agriculture

MDPI

Review

Review of Energy-Efficient Embedded System Acceleration of Convolution Neural Networks for Organic Weeding Robots

Vitali Czymmek *, Carolin Köhn, Leif Ole Harders and Stephan Hussmann *

Faculty of Engineering, West Coast University of Applied Sciences, Fritz-Thiedemann-Ring 20, 25746 Heide, Germany; leifharders@fh-westkueste.de (L.O.H.)

* Correspondence: czymmek@fh-westkueste.de (V.C.); hussmann@fh-westkueste.de (S.H.)

Abstract: The sustainable cultivation of organic vegetables and the associated problem of weed control has been a current research topic for some time. Despite this, the use of chemical and synthetic pesticides increases every year. This is to be solved with the help of an automated robot system. The current version of the weeding robot uses GPUs to execute the inference phase. This requires a lot of energy for an 8-track robot. To enable autonomous solar operation, the system must be made more energy efficient. This work aims to evaluate possible approaches and the current state of research on implementing convolution neural networks on low power embedded systems. In the course of the work, the technical feasibility for the implementation of CNNs in FPGAs was examined, in particular, following the example of a feasibility analysis. This paper shows that the acceleration of convolution neural networks using FPGAs is technically feasible for use as detection hardware in the weeding robot. With the help of the current state of research and the existing literature, the optimization possibilities of the hardware and software have been evaluated. The trials of different networks on different hardware accelerators with diverse approaches were investigated and compared.

Keywords: convolution neural networks (CNN); FPGA; embedded systems; machine learning; edge devices; real time image processing; agricultural machinery and equipment for precision farming

Citation: Czymmek, V.; Köhn, C.; Harders, L.O.; Hussmann, S. Review of Energy-Efficient Embedded System Acceleration of Convolution Neural Networks for Organic Weeding Robots. *Agriculture* **2023**, *13*, 2103. https://doi.org/10.3390/agriculture13112103

Academic Editor: Galibjon M. Sharipov

Received: 5 October 2023
Revised: 1 November 2023
Accepted: 2 November 2023
Published: 6 November 2023

1. Introduction

One of the global goals in organic farming is to minimize the use of chemical or synthetic pesticides to protect the environment. For example, at present, weeding regulation on carrot fields is performed by hand. This manual weed control is very expensive. Our cooperation partner, Westhof Bio GmbH in Germany, for example, spends over 170,000 EUR per year for the manual elimination of weeds by human workers. Furthermore, it is more and more difficult to find workers for this task. The Westcoast University of Applied Sciences with its research projects is working on the problem in the form of a weed control robot that autonomously takes pictures, detects, and classifies plants, and then destroys the weeds. Our motivation for qualitative research into the implementation of artificial neural networks is the sustainable cultivation of organic vegetables and the associated problem of weed destruction, which is to be solved with the aid of the 8-track automated robot system seen in Figure 1.

The development of a fully automated weeding robot makes it possible to eliminate the need for chemicals. One challenge is plant classification and weed control close to the crops being grown. Currently, the robot is suitable for cultivated carrots, which are a challenge for automated robotic systems due to their nature. The robot shown in Figure 1 is a solar and diesel-powered weeding robot prototype. It is designed to weed plant rows simultaneously using identical weeding modules following a typical layout for agricultural machinery. The system consists of a manually steered carrier system and autonomous weeding modules. These sensor/actuator modules comprise a vision based weed detection mechanism (sensor) and a mechanical weeding mechanism (actuator). A camera takes an

image, then the image is run through a convolution neural network, and then the position of the weed is sent to the destruction unit and removed by means of an array of pneumatic cylinders and electric motors. Real-time implementation and large data volumes as well as detection and classification despite different weather conditions and stages are critical aspects [1]. The goal is the cost-effective, automated, and accurate implementation of weed control using artificial neural network implementations. The current version of the weeding robot uses GPUs to execute the inference phase. This work aims to evaluate possible approaches and the current state of research on implementing convolution neural networks on low power embedded systems. By addressing the critical need for reduced energy consumption in autonomous robotic systems, this study evaluates the technical feasibility of implementing CNNs in field-programmable gate arrays (FPGAs). This research not only emphasizes the development of sustainable solutions for organic weed control, but also offers valuable insights into the optimization of both hardware and software components. In the course of this work, the technical feasibility for the implementation of CNNs in FPGAs will be examined, in particular, following the example of a feasibility analysis. For this purpose, the basics of artificial neural networks, especially CNNs, will be presented first. Next, hardware options for CNN acceleration will be compared. In the context of the implementation of CNNs on FPGAs, optimization approaches will be presented. Hardware options will be cited and, with respect to the weeding robot, the implementation of a YOLO network using FPGAs will be explored.

