electronics

MDPI

Article

Best Practices for the Deployment of Edge Inference: The Conclusions to Start Designing

Georgios Flamis *, Stavros Kalapothas * and Paris Kitsos *

ECSA Laboratory, Electrical and Computer Engineering Department, University of Peloponnese, 26334 Patras, Greece

* Correspondence: g.flamis@go.uop.gr (G.F.); s.kalapothas@go.uop.gr (S.K.); kitsos@uop.gr (P.K.)

Abstract: The number of Artificial Intelligence (AI) and Machine Learning (ML) designs is rapidly increasing and certain concerns are raised on how to start an AI design for edge systems, what are the steps to follow and what are the critical pieces towards the most optimal performance. The complete development flow undergoes two distinct phases; training and inference. During training, all the weights are calculated through optimization and back propagation of the network. The training phase is executed with the use of 32-bit floating point arithmetic as this is the convenient format for GPU platforms. The inference phase on the other hand, uses a trained network with new data. The sensitive optimization and back propagation phases are removed and forward propagation is only used. A much lower bit-width and fixed point arithmetic is used aiming a good result with reduced footprint and power consumption. This study follows the survey based process and it is aimed to provide answers such as to clarify all AI edge hardware design aspects from the concept to the final implementation and evaluation. The technology as frameworks and procedures are presented to the order of execution for a complete design cycle with guaranteed success.

Keywords: computing methodologies; Artificial Intelligence; Machine Learning; general and reference; cross-computing tools and techniques; hardware: integrated circuits; very large scale integration design; hardware validation

Citation: Flamis, G.; Kalapothas, S.; Kitsos, P. Best Practices for the Deployment of Edge Inference: The Conclusions to Start Designing. *Electronics* **2021**, *10*, 1912. https://doi.org/10.3390/electronics10161912

Academic Editor: Rashid Mehmood

Received: 29 June 2021
Accepted: 2 August 2021
Published: 9 August 2021

1. Introduction

One of the main challenges Artificial Intelligence (AI) and Machine Learning (ML) designs are facing is the computational cost. This condition is forcing towards improving hardware acceleration in order to offer the high computational power, required from application fields that should be supported [1–5]. Recently, hardware accelerators are being developed in order to provide the needed computational power for the AI and ML designs [6]. In the literature, hardware accelerators are built using Field Programmable Gate Arrays (FPGA), Graphic Processor Units (GPU) and Application Specific Integrated Circuits (ASIC) to accelerate the computationally intensive tasks [7]. These accelerators provide high-performance hardware while aiming to preserve the required accuracy [8].

GPU is largely used at the training stage of the AI/ML design, because it provides floating point accuracy and parallel computation [9,10]. It also has a well-established ecosystem, as software/hardware collaborative scheme. At inference stage of AI/ML design, GPU consumes more energy which cannot be tolerated in edge devices, as other platforms can be more energy efficient, also in area of chip usage.

Reconfigurability, customizable dataflow, data-width, low-power and real-time makes FPGA an appealing platform for the inference phase of AI/ML designs [11,12]. The performance of the AI/ML accelerator can be limited by computation and memory resource on FPGA. Various techniques have been developed to improve the bandwidth and resource utilization, as it will be presented in the paragraphs that follow.

Another way to accelerate the AI/ML design is the customizable and dedicated hardware, the well known Application-Specific Integrated Circuits (ASIC) [13,14]. They are

specifically tailored to accelerate one or a certain type of algorithm [15–19]. The acceleration effect is usually good and the power consumption is low, but at the same time, the reconfigurability is poor and the development cost is high. Furthermore, hardware design and longer development cycle may increase the cost.

Among the plurality of surveys that have been recently published and are individually mentioned in the main body of this work, the target of this study is the collection of the complete end-to-end AI/ML design considerations. They are presented in the order of execution, but without eliminating it to a single task sequence. In Figure 1, this structure is presented. The left-most block of Figure 1 contains the basic steps of the AI/ML design, modeled and evaluated as to resolve a specific problem. In details, it is shown in Figure 2, as an iterative process that depends on the selected dataset and produces a floating-point model. The model of choice is trained with a known dataset and validated with a smaller portion of it. Testing on new data that the model never saw before will allow to complete this phase after a number of iterations. In Section 2, the details of this process are described. In the middle, is the definition of the hardware into which the model will be transferred. The deployment to hardware is strongly connected with the architecture and the type of hardware that has been selected. Thus, the decision on it should be done at an early stage. The available options are presented in Section 3 of this work. The methods to optimize the AI/ML design for efficient deployment to the hardware are covered to the right-most block of Figure 1. At the size of edge AI on which this work is focused, the hyperparameters tuning is necessary to compete the limitations of the hardware capabilities. The overall process may be iterated several times for optimal performance.

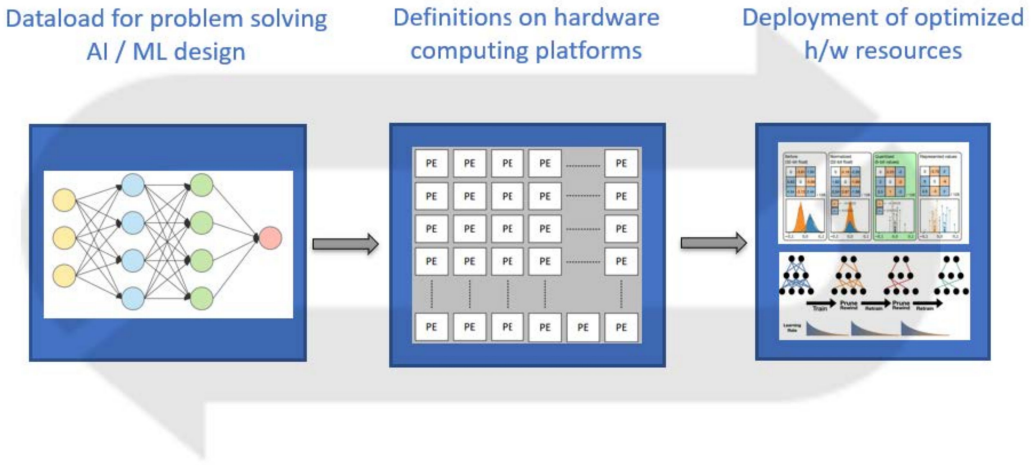

Figure 1. The end-to-end AI/ML design considerations.

The rest of this study is structured as below:

- First, the relationship to dataload that the AI/ML design will be processing is defined. The data are almost everything for deep learning and the best choices of the required resources are made when data have good quality. The selected dataload should be representing the data that the AI/ML design will confront when it is implemented, otherwise it will be flawed.
- Second comes the selection of the AI/ML design architecture as modeled in high level and the design constraints are defined. There are reasons to select between a very flexible FPGA implementation or an ASIC deployment for the purpose of the AI/ML design execution.
- Last but not least, the optimization and verification of the hardware design is discussed, with quantization, power consumption and area being the most important parts.

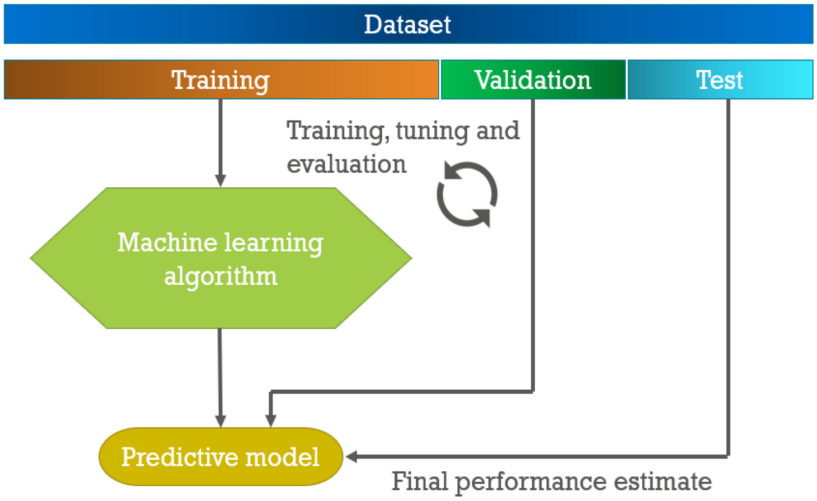

Figure 2. Basic steps of the AI/ML design.

