

Classification and Thematic Descriptors	References
EI Techniques	
Machine Learning and Neural Network	[5–17]
Deep Learning	[18–36]
Expert System	[36–39]
Fuzzy Intelligence	[40–46]
Swarm Intelligence	[47–51]
SOM and Extreme Learning	[52–58]
Applications for EI	
Object Detection and Recognition	[59–66]
Indoor Localization and Surveillance Monitoring	[67–73]
Others	[74–76]
EI Hardware and platforms	
Xilinx	[77–81]
Altera	[82–85]
Archronix	[86]
Challenges for EI	
Challenges	[87–97]

3. Embedded Intelligence (EI) Techniques

This section gives some discussions for FPGA-based EI techniques from six perspectives: (1) machine learning and neural networks; (2) deep learning; (3) expert systems; (4) fuzzy intelligence; (5) swarm intelligence; and (6) self-organizing map (SOM) and extreme learning machine (ELM).

3.1. Machine Learning and Neural Network

Field programmable gate array (FPGA) technologies are becoming popularized in machine learning (ML) and neural network (NN) applications. FPGA technologies and platforms have the benefit of parallel computations, lower power consumption and short turnaround time as compared to ASIC (application specific integrated circuits) for practical implementations. Some extensive works on FPGA-based technologies for sensor-based systems can be found in [92–94]. This section will focus on FPGA-based systems focusing on machine learning and neural network implementations.

The authors in [5] proposed the Ubiquitous ML Accelerator with Automatic Parallelization on FPGA. The concept of automatic parallelization uses out-of-order techniques where the instruction set command is customized. This technique can be used for either blocking or nonblocking interfaces. These can apply across different heavy data processing application such as deep learning, clustering, etc. The accelerator utilizes the concept of 2D meshed architecture that consists of multiple components: (1) instruction buffer to store the customized instruction; (2) task scheduler to assign tasks in real-time; (3) out-of-order core to fully utilize the resources in order to achieve optimum parallelism; (4) network interface to communicate with other blocks. The temporary data will be stored as off-chip memory known as DMA and DRAM controller [6].

The out-of-order techniques make use of the ability of customized commands where all the blocks as shown in the architecture above require very specific commands to execute. Thus, commands have two main categories—blocking and nonblocking interfaces. Blocking interfaces refer to a task that must wait until entire execution completes, whereas nonblocking commands can execute the mechanism concurrently. At the same time, the

out-of-order scheduler needs to ensure the intertask data dependence scheme to exploit the potential parallelism. By using this technique, it is able to speed up 25 times as compared to high performance intel processors, and thus it is suitable for highly intensive data processing applications.

Since FPGA has inherent parallelism and deeply pipelined computation capability, a highly scalable and parameterizable FPGA-based Stochastic Gradient Descent (SGD) is able to be used for different applications such as linear model training; this training model is commonly used for classification, compressive sensing and image construction. In terms of hardware efficiency, SGD techniques were able to achieve the convergence quality, with a slight tradeoff for the algorithm precision (e.g., using fixed-point implementations [7]). By quantizing data, it is able to increase the overall efficiency of the embedded system as the system is not required to read all the data, which requires huge amount of memory. The reason is all of the data are being quantized in advance so that all the data can be characterize into one cache line. This technique is able to reduce bandwidth utilization significantly and, at the same time, maintain the satisfactory efficiency. The number of alterations in the data are increased and improve the statistical efficiency. Furthermore, all the data quantization must happen before the precalculation step so that the precalculation step does not require a large amount of data. The computation pipeline for quantized data are another key factor for increasing the performance of the FPGA because all the data can be computed in parallel after a certain number of iterations. The selection of stochastic data will decide the width of the pipeline which needs to set before synthesis. The authors in [8] gave a comparison of the SGD with quantization techniques in FPGA (in their approach using single-precision) having similar performance compared to 10-core CPU.

Support vector machines (SVM) are one of the most commonly used methods for machine learning that build a hyperplane to classify data between two classes [9], although a decomposition technique is commonly used to improve the performance of the SVM [10]. However, utilizing the cascaded multiprecision training flow means that a system is able to contribute towards the training acceleration using hardware platforms by take advantages of FPGA on chip memory blocks and parallelization potential. Cascaded multiprecision training techniques are basically all the data will go through lower precision phases which require lesser bandwidth and faster execution time. After that all the mismatch data will go through higher precision phases. The Gilbert algorithm is used to solve the nearest point problem where all the computations are computed under the processing element (PE) [11]. Based on the evaluation, the cascade technique was able to outperform this while maintaining comparable accuracy.

Another research paper on machine learning proposed by [12] focused on classification techniques. In general, graph-based machine learning techniques found that this is the most powerful technique for classification. Optimum path forest (OPF) is considered one of the graph-based classifiers quite similar to support vector machine (SVM) techniques but has a faster execution time and is easier to deploy [13]. This method used the graph partition task as a competitive process by making the unknown samples dominant based on the path cost function. Vector-based operation is used in OPF, which is a significant benefit for FPGA and ideal for hard real-time signal processing applications. The main architecture of OPF uses the single instruction multiple data (SIMD) approach with a group of processing elements (PE) and custom memory controllers for handling the data distribution given by the PE [14]. OPF consists of two main subsections, which are the training stage and the classification stage. Euclidean distance is used in the training stage to classify the most fitting element. At the same time, a minimum spanning tree (MST) is extracted from the graph after the training is completed. On the other hand, the classification stage is the process of the minimization of dissimilarity. Figure 1 shows the overall FPGA architecture used in OPF classification.

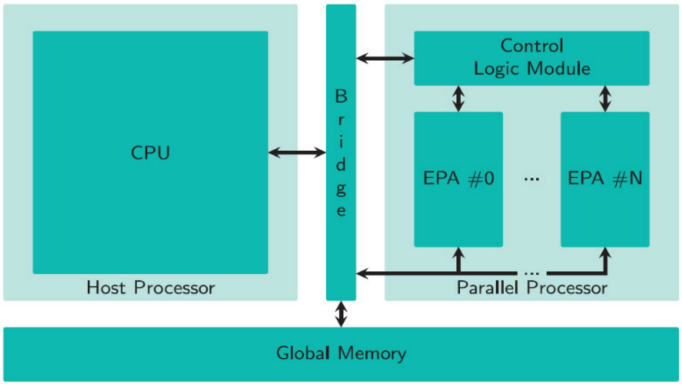

Figure 1. Overall FPGA architecture for optimum path forest (OPF) classification [12] (Reprinted with permission from ref. [12]. Copyright 2021 Elsevier).

In general, the host processor is used to control the execution flow, managing data due to higher bandwidth and patching commands to the parallel processor (PP). There is a bridge to connect the CPU and FPGA where the main features are for communication and data distribution to reduce data access latency and create a direct channel to the DMA. The parallel processor plays an important role because the EPA inside the PP is used for computing the core functionality of the entire application. Figure 2 shows that each EPA consists of their own local shared memory (LSM) linked with a memory management (MM) module to protect the memory from corruption. All intensive tasks are handled by the EP inside the EPA, with all the OPF classification algorithm being computed here. The parallel architecture can be easily used for any other application or algorithms simply by redesigning the EP [12]. The architecture is focused on hardware implementation that decreases global memory access and increases the computation to memory access ratio which improves the overall execution time from 2.18 to 9 times faster than the software version [12].

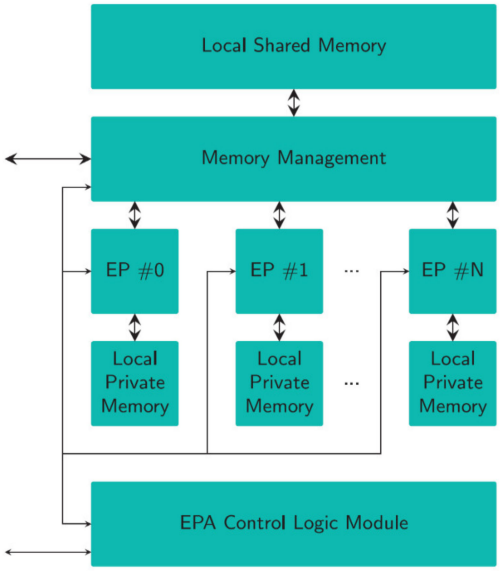

Figure 2. Block diagram for EPA [12] (Reprinted with permission from ref. [12]. Copyright 2021 Elsevier).

The conditional restricted Boltzmann machine (CRBM) is proposed by [15] for arbitrarily large models that are based on data distribution. By using FPGA on the CRBM, it is able to save on power consumption and increase performance by approximately five times higher than CPU solutions. A parametrizable framework is used for CRBM related

applications. Furthermore, the bipartite graph is the part of CRBM where the nodes of one part are input, also known as the visible layer, while the hidden layer refers to the node at the output [15]. Different applications will implement different algorithms based on size, such as batch gradient descent, minibatch gradient descent and stochastic gradient descent. A 32-bit floating point is used; thus, multiple blocks are required for optimization in order to compute general matrix multiplication (GEMM) [16]. Batch gradient descents will take the average of the gradient and then the mean will update as a parameter, but this does not work well in large dataset. Thus, SGD become a good option because only one dataset will be used at all times and the parameter frequently updates. Furthermore, the mini batch gradient descent is to able to use vectored implementation where the other two algorithms are unable to use it. The technique for this accelerator is dividing the matrices into blocks to improve utilization of adjacent data.

The core block for the processing the GEMM is the processing element (PE) where all the multiplication and accumulation happen. The function of the PE is also processing the data shared by the adjacent PE. All the data are transferred and handled by the FIFO buffer. Furthermore, double buffering is used at the input and output, so that all the function can execute parallel with data transfer. Their work showed a 15% execution time improvement by using host pipes to reduce the data transfer overhead between CPU and FPGA [15].

A recent work by the authors in [17] proposed a low-powered processor for embedded AI systems termed as Intellino. Their approach and experimental implementations investigated two AI algorithms (*k*-nearest neighbor and radial basis function (RBF) neural network) and were able to achieve low-power and hardware complexity by excluding heavy computational requirements such as multiplication operations. The authors also proposed optimizations for distance-based AI algorithms which are suitable for practical hardware realizations on FPGA platforms. Their experimental implementation on FPGA showed that Intellino required a 1.5 MHz operating frequency leading to a reduction in power consumption for the AI embedded system.

3.2. Deep Learning

Deep learning techniques generate a network that can learn and train by itself at the same time as constructing precise decisions based on the training data provided. In recent years, the usage of FPGA on deep learning has been increasing significantly in deep learning related applications due to the capability for maximizing parallelism and the energy efficiency. Although FPGA has relatively lower memory, bandwidth and computing resources compared with GPU, it is able to provide high throughput with reasonable accuracy. This subsection will discuss FPGA-based deep learning architectures, and in particular for convolutional neural network (CNN).

The authors in [18] proposed a hardware accelerator for compressed CNN (referred to as CoNNA) which is able to accelerate pruned, quantized CNN. The coarse-grained reconfigurable architecture is applied in CoNNA, which allows immediate selection of the CNN network [19]. Based on the analysis of the research, CoNNA is able to speed up at least 3.08 times and up to 14.1 times faster than the MIT Eyeriss Accelerator [20]. The specialty of CoNNA is that it used different methods in parallelizing the CNN operations. The individual convolution operation is executed sequentially but each convolution has a single processing unit used to calculate the multiply accumulate operation. However, all of the convolutions consist of a number of processing units that are able to compute in parallel. In addition, to increase CNN processing performance, CoNNA was able to compress CNN networks and featured maps directly. CoNNA's power consumption is reduced significantly although CoNNA stores feature maps in the external memory, but these consist of a cache line to store temporary data and reduce the data transfer between the accelerator and the external memory [18]. CNN pruning will be the key technique for this accelerator. All of the redundant weights memory will be removed, thus the memory can be significantly reduced. This compression is also done by skipping any zero at input

feature maps and convolutional kernels. By skipping all the zeros, the system was able to process the input instances at higher rate due to the lower precision required [21].

The reconfigurable computing unit (RCU) is mainly aiming to compute all the arithmetic algorithms done by different layers. Within the CoNNA CNN accelerator, an input stream manager (ISM) will distribute the data to different parties in the CNN in compressed format; an output stream manager (OSM) will handle the compressed data and decide the data to be stored on external memory or on chip caches. All the operations and interfacing will be the function of configuration and control unit (CCU) [18]. Lastly, since CoNNA processes the network in sequential manner layer by layer, it is crucial to store the mode appropriately based on the CNN description linked list that is able to be reconfigurable at the RCU units. To make it more efficient all the available blocks will be channeled for RCU to process in parallel. In conclusion, CoNNA is able to utilize the compressed CNN technique to excel the FPGA performance as compared to the traditional standard CNN method in GPU.

Other authors then proposed a compressed CNN method for hardware accelerator; there is another research paper that uses fast and scalable architecture which targets low density FPGAs [22]. The two main techniques used in the architecture are fixed-point arithmetic and image batching. The proposed techniques are able to save the computational memory and memory bandwidth, which allow for a focus on real-time processing applications such as surveillance monitoring and object detection [23]. Fixed point quantization is part of the optimization techniques for which the activations and weights are represented in fixed point format regardless of the size of the window. Every layer can have its own scale factor, while the image batch being in convolution layers means multiple images executing in a batch able to reduce the memory bandwidth [22]. Figure 3 shows an overview of the proposed architecture.

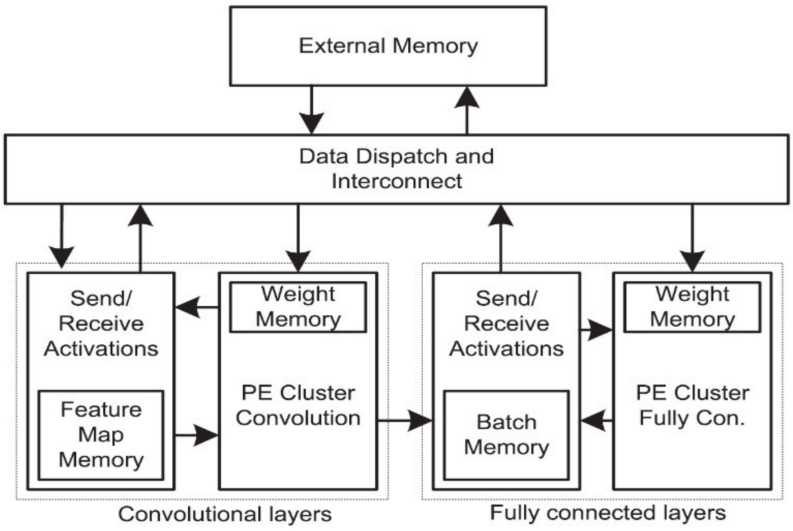

Figure 3. Overview of the architecture with separate blocks for convolutional and fully connected layers [22]. (Reprinted with permission from ref. [22] Copyright 2021 Elsevier).