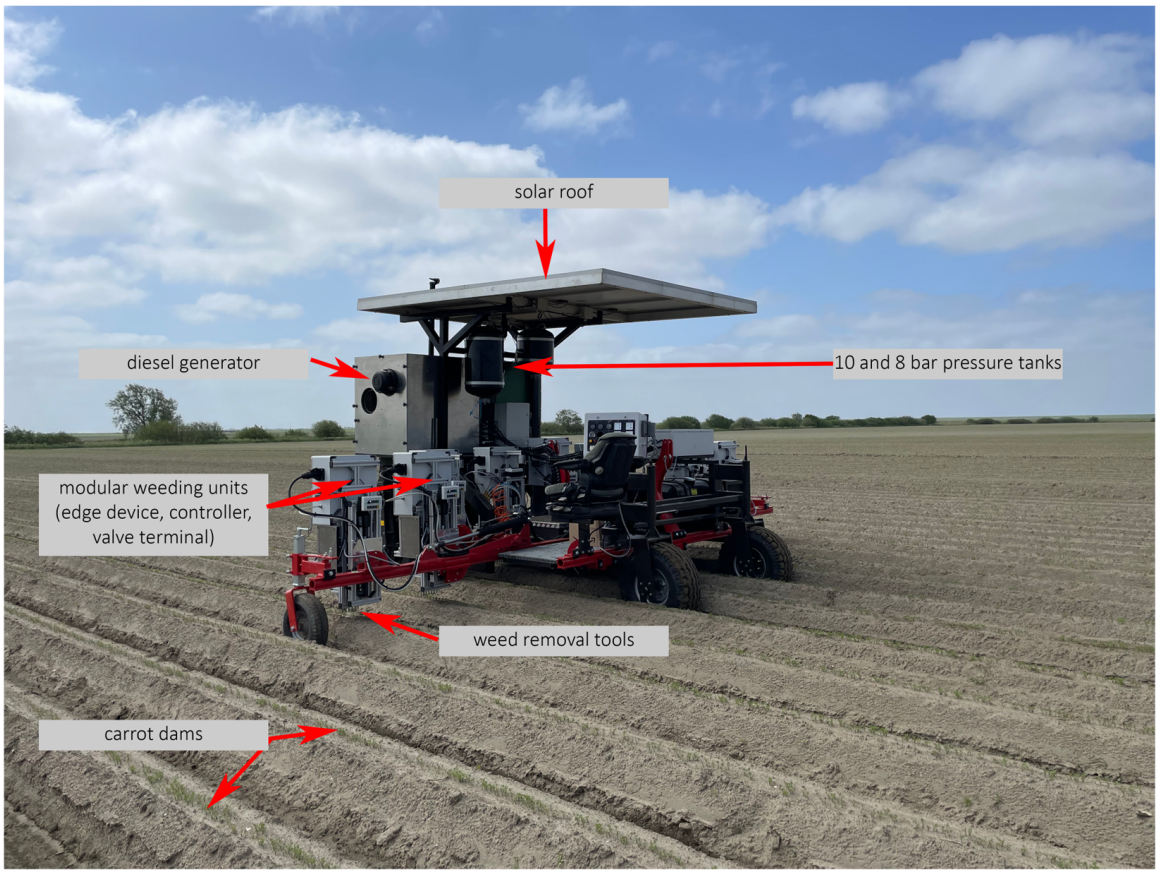

Figure 1. Image of the developed 8-track automated weeding robot on the field of our cooperation partner Westhof Bio GmbH in Germany.

2. Basics of Artificial Neural Networks

In this chapter, artificial neural networks are examined. CNNs are especially focused on due to the application focus on image processing. After an overview of the subject, the layers and computations within the networks as well as the training and inference phases

are discussed in more detail. Application areas highlighting the importance of artificial networks and different architectures are presented.

Instead of using elaborate algorithms for AI applications, deep learning can be used to extract data from large data sets. The learning capabilities of computers are enabled by artificial neural networks. These are based on biological neural networks or the central nervous systems of living organisms [2]. Deep neural networks and deep learning are a subfield of artificial intelligence. The learning process here does not require explicit programming. Arthur Samuel defined machine learning (ML) in 1959 as a way to make computers capable of learning without explicit programming. A new problem is abstracted through a training process [3]. Deep neural networks have multiple hidden layers and can be used to analyze large amounts of data after training. Unlike conventional solutions and algorithms, neural networks are flexible [4]. Particularly popular forms are RNNs for handling problems involving time sequences and CNNs for learning spatial features [5]. Convolutional neural networks, also referred to as ConvNets or CNNs, are a widely used architecture for parallel processing techniques enabled by GPUs or FPGAs. CNNs are suitable for image processing.

2.1. CNN Structure

A typical CNN structure consists of a number of layers. Each layer is responsible for inputting a data set, a feature map (FM), and produces a new set of FMs [6]. In the inference phase of CNNs, a computationally intensive convolutional layer is used to extract features, such as lines or edges. Pooling layers are used to reduce mismatches by averaging common features in an image region. A handoff is made to additional convolutional and pooling layers. The number of CNN layers affects image recognition accuracy and performance requirements. If sufficient memory bandwidth is available, the layers can be processed in parallel. This schematic structure can be seen in Figure 2.