2. Dataload for Problem Solving

There is a large number of software libraries and frameworks developed for high-performance execution of the training stage in AI/ML designs with target to achieve best accuracy and improve the efficiency. Toolkits such as PyTorch [20,21] and TensorFlow [22–24] are popular with the aim of increasing the productivity on this scope by providing high-level APIs. They are defined as software level optimizations that are performed iteratively, where, in each iteration, a small number of parameters are removed and the resultant network is fine-tuned by retraining for a limited number of epochs. The accuracy of the network is analyzed on the basis of the accuracy loss and from there the decision whether to continue it taken. Despite the power cost of GPUs, the high performance execution of training can be efficiently achieved, due to their multi-core and fast memory architecture [25–27].

Proper allocation of the effort during the iterations is very important. Collecting more data or prepossessing the existing with different optimization algorithms [28] or fine-tuning the algorithm operation is necessary at this stage.

The dataset should include the useful data required to resolve the particular problem that the AI/ML system will be designed for. In most cases, it will be large variety of data that are not all useful. The distribution of the data is the best method to approximate this condition. When the distribution matches the target of the AI/ML system, the design can be reduced to the necessary size. The best representation of the data can be achieved by clipping the far outlier values with an equal min/max range. The discipline of trying to understand data can be found in a visual context so that patterns, trends and correlations that might not otherwise be detected can be exposed. The features selected to train the AI/ML design should always be natural and meaningful to the context of the purpose of the design and the input data should be scaled in a common range between them. In addition, unnatural values or those with conventional meanings should be avoided. Normalizing the values is a good approach to optimize the input data in the concept to be suitable for training [29,30].

Cleaning up a dataset from duplicated entries is an important action, but the exact method to do it relies on the actual AI/ML design purpose. In particular, the data preparation functions that have been found useful for the purpose of an AI/ML design are focused on statistical methods to apply the most optimal structure, distribution and the quality of the dataset prior starting the training of the model [31].

Pandas is the Data Analysis Library in Python [32]. It is a tool of analyzing and cleaning up large datasets as it can print out the summary of most basic statistical information for each feature in the dataset, such as minimum, maximum, mean and quarterly

values [33]. A sanity check of the dataset is an important prerequisite as it will ensure the proper distribution of the data is the expected.

TensorFlow Extended (TFX) is a good example of the frond to back flow that performs the data preparation, followed from a formalized validation and deployment of the AI/ML design [34].

The quality of the AI/ML design can be significantly improved with error analysis [35]. A single scalar metric that measures end to end quality is necessary as it can be used to plot the learning curves that most toolflows provide. Identification of the situation that the AI/ML design is under-fitting or over-fitting the training data can be done before running a long time training session. Improvements can be applied by spotting patterns from the analysis [36,37].

Sometimes it is impractical or impossible to collect new raw datasets for use as training data. There is a number of methods developed to allow synthesize artificial data. Data Augmentation is the most popular method that can be applied as a data-space solution to the problem of limited data [38,39]. In AI/ML designs, the validation error must continue to decrease with the training error. Data Augmentation is a very powerful method of achieving this. The augmented data will represent a more comprehensive set of possible data points, thus minimizing the distance between the training and validation set, as well as any future testing sets. More methods to improve the generalization performance of the AI/ML design are focusing on the architecture of the model. These are functional solutions known as regularization techniques that aim at making the model generalize better. In particular these are the methods of Dropout [40], Batch normalization [41,42], Transfer learning [43], L2-regulazation [44], Hierarchical Guidance and Regularization (HGR) learning [45], translation [46], horizontal flipping [47], noise disturbance [48] and a plethora of others are available in the bibliography. Recently, the method of dynamic feature selection has been introduced with minimum redundancy information for linear data [49]. It is an effective feature selection method that can select a feature subset from higher order features. The selected feature subset reflects high correlation and low redundancy in various real-life datasets. Similarly, Zheng et al. propose a full stage data augmentation framework to improve the accuracy of deep CNNs, which can also play the role of model ensemble without introducing additional model training costs [50].

3. Definitions on Hardware Computing Platforms

AI/ML designs are in principle computationally expensive. Thus, numerous architecture designs were laid down by the researchers and also contributed commercially to promote hardware acceleration. In the state-of-the-art for accelerating deep learning algorithms a plethora of hardware platforms are supported such as, field programmable gate array (FPGA), application specific integrated circuit (ASIC) and graphic processing unit (GPU). The features that are most attractive to hardware designers are the benefits gained from increased performance, high energy efficiency, fast development round and capability of reconfiguration. This section is targeted to the differentiation and benefits of the usage of FPGAs, GPUs and ASICs based processors for AI/ML applications and more specifically focused on the inference for edge applications, with Deep Neural Networks (DNN) and Convolutional Neural Networks (CNN).

3.1. FPGA-Based Accelerators

An FPGA architecture consists of the programmable input/output (I/O) and the programmable logic core. Inside the FPGA chip the I/O blocks are physically located in the edge of the design to facilitate the connection with external modules. The programmable core fabric includes programmable logic blocks and programmable routing interconnections. In the basic FPGA architecture, multiple DSP blocks and RAM blocks are also embedded thus consisting the Processing Elements (PE) of the device.

FPGAs are generally accepted for computational-intensive applications [51–53] and many research papers have demonstrated that FPGAs are very suitable for latency-sensitive

and real-time inference jobs [54–56]. FPGAs also provide high throughput at a reasonable price with low-power consumption and reconfigurability. The highest energy consumption is the result of accessing the off-chip DRAM memory for data movement; therefore, the use of Block RAM (BRAM) is preferred, where low latency can be also achieved. There are also low-power techniques applied at RTL, such as clock-gating and memory-splitting, for efficient code generation of CNN algorithms. In [57], it is shown that FPGAs can maximize parallelism of feature map and convolutional kernels with high bandwidth and dynamical configurations to improve performance. From other observations [58] it has been concluded that DSP blocks and BRAM elements are more crucial to increase parallelism than flip-flops (FFs) and Look-up tables (LUTs), which can be further exploited and must be taken into consideration when designing FPGAs in the future.

When designing CNNs to fit inside an FPGA, several considerations should be taken into account including resource optimization such as, compression (weight quantization and network pruning) [59] and similarly, implementing Binary Neural Networks (BNN) which are a type of CNNs where the MAC operations are transformed using weight values of +1 and −1 implementing less expensive computations such as XNOR and bit counting. Therefore, the aforementioned techniques can introduce efficient resource utilization without significant accuracy drop [60] in implementations using FPGAs. Other quantization methods include, 32 bit floating-point (FP32) to 8 bit fixed-point (INT8) vector representations, although further network quantization is limited because binarized activation functions, as well as weights show a significant accuracy drop (12.4%) [61,62]. In Figure 3, is the high-level workflow of quantization and pruning method applied to the weights and network of a CNN in order to fit inside an FPGA. More details are presented in the third section of this study.

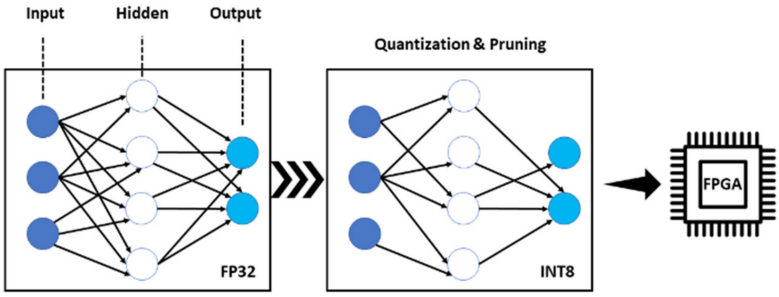

Figure 3. Compression steps prior implementation to FPGA.

Minakova et al. at [63] present a novel method to reduce the memory required to store intermediate data exchanged between CNN operations. The method is orthogonal to pruning and quantization, and hence they can be combined. In particular it is proposed to convert each CNN layer into a functionally equivalent Cyclo-Static Dataflow model (CSDF) [64] that eliminates the memory footprint from 17% to 64%. The method is attached to the PEs that are related with the execution of the specific layer. The model is shown in Figure 4a, as adapted from the relative study.