Although the convolutional layers and full connected layers are quite similar, they have to be broken into two different modules and executed in parallel. The input image and intermediate feature maps are stored in the on-chip memory. If the on-chip memory is too small to store the image, then the image will break into small pieces and be convolved separately. Besides that, OFM that is not able to fit into on chip memory will be stored in external memory through the data dispatch and interconnected. All the arithmetic will be computed by PE that run in parallel. The convolution layers' PE cluster consists of the matrix and interconnection network for forward input and output activation between PE, as shown in Figure 4.

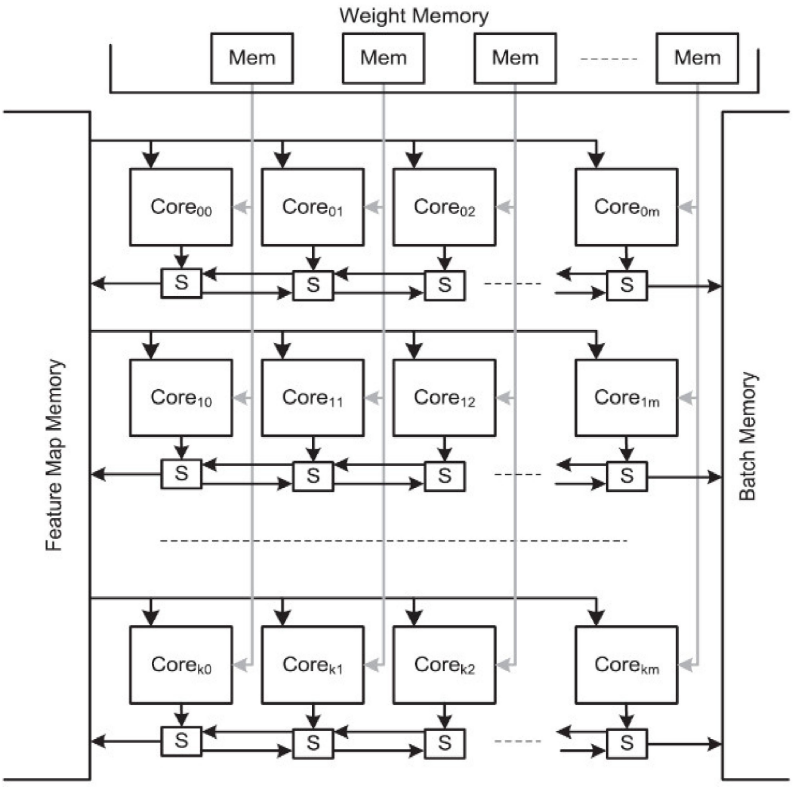

Figure 4. Interconnection network between processing element (PE) where S is referring to the switch [22] (Reprinted with permission from ref. [22]. Copyright 2021 Elsevier).

If the next layer is the convolution layer, then it will be sent back to feature map memory; otherwise, it will be sent to the FC layer, which consists of the same structurelike cluster of convolutions. The entire data transfer is performed in two small modules known as switch. The 8-bit mixed fixed-point representation for weights and activation are able to improve performance and reduce the loss of accuracy in hardware [22]. At the same, this architecture is able to increase the throughput by focusing on image batching at the FC layers and fixed-point arithmetic at the convolution layers. Another work by the authors in [98] proposed a framework, termed as Finn, which focused on the efficient mapping of binarized neural networks to FPGA hardware. Their experimental results on a ZC706 embedded FPGA platform showed that their hardware could give very low power consumption (less than 25 W) while maintaining high classification performance on standard datasets (MNIST, CIFAR-10, SVHN).

In the research [24], a depthwise separable convolution neural network on FPGA is being reviewed due to its ability to reduce the number of model parameters and improve calculation speeds, which makes it widely used in mobile edge computing applications such as Xception [25] and MobileNet [26]. A custom computing engine architecture is proposed to handle the data transfer by using double buffering-based memory channels. Besides that, dividing the matrix multiplication to small form factors is applied in fully connected layers: this is known as the data tiling technique. A depthwise separable CNN is actually a factorization from the traditional convolution to depthwise convolution and pointwise convolution. A single filter is applied to every channel of the IFM that is used for depthwise convolution, while convolution to build new features is created by combining the depthwise convolution in pointwise convolution [24]. The overview of the accelerator is shown in Figure 5.

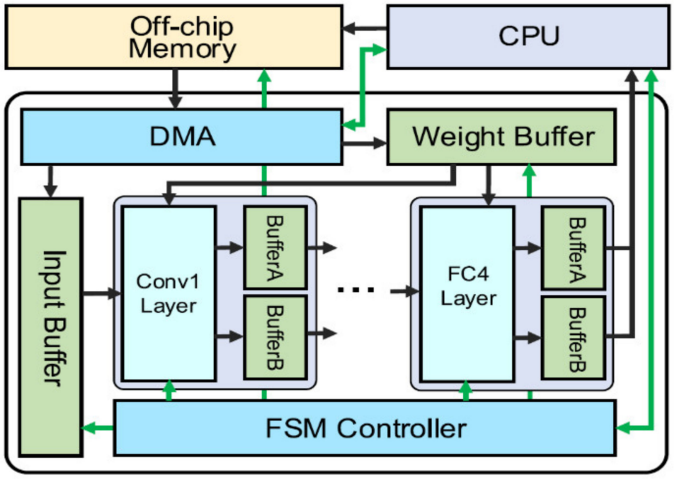

Figure 5. Depthwise separable architecture hardware accelerator [24] (Reprinted with permission from ref. [24]. Copyright 2021 Elsevier).

By storing the intermediate result between layers on the chip, it is possible to save the data access time and directly increase the throughput. The above shows all the blocks available in the architecture, where the green color represents the control paths and the black represents the data paths. Coordination and communication with the external CPU are done by FSM controller. The most computation-intensive parts are the convolution layers, which include both depthwise and pointwise convolution units.

On the other hand, DCU (Figure 6) consists of multiply accumulate unit (MAC) computing, accumulators, nonlinearity activation modules and max pooling modules. The nonlinearity module carries out the ReLU function, and down-sampling of the data are computed at the max pooling module. Figure 7 shows how the pipeline architecture works where it acts as one of the most essential techniques for this depthwise separable CNN. From Figure 7, we can see that CONV1 to 3 are executed in pipelined at FC4 layers. Other optimization techniques such as parallelizing computation of multiple outputs and inputs feature maps as well as the convolutional kernel that provided the potential of parallelism. Last but not least, low precision numeric computing is used. Based on the research, their experimental work showed that 16-bit fixed point multiplication consumed 6.2 times less energy than the 32-bit floating point, with a small accuracy loss of 0.5%. The work concluded that the design could improve performance times by up to 17.6 times faster than CPU while using 29.4 times less energy than GPU for a $32 \times 32 \times 3$ image size [24].

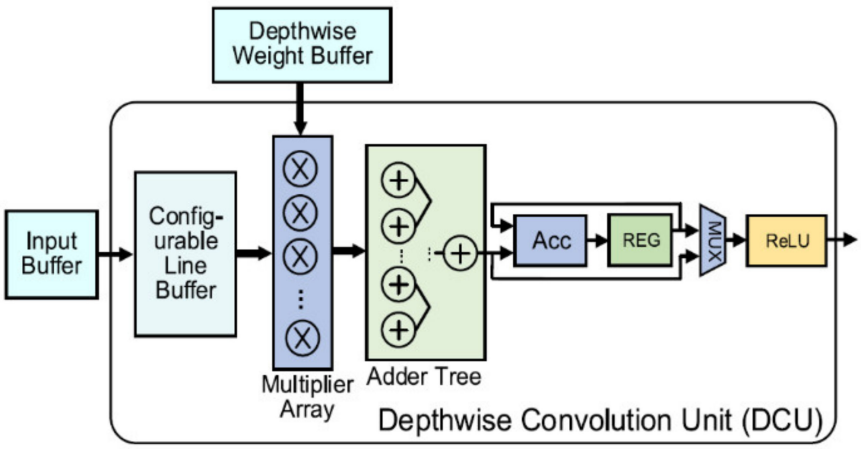

Figure 6. Depthwise convolution unit (DCU) [24] (Reprinted with permission from ref. [24]. Copyright 2021 Elsevier).

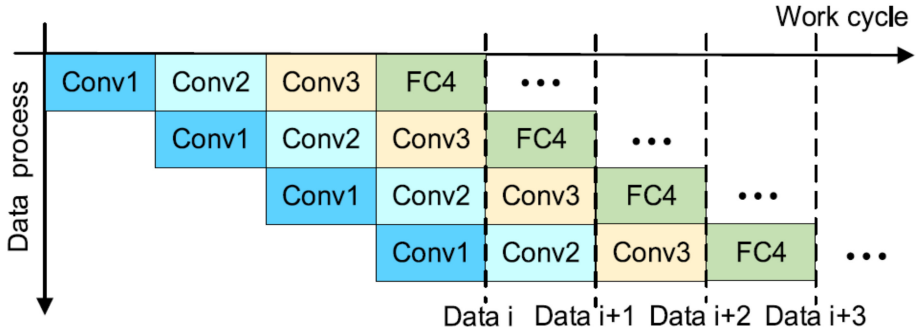

Figure 7. Pipelined optimization technique [24] (Reprinted with permission from ref. [24]. Copyright 2021 Elsevier).

Besides that, reducing parameter requirements is another field that researchers are trying to explore. SqueezeNet and LetNet are some of the examples using condensed parameter requirement techniques in their hardware accelerator design [27]. In this study, the authors in [28] proposed to leverage the previous research by reducing convolutional layer kernels and size and removes most of the fully connected layers. All the layers that are removed will be replaced by a layer called the fire module layer [29] which is a combination of the squeeze layer and expand convolution layers.

The network also uses the down sampling method to reduce size and maintain higher accuracy. The number of parameters is calculated by multiplication of the number of parameters in one kernel by the number of output channels in that layer. Besides that, AC operation is calculated based on the number of weight vectors per square of the output feature dimension of the layer [28].

In term of bit width, 8-bit encoded weight vectors and 8-bit encoded image pixels are used. Each of the layers have a different number of parallel memories subject to the size of kernel. By using this fire module technique, it was able to reduce the parameter by 11.2 times fewer than standard network without impacting classification accuracy [28]. Another method has been published as a deep learning technique is referred as ALAMO. ALAMO uses loop unrolling, pipelining, fixed point function and prefetching techniques as CNN accelerators [30]. Figure 8 shows the architecture of the module.

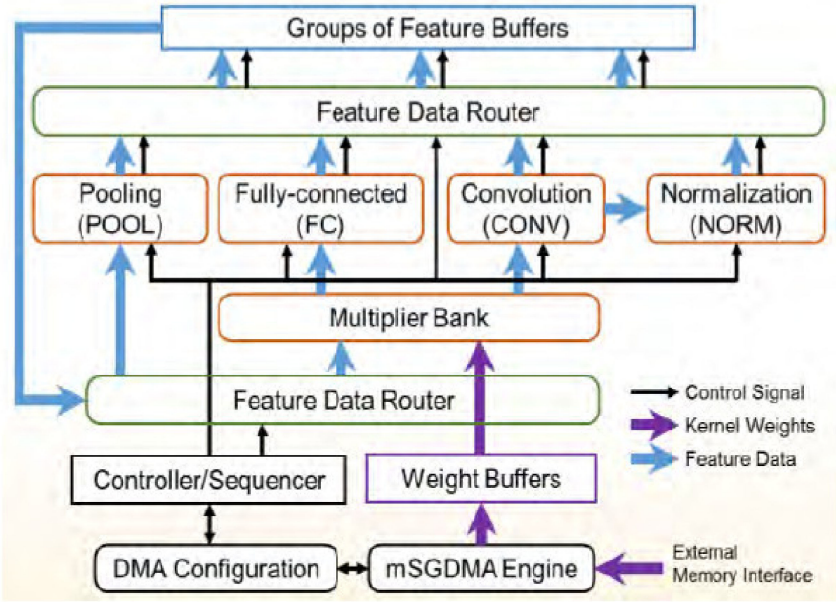

Figure 8. Overall architecture of ALAMO [30] (Reprinted with permission from ref. [30]. Copyright 2021 Elsevier).

Selection of the data read and data write is overseen by the data router, whereas the feature buffers are used to hold the FM using on chip RAMs. Internally there is a weight buffer used to control the weight for conventional layer (CONV) and fully connected (FC) layers. The memory system is designed to store kernel weights and features. All the computation functions, such as the adder tree and ReLU components, are stored inside the CONV module. The ReLU component check inputs pixel sign bits to either output zero or the data pixel itself, while the average or maximum operation is done by the Pool module. The NORM module and FC module are used to maintain the required components to perform the operations of local response normalization and share the multiplier bank module with the CONV module to perform matrix–vector multiplication (MVM) [30].

Another proposed method, termed DarkFPGA by [32], uses batch-level parallelism hardware architecture. Batch-oriented data patterns and tiling techniques are used to optimize the hardware accelerator, which is able to effectively exploit parallelism. In this training, the researcher is targeting how to use low precision training by using an 8-bit integer. According to all the research, by using low point fixed point, it is them possible to increase the speed and power consumption. In addition, batch-oriented data patterns, which refer to channel height width batch (CHWB) patterns, are implemented, using batch level parallelism but utilizing bandwidth as a compromise.

Although tiling is another common optimization technique used to optimize memory bandwidth and computer resource utilization, in this work tiling is mainly used to break the large input feature to smaller tiles of data. Once the data can be fitted into the on-chip memory of an FPGA, the approach will tile the data into four main dimensions, which are batch tile, channel tile, filter tile and image tile. The parameter selection for tiling is crucial to maximize the performance of the design. The batch level parallelism is handled by PE which, in parallel, is using GEMM kernel [33]. Both forward and backward propagation on a homogenous FPGA system were able to improve the training by using a DarkFPGA accelerator with batch level parallelism.

Quantization and compression techniques are widely used as hardware accelerator techniques; however, the disadvantage is the potential of degradation to very low precision in complex problem. In conjunction with the disadvantages highlighted above, Addnet is proposed by [34] to demonstrate the reconfigurable constant coefficient multipliers (RCCM). RCCM claims that it is a better alternative by replacing multiplication input values with only adders, subtractions, bit shifters and multiplexers [35]: for example, a constant coefficient multiplier in a circuit known as y = cx where c is constant. It can translate an equation of y = 6x with just addition and shift registers such as (x << 2) + (x < 1) = 6x. The result of implementing RCCM and AddNet was the ability to save resources by up to 50% more than a standard 8-bit quantized network with similar accuracy [34]. Likewise, all FPGA devices consist of LUTs followed by a fast carry chain, thus using MUX with no additional cost as long as the correct RCCMs are selected to simplify the training process. MAC operations are used in parallel to compute the convolution through the input feature and weight buffer. Additional topology is used in this research to replace a standard 8-bit multiplier with the AddNet constant coefficient multiplier. The accumulator will accumulate all the data before transfer to ReLU. This approach gives better accuracy than the network that constrain weight parameters to have binary value. Multiplication is an expensive solution for resources; thus, this implementation replaces it with traditional resources like shifters, adders, etc. It is a new dimension for the optimization of neural networks.