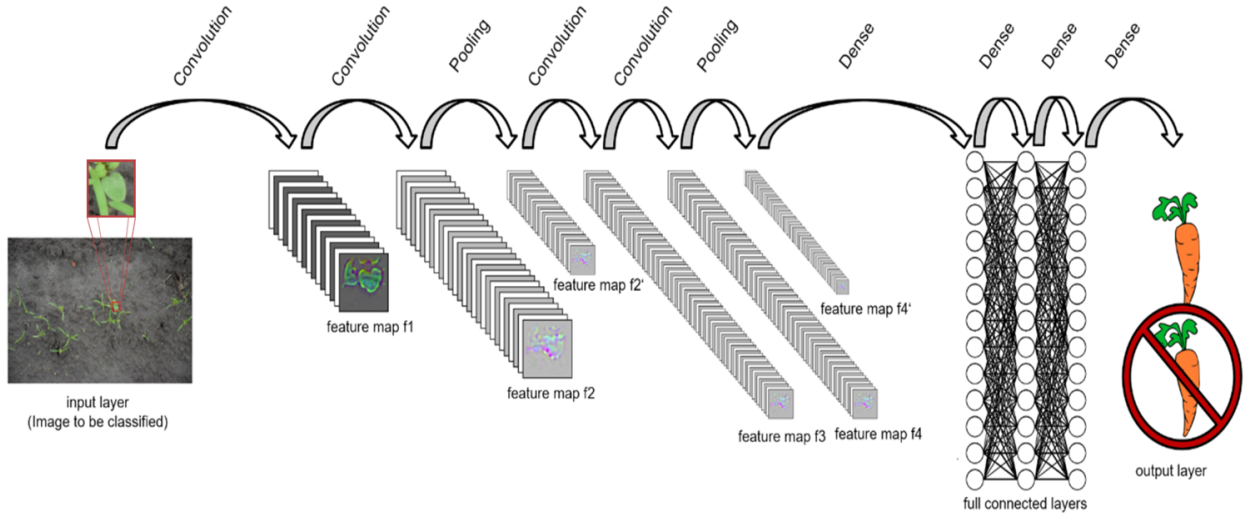

Figure 2. Example CNN for classifying carrot plants in organic farming.

Repeated use of weights is called division of weights. This approach will be discussed in more detail later in this paper. Thus, the memory requirement for weights can be reduced. For the structuring of the computation and weight division, a convolution can be used. In a convolution, the weighted sum for each output activation is calculated by the neighborhood of the input activations. This approach is used in convolutional layers. In a convolution operation, a matrix smaller than the original image matrix is produced by dragging a filter (feature identifier kernel) over the image to produce a result. A filter is an array of weights or parameters computed during the training phase. The values of the filter are multiplied by the original pixel values. First, an element-by-element multiplication is performed.

Then, the products are summed up to produce a number. The inputs and outputs of the convolution layer are a series of FM arrays [7]. Figure 3 shows the trained filter of the first convolution layer.

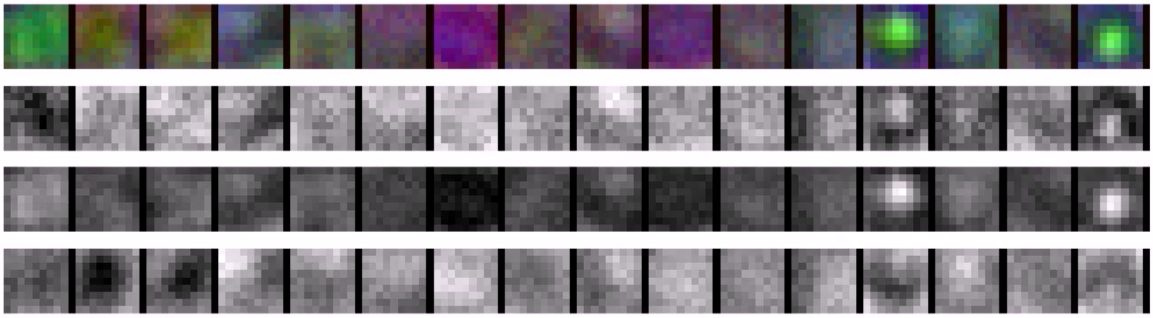

Figure 3. Filters of the first convolution layer.

Input activations of the layer are structured as a series of input feature maps (ifmaps) called channels. Each channel is convolved with a different 2D filter. Particularly good results are achieved by a deep hierarchy of layers. Such a hierarchy is particularly suitable in image processing, speech recognition, and robotics. The set of 2D filters used is combined as one 3D filter. Convolution results are summarized for each point across all channels. A 1D distortion (bias) can be added to the filter results. The result of the convolution is an output activation. Output activations include one channel of output feature maps (ofmap). More 3D filters can be applied to the same input, creating more output channels. Multiple ifmaps can be processed into a batch. This approach can improve the potential reuse of filter weights. Filters are composed of weights. Ifmaps and ofmaps are composed of activations.