The massive amount of storage space provided by the external memories comes with the cost of high access time and usually non-constant access time. In this concern, Benes et al. in [65] proposes an architecture that optimizes memory accesses in such a way that the Read Modify Write operations ideally do not require pipeline stalls even when memory accesses are randomly distributed over the whole external memory. The core of the proposal is the setup of a Transaction table that contains multiple entries with same address. The Transaction table format is shown in Figure 4b.

The strong research interest to address the memory requirements of the AI/ML designs has driven to the investigation of memory aware designs such as μ-Genie proposed from Stramondo et al. in [66]. The effective use of the available resources is achieved with co-designed memory system and processing elements. The template of the functional unit for this purpose is adapted for the relative study in Figure 4c.

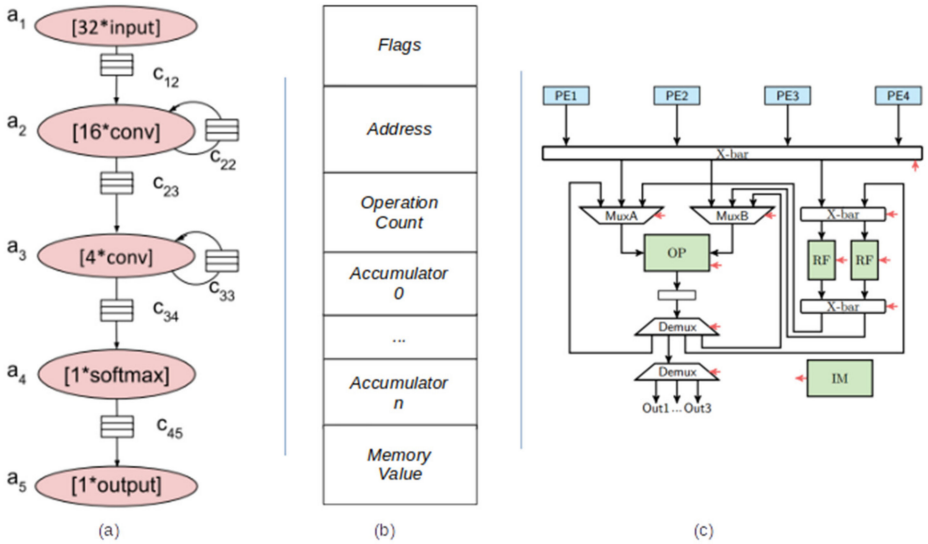

Figure 4. Memory constraints and optimization methods, (**a**) Cyclo-Static datafle model for intermediate data storage in reduced memory size, (**b**) transactions in read-modify-write operations for optimized memory access, (**c**) co-designed memory system and processing elements.

Further challenges include streamlining the software architectures to facilitate the development for efficient mapping of AI models, using hardware description language (HDL), into FPGAs. Towards this process, several frameworks have been introduced such as the HADDOC2 [67] that models the target CNN into a data flow graph. The optimized low-level architecture mainly uses LEs instead of DSPs. Furthermore, multiplications by zero are being totally removed from the computation (replaced by simple signal connections), as well as the multiplications by two are being implemented via shift registers. The hls4 ml [68] translates ML algorithms for FPGA implementation using Python. It is based on an open-source code design workflow supporting network pruning and quantization and with the latest work [69], its capabilities are extended towards low-power inference implementations. In fpgaConvNet [70] architecture, one processing stage per layer is employed and a set of transformations are proposed to facilitate the CNN mapping on FPGAs. In CNN2Gate [71], the computation flow of network layers along with the weights and biases is extracted and a fixed-point quantization is applied. Necessary data are exported using vendor specific OpenCL to synthesize the build. Ultimately, the dependency on developer's low-level hardware knowledge is abstracted. FINN [72], also incorporates a data-driven approach to generate a custom streaming architecture based on binary neural network (BNN) structure. Each convolution layer has its own computation engine and all engines are inter-connected together to create a pipeline. Together with development kits (like Vitis AI [73], Dnnweaver2 [74]) to transform the network model into RTL code. Recently, researchers have addressed the cost-effective implementations of FPGA clusters in AI and ML applications [75]. In Table 1, the most important FPGAs frameworks with the basic transformation techniques are given.

Table 1. Available frameworks for FPGAs.

Frameworks	Transformation Technique
HADDOC2	Models CNN as a dataflow graph
hls4ml	Translating NN into FGPA firmware using Python and HLS
fpgaConvNet	Assign one processing stage per network layer
Dnnweaver2	Use of dataflow graph
CNN2Gate	Automatic synthesis using OpenCL
FINN	Generate data streams based on Binary NN

3.2. GPU-Based Accelerators

The GPU architecture contains a programmable engine with massively parallel processors (ALUs) for computer graphics rendering that includes multiple buffers with high precision (64-bit double) support and fast DRAM modules. Recently, GPUs are also used for general purpose computing applications (GP-GPU). Therefore, GPUs are widely used as hardware accelerators providing high processing speed and reduced execution time. GPUs process data in single-instruction and multiple-thread (SIMT) have a large number of parallel ALUs which fetch data directly from memory and do not communicate directly with each other. These programmable cores are supported by a high memory bandwidth and thousands of threads [76]. Typical applications that GPUs excel at, include signal and image processing which take advantage of their multiple native floating-point cores together with their high-parallelism. In Figure 5, a layout diagram of the main components inside a GPU is illustrated. In order to compute in parallel, concurrent threads must be created on the GPU. Each thread is executed on a GPU core (ALU) and multiple GPU cores are grouped together into blocks. Threads operating at the same block can inter-exchange data through shared memory. At runtime, a list of blocks that need to be executed must be maintained by the system. However, each block execution is independent from each other.

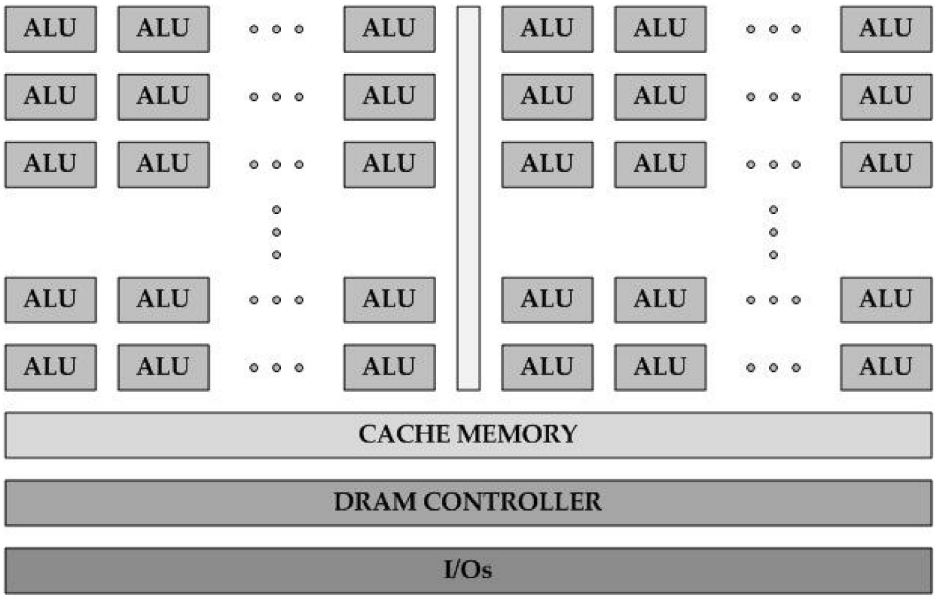

Figure 5. GPU internal components.

CNN implementation on a GPU is materialized using a high-level language such as C/C++. Thus, GPUs can be programmed easier by developers, who are abstracted from the hardware layer. NVIDIA also created CUDA [77], a parallel programming platform to help developers accelerate general computing applications. In addition, TensortRT, is a deep learning inference optimizer, based on CUDA, that supports INT8 and FP16 representations to reduce precision, as well as latency [78]. Typically, the process to fit a CNN into a GPU requires two phases (i) training and (ii) inference. In most cases, high-end GPUs are utilized for model training [79–82], mainly based on voltage reduction and frequency scaling which can achieve better performance with power capping.