In summary of all the reviews in this subsection, compared to ASIC implementations, FPGA is suitable for deep learning in embedded intelligence due to the flexibility of the platform to allow the developer or researcher to develop different deep learning application in very short timeframe. The rich set of programmable logic cells and embedded components, such as DSP, allow it to perform arithmetic intensive operations. There are still many other techniques being researched and deployed in the market for deep learning, such as rearranging memory data, double buffering, graph partitioning, the roofline model, etc.

3.3. Expert System

Expert systems in AI can resolve many issues via reliable and interactive computer-based decision making by reference to expert experience [36]. Traditional methods of expert systems using software algorithms, however, are limited by the processing speed of the complex system. Therefore, an expert system is difficult to apply in different applications due to the process being slow and complex. One of the most commonly used applications for expert systems is fault diagnosis. The proposed method shows that hardware implementation is able to achieve speeds of at least 6x faster than software implementation. The high level of the expert system and the realizing process of the expert system with FPGA are discussed in [37].

The whole structure of the expert system consists of three main parts, which are the master controller, FPGA and EEPROM. EEPROM is used to store memory. FPGA will used to handle all the reasoning processing based on the design rules. At the same time, it will also provide the process coordination for EEPROM to access the parameter information. EEPROM bus switching is also very important as it will handle the block according to the reasoning process. The reasoning process focuses on the expert knowledge and experience and reflects the relationship between sign and status. It will be completed by using basic logic directly. Furthermore, there are two main types of interface, which are known as the serial interface and parallel interface. Serial interface design shows that this approach is seven times slower in the communication speed as compared to a parallel interface, but fewer ports are occupied, whereas the parallel interface occupies many system ports but its read–write speed is very high [37]. There is another piece of research done by [38] on expert systems based on expert reasoning rules and flow charts of expert adaptive control. To improve precision, PID control and multistrategy expert knowledge are used [39]. The reasoning mechanism based on the production rules is shown as U = f(E, K, I, G) where E is the input information, K is the empirical fact and U is the output of control behaviour [38].

The PWM of the system is used in the Altera chip to generate square wave output. The facts in the expert knowledge base sampled from the different layers through the digital PID controller to transform into FPGA. The proposed technique was able to smoothen the signal without overshoot; however, it is not suitable for complex systems. In summary, an expert system in only suitable for small scale EI applications even though FPGA is able to speed up the entire process.

3.4. Fuzzy Intelligence

Fuzzy logic systems aim to model complex systems which cannot be modeled mathematically. A fuzzy system mainly consists of four main blocks such as a fuzzifier, rule base, inference engine and defuzzification [40].

The fuzzifier is mainly used to convert the crisp input value to a fuzzified value. Besides that, the inference engine is basically a kernel that handles fuzzy rule based on the knowledge base. Lastly, defuzzification is used o output from the fuzzy system. The most commonly used defuzzification methods are known as center of maximum (COM) [42], center of gravity (COG) [41] and mean of maximum (MOM) [43].

- COM typically used in closed loop control applications due to the small change in the input, which will not impact the output signal.
- COG strength is the capability to accelerate the CMOS arithmetic architecture.
- The MOM method selects the typical value of the most valid output linguistic term instead of averaging the degree of membership function (MF).

FPGA is suitable for implementing fuzzy logic systems due to its ability to implement various arithmetic equations and make fast decision in real-time environments. Furthermore, pipeline digital architecture is an architecture that utilizes the advantages provided by FPGA in image contrast techniques. Grayscale images only consist of one component instead of an RGB image, thus it is suitable to the use zero order Takagi–Sugeno fuzzy model to enhance the image contrast [44]. The specialty of this model is the capability to avoid the division operation in the defuzzification block, which is very computationally

intensive. The Xilinx DSP48E2 slice is proposed in this digital architecture by optimizing the DSP blocks [44]. The overall structure of the DS48E2 slice is shown in Figure 9.

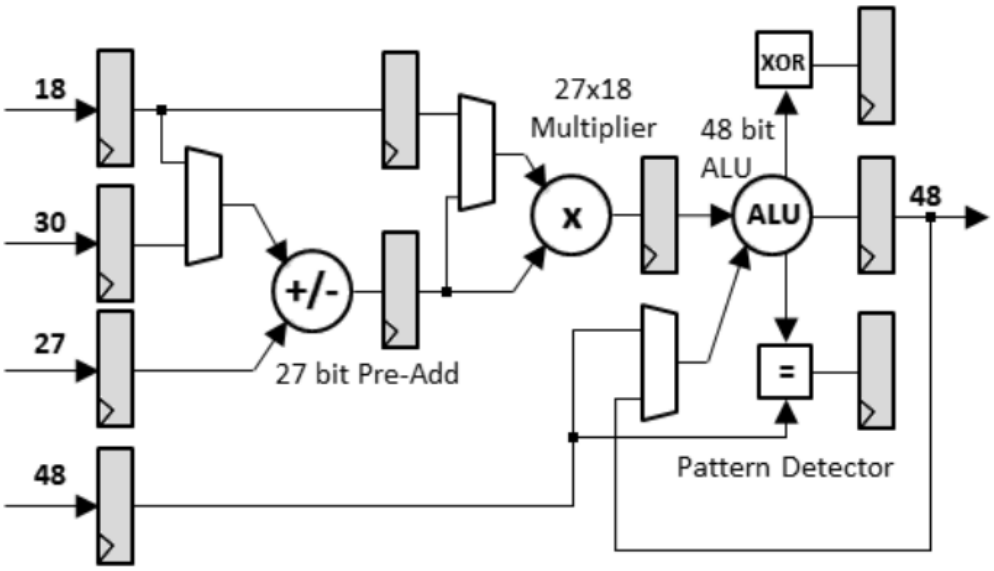

Figure 9. DS48E2 slice internal structure [44] (Reprinted with permission from ref. [44]. Copyright 2021 IEEE).

The main fuzzy model consisting of range detector blocks compares the input pixel value, whereas the active rule detector blocks are searching the active fuzzy rules from the fuzzy rule base by referring to the input pixel. Lastly, the fuzzy core block is responsible for the entire implementation by computing the output pixels according to the fuzzy model. Figure 10 shows the fuzzy model for digital architecture.

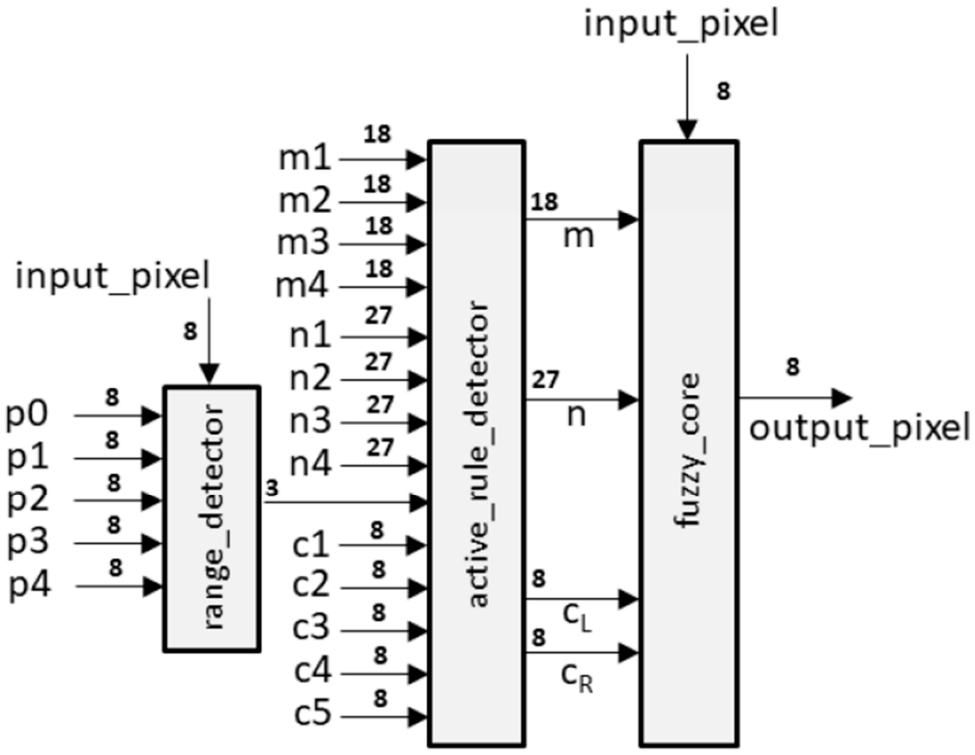

Figure 10. Fuzzy model for digital architecture [44] (Reprinted with permission from ref. [44]. Copyright 2021 IEEE).

The pipeline is the core structure for the digital architecture. The fuzzification block implements the equation by taking the input pixel and computing the degree depending on the corresponding parameters. Defuzzification blocks implement the equation in order to compute the output based on the active fuzzy rules.

The fuzzy core block's main configuration is illustrated in Figure 11. In addition, another approach used by researchers is to use hardware IP cores designed to be placed into FPGA acting as hardware accelerated FIS system. Placing the AXI internal bus on the main bus enables it to communicate across all signaling.

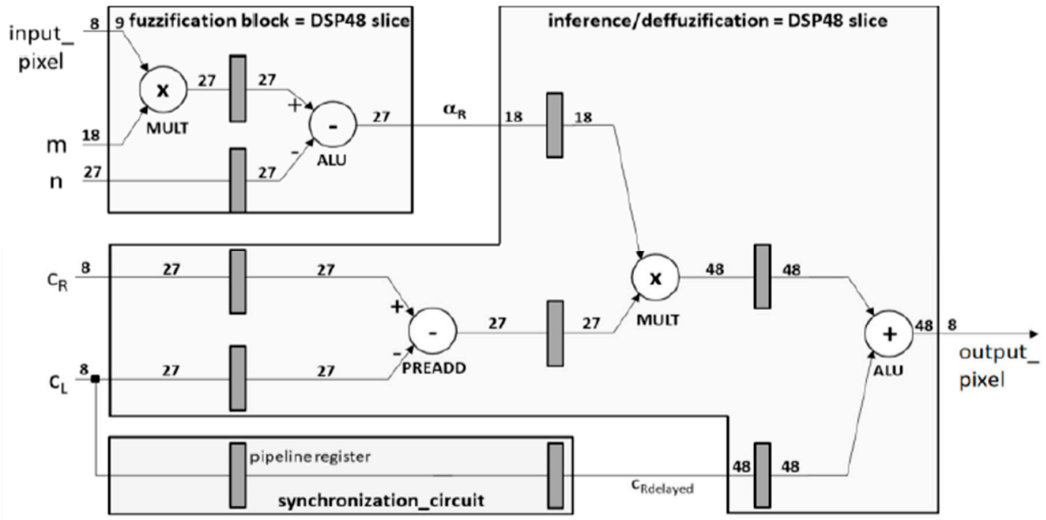

Figure 11. Fuzzy core block main structure [44] (Reprinted with permission from ref. [44]. Copyright 2021 IEEE).

Furthermore, an accelerated fuzzy membership function (MF) is used on FPGA in fuzzification. The hardware implementation for fast multipliers is one of the methods for implementing MF. This method is done by using fixed point calculation.

In terms of rule evaluation, clipping is more efficient in terms of hardware implementation. Typically, all rules will be an accumulation of the lookup table and decision making will based on going through the lookup table. Next is the aggregation of the rule output: Mamdani and Sugeno aggregation can be used as hardware accelerators. Mamdani aggregation typically uses center of gravity (CoG) techniques. [46].

Lastly we will consider the most popular defuzzification technique, the centroid method. The advantages of this technique are it relative easiness to implement and generate accurate output. Based on the research, MoM can be optimized with respect of the cost of memory usage and can be performed in one clock cycle. In summary, with this method the designer interacts with 32-bit variable at the address of interface modules and uses scalability to achieve higher levels of abstraction [46].

3.5. Swarm Intelligence

Swarm intelligence (SI) techniques are heuristic stochastic search processes. The collective behavior of decentralized and self-organizing systems originates from SI techniques. At a later stage, a collection of algorithms is proposed to optimize problems such as particle swarm optimization (PSO) and ant colony optimization (ACO) [47]. PSO formulated the process into a mathematical model with two equations which are analogous to the position and velocity of particles. PSO is easy to adapt into any process as long as the population is initialized in agreement of the strategy [48]. For the ACO method, a graph is used to represent the solution space [49]. Updating the individual solutions, evaluating each individual and communication between individuals for information sharing purposes are the main three stages of SI. To optimize the performance of memory and assess the SI, hierarchical convergence and implementation of *combine_x* and *reduce_x* functions is used.

In order to increase the throughput of a memory accessing feature of the hardware platform, a quantum-behaved particle swam optimization (QPSO) algorithm is implemented [50]. FPGA is a hardware platform used for the parallel implementation of SI with good flexible pipelines. In order to optimize or improve the usage of the resources, the fitness evaluation function is used. The fitness value shared by the parallel swarm unit modules is only determined to improve the bandwidth. In addition, PSO technique are implemented on single FPGA with its own position update unit and fitness evaluation unit. Figure 12 shows the rescheduled dataflow of quantum-behaved particle swarm optimization (QPSO).

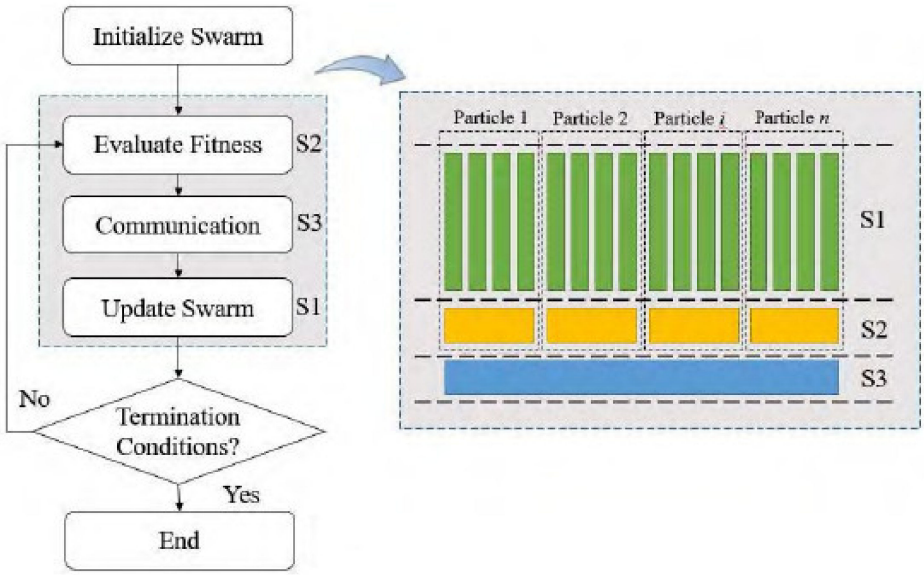

Figure 12. Flow chart of quantum-behaved particle swarm optimization (QPSO) [50] (Reprinted with permission from ref. [50]. Copyright 2021 IEEE).

SI is optimized in FPGA by optimizing the baseline of the general framework of swarm intelligence (FASI) in FPGA by setting the tuning knob CN to 1 and R to 0 [51]. In addition, full pipelined FASI in FPGA, where the reductions and combinations are pipelined together with the threads, as shown in Figure 13, allows the map dimension, map particle, combiner and reducer global to be pipelined together for higher throughput.