In common CNN networks, between five and a thousand convolutional layers are applied, as well as one to three FC layers after the convolutional layers for classification reasons. Filters are also applied to ifmaps in the FC layer. Other optional layers can be applied, such as pooling layers, normalization layers, or activation functions. Activation functions are comparable to the action potential in neurons. If an action potential is present, it instructs the neuron to act in a biological neural network. For activation functions, sigmoid and TanH-functions, which are non-linear and require much training time, or the ReLU (rectified linear unit) function, which requires less training time and has less computational complexity, can be used. ReLU is known for its simplicity and ability to accelerate training. Other forms include leaky ReLU, parametric ReLU, and exponetial ReLU. For example, batch normalization (BN) or local-response normalization (LRN) can be applied in a normalization layer. The normalization layer is used between the convolutional and FC layers.

In the pooling or subsampling layer, the number of parameters and computations is reduced. In the pooling layer, calculations are performed that are used to reduce the dimensionality of the feature maps. For this purpose, pooling layers are applied between convolutional layers. Max, avg, or min operations can be applied, and selecting the maximum value for a filter is a best practice.

The different layers of the CNN, consisting of convolutional, ReLU, activation, pooling, and FC layers, have different requirements. Convolutional layers have high computational intensity due to multiplication and addition operations (MACs). FC layers require large memory due to the weights that are trained [8].

2.2. Training and Inference Phase

To increase the accuracy of CNN networks, large amounts of data are required for the training phases and the determination of weights and thresholds. The complex models resulting from the training phases are often implemented on GPUs for training or classification purposes. A distinction is made between the training and inference phases. Inference

is often performed on embedded devices with limited resources. Training requires large data sets and computational resources.

First, lines and edges are abstracted. Later shapes are composed from these attributes. Subsequently, an object or a scene is derived from the shapes. The program is not changed during the learning process, instead the weights are determined during training. After the training phase, the intended task can be executed. Inference means executing the program with the weights determined in the training phase. The distance between the correct score of the result and the score produced by the network is defined as the loss. The goal is to minimize the average loss. To achieve this and to optimize the weights accordingly, the largest possible training set must be used. A heuristic optimization procedure called gradient descent can be used, iteratively repeated. An efficient method that can be used in this process is backpropagation. This involves passing values backwards through the network to gain an understanding of how the loss is affected by the weights.

2.3. CNN Architectures

The various CNN architectures differ according to the number of layers, layer types, connections between layers, and layer shapes, in terms of filter size, the number of channels, and filters. The CNN architecture is selected depending on the application purpose and hardware. The performance and accuracy of each architecture are used as a guide. Architecture performance is usually measured using the ImageNet Challenge. One of the first CNN architectures was LeNet in 1989. In 2012, AlexNet was the first CNN architecture to win the ImageNet Challenge. Other well-known architectures include Overfeat and VGG-16. In networks such as GoogLeNet and ResNet, the number of layers continues to increase, making reasonable hardware implementation increasingly difficult. For this reason, efforts are being made to compress the networks. Binarized neural networks (BNNs) are a possible optimized form of deep neural networks where the value of neurons and weights is either 1 or -1, allowing representation using a single bit [9].

3. Hardware Accelerator Based on FPGAs

In this section, first various hardware accelerators, including GPUs, CPUs, ASICs, and FPGAs, are presented and primarily compared with the FPGA. Next, the characteristics of FPGAs are highlighted by showing the various advantages and disadvantages of FPGAs. Finally, at the end of this section, an overview of the state-of-the-art FPGA accelerators is provided. This section highlights the primary challenges in using FPGAs as hardware accelerators, which are addressed in the next section for the purpose of successfully implementing CNNs on FPGA mesh accelerators.

3.1. Comparison of Hardware Accelerators

Performing inference using external servers or the cloud has disadvantages in terms of data security and the robustness of the data connection. Direct evaluation on the end device can shorten the response time. Sufficient computational resources are required for industrial automation and for implementing machine vision by using CNNs. FPGAs have advantages due to their flexible and configurable architecture. In contrast, the architecture of GPUs is fixed. The FPGA architecture reduces latency to a minimum due to its deterministic design. Efficient resource consumption can be achieved by building each CNN layer close to the FPGA's memory.

Focusing on the advancement of network models has resulted in large CNN models suitable for the image classification of images as large as 224×224 pixels, requiring 39 billion floating point number operations (FLOPs) and over 500 MB of model parameters. The computational complexity is proportional to the size of the input image. As a result, processing images with a high resolution can require more than 100 billion operations. To run such complex network models, a suitable computational platform must be selected. CPUs can perform 10–100 GFLOP/s, but their energy efficiency is usually less than 1 GOP/s, which makes it difficult to achieve the high performance that cloud applications require as

well as the low power consumption that mobile applications require. GPUs, on the other hand, can achieve peak performance at up to 10 TOP/s, making them a good choice for applications that require high performance. FPGAs can enable high parallelization and simplify the computational process for some neural network models by building their hardware logic. Another approach to the practical implementation of neural networks, is to simplify the network models in a hardware friendly way. This is not intended to bring about any reduction in the accuracy of the network model. FPGAs can achieve better energy efficiency than CPUs or GPUs [10]. A GPU is effective at processing the same set of operations in parallel single instruction, multiple data (SIMD). It has a well-defined instruction set and fixed word sizes. For example, single or double precision integer and floating point values. An FPGA on the other hand is effective at processing the same operations in parallel multi instructions, multiple data (MIMD). It does not have a predefined instruction set or a fixed data width. Figure 4 shows a schematic example of these three types.