It is also observed, that among different CNN implementations using GPUs, performance is dependent on the degree of the parallelism in the execution threads, as well as in the efficient allocation of resources (registers, shared memory) [83]. For example, if Error Code Correction (ECC) in RAM is disabled, as bit-level accuracy is not mandatory for CNN applications, this can lead to a 6.2% decrease in memory utilization [84]. In Table 2, a list of CNN transformation frameworks with GPU support is shown.

Table 2. Available frameworks for GPUs.

Frameworks	Transformation Technique
Cuda-convnet2 [85]	CNN implementation using CUDA
cuDNN [86]	Nvidia's CUDA SDK for CNN implementation
OpenCV DNN [87]	OpenCV library with DNN and optimized for GPU
Tensorflow [88]	CNN framework with GPU (CUDA) performance optimization
PyTorch [89]	Optimized tensor library for DL using GPU (CUDA)
DeepSense [90]	Mobile GPU-based CNN optimization using OpenCL

3.3. ASIC-Based Accelerators

Contrary to the way AI/ML design is formulated to the FPGA or GPU hardware structures for the purpose of hardware acceleration, the main way to meet the design target using ASIC is to customize the dedicated hardware acceleration circuits [91]. An optimized and specialized hardware implementation can reduce system cost by optimizing the needed resources and decreasing power requirements while improving the performance [92].

Similar to the reconfigurable, FPGA or FPGA like architectures, dedicated AI/ML designs have also adopted fixed-point representation to eliminate the size of the exchanged data [93,94]. In most cases some flexibility is achieved with specialized MAC units that can be parallelized.

The AI/ML design of dedicated architectures cannot be changed after fabrication, they include a separate processing unit with generic role for all other functions than the specialized MAC operations. The fabrication process is time consuming and resource demanding, making it difficult to follow the rapid evolution of AI/ML designs.

The performance variation is a strong selection criterion as it depends on the technology, data size and area efficiency of the AI/ML design. These different features justify the differences in peak performance among the architectures [95]. Several vendors involved with the design of AI/ML ASIC-based accelerators have selected their own hardware architecture. The most indicative cases are listed in Table 3 that follows.

Table 3. Available AI/ML specialized accelerators as ASIC.

Vendor	Hardware Architecture
Greenwaves [96]	Composed of one Fabric Controller (FC)core and eight cores in a cluster as extended version of the RISC-V instruction set and separate data cache for each.
BrainChip [97]	It is based to spiking-neural-network (SNN) technology that connects 80 event-based neural processor units and it is optimized for low-power edge AI.
Maxim Integrated Products [98]	It is a hardware-based CNN accelerator that enables battery-powered applications to execute AI inferences. It is combined with a large on-chip system memory
DSP group [99]	It is a standalone hardware engine that is designed to accelerate the execution of NN inferences. It is optimized for efficiency to ensure ultra-low power consumption for small to medium size NNs
Synaptics [100]	proprietary power and energy optimized neural network and domain specific processing cores, significant on-chip memory
Synsense [101]	A scalable, fully-configurable digital event-driven neuromorphic processor with 1 M ReLU spiking neurons per chip for implementing Spiking Convolutional Neural Networks
Renesas [102]	Dynamically Reconfigurable Processor (DRP) technology is special purpose hardware that accelerates image processing algorithms by as much as 10×, or more

Recently, resistive memory based Bayesian neural networks have been found applicable in edge inference hardware. In [103], Dalgaty et al. propose to use trained Bayesian models with probabilistic programming methods to overcome the inherently random process. Hence, the intrinsic cycle to cycle and device to device conductance variation are limited.

4. Deployment of Optimized h/w Resources

Dedicated hardware solutions are the most performance efficient, as they aim to achieve the highest performance at the lowest cost. The accelerator-based devices can be very limited in terms of configurability and thus unable to adapt the processing dataload to each particular network and model optimizations. Reconfigurable computing is an alternative solution providing the required configurability and high performance. The lower cost and high energy efficiency are the result of a dedicated hardware optimization for inference.

The performance efficiency also depends on the network [104], hence there are several methods developed to optimize the deployment and accordingly the hardware resources [105].

Targets for the deployment of a network at the edge in embedded systems are the configurability, the low size and lowest power. Moreover, it is critical to control the size of the memory that the AI/ML design consumes. The two techniques to achieve this are pruning [106] and quantization [107]. The first sets to zero all near-zero weights, which effectively reduces the size of the weights table, and hence it allows for compression. The second replaces the 32-bit floating point arithmetic used at training with lower precision representations that executes to the hardware, such as 8-bit fixed-point. They are also referred as hardware aware optimizations.

The main idea of network pruning is to delete unimportant parameters, since not all parameters are important in highly precise deep neural networks. Consequently, connections with less weights are removed, which converts a dense network into a sparse one [108]. It is as simple as to remove the weights of a trained network that are lower than a threshold and then retrain. The appropriate threshold selection to prune the network is an iterative process involving training that may consume significant time due to the re-iterate steps. Manessi et al. in [109], propose the differentiability-based pruning with respect to learning thresholds, thus allowing pruning during the backpropagation phase. Han et al. in [110] propose pruning that is lossless in accuracy for state of art CNN models, as the method retrains the sparse network. Yu et al. in [111] propose Scalpel that consists of SIMD-aware weight pruning and node pruning; hence, it customizes pruning for different hardware platforms based on their parallelism. Further compression is achieved when pruning is combined with quantization that is analyzed next.

Quantization of the parameters takes an existing neural network and compresses its parameters by changing from floating-point numbers to low-bit width numbers, thus avoiding costly floating-point multiplications [112,113]. Analysis has shown that data represented with fixed-point preserves the accuracy close to that obtained with data represented with floating-point [114]. A deeper optimization allows different fixed-point scales and bitwidths for different layers [115,116]. Data quantization can be taken to the limit with some or all data represented in binary. Although there is a trade-off between the network size and precision, quantization gives a significant improvement to the point that results can be almost identical to the full-precision network. The quantization of 8-bit seems to be a good choice, as it practically compromises between the memory size and the retraining accuracy. More aggressive quantization using 4, 2 or even 1-bit resolution is possible, but it requires more special distribution methods.

Quantization and pruning approaches can be considered individually as well as jointly by pruning the unimportant connections and then quantizing the network. After pruning and quantization, the dense computation becomes sparse computation and after quantization the arithmetic alignment should be preserved. The sparsity can reduce the efficiency of the hardware; hence, the precision and efficiency trade off remain till the end of the AI/ML design.

Among the pruning and quantization methods, the model compression to accelerate the hardware compute capabilities at the lower possible cost has been explored with significant results. Albericio et al. in [117] proposes a method for zero-skipping, based on the observation that in practice, many of the neuron values turn out to be zero; thus,

the corresponding multiplications and additions do not contribute to the final result and could be avoided. Aimar et al. in [118] have developed NullHop, a CNN accelerator that exploits activation sparsity per its ability to skip zeroes and further compression through the sparsity map and the nonzero value list. Dynamic sparsity of activation is exploited by Han et al. in [119] with EIE accelerator design. Zhang et al. propose Cambricon-X in [120], a sparse accelerator to exploit the sparsity and irregularity of NN models for increased efficiency with a PE-based architecture consisting of multiple Processing Elements. Last but not least, Yao et al. in [121] have proposed DeepIoT that compresses commonly used deep neural network structures for sensing applications through deciding the minimum number of elements in each layer.

More recently, the inference of State-Space Models (SSM) has appeared as optimal approach to the deployment of models that are dynamic, large, complex or uncertain due to the lossy data acquisition from sensors or microphones that are common in edge AI systems [122]. Towards the increased computational cost and large parameters space, Imani in [123] proposes a two stages Bayesian optimization framework that consists of dimensionality reduction and sample selection processes. An eigenvalue decomposition based technique is developed for mapping the original large parameter space to reduced space in which the inference function has the highest variability.