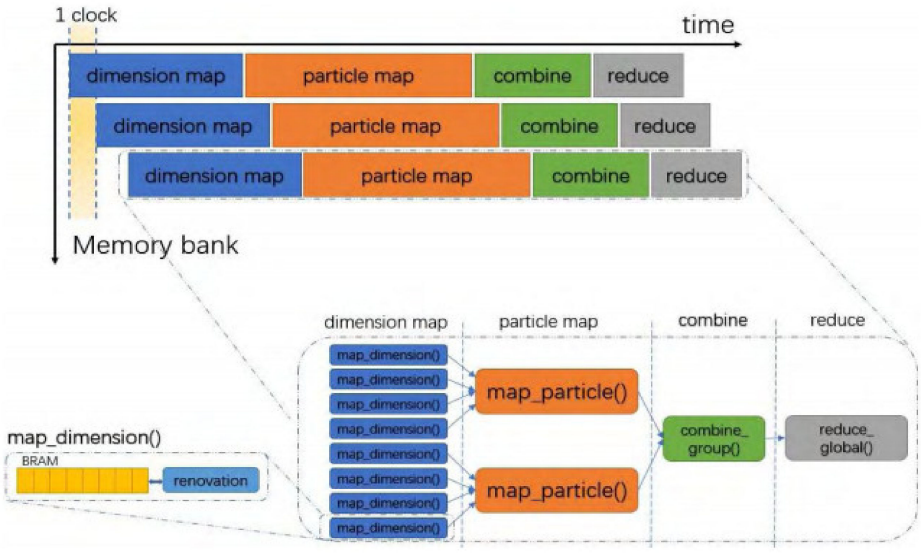

Figure 13. Full pipelined general framework of swarm intelligence (FASI) on FPGA [50]. (Reprinted with permission from ref. [50]. Copyright 2021 IEEE).

Another advantage of FPGA is the capability of parallelism and allowing a one clock initiation interval pipeline (the input data available in one clock cycle). Figure 14 illustrates how BRAMs will be read by the map dimension and how the global best values will be generated by combining groups in every clock [50].

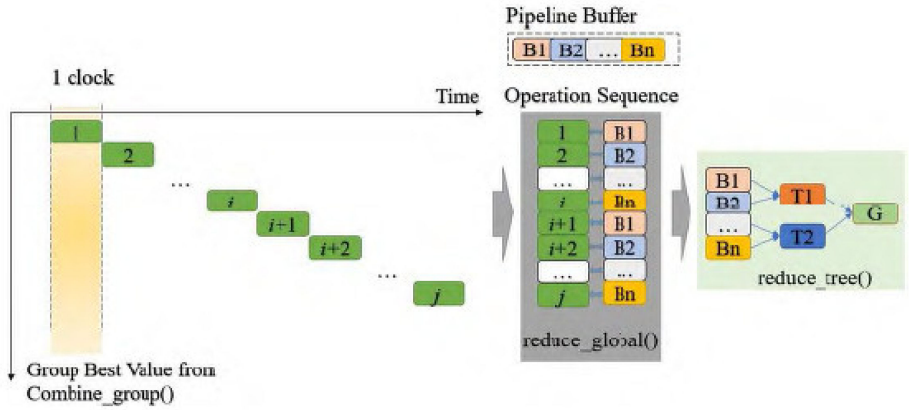

Figure 14. Clock initiation interval by using pipeline buffer [50] (Reprinted with permission from ref. [50]. Copyright 2021 IEEE).

In order to efficiently accelerate QPO, the buffer is designed as a temporary global best value in each of its elements. Another technique is used to improve the BRAM bandwidth in the FPGA by using a pattern of assembling. The bandwidth of the BRAM can be calculated based on the depth per vector (D) multiplied by word width. The BRAM pattern is shown in Figure 15.

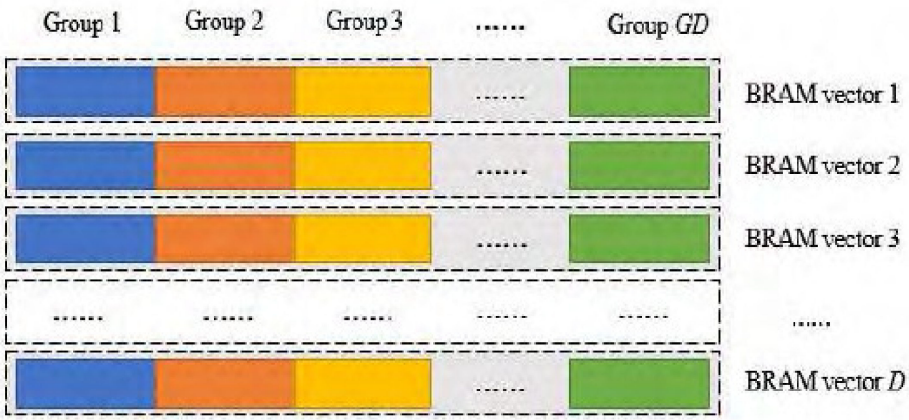

Figure 15. Pattern for BRAMs [50] (Reprinted with permission from ref. [50]. Copyright 2021 IEEE).

In conclusion, swam intelligence implementations require heavy data transmission and the hierarchal memory architecture of hardware platforms. Based on the analysis, FPGA achieves better throughput and also the highest speedup compared to other systems such as CPU and GPU.

3.6. SOM and Extreme Learning Machine

Self-organizing maps (SOM) are an unsupervised learning model that are being studied by many researchers. In traditional SOMs, CPU is used to process all the data. The research by [52] mainly targets parallel processing using hardware implementation. The proposed architecture will use the FPGA platform and show that it is able to improve the overall acceleration by three to four orders of magnitude as compared to CPU execution. The traditional SOM method is more focused on a binary version of SOM, using Hamming

distance to identify the best match units (BMU) [53], allowing for the use of high speed computation of the SOM learning phase by executing large SOM using a moderate density of FPGAs to speed up the data mining operations. Some researchers [54] have used multiple FPGA chips composed into single chipset to accelerate the SOM training process. In this approach, the strategy is aims to improve the data transfer between blocks and the flexibility for use in different networks and applications. The main component that is used in the research is using multiplexer components. The benefit of using multiplexers is due to the ability to reuse the resources for larger networks, such as reusing the fixed size blocks for computing large amounts of data, and reusing the circuit for decision making and the weight update component [52]. Therefore, by applying a multiplexer, it allows the single FPGA chip to be used for large network. Another improvement to the architecture is using pipelined architecture for training algorithms so that the computations can be performed in parallel. Furthermore, the Manhattan distance algorithm is used instead of Euclidean distance due to it having better performance in the evaluation of the relevance of small distances with the increase in computations for *n*-dimensional spaces. There are different types of architecture for SOM, such as a centralized model, a hybrid model and a decentralized model. In the paper by [55], focusing on centralized architecture due to low density, a FPGA chip is required to achieve higher processing speed. Figure 16 shows the schematic for SOM.

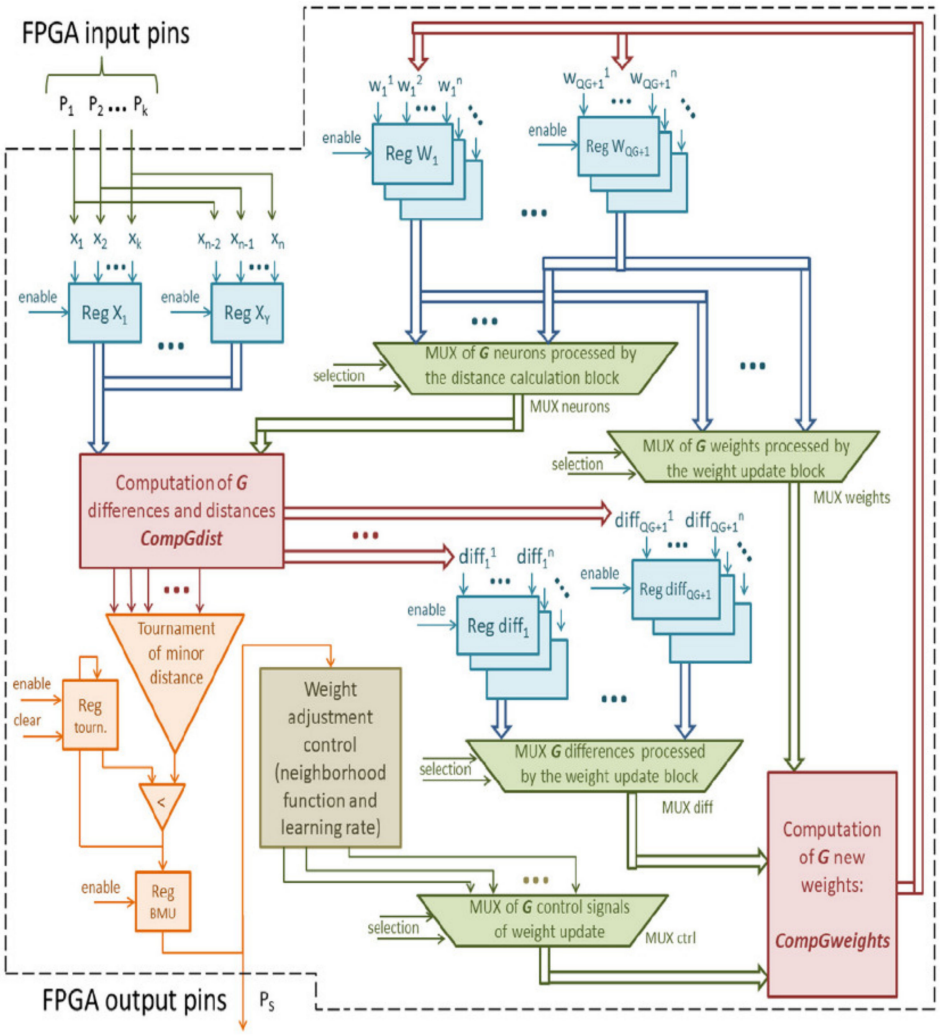

Figure 16. Schematic of Self-Organizing Maps [52] (Reprinted with permission from ref. [52]. Copyright 2021 Elsevier).

Based on the diagram above, another element is a register circuit that is used for temporary storage of intermediate computed values so that pipelined processing architecture can be utilized in the execution. This technique aims to achieve the speed requirements of high-performance applications. Another part of SOMs is the admission of the input vectors and identification of BMU, where almost all the computation occurs. Besides that, the weight adjustment control block is used to control the mode of the SOM and also to generate the weight update signals. The control unit is responsible for data transfer across the entire architecture [52]. As a conclusion, the proposed architecture of SOMs was able to achieve a higher magnitude order relation than software implementation of the same order.

Extreme learning machine (ELM) is another method that is commonly used in classification accuracy and speed. ELM is typically used in single layer feedforward neural (SLFN) networks [56]. The benefit of using ELM is that it does not need to adjust all the parameters randomly; it can automatically find the best behavior. In this research, the fixed-point and QR composition techniques are used again to reduce the FPGA resources. The main computations are matrix inversion, matrix transposing and matrix multiplication. QR decomposition (QRD) is applied for matrix inversion with three different algorithms: Gram–Schmidt, householder reflection and given rotation. Different computational techniques are used for the training, which used fixed point arithmetic [57]:

- ELM1: Using a serial computation procedure for a square matrix. The advantage is to share arithmetic units and reduce logic occupation;
- ELM2: Is the improvement of previous versions by using parallelization. Double up the arithmetic units to compute the arithmetic concurrently;
- ELM3: A rectangular matrix decomposition is used; basically, some of matrix multiplication state is removed from the block and the control unit is modifying for data addressing and flow into computation units.

The main objective for developing a hardware implementation in embedded systems is due to the ability of parameterizable, accurate and balance performance compared to software platform. Based on the research, QR decomposition and modified Gram–Schmidt methods were able to cope with matrix inversion techniques, since this technique optimizes resource usage in embedded systems and stability [58]. Furthermore, the train module, artificial neural network (ANN) module, RAM memories for storage and data flow will be the four main hardware structure blocks for the expert system.

The entire block diagram consists of a general control unit, basic arithmetic unit, RAM address generation, RAM for temporary storage and matrix operation state machine. In addition, the four main blocks for the arithmetic unit will be based on vector multiplication, root square, division and multiplication subtraction. Based on the three different architecture approaches, ELM2 consumes 33% less RAM resources compared to ELM1, and ELM3 and ELM1 consume fewer DSP blocks due to sequential operations. Lastly, by applying the QR decomposition and pipelined architecture, it can save up to twenty-two times power consumption with similar performance as compared to the traditional ELM method [57].

4. Embedded Intelligence (EI) Applications

In this section, multiple embedded intelligence applications are reviewed and investigated. From the research, it is possible to see that FPGA is widely used in computer vision applications such as facial recognition, object detection and enormous vehicle detection, etc. All the applications stated above are required to achieve computational power efficiency while maintaining the accuracy of the system.

4.1. Object Detection, Posture Recognition and Facial Recognition

Computer vision has been emerging with the development of deep learning. The deep learning method is proposed for facial expression recognition from images and setting of neural networks with FPGA implementation. The Zynq-7000 All Programmable SoC Video

and Imaging Kit was chosen for the architecture implementation. The proposed algorithm for the face recognition application and architecture, is built-up from eight parts.

The VITA-2000 camera module, which is used for video capturing of faces and emotion recognition, block processes the image data with trained with the CNN model. The AXI Lite and AXI buses data flows are controls by an ARM Processor, while Gigabit Ethernet is used as a bridge for communication between the ARM Processor and the host computer. The human faces that are extracted from the video stream, the trained weights, and the sub and final results of classification process are stored in the memory controller and DDR3. The data are stored sequentially in the memory. The AXI VDMA controller is in charge of the data transfer between CNN Core and Board Memory. The algorithmic steps for the CNN method are computed by the CNN core [59].

1. The facial features are calculated to detect faces in single frame.
2. The serialization of the face into a vector.
3. The result vector is calculated after passing the CNN Core.

The computation of softmax function and label will appear. The calculation for label of input image is performed in the recognition mode, and images passing the CNN core processor block will be labeled and shown on the screen. The CNN core is used to store the actual step able to optimize the BRAM memory usage.

The FPGA simulation (inference) is performed with Vivado HLS, which enables a number of built-in functions used to simulate the training and deployment of models in FPGA systems by extracting the IP core. The data are then passed through the processing block after the simulation is computed and if the result is satisfactory then the IP core will be used for implementation. The descriptions of the overall architecture of parameterized modules are listed in following. The VITA receiver, which comprises of a task of reading images or data from the camera where the addresses are spawned to read the input data in each clock cycle, and all the input data will be stored at scheduler and then pass the data to computation kernels like the image pipe. Lastly, the emotion recognition process is performed. The output data are stored in the RAM of the address generator in the DDR memory. The main scheduler of the system is the APU (application processing unit) Dual Core Cortex A9 + OCM (on-chip memory). To operate the system, the combination of APU Dual Core Cortex A9 + OCM, AMBA Switches and AXI Lite Interconnect is used. One or more AXI memory-mapped master devices are connected to one or more memory-mapped slave devices by AXI Lite Interconnect. The processed data will then transfer to HDMI output block to presented on the screen once the process is completed [59].