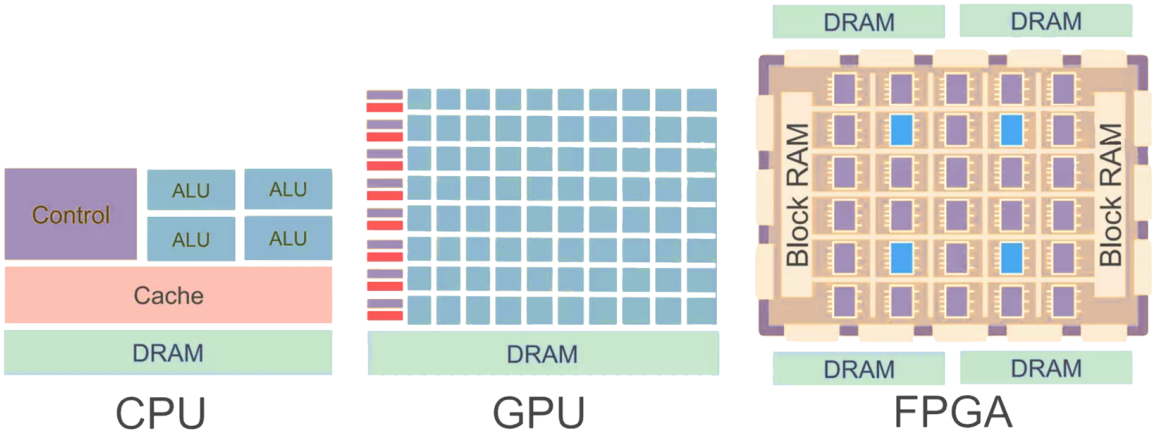

Figure 4. Schematic overview of parallel processing in CPUs, GPUs and FPGAs.

The selected hardware should enable high throughput as well as high energy efficiency. Important components of the convolutional layers and FC layers are multiply- accumulate operations (MACs). Parallelization is necessary for good performance. Temporal parallelization is most often used with CPUs and GPUs, where computational transformation on the kernel reduces the number of multiplications and thus increases throughput. SIMD and SIMT allow the parallelization of MACs. All ALUs share the same control unit and memory (register file, RF). Convolutional as well as FC layers are mapped as matrix multiplication, which can be optimized by libraries for CPUs, such as OpenBLAS or Intel MKL, and for GPUs, such as cuBLAS or cuDNN. In spatial parallelization, instead of centralized control for a large set of ALUs, a processing chain is formed. Thus, data can be passed directly and ALUs can have their own control unit and local memory. An ALU with its own local memory is called a processing engine (PE). Spatial parallelization is mostly used for deep neural networks on ASICs and FPGAs [11]. Table 1 shows a comparison between the abovementioned hardware accelerators.

Table 1. Comparison of CPUs, GPUs, FPGAs and ASICS for machine learning.

Criteria	CPU	GPU	FPGA	ASIC
Processing Peak Power	moderate	high	very high	highest
Power Consumption	high	very high	low	very low
Flexibility	highest	moderate	high	lowest
Training	poor	best yet	not efficient	not available
Inference	poor	good	best	potentially best

3.2. FPGA Features

An FPGA consists of a gate of programmable logic blocks, including general logic, memory, and multiplier blocks along with programmable conductive fabrics that allow the custom blocks to be connected. The gate is surrounded by input and output blocks that connect the chip to the external circuitry. FPGAs can be electronically programmed [12]. Additionally, FPGAs have embedded components such as digital signal processing (DSP) blocks. These are used to perform arithmetically intensive operations such as MACs (multiply- accumulate), block RAMs (BRAMs), look-up tables (LUTs), flip-flops (FFs), clock management units, and high-speed I/O. The important components of an FPGA accelerator are the processing elements (PE) and on-chip RAMs (M20Ks). There are two types of RAM resources in an FPGA: distributed RAM (DRAM) and block RAM (BRAM). DRAMs are implemented by LUTs, which wastes LUT resources and slows down the speed of read and write data, BRAMs are integrated as internal RAM resources on the chip, which allows read and write operations to be sped up [13].