Under this variety of optimizations for performance, the next requirement is to have enough metrics for evaluation and selection of the most optimal solution. In [124,125], the method of Eyexam is proposed as a systematic way to capture the performance limitation of the AI/ML designs. It is based on the roofline model and extends it with seven sequential steps to conclude to the fine-grained profile of the AI/ML design. At step 1, the upper bound of the performance is defined as the finite size of the workload. The order of operations determines step 2 and it is strongly related to the dataflow. The minimum number of processing elements in which the performance restrictions appear is evaluated as step 3. The physical dimensions and the storage capacity are examined in steps 4 and 5. The data bandwidth required for each computation to be exchanged with the memory and the effects of data variation are visited in steps 6 and 7 respectively.

In a comprehensive view of the complete flow for efficient inference, the accuracy of the AI/ML design is the metric that is necessary to be met and interpreted in the content of difficulty to resolve the given problem or task with the provided dataload. Throughput and latency are the giving useful input to indicate the amount of data that can be processed or the amount of time consumed to finish a task. Energy efficiency, power consumption and cost are those to guarantee the success of the design, while flexibility and scalability refer to designs in which flexibility is important.

5. Conclusions

Due to the complexity of the AI/ML designs and the limitations of the hardware to achieve lower power, lower latency and better accuracy in edge inference, the complete end-to-end AI/ML design flow has been presented in this study. It follows a software/hardware synergy for training and inference with the aim of highlighting the design challenges and providing guidance through the available solutions. The role of the dataload as the starting point for resolving a problem is a large area that can affect the size and the performance in hardware execution. The varying choices for the hardware implementation have been explored and the individual advantages have been highlighted, thus allowing the proper and most optimal choice. Last but not least the major optimization techniques have been presented, as the energy efficiency accompanies the design cost without compromising the quality in terms of accuracy. It is clear that the edge inference brings challenges and new design requirements for the network models, network optimizations and new computing devices. The practices towards the deployment on edge inference should be defined at the beginning and iterated inside the large circle of Figure 1 untill the target is met.

Author Contributions: Conceptualization, G.F. and S.K.; methodology, P.K.; software, G.F.; validation, G.F., S.K. and P.K.; formal analysis, P.K.; investigation, G.F.; resources, S.K.; data curation, P.K.; writing—original draft preparation, G.F.; writing—review and editing, S.K.; visualization, P.K.; supervision, P.K. 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.

References

1. LeCun, Y.; Bengio, Y.; Hinton, G. Deep learning. *Nature* **2015**, *521*, 436–444. [CrossRef]
2. Farabet, C.; Poulet, C.; Han, J.Y.; LeCun, Y. CNP: An FPGA-based processor for Convolutional Networks. In Proceedings of the 2009 International Conference on Field Programmable Logic and Applications, Prague, Czech Republic, 31 August–2 September 2009; pp. 32–37.
3. Sankaradas, M.; Jakkula, V.; Cadambi, S.; Chakradhar, S.; Durdanovic, I.; Cosatto, E.; Graf, H.P. A Massively Parallel Coprocessor for Convolutional Neural Networks. In Proceedings of the 2009 20th IEEE International Conference on Application-Specific Systems, Architectures and Processors, Boston, MA, USA, 7–9 July 2009; pp. 53–60.
4. Abu Talib, M.; Majzoub, S.; Nasir, Q.; Jamal, D. A systematic literature review on hardware implementation of artificial intelligence algorithms. *J. Supercomput.* **2021**, *77*, 1897–1938. [CrossRef]
5. Guo, K.; Zeng, S.; Yu, J.; Wang, Y.; Yang, H. A Survey of FPGA Based Neural Network Accelerator. *arXiv* **2017**, arXiv:1712.08934.
6. Seng, K.; Lee, P.; Ang, L. Embedded Intelligence on FPGA: Survey, Applications and Challenges. *Electronics* **2021**, *10*, 895. [CrossRef]
7. Li, Z.; Zhang, Y.; Wang, J.; Lai, J. A survey of FPGA design for AI era. *J. Semicond.* **2020**, *41*, 021402. [CrossRef]
8. Capra, M.; Bussolino, B.; Marchisio, A.; Shafique, M.; Masera, G.; Martina, M. An Updated Survey of Efficient Hardware Architectures for Accelerating Deep Convolutional Neural Networks. *Future Internet* **2020**, *12*, 113. [CrossRef]
9. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. ImageNet classification with deep convolutional neural networks. *Adv. Neural Inf. Process. Syst.* **2012**, *25*, 1097–1105. [CrossRef]
10. Simonyan, K.; Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv* **2014**, arXiv:1409.1556.
11. Diniz, W.F.; Fremont, 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]
12. 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.
13. Rankin, D.; Krupa, J.; Harris, P.; Flechas, M.A.; Holzman, B.; Klijnsma, T.; Pedro, K.; Tran, N.; Hauck, S.; Hsu, S.-C.; et al. FPGAs-as-a-Service Toolkit. In Proceedings of the 2020 IEEE/ACM International Workshop on Heterogeneous High-Performance Reconfigurable Computing (H2RC), Atlanta, GA, USA, 13 November 2020.
14. Moolchandani, D.; Kumar, A.; Sarangi, S.R. Accelerating CNN Inference on ASICs: A Survey. *J. Syst. Arch.* **2021**, *113*, 101887. [CrossRef]
15. Reuther, A.; Michaleas, P.; Jones, M.; Gadepally, V.; Samsi, S.; Kepner, J. Survey of Machine Learning Accelerators. In Proceedings of the 2020 IEEE High Performance Extreme Computing Conference (HPEC), Boston, MA, USA, 22–24 September 2020; pp. 1–12.
16. Cavigelli, L.; Gschwend, D.; Mayer, C.; Willi, S.; Muheim, B.; Benini, L. Origami: A convolutional network accelerator. In Proceedings of the Great Lakes Symposium on VLSI, Pittsburgh, PA, USA, 20–22 May 2015; pp. 199–204.
17. Chen, Y.H.; Krishna, T.; Emer, J.S.; Sze, V. An energy-efficient reconfigurable accelerator for deep convolutional neural networks. In Proceedings of the 2016 IEEE International Solid-State Circuits Conference (ISSCC), San Francisco, CA, USA, 31 January–4 February 2016; pp. 262–263.
18. Shafiee, A.; Nag, A.; Muralimanohar, N.; Balasubramonian, R.; Strachan, J.P.; Hu, M.; Williams, R.S.; Srikumar, V. ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars. In Proceedings of the 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA), Seoul, Korea, 18–22 June 2016; pp. 14–26.
19. Andri, R.; Cavigelli, L.; Rossi, D.; Benini, L. YodaNN: An Ultra-Low Power Convolutional Neural Network Accelerator Based on Binary Weights. In Proceedings of the 2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Pittsburg, PA, USA, 11–13 July 2016; pp. 236–241.
20. Gokmen, T.; Vlasov, Y. Acceleration of Deep Neural Network Training with Resistive Cross-Point Devices: Design Considerations. *Front. Neurosci.* **2016**, *10*, 333. [CrossRef]
21. Ketkar, N. Introduction to PyTorch. In *Deep Learning with Python*; Apress: Berkeley, CA, USA, 2017; pp. 195–208. [CrossRef]
22. Goldsborough, P. A Tour of TensorFlow. *arXiv* **2016**, arXiv:1610.01178.
23. Abadi, M.; Isard, M.; Murray, D.G. A computational model for TensorFlow: An introduction. In Proceedings of the 1st ACM SIGPLAN International Workshop on Machine Learning and Programming Languages, Barcelona, Spain, 18 June 2017; pp. 1–7.
24. David, R.; Duke, J.; Jain, A.; Reddi, V.J.; Jeffries, N.; Li, J.; Kreeger, N.; Nappier, I.; Natraj, M.; Regev, S.; et al. Tensorflow lite micro: Embedded machine learning on tinyml systems. *arXiv* **2020**, arXiv:2010.08678.
25. Demosthenous, G.; Vassiliades, V. Continual Learning on the Edge with TensorFlow Lite. *arXiv* **2021**, arXiv:2105.01946.