The main target in the training process is to train the neural network to learn optimal parameters itself, and the data are classified into seven classes. No overfitting phenomenon is observed for the model, as determined by the dropout that followed the layers in the neural network. The FER dataset with 48×48 pixel grayscale images of faces is used to test the FPGA implementation of the CNN method [60]. With the facial images, about 60.3% recognition efficiency is achieved, using 200 MHz clock frequency for 400 faces/second throughputs.

Furthermore, there are other research works [61] to review about object detection in embedded intelligence. Computational time and probability of detection, and frames per second are the crucial parameters for the performance of detection and recognition. There are some methods which use deep learning, such as single shot detector (SSD), YOLO (You Only Look Once) and faster region CNN (FRCNN) [62]. These CNN models require high processing times, where Xilinx PYNQ Z2 board and Intel Movidius Neural Compute Stick (NCS), a portable and low-powered device together play a major role to speed up the process. Other than that, Intel Movidius NCS provides built-in API, allow researchers to deploy their application in a shorter timeframe. SDD makes use of CNN's pyramidal feature hierarchy, which provides a good balance between accuracy and speed. The image is used only once and creates a feature map. The convolutional is then applied to predict the bounding box and detect target objects [63]. The VGG-16 model per-trained is applied on MobileNet architecture to extract the feature maps. On the other hand, YOLO uses single CNN that predicts multiple bounding boxes and class probability at the same time.

The separate components of object detection are unified into a single neural network. The entire image is applied to assume each bounding box. For FRCNN, it uses a CNN (named the Region Proposal Network) to create the bounding boxes. The algorithm applies anchor boxes to change object detection with different aspect ratios, and go through numerous pooling layers to acquire the classifier image [64].

Object detection using PYNQ Z2 and NCS can be seen in the flow chart, where the NCS and USB camera are connected to the PYNQ Z2 board. The graph file is created on NCS, which uses it for object detection. The object detection then predicts the bounding boxes over the detected target objects. The bounding boxes are labelled accordingly. In term of frames per second (FPS) of the object detection model, SSD is the highest, with 3.98 FPS, while YOLO is the fastest for the computational time, with 12.68 ms. FRCNN is the most accurate, with 100% for probability of detection [61].

Next, flex sensors with a FPGA implemented artificial neural network (ANN) are applied to a smart chair sitting posture recognition system. With help from machine-learning technologies and novel sensors, the sitting posture recognition system has become an interesting topic due to a lot of working adults having bad sitting posture habits. Wrong sitting postures will cause a lot of unnecessary health problems. A low-complexity hardware system with six passive flex sensors is attached to a chair with 5 Hz of sampling rate, and a simple voltage divider is connected to each sensor. All the sensors will be outputting the analog voltage based on the resistance and the resistance is the response based on the deformation of the flex sensors that are mounted on the chair. The analog voltage is then converted to digital format by using the ADC board, which collects the analog voltage from flex sensors. A FPGA with machine learning algorithm is then used to process the digital signal [65].

A two-layer artificial neural network (ANN) is implemented as the classifier which categorizes seven different health-related sitting postures. The ANN has six inputs, which are the six flex sensors connected to the chair. The six inputs are connected to the internal nodes in the hidden layer, and then connect to the output layer. The output layer generates a classification result, which identifies seven sitting postures. The best combination of the algorithm in the application with 10-fold cross-validation is using two layers of ANN and 28 nodes in the hidden layer. The flex sensor that is used in the application is FS-L-0055-253-ST from Spectra Symbol with flat resistance of 25 k Ohm and bending resistance that can vary from 45 k to 125 k Ohms. A 5 V power supply is used for the voltage dividers, where the changes of resistance are converted to the change of voltages [65].

A total of 11 participants were used in the experiment to evaluate the performance of the smart chair sitting recognition system. The participant is required to sit on the smart chair with seven difference postures for 30 s on each posture, and data are recorded. Nine people are assigned to the first part of the experiment, which aims to train and evaluate as well as to cross-validate the ANN model, while another two people are to validate the generalization of the ANN model to prevent over-fitting. A total of 9791 samples are collected from each sensor, with 67% (which is 6527 samples) used for training the model and 33% (which is 3264 samples) for evaluation of trained floating-point model and quantized fixed-point model. The floating-point model achieved an accuracy of 97.78%, while the fixed-point model achieved an accuracy of 97.43%. The processing system has dynamic power consumption of 7.35 mW and a propagation delay of 8.714 ns, when the clock frequency and sampling rate are 5 Mhz and 5 sample/second respectively. The processing system uses the FPGA board of Spartan 6 XC6SLX9, with 75 slice of registers, 659 slices of LUTs, and 911 of Flip-Flop pairs. A low power consumption system is successfully designed and implemented with good accuracy by using a 9-bit fixed point model [65]. Figures 17 and 18 show the encoder–decoder neural network and the binary encoder–decoder neural network.

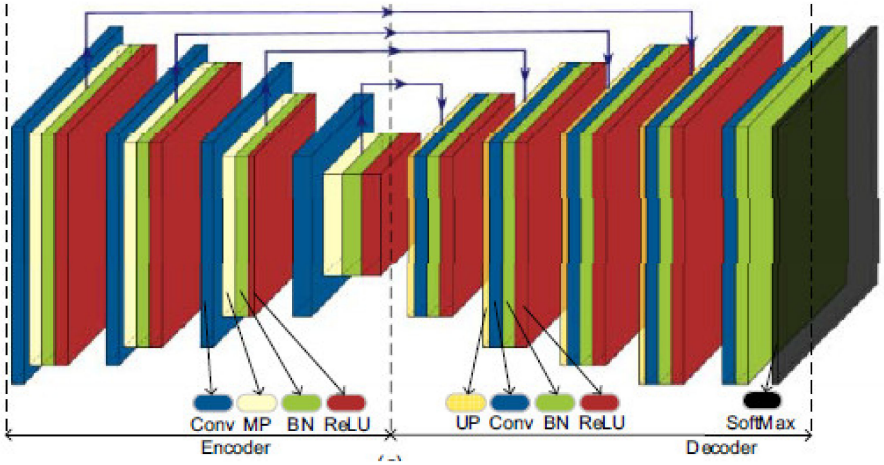

Figure 17. Encoder–decoder neural network [66] (Reprinted with permission from ref. [66]. Copyright 2021 IEEE).

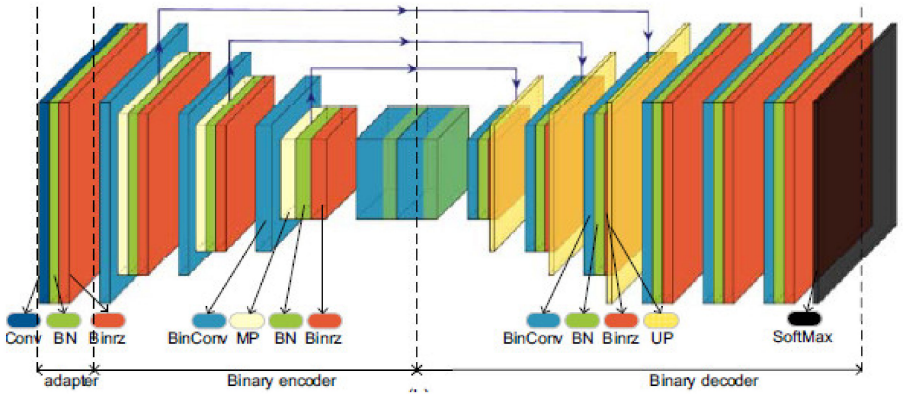

Figure 18. Binary Encoder–decoder neural network [66] (Reprinted with permission from ref. [66]. Copyright 2021 IEEE).

In addition, another example of object detection application is real-time scene text recognition. Autonomous driving systems make use of the scene text in images and videos for essential information. Road signs are extracted by the control. The overall architecture uses a FPGA-based accelerator, which includes a streaming pipeline architecture and ten blocks with three different types of adapter, encoder and decoder in the main BSEG logic. Figure 19 shows the overall architecture for this application.

Those blocks have a process element (PE) array, and the PEs perform operations such as convolution, binaries convolution, max pooling and more by using the compute-in-memory architecture. The FPGA platform from Xilinx VCU118 board, which includes an XCVU9P FPGA, is used to design the RTL-level. For system-level verification, Synopsys VSC and Verdi are applied, while for RTL synthesis and implementation, Vivado 2018.2 is used. The experimental results show that a throughput of 307.7 fps with interval of 3.25 ms and a latency of 3.875 ms are achieved with the trained binary quantization and computing-in-memory architecture. The FPGA-based encoder–decoder accelerator consumes 8.95 W of dynamic power, equivalent to energy efficiency of 351.7-GOPs/W. In conclusion, the real-time recognition system with FPGA was able to improve 14, 19 and 3 times better in term of area on chip usage whereas 57, 24 and 8 times more energy efficient when compared to other models that use GPU as reference [66].

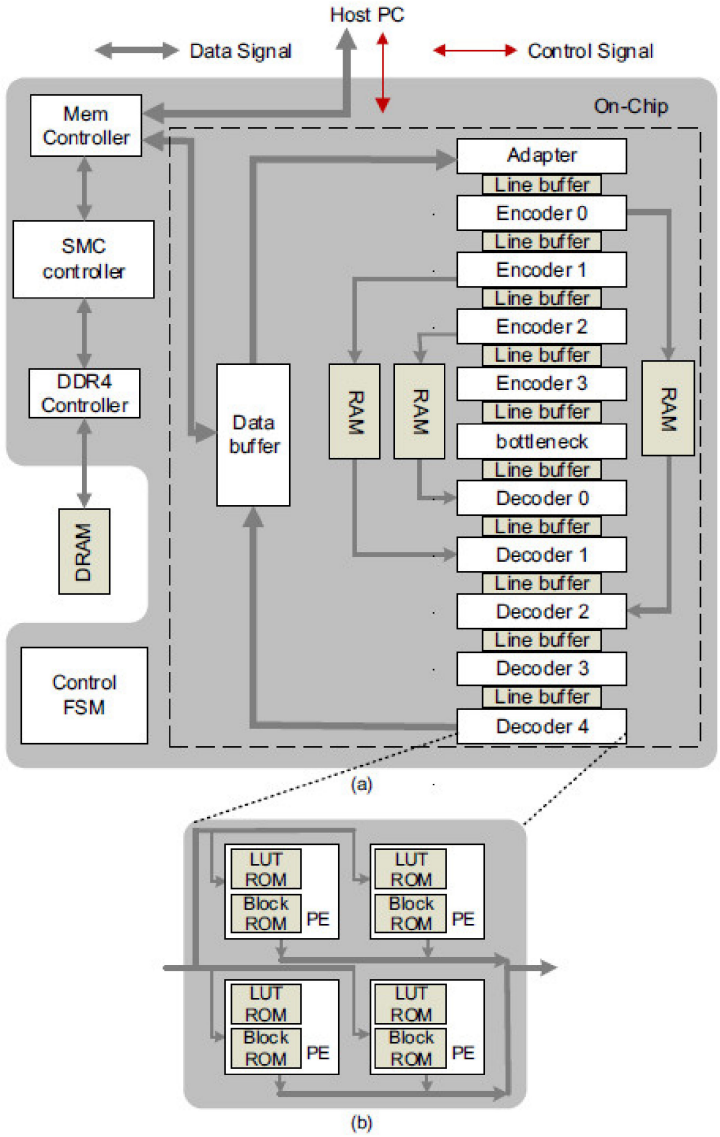

Figure 19. FPGA accelerator hardware framework consisting of AXI master interface, Vivado IPs and processing logic [66]: (**a**) The top-level system is acted as an AXI master to read/write data; Dual-port RAMs are used to store the pooling index; (**b**) The computing-in-memory blocks: the LUT and Block ROMs are used to store weight and BN-Binz reference parameters. Noted that the PE number in different blocks is different (Reprinted with permission from ref. [66]. Copyright 2021 IEEE).

4.2. Indoor Localization and Surveillance Monitoring

FPGA have been widely used in some other applications like surveillance monitoring, localization, etc. Location based service (LBS) is important with the current emerging Internet and mobile devices. Indoor localization is challenging due to radio propagation environments such as shadowing, multipath effect, delay distortion and fading [67]. To have high accuracy and efficiency of the LBS for indoor environments, fingerprint-based indoor localization is introduced by using scalable deep neural network (DNN) architecture implement together with denoising auto-encoder (SDNNLoc) based on FPGA [68]. Denoising autoencoders encode the data so that the fingerprint database can be updated in real-time. The indoor localization uses WiFi RSS fingerprints as the overall framework, which comprises of three stages. The first stage is the offline database-building phase where an interested location area is defined for the collection of offline RSS fingerprints and building of baseline fingerprint database [69]. The next stage is online location prediction and target tracking phase. The locations of the targets are extracted and the modification

of denoising autoencoder algorithm and sampling optimization is implemented for the target motion trajectory. Lastly, the fingerprint is updated online and the positioning phase is where the target movement tracking restores the fingerprint database.

FPGA is selected as hardware accelerator over ASICs due to its flexibility and fast implementation time. When compared with the power consumption to CPUs and GPUs, the FPGA uses less power. A pipelined reconfigurable architecture is proposed with the full utilization of reconfigurable interconnect resources on die and reconfigurable functional blocks of the FPGA chip. The DNN model file featured with hyperparameters are compiled on a host computer and is dynamically mapped to the FPGA. The scalability of the architecture is explained with the use of all cell nodes of a DNN by letting a single worker run its computational load, or concurrent workers are distributed among the layers, thus the data transportation delay is hidden from one layer to the next. The distributed process is performed at a desired grain, and makes the architecture suitable for the multibuilding indoor localization situations. The complexity of computation is decreased tremendously. The hardware that used for fingerprint-based indoor localization is GeForce 1080 Ti GPU for the computing server implementation, and Keras-2.2.5 (with Tensorflow-gpu-1.20) to train the deep learning model using Python-3.6.6 [70]. Two university buildings are selected for the system deployment with over 100 WiFi access points (APs) installed to collect data for evaluation. 59 APs are fully accessed via the collection of data from four participants with the Android App to collect the fingerprints on different Android mobile phones. A server is used for uploading of fingerprints for dataset processing. The performance evaluation is focused on the runtime of the SDNNLoc and compared with two other baseline systems, which are Horus [71] and DeepFi [72]. The SDNNLoc system is more accurate than the other two, with fewer location errors. The usage of the DNN method introduces noise interference to the input datasets, which avoids the over-fitting problem for training. The running time per location for the SDNNLoc is almost the same as the DeepFi system, where both use the DNN based technique. However, Horus uses the simple computation complexity of Bayesian inference method, thus having better performance than SDNNLoc. The implementation of FPGA as compared to the previous original version of SDNNLoc is faster by 5.8 times.