FPGAs and ASICs operate at a lower clock frequency, resulting in lower energy consumption [10]. First-in-first-out (FIFO) buffer memory hardware structures can be used. FPGAs are suitable for battery-powered devices. The performance of CNNs on FPGAs is measured in giga operations per second (GOP/s) [14]. FPGAs enable the implementation of irregular parallelism and user-defined data types. As a result, higher computational throughput can be achieved [6]. FPGAs show good performance, but a balance must be struck between latency, precision, and hardware complexity [14]. FPGAs offer a good trade-off between performance and cost. Deep neural networks such as AlexNet and VGG are based on multiplications of dense floating point number matrices (GEMM) with 32-bit floating point numbers (FP32). These are well suited for acceleration on GPUs. FPGAs offer superior energy efficiency. On the downside, they do not offer the performance of GPUs. Crucial in FPGA accelerators are a high number of DSPs and on-chip RAMs (M20K memory blocks) as well as a high memory bandwidth (HBMs) and an improved frequency, e.g., by a HyperFlex core architecture, to match the performance for floating point operations to the performance of GPUs. Corresponding networks such as TNNs, ResNets or BNNs show better performance on FPGAs than on GPUs [15]. FPGAs show better computational density per watt for additions up to 16-bit and 32-bit fixed-point numbers than GPUs, whose computational density is better suited for floating-point number operations.

System-on-modules (SoMs) or system-on-chips (SoCs) combine an FPGA with an ARM processor. This simplifies the development of artificial intelligence systems. Instead of running inference in the cloud, it can be executed at the edge by using accelerators such as FPGAs, GPUs, DSPs, and ASICs. Additional tasks can be executed through the processor. An SoC combines an FPGA with a CPU on a chip and thus offers additional advantages for embedded applications [16]. The FPGA serves as a hardware accelerator for executing inference. The CPU takes over control tasks. This exploits the advantages of FPGAs over GPUs and ASICs: easy integration of different interfaces and sensors as well as the necessary flexibility to apply neural networks is enabled. SoCs are used to achieve low and deterministic latency. Thus, SoCs are suitable for real-time applications. In addition, good energy efficiency is achieved. Besides executing inference, communication with the host computer and other peripherals can be enabled simultaneously with actuator control [17].

A high-performance CNN processor called a deep learning processing unit (DPU) can be used [18]. To improve performance on FPGAs, DPUs have been released by Xilinx. Unlike other FPGAs that focus on specific functions of CNNS, DPUs support the basic functions of deep learning. A nested scheduler for task distribution between ARM and different DPUs plays an important role. For this purpose, Xilinx has released a hybrid CPU-FPGA MPSoC [19].

3.3. State-of-the-Art FPGA Accelerators

Various manufacturers offer FPGAs for accelerating neural networks. These include, for example, Xilinx, Intel, Efinix, Lattice, Semiconductor, Quick Logic, Achronix, Cypress Semiconductor, Phytec, NXP Semiconductor and STMicroelectronics. Often, an SoC is applied to provide the most efficient acceleration possible. Many manufacturers also offer development kits for selected products that facilitate the use of the FPGAs or SoCs through a variety of interfaces and provided supplements to the boards. These packages usually include software packages. An overview of a selection of manufacturers and the products offered is shown in Figure 5. The selected products are described in more detail below.

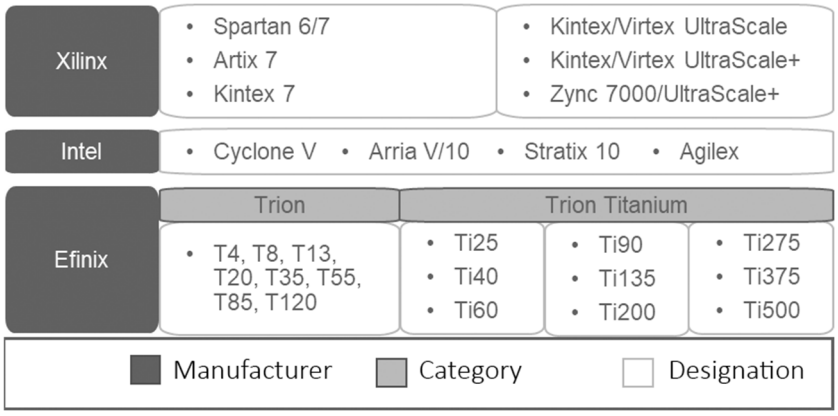

Figure 5. Overview of the state-of-the-art FPGA accelerators.

Xilinx's UltraScale architecture combines functionality with performance to implement applications that require I/O and memory bandwidth, large data flow, and DSPs. A portfolio is offered for various application purposes for both high required capacity, bandwidth, or performance as well as low space requirements and low cost. Spartan 6 FPGAs are suitable for I/O optimization, Spartan 7 FPGAs for I/O optimization with high performance per watt, Artix 7 FPGAs provide high DSP bandwidth, and Zynq 7000 is designed for system optimization with processor integration. An FPGA accelerator for BNNs to classify a Cifar10 dataset can achieve 2.4 TOP/s when using a Zynq 7Z045 with 11 W [6].