26. Marchisio, A.; Hanif, M.A.; Khalid, F.; Plastiras, G.; Kyrkou, C.; Theocharides, T.; Shafique, M. Deep Learning for Edge Computing: Current Trends, Cross-Layer Optimizations, and Open Research Challenges. In Proceedings of the 2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Miami, FL, USA, 15–17 July 2019; pp. 553–559.
27. Awan, A.A.; Subramoni, H.; Panda, D.K. An In-depth Performance Characterization of CPU- and GPU-based DNN Training on Modern Architectures. In Proceedings of the Machine Learning on HPC Environments, Denver, CO, USA, 13 November 2017; p. 8.
28. Chen, X.; Chen, D.Z.; Hu, X.S. moDNN: Memory optimal DNN training on GPUs. In Proceedings of the 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresded, Germany, 19–23 March 2018.
29. Shorten, C.; Khoshgoftaar, T.M. A survey on Image Data Augmentation for Deep Learning. *J. Big Data* **2019**, *6*, 1–48. [CrossRef]
30. Jayalakshmi, T.; Santhakumaran, A. Statistical normalization and back propagation for classification. *Int. J. Comput. Theory Eng.* **2011**, *3*, 1793–8201.
31. Bhanja, S.; Das, A. Impact of data normalization on deep neural network for time series forecasting. *arXiv* **2018**, arXiv:1812.05519.
32. Koval, S.I. Data Preparation for Neural Network Data Analysis. In Proceedings of the IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus), St. Petersburg, Russia, 29 January–1 February 2018.
33. Bernard, J. Python Data Analysis with Pandas. In *Python Recipes Handbook*; Apress: New York, NY, USA, 2016; pp. 37–48.
34. Stancin, I.; Jovic, A. An overview and comparison of free Python libraries for data mining and big data analysis. In Proceedings of the 2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, Croatia, 20–24 May 2019; pp. 977–982.
35. Katsiapis, K.; Karmarkar, A.; Altay, A.; Zaks, A.; Polyzotis, N.; Ramesh, A.; Mathes, B.; Vasudevan, G.; Giannoumis, I.; Wilkiewicz, J.; et al. Towards ML Engineering: A Brief History of TensorFlow Extended (TFX). *arXiv* **2020**, arXiv:2010.02013.
36. Scheffer, T.; Joachims, T. Expected Error Analysis for Model Selection. In Proceedings of the International Conference on Machine Learning (ICML), Bled, Slovenia, 27–30 June 1999.
37. Roelofs, R.; Fridovich-Keil, S.; Miller, J.; Shankar, V.; Hardt, M.; Recht, B.; Schmidt, L. A Meta-Analysis of Overfitting in Machine Learning. In Proceedings of the 33rd Conference on Neural Information Processing Systems (NeurIPS), Vancouver, BC, Canada, 8–14 December 2019.
38. Aderson, M.R.; Cafarella, M. Input selection for fast feature engineering. In Proceedings of the IEEE 32nd International Conference on Data Engineering (ICDE), Helsinki, Finland, 16–20 May 2016.
39. Shin, H.-C.; Roth, H.R.; Gao, M.; Lu, L.; Xu, Z.; Nogues, I.; Yao, J.; Mollura, D.; Summers, R.M. Deep Convolutional Neural Networks for Computer-Aided Detection: CNN Architectures, Dataset Characteristics and Transfer Learning. *IEEE Trans. Med. Imaging* **2016**, *35*, 1285–1298. [CrossRef]
40. Kukačka, J.; Golkov, V.; Cremers, D. Regularization for Deep Learning: A Taxonomy. *arXiv* **2017**, arXiv:1710.10686.
41. Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; Salakhutdinov, R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *J. Mach. Learn. Res.* **2014**, *15*, 1929–1958.
42. Benz, P.; Zhang, C.; Karjauv, A.; Kweon, I.S. Revisiting Batch Normalization for Improving Corruption Robustness. In Proceedings of the 2021 IEEE Winter Conference on Applications of Computer Vision (WACV), Hawaii, HI, USA, 4–8 January 2021; pp. 494–503.
43. Weiss, K.; Khoshgoftaar, T.M.; Wang, D. A survey of transfer learning. *J. Big Data* **2016**, *3*, 1–40. [CrossRef]
44. Zamir, A.R.; Sax, A.; Shen, W.; Guibas, L.J.; Malik, J.; Savarese, S. Taskonomy: Disentangling task transfer learning. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–21 June 2018; pp. 3712–3722.
45. Kendall, A.; Gal, Y. What uncertainties do we need in bayesian deep learning for computer vision? In Proceedings of the Advances in Neural Information Processing Systems (NIPS), Long Beach, CA, USA, 4–9 December 2017; pp. 5574–5584.
46. Zhang, Z.; Xu, C.; Yang, J.; Tai, Y.; Chen, L. Deep hierarchical guidance and regularization learning for end-to-end depth estimation. *Pattern Recognit.* **2018**, *83*, 430–442. [CrossRef]
47. Zheng, Q.; Tian, X.; Yang, M.; Wang, H. Differential learning: A powerful tool for interactive content-based image retrieval. *Eng. Lett.* **2019**, *27*, 202–215.
48. Kobayashi, T. Flip-invariant motion representation. In Proceedings of the IEEE International Conference on Computer Vision (ICCV), Venice, Italy, 22–29 October 2017; pp. 5628–5637.
49. Zhang, K.; Zuo, W.; Chen, Y.; Meng, D.; Zhang, L. Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising. *IEEE Trans. Image Process.* **2017**, *26*, 3142–3155. [CrossRef] [PubMed]
50. Zhou, H.; Wen, J. Dynamic feature selection method with minimum redundancy information for linear data. *Appl. Intell.* **2020**, *50*, 3660–3677. [CrossRef]
51. Zheng, Q.; Yang, M.; Tian, X.; Jiang, N.; Wang, D. A Full Stage Data Augmentation Method in Deep Convolutional Neural Network for Natural Image Classification. *Discret. Dyn. Nat. Soc.* **2020**, *2020*, 1–11. [CrossRef]
52. Shawahna, A.; Sait, S.M.; El-Maleh, A. FPGA-Based Accelerators of Deep Learning Networks for Learning and Classification: A Review. *IEEE Access* **2019**, *7*, 7823–7859. [CrossRef]
53. Hao, C.; Chen, Y.; Zhang, X.; Li, Y.; Xiong, J.; Hwu, W.; Chen, D. Effective Algorithm-Accelerator Co-design for AI Solutions on Edge Devices. In Proceedings of the 2020 on Great Lakes Symposium on VLSI (GLSVLSI '20), Online, 7–9 September; pp. 283–290. [CrossRef]