Another application with a convolutional neural network is implemented together with FPGA in the advanced driver assistance system (ADAS) and video surveillance analytics application, for their energy efficiency and computational processing speed [73]. Sixteen channels can be processed by the system with continuous input of video of 1080 p resolution. The ADAS system can be used on vehicles, for the detection of pedestrians, for tracking purposes or for attributes analytics, while for video surveillance it is implemented in face detection and recognition. Xilinx MPSoC ZU9 is selected as the FPGA hardware. SSD and Densebox are used for the deep learning algorithm with the highest accuracy.

The ADAS and video surveillance analytics systems using deep learning algorithms on FPGA demonstration system has three main features that show the fast speed, high flexibility and low power consumption of the deep learning processing with FPGA. The first is where FPGA board is used with deep learning algorithms to process the 16 channels of 1080 p videos and the results are shown on the monitor. Level-3 ADAS functionalities can be made possible where the cameras have a surrounding view of the vehicle. Next, the facial recognition in the conference hall is performed by implementing the cameras to capture the real-time video, and then FPGA is used to analyze and display the result on the screen [73]. The deploying of the CNN network architecture can be made in several seconds for the FPGA system, as the customized design of the deep learning processing unit that sped up the CNN models.

4.3. Other EI Applications

With the implementation of deep learning techniques together with FPGA in radar automatic target recognition, it is shown that the high classification accuracy can be retained; besides that, the execution speed, power consumption, and memory requirements are

enhanced as well. The embedded application of the millimeter-wave (mmW) radar-based human activity classification applies an acceleration method of the convolutional neural network (CNN) on the FPGA hardware. The characterization of time-frequency is used to represent the human micro-Doppler (mD) signals, where micromotions of targets causes the frequency modulation in radar echoes. Human activity classification makes use of the time-frequency that representation of mD signals because of the uniqueness for different targets with micromotions [74]. The mmW radar is appropriate to be used due to the short wavelength where it benefits the mD measurement in time-frequency domains by causing wide dynamic range of mD frequency corresponding to human gait patterns. Frequency-modulated continuous wave (FMCW) waveforms with sawtooth-shaped frequency sweep are analyzed and used as an input to the CNN [75].

The classification which applied by CNN from the network architecture viewpoint includes the convolutional layer (CONV), pooling layer, activation function, fully connected layer (FC), and classifier. The successive groups of these layers include some linear and nonlinear processing, which is performed for the input spectrogram to map with the one-dimensional feature space. The classifier will handle all the prediction and forecast based on the data that is trained. Backpropagation in training and forward propagation in test are the two directions for the information flow in CNN. A series of quantization strategies are applied to various types of data that pass into the CNN model, as due to the consideration of bit width constraints of the FPGA, where FPGA works in fix-point mode, and limitation of computation resources and on-chip memory. Due to consideration of reasonable compromise of precision and hardware efficiency [76], the feature maps and weight parameters are using 16-bit fixed-point number representation.

The execution speed is improved in forward propagation through the network by the computation procedure of CNN and the pipelining technique on FPGA. The computational implementation included the channels and layers of the CNN with several parallel processing schemes that would reduce the waiting time and caching time indirectly and increase the speed of the network on FPGA. There are various parallel processing strategies that are proposed for the data flow and computation patterns in the network model. Firstly, the interchannel parallel computing where each channel is independently applied for those two- or three-dimensional convolution kernels, and the same input data shared among the multiple channels, within a convolutional layer [75]. The multichannel convolution between kernels and input data are executed in parallel.

The convolution group parallel computing, which explains the output of previous convolutional layers, is identical with the dimensions of input of pooling manipulation that are induced by the activation results. All caches before the pooling are not required and the data cluster is transferred in parallel to save the time of operation in a single channel. The removal of the softmax step could benefit the computation reduction, as when the dimensionality of the last fully connected layer output is 4×1, the classification decision can be optimized by the logic, which clearly benefits the FPGA that consists of a lot of logic gates [75].

Secondly, there is in-channel parallel computing where input data determines the depth of kernels in each channel of convolutional layer. For the in-channel convolution, two parallel schemes (which are data parallelism and computation parallelism) are implemented. In addition, interlayer parallel computing also can be viewed as a time series structure from the overall viewpoint of the CNN model. The human target-generated spectrogram of mmW radar mD signals is transferred and handled sequentially between layers [75]. The execution of some layers is not started until the previous one is completed, as according to the computation patterns in CNN. Pipelining is used as a technique for CNN related application as it computes all the functions in parallel.

The proposed method is validated with a variety of human activities measured using mmW radar data. For example, a Texas Instruments AWR1443 automotive radar sensor is used where it operated at about 77 GHz, with 1.536 GHz of effective bandwidth of radar signals. Nine individuals were used to carry out the measurement repeated at 3 m in

front of the mmW radar. The nine persons are required to perform four actions including walking, jogging, jumping and walking with a stick, with two aspect angles of human gaiting direction such as 0 and 90 degrees. There are in total more than 11,000 items in the recorded dataset that is collected along the measurements. The recording duration is approximately 2.1 s. Besides that, some of the measured mmW radar data from seven out of nine persons performing nine actions in place are used for the generation of a training dataset for the learning parameters of the CNN model. The remaining data of two persons is used for the test dataset. The Xilinx Zynq XC7Z045 board is selected as the system platform to validate the proposed acceleration method for the performance assessment. By implementing both interlayer and convolutional group parallelism in the overall network perspective, more efficient usage of caches can be achieved, where no cache is needed for max pooling layers. The entire CNN network and every convolution layer are accelerating on the Xilinx FPGA device. In this application, the pipeline processing and multiple parallelism strategies show that there exists much runtime overlap between layers; thus, it is faster than the serial computing of the CNN execution. When compared to the NVDIA GPU, the FPGA with the proposed acceleration method is 30.42% faster in the averaged execution time of CNN in test, and a 0.27% reduction in the classification accuracy is seen with 87.81% [75].

5. FPGA Hardware and Platforms

Based on the market share research, Xilinx is the biggest FPGA manufacturer with 3.06 billion as of the 2019 annual reports [99], followed by Intel, who owned 1.987 billion in the same year [100]. Other than these top players on the FPGA, there are some other FPGA manufacturers such as Lattice Semiconductor and Archonix Semiconductor [101]. Xilinx, Intel and Archonix offer high performance FPGA platforms which include the ARM Processor. For GPU and even machine learning processor examples, Intel are offering VPU mainly target image processing. Table 2 shows a summary of FPGA hardware platforms.

Table 2. Summary Table of FPGA platforms for Intel, Xilinx and Archonix.

FPGA	Series	Company	(KLEs)	DSP Slices	Description
Agilex	AGF027	Intel	2692	8528	Quad core ARM Cortex-A53, PCIe Gen5
Agilex	Agilex M	Intel	-	-	-
Stratix 10	DX 2800	Intel	2753	5760	Quad-core ARM Cortex-A53 HPS, HBM2 16G Intel Optane DC Persistance Memory
Stratix 10	GX 2800	Intel		3456	DDR4
Arria 10	GT1150	Intel	1150	1518	DDR4
Arria 10	SX660	Intel	660	1688	Dual Core ARM Cortex-A9 MPCore, DDR4
Cyclone 10	10CX220	Intel	220	192	DDR3
MAX 10	10M50	Intel	50	-	-
Virtex UltraScale+	VU19P	Xillinx	8938	3840	DDR4, server Class DIMM
Virtex UltraScale+	HBM VU57P	Xillinx	2852	9024	DDR4 Server Class DIMM
Kintex Ultrascale+	KU19P	Xillinx	1843	1080	DDR4
Artix-7	XC7A200T	Xillinx	215	740	DDR3
Spartan-7	XC7S100	Xillinx	102	160	DDR3
Spartan-6	XC6SLX150T	Xillinx	23	180	-
ZynQ	Z-7020 XC7Z020	Xillinx	85	220	Dual-Core ARM Cortex-A9 MPCore, DDR3
Zynq UltraScale+	ZU7EV	Xillinx	504	1728	Quad-Core ARM Cortex-A53 MP Core, Dual-core ARM Cortex-R5 MPCore and Mali 400 MP2 GPU, DDR4
Zynq UltraScale+	ZU7CG	Xillinx	504	1728	Dual-Core ARM Cortex-A53 MP Core and Dual-core ARM Cortex-R5 MPCore, DDR4
Speedster7t	AC7t6000	Achronix	2600	1760@MLP	MLP(Machine Learning Processor),GDDR6, PCIe Gen5

5.1. Xilinix

The table above shows that Xilinx offer a wide range of FPGA platforms starting from low cost Spartan 6 with 45 nm technology, Spartan 7 with 28 nm technology, up to high end solutions such as Virtex Ultrascale [77]. At the same time, Xilinx also offer a hybrid version of platform that merges ARM processors with FPGA where ARM is used to handle the communication with external processing and FIFO, while FPGA focuses on parallel computation. Those platforms are known as Zynq series, a high end version of which cooperate with the GPU to boost performance. Some example of application known as Twitch is a live video streaming company which offer high quality broadcast video is using Xilinx Ultrascale FPGA platform for hardware acceleration [78]. With this hardware acceleration platform, it was able to improve the overall performance by thirty times as compared to CPU implementation.

Another research done by [59] is using low cost computational FPGA Soc Zynq-7000 as platform for hardware accelerator which prove that it able significantly improve the real-time face recognition application. The architecture is utilizing ARM processor for all the communication and data transferring, whereas FPGA will act as a core for CNN computation for the overall architecture when using Zynq–7000.

Furthermore, another framework (known as DarkFPGA) makes assessments using the Maxeler MAX5 platform that is built up with Xilinx Ultrascale+ VU9P FPGA. At the same time, three 16 GB DDR4 RAM are implemented in the platform as off-chip memory with a maximum bandwidth of 63.9 GB/s and 200 MHz frequency is used for the hardware accelerator. Furthermore, 8-bit integers and sum square error loss function (SSE) are used to train the network [79]. The key features for the framework consist of a scalable accelerator architecture, a software definable hardware accelerator and an optimization hardware design tool.

Researchers have compared the above with other platforms, such as GPU and CPU. The summary of the result shows that FPGA achieves over 200 times speedup over CPU based implementation, but is 2.5 times slower when compared to GPU architecture in terms of overall performance. If power consumption is taken into consideration, FPGA is 6.5 times more energy efficient when compared to GPU.

Another research on independently recurrent neural networks (RNN) on FPGA based accelerator uses the Xilinx Zynq zc706 FPGA platform. Different blocks inside the accelerator multiply the units required, e.g., 128 MAC units to calculate 16-bit signed multiplications and 32-bit signed accumulations. When the overall performance of the architecture is compared Intel CORE i5 CPU with 2 GHz working frequency, it is seen that that FPGA is 26 times faster in performance than Intel CORE i5. A reason is that FPGA is using a 16-bit fixed point while Intel is using 32-bit floating point. In the paper, it is shown that by running at the clock frequency of 125MHz it was able to become seven times more energy efficient than the LSTM reference for similar performance [81].

5.2. Intel (Altera)

Besides Xilinx, Intel is also another key player for FPGA platform in embedded intelligence. Intel offers a wide range of FPGA chips starting from MAX 10, Cyclone 10 until Arria and Agilex. Intel provides two options for Arria and Agilex chips that either purely focus on FPGA or work in combination with ARM SoC. At the same time, Intel also offers other platforms to speed up the growth of AI such as vision processing units (VPUs), which support the demand of computer and edge AI workloads with higher efficiency and shorten the development time by using the OpenCL tool. Besides that, Intel also invented a platform known as Neural Compute Stick 2 by utilizing VPU technology, which allows developers to fine tune and deploy a CNN on low power applications that require real-time monitoring.

VK, which uses the Intel FPGA platform, is one of the largest social networks in Russia. Intel Arria 10 together with Intel PAC are hardware accelerated processing servers that provide quick image conversion and storage requirements [82]. When a user requests an

image, all images will be converted by the CTAccel image processor. The image conversion is able to boost performance as only single resolution images are required by the system. The original method that used customer in data, storage whereas by implementing FPGA for image compression it is able to improve the overall dataflow [82].

For another work, a hybrid CPU–FPGA based computing platform for machine learning is proposed by [83] to implement the LeNet-5 machine learning algorithm. The proposed solution indication that the training phase is handled by the GPU, whereas FPGA is used for the inferencing phase due to FPGA consisting of logic circuits that are able to fully optimize the performance of the hardware accelerator. In this Mid-range Arria, FPGA is adopted as each inference only requires a small quantity of data and less frequent communication with with host. Based on the research, the FPGA Arria 10 speed is six times faster than GPU and 44 times faster than CPU in the inferencing process [83].

Another novel FPGA accelerator design for real-time application with ultralow power CNN uses Intel Arria 10 FPGA. Titan X GPU is a reference platform for comparison and the framework is YOLO V2 [84]. The overall result shows that FPGA is able to achieve 3.3 times more power efficiency than Titan X GPU, whereas it has 418 times more power efficiency than CPU. The overall hardware architecture uses the CNN Model of AlexNET and VGG16 as the standard for comparison. It uses a clock frequency of 200 MHz that consists of 1366 BRAM and 410 DSP. Besides that, it consists of 360 K LUTs with 523.7 K FIFO to allow temporary storage of data to minimize data transfer to the external memory. With this implementation, Arria 10 is able to save 20% of power, thus making it suitable to use in EI related applications such as road object recognition. In conclusion, large, reconfigurable processing elements with low power consumption have make FPGA widely used recently due to the rapid growth in demand in AI, so that 92% of the execution time is spent on MAC computations during forward inference in YOLO framework [85].

5.3. Achronix

Another FPGA company that is growing rapidly is Achronix. The proposed FPGA heterogeneous computing platform uses the master–slave model. The master issues commands to the slave to process the data according the compiled kernel program through PCIe. The processed data from the device's DDR will be read by the host.

Achronix SPEEDSTER 7 t which uses a 7 nm process, focuses on three main features; 2D network on chip (NoC), high-speed interfaces and machine learning processors, which enable Achronix to gain the 5G infrastructure, computational storage, test and measurement, AI/ML, networking and compute acceleration market. It also provides the VectorPath Accelerator card with ACE design software. Notably, only Achronix offers its FPGA with GDDR6 memory to deliver 4Tbps memory bandwidth. Its machine learning supports fully factorable integer multiplier/accumulators to speed up the AI/ML application [86].

6. Challenges

The rapid development of embedded intelligence (EI) for machine learning applications such as image processing, surveillance monitoring, etc. is causing the system to become more complex. Neural network models are also growing larger and larger, which is expressed in the algorithmic computation of significant numbers of model parameters. Therefore, the comparison between CPU, GPU and FPGA had come into the picture in term of hardware resources, power consumption, design difficulty and accuracy [87]. There are several challenges which need to be addressed and overcome for practical use of the FPGA-based EI which has been identified and discussed in the earlier sections.