Using a Z-turn evaluation board with a Xilinx Zynq 7000 SoC and optimizing the computations to 8-bits, a frequency of 160 MHz and a power consumption of 1.77 W is achieved, resulting in a throughput of 40.96 GOP/s. This uses 134 calculation units and 601 kB of internal memory. The reduction from 32-bit floating-point number precision to 8-bits fixed-point number precision results in a 1% loss in accuracy for AlexNet while reducing power consumption and cost. In addition, the data flow is made more efficient by reducing the data transfer between off-chip and on-chip memory [20]. The Virtex UltraScale + VC 707 contains up to 500 MB of BRAMs arranged in small units. It enables high bandwidth with low power consumption compared to a platform with SRAM and off-chip memory [21]. Zebra was developed by Minsology for Xilinx FPGAs and is available as closed-source software. Zebra takes a Caffe application available for CPUs or GPUs and runs it on Xilinx FPGAs using the Zebra runtime. This can be used to implement different existing applications on different hardware platforms. Like Zebra, XfDNN can be customized to the number of cores [22]. Zebra achieves better performance and efficiency than Intel's OpenVINO tool.

Intel offers hardware solutions and development boards. Intel Agilex SoC FPGAs contain a quad-core ARM Cortex-A53 processor. The Intel Stratix series is equipped with 64-bit quad-core ARM Cortex-A53 processors. The Intel Arria series offers a compromise between cost and energy efficiency. The Intel Arria 10 SoC is the further development of the Arria V series and is equipped with a dual-core ARM Cortex-A9. When classifying an ImageNet dataset, a throughput of 1.9 TOP/s can be achieved when implementing a BNN

on a Stratix V GSD. Compared to the performance of an Arria 10 FPGA, a 14 nm ASIC, a Xeon server CPU, an Nvidia Titan X server GPU, and an Nvidia TXI mobile GPU, the FPGA exhibits better efficiency than the GPU and CPU and higher flexibility than the ASIC due to the operations of the BNN network used [9]. When implementing Ternary ResNet, Int6, and BNNs, the Stratix 10 series shows both better performance and power efficiency than the Titan X Pascal GPU [23].

Trion FPGAs from Efinix have between 4000 to 120,000 logical elements, memory blocks and DSP blocks. They can be applied for various purposes that require logical units or computation acceleration and deep learning. Trion Titanium FPGAs are used for various applications. The Ti25, Ti40, and Ti60 are used for mobile devices with low power requirements and a large number of I/Os. The Ti90, Ti135, and Ti200 are used for hardware acceleration, edge computing, and machine learning [24]. The Ti275, Ti375, and Ti500 are suitable for industrial automation or automotive applications. As the complexity of applications increases, the number of logical elements increases most noticeably from 24,000 in the Ti25 to 508,000 in the Ti500 and the number of DSP blocks from 67 in the Ti25 to 19,920 in the Ti500.

In [25], a training methodology termed the neural network design parameter extraction (NNDPE) program, facilitated the rapid implementation of artificial neural networks (ANNs) on FPGA platforms. This methodology enabled the efficient extraction of ANN design parameters, leading to the development of a high-precision ANN hardware architecture operating on the Virtex-7 FPGA at the impressive clock frequency of 150.76 MHz, significantly outperforming software implementations running on CPU cores.

In the Ref. [26] the transformative impact of deep learning on various applications and highlights the challenges in implementing neural network inference on resource-constrained hardware platforms is shown. It delves into the analysis of network architecture and FPGA characteristics, emphasizing five key acceleration strategies, including computing complexity, parallelism, data reuse, pruning, and quantization.

Table 2 below provides a comparison of the abovementioned FPGA accelerators focusing on their specific characteristics, processing power, and efficiency, especially in the context of optimizing CNNs. The comparison covers zynq-7000 (Xilinx Inc., San Jose, CA, USA), Arria 10 (Intel Corporation, Santa Clara, CA, USA) and Stratix 10 (Intel Corporation, Santa Clara, CA, USA), and Artix-7 (Xilinx Inc., San Jose, CA, USA), and highlights their individual capabilities and performance metrics that are critical to evaluating their suitability for advanced CNN-based computational tasks.

Table 2. Comparison of key FPGA accelerators for CNN implementations: processing power and efficiency analysis.

FPGA	Processing Power	Power Efficiency
Xilinx Zynq-7000	moderate	moderate
Intel Arria 10	high	moderate
Intel Stratix 10	very high	low
Xilinx Artix 7	low	high
Efinix Trion T120	moderate	moderate

4. CNN Optimizations for Implementation in FPGAs

In the previous section, the feasibility of accelerating CNNs on FPGAs was presented. In order to make the implementation efficient and successful, some optimization approaches are explained in this chapter. After the basic approaches are described and the goal of the optimization approaches is outlined, selected approaches are examined in more detail in Sections 4.1 and 4.2.