54. Putnam, A.; Caulfield, A.M.; Chung, E.S.; Chiou, D.; Constantinides, K.; Demme, J.; Esmaeilzadeh, H.; Fowers, J.; Gopal, G.P.; Gray, J.; et al. A reconfigurable fabric for accelerating large-scale datacenter services. *Commun. ACM* **2016**, *59*, 114–122. [CrossRef]
55. Zhang, C.; Li, P.; Sun, G.; Guan, Y.; Xiao, B.; Cong, J. Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks. In Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '15), Monterey, CA, USA, 22–24 February 2015; pp. 161–170. [CrossRef]
56. Tridgell, S. Low Latency Machine Learning on FPGAs. Ph.D. Thesis, University of Sydney, Sydney, Australia, 2020.
57. Kim, Y.; Kim, H.; Yadav, N.; Li, S.; Choi, K.K. Low-Power RTL Code Generation for Advanced CNN Algorithms toward Object Detection in Autonomous Vehicles. *Electronics* **2020**, *9*, 478. [CrossRef]
58. Li, S. Towards Efficient Hardware Acceleration of Deep Neural Networks on FPGA. Ph.D. Thesis, University of Pittsburg, Pittsburg, PA, USA, 2018.
59. Samal, K. FPGA Acceleration of CNN Training. Master's Thesis, Georgia Institute of Technology, Atlanta, GA, USA, 2018.
60. Pitsis, A. Design and Implementation of an FPGA-Based Convolutional Neural Network Accelerator. Diploma Thesis, Technical University of Crete, Crete, Greece, 2018.
61. Su, C.; Zhou, S.; Feng, L.; Zhang, W. Towards high performance low bitwidth training for deep neural networks. *J. Semicond.* **2020**, *41*, 022404. [CrossRef]
62. Rastegari, M.; Ordonez, V.; Redmon, J.; Farhadi, A. *XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks*; Springer: Berlin/Heidelberg, Germany, 2016; pp. 525–542.
63. Minakova, S.; Stefanov, T. Buffer Sizes Reduction for Memory-efficient CNN Inference on Mobile and Embedded Devices. In Proceedings of the 2020 23rd Euromicro Conference on Digital System Design (DSD), Kranj, Slovenia, 26–28 August 2020; pp. 133–140.
64. Bilsen, G.; Engels, M.; Lauwereins, R.; Peperstraete, J. Cyclo-static dataflow. *IEEE Trans. Signal Process.* **1996**, *44*, 397–408. [CrossRef]
65. Benes, T.; Kekely, M.; Hynek, K.; Cejka, T. Pipelined ALU for effective external memory access in FPGA. In Proceedings of the 2020 23rd Euromicro Conference on Digital System Design (DSD), Portoroz, Slovenia, 26–28 August 2020; pp. 97–100.
66. Stramondo, G.; Gomony, M.D.; Kozicki, B.; De Laat, C.; Varbanescu, A.L. µ-Genie: A Framework for Memory-Aware Spatial Processor Architecture Co-Design Exploration. In Proceedings of the 2020 23rd Euromicro Conference on Digital System Design (DSD), Portoroz, Slovenia, 26–28 August 2020; pp. 180–184.
67. Abdelouahab, K.; Pelcat, M.; Sérot, J.; Bourrasset, C.; Berry, F. Tactics to Directly Map CNN Graphs on Embedded FPGAs. *IEEE Embed. Syst. Lett.* **2017**, *9*, 113–116. [CrossRef]
68. Duarte, J.M.; Han, S.; Harris, P.; Jindariani, S.; Kreinar, E.; Kreis, B.; Ngadiuba, J.; Pierini, M.; Rivera, R.; Tran, N.; et al. Fast inference of deep neural networks in FPGAs for particle physics. *J. Instrum.* **2018**, *13*, P07027. [CrossRef]
69. Fahim, F.; Hawks, B.; Herwig, C.; Hirschauer, J.; Jindariani, S.; Tran, N.; Carloni, L.P.; Di Guglielmo, G.; Harris, P.; Krupa, J.; et al. hls4ml: An Open-Source Codesign Workflow to Empower Scientific Low-Power Machine Learning Devices. *arXiv* **2021**, arXiv:2103.05579.
70. Venieris, S.I.; Bouganis, C.S. fpgaConvNet: A framework for mapping convolutional neural networks on FPGAs. In Proceedings of the 2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), Washington, DC, USA, 1–3 May 2016; pp. 40–47.
71. Ghaffari, A.; Savaria, Y. CNN2Gate: An Implementation of Convolutional Neural Networks Inference on FPGAs with Automated Design Space Exploration. *Electronics* **2020**, *9*, 2200. [CrossRef]
72. 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 (FPGA '17), Monterey, CA, USA, 22–24 February 2017; pp. 65–74. [CrossRef]
73. Vitis AI. Xilinx Viti AI. Available online: https://www.xilinx.com/products/design-tools/vitis/vitis-ai.html (accessed on 10 January 2021).
74. Rupanetti, D.; Nepal, K.; Salamy, H.; Min, C. Cost-Effective, Re-Configurable Cluster Approach for Resource Constricted FPGA Based Machine Learning and AI Applications. In Proceedings of the 2020 10th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, USA, 6–8 January 2020; pp. 0228–0233. [CrossRef]
75. Sharma, H.; Park, J.; Mahajan, D.; Amaro, E.; Kim, J.K.; Shao, C.; Mishra, A.; Esmaeilzadeh, H. From high-level deep neural models to FPGAs. In Proceedings of the 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Taipei, Taiwan, 15–19 October 2016; pp. 1–12.
76. Che, S.; Li, J.; Sheaffer, J.W. Accelerating Compute-Intensive Applications with GPUs and FPGAs. In Proceedings of the Symposium on Application Specific Processors, Anaheim, CA, USA, 8–9 June 2008; pp. 101–107. [CrossRef]
77. NVIDIA. Cuda-Zone. Available online: https://developer.nvidia.com/cuda-zone (accessed on 22 January 2021).
78. Nvidia Corporation. TensorRT. Available online: https://developer.nvidia.com/tensorrt (accessed on 22 January 2021).
79. Song, M.; Hu, Y.; Chen, H.; Li, T. Towards Pervasive and User Satisfactory CNN across GPU Microarchitectures. In Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA), Austin, TX, USA, 4–8 February 2017; pp. 1–12. [CrossRef]

80. Van Essen, B.; Macaraeg, C.; Gokhale, M.; Prenger, R. Accelerating a Random Forest Classifier: Multi-Core, GP-GPU, or FPGA? In Proceedings of the IEEE 20th International Symposium on Field-Programmable Custom Computing Machines, Toronto, ON, CA, 9–12 May 2012; pp. 232–239. [CrossRef]
81. Czarnul, P.; Proficz, J.; Krzywaniak, A. Energy-Aware High-Performance Computing: Survey of State-of-the-Art Tools, Techniques, and Environments. *Sci. Program.* **2019**, *2019*, 8348791. [CrossRef]
82. Price, D.C.; Clark, M.A.; Barsdell, B.R.; Babich, R.; Greenhill, L.J. Optimizing performance-per-watt on GPUs in high performance computing. *Comput. Sci. Res. Dev.* **2016**, *31*, 185–193. [CrossRef]
83. Li, X.; Zhang, G.; Huang, H.H.; Wang, Z.; Zheng, W. Zheng Performance Analysis of GPU-Based Convolutional Neural Networks. In Proceedings of the 45th International Conference on Parallel Processing (ICPP), Philadelphia, PA, USA, 16–19 August 2016; pp. 67–76. [CrossRef]
84. Li, D.; Chen, X.; Becchi, M.; Zong, Z. Evaluating the Energy Efficiency of Deep Convolutional Neural Networks on CPUs and GPUs. In Proceedings of the IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom) (BDCloud-SocialCom-SustainCom), Atlanta, GA, USA, 8–10 October 2016; pp. 477–484. [CrossRef]
85. cuda-convnet2. Available online: https://github.com/akrizhevsky/cuda-convnet2 (accessed on 22 June 2021).
86. Chetlur, S.; Woolley, C.; Vandermersch, P.; Cohen, J.; Tran, J.; Catanzaro, B.; Shelhamer, E. cudnn: Efficient primitives for deep learning. *arXiv* **2014**, arXiv:1410.0759.
87. Deep Learning with OpenCV DNN Module. Available online: https://learnopencv.com/deep-learning-with-opencvs-dnn-module-a-definitive-guide/ (accessed on 22 June 2021).
88. Abadi, M.; Barham, P.; Chen, J.; Chen, Z.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Irving, G.; Isard, M.; et al. Tensorflow: A system for large-scale machine learning. In Proceedings of the 12th (USENIX) Symposium on Operating Systems Design and Implementation (OSDI '16), Savannah, GA, USA, 2–4 November 2016; pp. 265–283.
89. Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Adv. Neural Inf. Process. Syst.* **2019**, *32*, 8026–8037.
90. Huynh, L.N.; Balan, R.K.; Lee, Y. DeepSense: A GPU-based Deep Convolutional Neural Network Framework on Commodity Mobile Devices. In Proceedings of the 2016 Workshop on Wearable Systems and Applications. Association for Computing Machinery, Singapore, 30 June 2016; pp. 25–30. [CrossRef]
91. Wang, E.; Davis, J.J.; Zhao, R.; Ng, H.-C.; Niu, X.; Luk, W.; Cheung, P.Y.K.; Constantinides, G.A. Deep Neural Network Approximation for Custom Hardware. *ACM Comput. Surv.* **2019**, *52*, 1–39. [CrossRef]
92. Misra, J.; Saha, I. Artificial neural networks in hardware: A survey of two decades of progress. *Neurocomputing* **2010**, *74*, 239–255. [CrossRef]
93. Fischer, M.; Wassner, J. BinArray: A Scalable Hardware Accelerator for Binary Approximated CNNs. In Proceedings of the 2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC), Online, 27–30 January 2021; pp. 0197–0205.
94. Mao, W.; Xiao, Z.; Xu, P.; Ren, H.; Liu, D.; Zhao, S.; An, F.; Yu, H. Energy-Efficient Machine Learning Accelerator for Binary Neural Networks. In Proceedings of the 2020 on Great Lakes Symposium on VLSI, New York, NY, USA, 8–11 September 2020.
95. Véstias, M. Processing Systems for Deep Learning Inference on Edge Devices. In *Smart Sensors, Measurement and Instrumentation*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 213–240.
96. Nascimento, D.V.; Xavier-de-Souza, S. An OpenMP translator for the GAP8 MPSoC. *arXiv* **2020**, arXiv:2007.10219.
97. Demler, M. Brainchip Akida Is a Fast Learne. In *Microprocessor Report*; The Linley Group: Mountain View, CA, USA, 2019.
98. Pell, R. Maxim launches neural network accelerator chip. *eeNewsEmbedded*, 8 October 2020.
99. DSPgroup DBM10 AI/ML SoC with DSP and Neural Network Accelerator. Available online: https://www.dspg.com/wp-content/uploads/2021/01/Power-Edge-AI-ML-SoC.pdf (accessed on 1 July 2021).
100. Synaptics. Synaptics Expands into Low Power Edge AI Applications with New Katana Platform. Available online: https://www.synaptics.com/company/news/katana (accessed on 2 August 2021).
101. SynSense. DynapSE2 a Low Power Scalable SNN Processor. Available online: https://www.synsense-neuromorphic.com/wp-content/uploads/2018/10/wp-dynapse2-opt-20181031.pdf (accessed on 2 August 2021).
102. Fujii, T.; Toi, T.; Tanaka, T.; Togawa, K.; Kitaoka, T.; Nishino, K.; Nakamura, N.; Nakahara, H.; Motomura, M. New Generation Dynamically Reconfigurable Processor Technology for Accelerating Embedded AI Applications. In Proceedings of the 2018 IEEE Symposium on VLSI Circuits, Honolulu, HI, USA, 18–22 June 2018; pp. 41–42.
103. Dalgaty, T.; Esmanhotto, E.; Castellani, N.; Querlioz, D.; Vianello, E. Ex Situ Transfer of Bayesian Neural Networks to Resistive Memory-Based Inference Hardware. *Adv. Intell. Syst.* **2021**, 2000103. [CrossRef]
104. Véstias, M. Efficient Design of Pruned Convolutional Neural Networks on FPGA. *J. Signal Process. Syst.* **2021**, *93*, 531–544. [CrossRef]
105. Qi, X.; Liu, C. Enabling Deep Learning on IoT Edge: Approaches and Evaluation. In Proceedings of the 2018 IEEE/ACM Symposium on Edge Computing (SEC), Bellevue, WA, USA, 25–27 October 2018; pp. 367–372.
106. Reed, R. Pruning algorithms-a survey. *IEEE Trans. Neural Netw.* **1993**, *4*, 740–747. [CrossRef] [PubMed]
107. Lin, D.; Talathi, S.; Annapureddy, S. Fixed Point Quantization of Deep Convolutional Networks. In Proceedings of the 33rd International Conference on Machine Learning, New York, NY, USA, 19–24 June 2016.