The most critical challenges for CPU–EI implementations are the slow computational speed with high power consumption making a nonideal platform for embedded intelligence. Typically, hybrid accelerators can be designed for CPU due to the simplicity in development and data transfer between external memory and other blocks. In addition, GPU platforms have been widely researched in the improvement of computer perfor-

mance and power efficiency, as the research shows that GPU is too power-intensive a solution to compute the equivalent network weights compared to CPU and FPGA solutions. Thus, GPU has difficulties for application in embedded intelligence systems that require slightly larger networks because low power consumption is an important priority for EI applications.

FPGA-based EI solutions are developing to become an ideal candidate in hardware accelerators for energy efficient neural network applications instead of solely relying on software solutions. Parallelism capability, flexibility and high performance with low energy consumption have become the key selling point for FPGA platforms. Based on the comparison, FPGA platforms can reach 10 times faster than CPU platforms, the lowest level having same performance as GPU. Furthermore, the ability of reconfigurability makes FPGA able to change the design frequently. The capability of reconfigurability has brought some challenges for researchers and developers due to its cost. Although it brings a lot of advantages on flexibility and high parallelism [88] it requires a lot of time in reconfiguration. FPGA reconfigurability can be classified into two types of reconfiguration, which are static and dynamic reconfiguration. Static reconfiguration refers to the compilation time, whereas dynamic reconfiguration is known as runtime reconfiguration, which potentially causes the execution delay and increases the run time.

In addition, challenges in programming are another increase in cost [88]. It requires very intensive resources to do hardware programming and it is very difficult for engineers or programmers to master the development cycle. Besides that, multiple FPGAs are required to compute large networks, thus the communication between blocks has been a challenge for researchers. At the same time, limits on chip memory are also a bottleneck for FPGA, thus, managing the available system resources to achieve high performance is also one of the challenges for researchers. In order to achieve this, there are many expectations and much research on different optimization techniques such as data optimization, access optimization and weight optimization. FPGA do not have built-in cache structure and the bandwidth typically is two generations older than high end GPU [89]. For example, a GPU Titan X uses GDDR5 but Arria 10 only uses DDR3. In summary, FPGA is a good platform as a hardware accelerator in embedded intelligence, however, it does have several drawbacks and challenges that need to be overcome. Researchers are putting more focus on improving it so that FPGA can have the most optimum performance in Embedded Intelligence applications with low power consumption. Another challenge is the development of automated tools to convert EI models (e.g., DNNs) to FPGA implementations. Some useful and illustrative works which have been performed for automated software tools can be found in [95–97]. The authors in [95–97] proposed automated tools (termed fpgaConvNet, CNN2Gate and Caffeine, respectively), which allow the mapping of CNN models onto FPGA hardware.

7. Conclusions

This paper has given a comprehensive overview of EI techniques, applications and various hardware platforms for FPGA-based architectures and implementations. For successful and practical FPGA solutions for EI deployment, the paper has also identified and discussed various challenges to be addressed and overcome, such as high computational processing, low energy/power consumption or high energy efficiency, and scalability considerations to accommodate different network sizes and topologies. As some starting directions and investigation points for researchers to overcome the various challenges, the paper has also discussed some useful techniques to address the challenges include pipelining, reducing data transfer latency by batching, reducing bandwidth requirements using fixed point precision, reducing memory usage using quantization, etc. The various discussions in the paper have shown that FPGA-based platforms are suitable for EI implementations and the acceleration of machine learning.

Author Contributions: Conceptualization, K.P.S. and L.M.A.; writing—original draft preparation, K.P.S., P.J.L. and L.M.A.; writing—review and editing, K.P.S. and L.M.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

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

Abbreviations

ADAS	Advanced driver assistance system
ANN	Artificial neural network
APU	Application Processing Unit
CCU	Configuration and control unit
CNN	Convolutional neural network
COG	Center of Gravity
COM	Center of Maximum
CPU	Central Processing Unit
CRBM	Conditional restricted Boltzmann machine
DMA	Direct memory access
DNN	Deep neural network
EI	Embedded Intelligence
ELM	Extreme learning machine
FIFO	First in first out
FPGA	Field Programmable Gate Array
GEMM	General matrix multiplication
GPU	Graphics Processing Unit
IIoT	Industrial Internet of Things
LBS	Location based service
LSM	Local shared memory
MAC	Multiply accumulate unit
mD	Micro doppler
MF	Membership function
MM	Memory management
MOM	Mean of Maximum
MST	Minimum spanning tree
OCM	On-chip memory
OPF	Optimum path forest
PE	Processing element
QRD	QR decomposition
RCU	Reconfigurable computing unit
RBF	Radial basis function
ReLU	Rectified linear unit
RNN	Recurrent neural network
SIMD	Single instruction multiple data
SGD	Stochastic gradient descent
SLFN	Single layer feedforward neural network
SOM	Self-organizing map
SVM	Support vector machine
VPU	Visual processing unit
YOLO	You only look once

References

1. Goodfellow, I.; Bengio, Y.; Courville, A. Machine learning basics. In *Deep Learning*; MIT Press: Cambridge, UK, 2016; Volume 1, pp. 98–164.
2. Parpinelli, R.S.; Lopes, H.S. New inspirations in swarm intelligence: A survey. *Int. J. Bio Inspired Comput.* **2011**, *3*, 1–16. [CrossRef]
3. Liu, S.; Liu, L.; Tang, J.; Yu, B.; Wang, Y.; Shi, W. Edge computing for autonomous driving: Opportunities and challenges. *Proc. IEEE* **2019**, *107*, 1697–1716. [CrossRef]

4. Trimberger, S.M.S. Three Ages of FPGAs: A Retrospective on the First Thirty Years of FPGA Technology. *IEEE Solid State Circuits Mag.* **2018**, *10*, 16–29. [CrossRef]
5. Wang, C.; Gong, L.; Li, X.; Zhou, X. A Ubiquitous Machine Learning Accelerator with Automatic Parallelization on FPGA. *IEEE Trans. Parallel Distrib. Syst.* **2020**, *31*, 2346–2359. [CrossRef]
6. Olson, C.B.; Kim, M.; Clauson, C.; Kogon, B.; Ebeling, C.; Hauck, S.; Ruzzo, W.L. Hardware Acceleration of Short Read Mapping. In Proceedings of the 2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines, Toronto, ON, Canada, 29 April–1 May 2012.
7. Lopes, F.F.; Ferreira, J.C.; Fernandes, M.A. Parallel implementation on FPGA of support vector machines using stochastic gradient descent. *Electronics* **2019**, *8*, 631. [CrossRef]
8. Kara, K.; Alistarh, D.; Alonso, G.; Mutlu, O.; Zhang, C. FPGA-accelerated dense linear machine learning: A precision-convergence trade-off. In Proceedings of the 2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), Napa, CA, USA, 30 April–2 May 2017; pp. 160–167.
9. Rabieah, M.B.; Bouganis, C.-S. FPGA based nonlinear support vector machine training using an ensemble learning. In Proceedings of the 2015 25th International Conference Field Programmable Logic and Applications (FPL), London, UK, 2–4 September 2015; pp. 1–4.
10. Bottou, L.; Lin, C.J. Support vector machine solvers. In *Large Scale Kernel Machines*; MIT Press: Cambridge, MA, USA, 2007; Volume 3, pp. 301–320.
11. Gilbert, E.G. An Iterative Procedure for Computing the Minimum of a Quadratic Form on a Convex Set. *SIAM J. Control* **1996**, *4*, 61–88. [CrossRef]
12. Diniz, W.F.; Frémont, V.; Fantoni, I.; Nóbrega, E.G. An FPGA-based architecture for embedded systems performance acceleration applied to Optimum-Path Forest classifier. *Microprocess. Microsyst.* **2017**, *52*, 261–271. [CrossRef]
13. Spadotto, A.A.; Pereira, J.C.; Guido, R.C.; Papa, J.P.; Falcao, A.X.; Gatto, A.R.; Cola, P.C.; Schelp, A.O. Oropharyngeal dysphagia identification using wavelets and optimum path forest. In Proceedings of the 2008 3rd International Symposium Communications, Control and Signal Processing, St Julians, Malta, 12–14 March 2008; pp. 735–740.
14. Farabet, C.; Poulet, C.; LeCun, Y. An fpga-based stream processor for embedded real-time vision with convolutional networks. In Proceedings of the 2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops, Kyoto, Japan, 27 September–4 October 2009; pp. 878–885.
15. Jakšić, Z.; Cadenelli, N.; Prats, D.B.; Polo, J.; Garcia, J.L.B.; Perez, D.C. A highly parameterizable framework for Conditional Restricted Boltzmann Machine based workloads accelerated with FPGAs and OpenCL. *Future Gener. Comput. Syst.* **2020**, *104*, 201–211. [CrossRef]
16. Kim, S.K.; McAfee, L.C.; McMahon, P.L.; Olukotun, K. A highly scalable restricted Boltzmann machine FPGA implementation. In Proceedings of the 2009 International Conference on Field Programmable Logic and Applications, Prague, Czech Republic, 31 August–2 September 2009; pp. 367–372.
17. Yoon, Y.H.; Hwang, D.H.; Yang, J.H.; Lee, S.E. Intellino: Processor for Embedded Artificial Intelligence. *Electronics* **2020**, *9*, 1169. [CrossRef]
18. Struharik, R.J.; Vukobratović, B.Z.; Erdeljan, A.M.; Rakanović, D.M. CoNNa–Hardware accelerator for compressed convolutional neural networks. *Microprocess. Microsyst.* **2020**, *73*, 102991. [CrossRef]
19. Luo, J.H.; Wu, J.; Lin, W. Thinet: A filter level pruning method for deep neural network compression. In Proceedings of the IEEE International Conference on Computer Vision, Venice, Italy, 22–29 October 2017; pp. 5058–5066.
20. Chen, Y.H.; Krishna, T.; Emer, J.S.; Sze, V. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE J. Solid State Circuits* **2016**, *52*, 127–138. [CrossRef]
21. Anwar, S.; Hwang, K.; Sung, W. Structure pruning of deep convolutional neural networks. *Emerg. Technol. Comput. Syst.* **2017**, *13*, 1–18. [CrossRef]
22. Véstias, M.; Duarte, R.P.; de Sousa, J.T.; Neto, H. A Fast and Scalable Architecture to Run Convolutional Neural Networks in Low Density FPGAs. *Microprocess. Microsyst.* **2020**, *77*, 103136. [CrossRef]
23. Shen, Y.; Ferdman, M.; Milder, P. Escher: A CNN accelerator with flexible buffering to minimize off-chip transfer. In Proceedings of the 2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), Napa, CA, USA, 30 April–2 May 2017; pp. 93–100.
24. Ding, W.; Huang, Z.; Huang, Z.; Tian, L.; Wang, H.; Feng, S. Designing efficient accelerator of depthwise separable convolutional neural network on FPGA. *J. Syst. Arch.* **2019**, *97*, 278–286. [CrossRef]
25. Chollet, F. Xception: Deep learning with depthwise separable convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017.
26. Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; Chen, L.C. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018.
27. LeCun, Y. Object Recognition with Gradient Based Learning. In *Shape, Contour and Grouping in Computer Vision*; Springer: Berlin/Heidelberg, Germany, 1999.
28. Hailesellasie, M.; Hasan, S.R.; Khalid, F.; Wad, F.A.; Shafique, M. FPGA-based convolutional neural network architecture with reduced parameter requirements. In Proceedings of the 2018 IEEE International Symposium on Circuits and Systems (ISCAS), Florence, Italy, 27–30 May 2018; pp. 1–5.

29. Hailesellasie, M.; Hasan, S.R. A fast FPGA-based deep convolutional neural network using pseudo parallel memories. In Proceedings of the 2017 IEEE International Symposium on Circuits and Systems (ISCAS), Baltimore, MD, USA, 28–31 May 2017; pp. 1–4.
30. Ma, Y.; Suda, N.; Cao, Y.; Vrudhula, S.; Seo, J.S. ALAMO: FPGA acceleration of deep learning algorithms with a modularized RTL compiler. *Integration* **2018**, *62*, 14–23. [CrossRef]
31. Ma, Y.; Suda, N.; Cao, Y.; Seo, J.S.; Vrudhula, S. Scalable and modularized RTL compilation of convolutional neural networks onto FPGA. In Proceedings of the 2016 26th International Conference on Field Programmable Logic and Applications (FPL), Lausanne, Switzerland, 29 August–2 September 2016; pp. 1–8.
32. Luo, C.; Sit, M.K.; Fan, H.; Liu, S.; Luk, W.; Guo, C. Towards efficient deep neural network training by FPGA-based batch-level parallelism. *J. Semicond.* **2020**, *41*, 022403. [CrossRef]
33. Moss, D.J.; Krishnan, S.; Nurvitadhi, E.; Ratuszniak, P.; Johnson, C.; Sim, J.; Mishra, A.; Marr, D.; Subhaschandra, S.; Leong, P.H. A customizable matrix multiplication framework for the intel harpv2 xeon+ fpga platform: A deep learning case study. In Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 25–27 February 2018; pp. 107–116.
34. Faraone, J.; Kumm, M.; Hardieck, M.; Zipf, P.; Liu, X.; Boland, D.; Leong, P.H. AddNet: Deep Neural Networks Using FPGA-Optimized Multipliers. *IEEE Trans. Very Large Scale Integr. Syst.* **2019**, *28*, 115–128. [CrossRef]
35. Demirsoy, S.S.; Dempster, A.G.; Kale, I. Design guidelines for reconfigurable multiplier blocks. In Proceedings of the 2003 International Symposium on Circuits and Systems, Bangkok, Thailand, 25–28 May 2003; Volume 4, p. IV.
36. Yu, H.J.; Chen, C.; Zhang, S.; Zhou, J.N. *Intelligent Diagnosis Based on Neural Network*; Metallurgical Industry Publishing: Beijing, China, 2000.
37. Xue, J.; Sun, L.; Liu, M.; Qiao, C.; Ye, G. Research on high-speed fuzzy reasoning with FPGA for fault diagnosis expert system. In Proceedings of the 2009 International Conference on Mechatronics and Automation, Changchun, China, 9–12 August 2009; pp. 3005–3009.
38. Mei, D.; Liu, Y. Design of a blowout expert control system based on FPGA. In Proceedings of the 2010 The 2nd International Conference on Computer and Automation Engineering (ICCAE), Singapore, 26–28 February 2010; Volume 4, pp. 402–405.
39. Duan, Y.H.; Yang, S. Fuzzy-PID control of stepping motor. *Comput. Simul.* **2006**, *2*, 290–293.
40. McKenna, M.; Wilamowski, B.M. Implementing a fuzzy system on a field programmable gate array. In Proceedings of the IJCNN'01. International Joint Conference on Neural Networks, Washington, DC, USA, 15–19 July 2001; Volume 1, pp. 189–194.
41. Mondal, S.; Chattapadhyay, P. Fuzzy Vs.neuro-fuzzy: Implementation on the reconfigurable FPGA system. In In Proceedings of the 2015 International Conference on Energy, Power and Environment: Towards Sustainable Growth (ICEPE), Shillong, India, 12–13 June 2015; pp. 1–5.
42. NI. Center of Maximum (CoM) (PID and Fuzzy Logic Toolkit). 2020. Available online: https://zone.ni.com/reference/en-XX/help/370401J-01/lvpidmain/center_of_max/#:~{}:text=In%20the%20Center%20of%20Maximum,the%20membership%20function%20was%20scaled (accessed on 3 July 2020).
43. NI. Mean of Maximum (MoM) (PID and Fuzzy Logic Toolkit. 2012. Available online: https://zone.ni.com/reference/en-XX/help/370401J-01/lvpidmain/mean_of_max/ (accessed on 20 August 2020).
44. Liviu, T. FPGA Implementation of a Fuzzy Rule Based Contrast Enhancement System for Real Time Applications. In Proceedings of the 2018 22nd International Conference on System Theory, Control and Computing (ICSTCC), Sinaia, Romania, 10–12 October 2018.
45. Mohammadi, M.; Shaout, A. Reconfiguratble Implementation of Fuzzy Inference System using FPGA. In Proceedings of the 2017 International Conference on New Trends in Computing Sciences (ICTCS), Amman, Jordan, 11–13 October 2017.
46. Van Broekhoven, E.; De Baets, B. Monotone Mamdani–Assilian models under mean of maxima defuzzification. *Fuzzy Sets. Syst.* **2008**, *159*, 2819–2844. [CrossRef]
47. Biswas, A.; Biswas, B. Swarm Intelligence Techniques and Their Adaptive Nature with Applications. *Comput. Intell.* **2014**, *319*, 253–273.
48. Eberchart, J.K.R. Particle Swarm Optimization. In Proceedings of the IEEE International Conference on Neural Networks, Perth, Australia, 27 November–1 December 1995.
49. Dorigo, C.M. Ant Colony Optimization Theory: A survey. *Theory Comput. Sci.* **2005**, *344*, 243–278. [CrossRef]
50. Li, D.; Huang, L.; Wang, K.; Pang, W.; Zhou, Y.; Zhang, R. A General Framework for Accelerating Swarm Intelligence Algorithms on FPGAs, GPUs and Multi-Core CPUs. *IEEE Access* **2018**, *6*, 72327–72344. [CrossRef]
51. Calazan, R.M.; Nedjah, N.; Mourelle, L.M. A hardware accelerator for particle swarm optimization. *Appl. Soft Comput.* **2014**, *14*, 347–356. [CrossRef]
52. De Sousa, M.A.D.A.; Pires, R.; Del-Moral-Hernandez, E. SOMprocessor: A high throughput FPGA-based architecture for implementing Self-Organizing Maps and its application to video processing. *Neural Netw.* **2020**, *125*, 349–362. [CrossRef] [PubMed]
53. Appiah, K.; Hunter, A.; Dickinson, P.; Meng, H. Implementation and applications of tri-state self-organizing maps on FPGA. *IEEE Trans. Circuits Syst. Video Technol.* **2012**, *22*, 1150–1160. [CrossRef]
54. Lachmair, J.; Mieth, T.; Griessl, R.; Hagemeyer, J.; Porrmann, M. From CPU to FPGA—Acceleration of self-organizing maps for data mining. In Proceedings of the 2017 International Joint Conference on Neural Networks (IJCNN), Anchorage, AK, USA, 14–19 May 2017; pp. 4299–4308.

55. De Sousa, M.A.D.A.; Del-Moral-Hernandez, E. Comparison of three FPGA architectures for embedded multidimensional categorization through Kohonen's Self-organizing maps. In Proceedings of the 2017 IEEE International Symposium on Circuits and Systems (ISCAS), Baltimore, MD, USA, 28–31 May 2017; pp. 1–4.
56. Afifi, A.; Ayatollahi, A.; Raissi, F. CMOL implementation of spiking neurons and spike-timing dependent plasticity. *Int. J. Circuit Theory App.* **2011**, *39*, 357–372. [CrossRef]
57. Frances-Villora, J.V.; Rosado-Muñoz, A.; Martínez-Villena, J.M.; Bataller-Mompean, M.; Guerrero, J.F.; Wegrzyn, M. Hardware implementation of real-time Extreme Learning Machine in FPGA: Analysis of precision, resource occupation and performance. *Comput. Electr. Eng.* **2016**, *51*, 139–156. [CrossRef]
58. Ganchosov, P.N.; Kuzmanov, G.; Kabakchiev, H.; Behar, V.; Romansky, R.P.; Gaydadjiev, G.N. FPGA implementation of modified Gram-Schmidt qr-decomposition. In Proceedings of the 3rd HiPEAC Workshop on Reconfigurable Computing, Prague, Czech Republic, 20 January 2016; pp. 41–51.
59. Phan-Xuan, H.; Le-Tien, T.; Nguyen-Tan, S. FPGA platform applied for facial expression recognition system using convolutional neural networks. *Procedia Comput. Sci.* **2019**, *151*, 651–658. [CrossRef]
60. Alizadeh, S.; Fazel, A. Convolutional Neural Network for Facial Expression Recognition. In Stanford University Report. *arXiv* **2016**, arXiv:1704.06756.
61. Kaarmukilan, S.P.; Poddar, S. FPGA based Deep Learning Models for Object Detection and Recognition Comparison of Object Detection Comparison of object detection models using FPGA. In Proceedings of the 2020 Fourth International Conference on Computing Methodologies and Communication (ICCMC), Erode, India, 11–13 March 2020; pp. 471–474.
62. Kaarmukilan, S.P.; Hazarika, A.; Poddar, S.; Rahaman, H. An Accelerated Prototype with Movidius Neural Compute Stick for Real-Time Object Detection. In Proceedings of the 2020 International Symposium on Devices, Circuits and Systems (ISDCS), Howrah, India, 4–6 March 2020; pp. 1–5.
63. Object detection: Speed and accuracy comparison (Faster R-CNN, R-FCN, SSD, FPN, RetinaNet and YOLOv3). Available online: https://jonathan-hui.medium.com/object-detection-speed-and-accuracy-comparison-faster-r-cnn-r-fcn-ssd-and-yolo-5425656ae359 (accessed on 20 August 2020).
64. Ren, S.; He, K.; Girshick, R.; Sun, J. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Trans. Pattern Anal. Mach. Intell.* **2016**, *39*, 1137–1149. [CrossRef]
65. Hu, Q.; Tang, X.; Tang, W. A Smart Chair Sitting Posture Recognition System Using Flex Sensors and FPGA Implemented Artificial Neural Network. *IEEE Sens. J.* **2020**, *20*, 8007–8016. [CrossRef]
66. Zhao, S.; An, F.; Yu, H. A 307-fps 351.7-GOPs/W Deep Learning FPGA Accelerator for Real-Time Scene Text Recognition. In Proceedings of the 2019 International Conference on Field-Programmable Technology, Tianjin, China, 9–13 December 2019; pp. 263–266.
67. Lin, K.; Chen, M.; Deng, J.; Hassan, M.M.; Fortino, G. Enhanced fingerprinting and trajectory prediction for IoT localization in smart buildings. *IEEE Trans. Autom. Sci. Eng.* **2016**, *13*, 1294–1307. [CrossRef]
68. Xie, T.; Jiang, H.; Zhao, X.; Zhang, C. A Wi-Fi-Based wireless indoor position sensing system with multipath interference mitigation. *Sensors* **2019**, *19*, 3983. [CrossRef] [PubMed]
69. Goswami, A.; Ortiz, L.E.; Das, S.R. WiGEM: A learning-based approach for indoor localization. In Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies, Tokyo, Japan, 6–9 December 2011; pp. 1–12.
70. Liu, C.; Wang, C.; Luo, J. Large-Scale Deep Learning Framework on FPGA for Fingerprint-Based Indoor Localization. *IEEE Access* **2020**, *8*, 65609–65617. [CrossRef]
71. Youssef, M.; Agrawala, A. The Horus WLAN location determination system. In Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services, Seattle, WA, USA, 5 June 2005; pp. 205–218.
72. Wang, X.; Gao, L.; Mao, S.; Pandey, S. DeepFi: Deep learning for indoor fingerprinting using channel state information. In Proceedings of the 2015 IEEE Wireless Communications and Networking Conference (WCNC), New Orleans, LA, USA, 9–12 March 2015; pp. 1666–1671.
73. Shan, Y. ADAS and video surveillance analytics system using deep learning algorithms on FPGA. In Proceedings of the 2018 28th International Conference on Field Programmable Logic and Applications (FPL), Dublin, Ireland, 27–31 August 2018; pp. 465–4650.
74. Chen, V. Doppler signatures of radar backscattering from objects with micro motions. *IET Signal Process.* **2008**, *2*, 291. [CrossRef]
75. Lei, P.; Liang, J.; Guan, Z.; Wang, J.; Zheng, T. Acceleration of FPGA based convolutional neural network for human activity classification using millimeter-wave radar. *IEEE Access* **2019**, *7*, 88917–88926. [CrossRef]
76. Farabet, C.; Martini, B.; Akselrod, P.; Talay, S.; LeCun, Y.; Culurciello, E. Hardware accelerated convolutional neural networks for synthetic vision systems. In Proceedings of the 2010 IEEE International Symposium on Circuits and Systems, Paris, France, 30 May–2 June 2010; pp. 257–260.
77. FPGA Leadership across Multiple Process Nodes, XILINX. 2020. Available online: https://www.xilinx.com/products/silicon-devices/fpga.html (accessed on 18 January 2021).
78. Xilinx. Xilinx Provides Twitch with Plug and Play VP9 Transcoding Solution. 2011. Available online: https://www.xilinx.com/publications/powered-by-xilinx/Twitch-Case-Study.pdf (accessed on 18 January 2021).
79. Wu, S.; Li, G.; Chen, F.; Shi, L. Training and inference with integers in deep neural networks. *arXiv* **2018**, arXiv:1802.04680.
80. Gao, C.; Zhang, F. FPGA-based Accelerator for Independently Recurrent Neural Network. In Proceedings of the 2018 IEEE 4th International Conference on Computer and Communications (ICCC), Chengdu, China, 7–10 December 2018; pp. 2075–2080.

81. Hochreiter, S.; Schmidhuber, J. Long short-term memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef]
82. VK Reinvents Storage for Social Networks, Intel. 2019. Available online: https://www.intel.com/content/www/us/en/customer-spotlight/stories/vk-storage-customer-story.html (accessed on 18 January 2021).
83. Liu, X.; Ounifi, H.A.; Gherbi, A.; Lemieux, Y.; Li, W. A hybrid GPU-FPGA-based computing platform for machine learning. *Procedia Comput. Sci.* **2018**, *141*, 104–111. [CrossRef]
84. Nakahara, H.; Yonekawa, H.; Fujii, T.; Sato, S. A lightweight yolov2: A binarized cnn with a parallel support vector regression for an fpga. In Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 25–27 February 2018; pp. 31–40.
85. Li, S.; Luo, Y.; Sun, K.; Yadav, N.; Choi, K.K. A Novel FPGA Accelerator Design for Real-Time and Ultra-Low Power Deep Convolutional Neural Networks Compared With Titan X GPU. *IEEE Access* **2020**, *8*, 105455–105471. [CrossRef]
86. Speedster 7t FPGAs, Achronix. 2020. Available online: https://www.achronix.com/product/speedster7t-fpgas (accessed on 18 January 2021).
87. Blaiech, A.G.; Khalifa, K.B.; Valderrama, C.; Fernandes, M.A.; Bedoui, M.H. A Survey and Taxonomy of FPGA-based Deep Learning Accelerators. *J. Syst. Archit.* **2019**, *98*, 331–345. [CrossRef]
88. Wang, T.; Wang, C.; Zhou, X.; Chen, H. An Overview of FPGA Based Deep Learning Accelerators: Challenges and Opportunities. In Proceedings of the 2019 IEEE 21st International Conference on High Performance Computing and Communications, Zhangjiajie, China, 10–12 August 2019; pp. 1674–1681.
89. Hao, C.; Chen, D. Deep neural network model and FPGA accelerator co-design: Opportunities and challenges. In Proceedings of the 2018 14th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT), Qingdao, China, 31 October–3 November 2018; pp. 1–4.
90. Liu, Z.; Li, Y.; Ren, F.; Goh, W.L.; Yu, H. Squeezedtext: A real-time scene text recognition by binary convolutional encoder-decoder network. In Proceedings of the AAAI Conference on Artificial, New Orleans, LA, USA, 2–7 February 2018.
91. Suda, N. Throughput-Optimized OpenCL-based FPGA Accelerator for Large Scale Convolutional Neural Network. In Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 21–23 February 2016.
92. Chew, L.W.; Chia, W.C.; Ang, L.M.; Seng, K.P. Low–memory video compression architecture using strip–based processing for implementation in wireless multimedia sensor networks. *Int. J. Sens. Netw.* **2012**, *11*, 33–47. [CrossRef]
93. García, G.J.; Jara, C.A.; Pomares, J.; Alabdo, A.; Poggi, L.M.; Torres, F. A survey on FPGA-based sensor systems: Towards intelligent and reconfigurable low-power sensors for computer vision, control and signal processing. *Sensors* **2014**, *14*, 6247–6278. [CrossRef] [PubMed]
94. Ang, L.M.; Seng, K.P.; Chew, L.W.; Yeong, L.S.; Chia, W.C. *Wireless Multimedia Sensor Networks on Reconfigurable Hardware*; Springer: Heidelberg, Germany, 2013.
95. Venieris, S.I.; Bouganis, C.S. fpgaConvNet: Mapping regular and irregular convolutional neural networks on FPGAs. *IEEE Trans. Neural Netw. Learn. Syst.* **2018**, *30*, 326–342. [CrossRef] [PubMed]
96. Ghaffari, A.; Savaria, Y. CNN2Gate: An Implementation of Convolutional Neural Networks Inference on FPGAs with Automated Design Space Exploration. *Electronics* **2020**, *9*, 2200. [CrossRef]
97. Zhang, C.; Sun, G.; Fang, Z.; Zhou, P.; Pan, P.; Cong, J. Caffeine: Toward uniformed representation and acceleration for deep convolutional neural networks. *IEEE Trans. Comput. Aided Design Integr. Circuits Syst.* **2018**, *38*, 2072–2085. [CrossRef]
98. Umuroglu, Y.; Fraser, N.J.; Gambardella, G.; Blott, M.; Leong, P.; Jahre, M.; Vissers, K. Finn: A framework for fast, scalable binarized neural network inference. In Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 22–24 February 2017; pp. 65–74.
99. Xilinx. Xilinx Reports Record Revenues Exceeding $3 Billion For Fiscal. 2019. Available online: https://www.xilinx.com/news/press/2019/xilinx-reports-record-revenues-exceeding-3-billion-for-fiscal-2019.html (accessed on 24 April 2019).
100. Alsop, T. Intel's Programmable Solution Group (PSG) revenue, Statista. Available online: https://www.statista.com/statistics/1096397/intel-programmable-solutions-group-revenue/ (accessed on 29 October 2020).
101. FPGA Market 2020 Global Industry study, SNN News. Available online: https://www.snntv.com/story/42322561/field-programmable-gate-array-fpga-market-2020-global-industry-study-by-overview-size-top-players-revenue-sharedevelopment-strategy-future-trends (accessed on 3 July 2020).