108. Xu, D.; Li, T.; Li, Y.; Su, X.; Tarkoma, S.; Jiang, T.; Crowcroft, J.; Hui, P. Edge Intelligence: Architectures, Challenges, and Applications. *arXiv* **2020**, arXiv:2003.12172.
109. Manessi, F.; Rozza, A.; Bianco, S.; Napoletano, P.; Schettini, R. Automated Pruning for Deep Neural Network Compression. In Proceedings of the 2018 24th International Conference on Pattern Recognition (ICPR), Beijing, China, 20–24 August 2018; pp. 657–664.
110. Han, S.; Mao, H.; Dally, W.J. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *arXiv* **2015**, arXiv:1510.00149.
111. Yu, J.; Lukefahr, A.; Palframan, D.; Dasika, G.; Das, R.; Mahlke, S. Scalpel: Customizing DNN pruning to the underlying hardware parallelism. *ACM SIGARCH Comput. Arch. News* **2017**, *45*, 548–560. [CrossRef]
112. Chen, J.; Ran, X. Deep Learning with Edge Computing: A Review. *Proc. IEEE* **2019**, *107*, 1655–1674. [CrossRef]
113. Li, Y.; Liu, Z.; Liu, W.; Jiang, Y.; Wang, Y.; Goh, W.L.; Yu, H.; Ren, F. A 34-FPS 698-GOP/s/W Binarized Deep Neural Network-Based Natural Scene Text Interpretation Accelerator for Mobile Edge Computing. *IEEE Trans. Ind. Electron.* **2019**, *66*, 7407–7416. [CrossRef]
114. Gysel, P.; Motamedi, M.; Ghiasi, S. Hardware-oriented approximation of convolutional neural networks. In Proceedings of the 4th International Conference on Learning Representations, San Juan, Puerto Rico, 2–4 May 2016.
115. Wang, J.; Lou, Q.; Zhang, X.; Zhu, C.; Lin, Y.; Chen, D. Design Flow of Accelerating Hybrid Extremely Low Bit-Width Neural Network in Embedded FPGA. In Proceedings of the 2018 28th International Conference on Field Programmable Logic and Applications (FPL), Lausanne, Switzerland, 29 August–2 September 2018; pp. 163–1636.
116. Vestias, M.P.; Duarte, R.P.; De Sousa, J.T.; Neto, H.C. A Configurable Architecture for Running Hybrid Convolutional Neural Networks in Low-Density FPGAs. *IEEE Access* **2020**, *8*, 107229–107243. [CrossRef]
117. Albericio, J.; Judd, P.; Hetherington, T.H.; Aamodt, T.M.; Jerger, N.D.E.; Moshovos, A. Cnvlutin: Ineffectual-Neuron-Free Deep Neural Network Computing. In Proceedings of the 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA), Seoul, Korea, 18–22 June 2016; pp. 1–13.
118. Aimar, A.; Mostafa, H.; Calabrese, E.; Rios-Navarro, A.; Tapiador-Morales, R.; Lungu, I.-A.; Milde, M.B.; Corradi, F.; Linares-Barranco, A.; Liu, S.-C.; et al. NullHop: A Flexible Convolutional Neural Network Accelerator Based on Sparse Representations of Feature Maps. *IEEE Trans. Neural Netw. Learn. Syst.* **2019**, *30*, 644–656. [CrossRef]
119. Han, S.; Liu, X.; Mao, H.; Pu, J.; Pedram, A.; Horowitz, M.A.; Dally, W.J. EIE: Efficient Inference Engine on Compressed Deep Neural Network. In Proceedings of the 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA), Seoul, Korea, 18–22 June 2016; pp. 243–254.
120. Hang, S.; Du, Z.; Zhang, L.; Lan, H.; Liu, S.; Li, L.; Guo, Q.; Chen, T.; Chen, Y. Cambricon-X: An accelerator for sparse neural networks. In Proceedings of the 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Taipei, Taiwan, 15–19 October 2016; pp. 1–12.
121. Yao, S.; Zhao, Y.; Zhang, A.; Su, L.; Abdelzaher, T. Deepiot: Compressing deep neural network structures for sensing systems with a compressor-critic framework. In Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems, Delft, The Netherlands, 6–8 November 2017; pp. 1–14.
122. Kantas, N.; Doucet, A.; Singh, S.S.; Maciejowski, J.; Chopin, N. On Particle Methods for Parameter Estimation in State-Space Models. *Stat. Sci.* **2015**, *30*, 328–351. [CrossRef]
123. Imani, M.; Ghoreishi, S.F. Two-Stage Bayesian Optimization for Scalable Inference in State-Space Models. *IEEE Trans. Neural Netw. Learn. Syst.* **2021**, 1–12. [CrossRef]
124. Sze, V.; Chen, Y.-H.; Yang, T.-J.; Emer, J.S. How to Evaluate Deep Neural Network Processors: TOPS/W (Alone) Considered Harmful. *IEEE Solid-State Circuits Mag.* **2020**, *12*, 28–41. [CrossRef]
125. Williams, S.; Waterman, A.; Patterson, D. Roofline: An insightful visual performance model for multicore architectures. *Commun. ACM* **2009**, *52*, 65–76. [CrossRef]