To maximize accuracy and throughput while minimizing energy consumption and costs, software and hardware must be considered as a unit to enable a holistic optimization approach. Acceleration methods can be divided into software and hardware improvement. Software optimization aims to reduce computation and bandwidth requirements

while maintaining accuracy. Algorithms, quantization, and weight reduction are used for this purpose. Algorithms are used to simplify or transform network models and the computational process. Data quantification involves weights and neurons to reduce bandwidth and memory requirements. Weight reduction involves the approximation of the weight matrix. In hardware improvement, the logical unit structure is adapted to the deep learning algorithms.

Optimization of the network can be achieved by reducing operands and operations. Operands can be reduced by using fixed-point numbers instead of floating-point numbers, nonlinear quantization, and weight distribution. The number of operations and model size can be achieved by techniques such as compression or pruning and use of compact network architectures.

FPGAs are capable of partial dynamic configuration, which makes FPGAs partially configurable. This can be applied to deep learning methods by configuring a part of the FPGA to run the next CNN layer while the rest is used to run the current CNN layer. For classical CNN networks such as AlexNet, FPGAs do not provide enough memory, resulting in weights being stored externally and having to be transformed for computation on the FPGA. This means that FPGAs should be improved, CNNs should be made less complex, and operations should be optimized [27].

Thanks to their parallelization capabilities and new development environments, neural networks can be enabled on FPGAs. One approach is the sequential implementation of convolutions and vector-matrix operations. Since a high number of computations affects latency due to the resulting complexity, parallelization is necessary. To increase throughput, the computations can be implemented in the form of a processing chain, which only slightly increases latency. DSPs used to execute MACs are limited to FPGAs. Even with sufficient hardware resources available, alternatives to classical deep neural network computations must be exploited. Workload analysis can be used to determine which computations can be structured in parallel [28].

Acceleration on FPGAs is achieved by algorithms, e.g., GEMM, Winograd or FFT, data path optimization, CNN network optimization, where sparse architecture is achieved by pruning, linear or binary quantization, and hardware generation, e.g., by HSL based on OpenCL or Vivado HLS based RTL.

4.1. Optimization Approaches Related to Approximations of the CNN Network

Neither the number of weights nor the number of operations in a CNN reflects energy consumption. Energy is consumed not only by computations, but also by memory access, e.g., fetching data from DRAM consumes more energy than the computation itself. Energy consumption is dominated by memory access for filter weights and feature maps [29]. When processing CNNs, memory bandwidth is often responsible for a bottleneck. For FC layers, execution may be memory bound due to the large number of weights. This results in a large number of memory reads. For convolutional layers, the large number of MAC operations results in many memory accesses. Each MAC operation requires at least two memory reads and one memory write. The three memory reads of MAC operations are for filter weight, FM activation, and partial sum. Reading the off-chip DRAM three times would be unfavorable for energy-efficient data flow. This compromises both throughput and energy efficiency since DRAM accesses require more energy than the computation itself. Accelerators with architectures for spatial parallelization reduce the energy required to move data. In the best case, MACs are performed without loading intermediate results.

Enhancing the efficiency and performance of CNN implementations on FPGAs demands careful consideration of various optimization strategies. Each approach comes with its own set of advantages and challenges, ranging from harnessing parallel processing capabilities and customizing hardware architectures to optimizing memory hierarchies and ensuring power efficiency. Balancing these factors is crucial for achieving optimal FPGA-based CNN implementations, as outlined in Table 3 below.

Table 3. Optimization approaches for CNN implementations on FPGAs: advantages and disadvantages.

Optimization Approach	Advantages	Disadvantages
Parallelism	Efficient utilization of concurrent processing	Complex design and synchronization challenges
Customization	Tailored hardware for specific architectures and data types	Requires specialized skills and extensive development time
Memory Hierarchy	Reduced latency and increased throughput	Challenging memory access patterns and design complexities
Quantization	Reduced memory and computation complexity	Potential accuracy degradation with aggressive quantization
Dataflow Optimization	Improved throughput and efficiency	Complexity for dynamic workloads and complex network structures
Resource Utilization	Efficient hardware usage and cost reduction	Constraints for accommodating changes and updates
Power Efficiency	Suitable for energy-constrained applications	Trade-offs with performance and design constraints
Flexibility and Reconfigurability	Adaptability to changing requirements and network variations	Overhead and complexity for reconfiguration

The goal of optimization approaches is to reduce computation and memory to increase throughput, latency, and energy efficiency. Approximation can reduce the required memory and computational complexity. Approximation algorithms can be divided into two types: quantization and weight reduction. Quantization methods reduce the precision of weights and/or activations (neuron output). Weight reduction removes redundant parameters through pruning or structural simplification, which simultaneously leads to a reduction in the number of activations in the network. Approximation increases throughput by increasing parallelization, reducing memory transfer, as well as workload. Quantization can be performed by using fixed-point numbers, binarization and ternarization, and logarithmic quantization. By reducing the precision in network compression, off-chip memory access occurs less frequently per operation. Data quantization is applied to 2-bit networks, TNNs, and 1-bit networks, BNNs [14]. Efficiency can be improved by compact data types. Extremely compact models such as BNNs that use a 1-bit data type offer advantages in memory size and bandwidth. In binary CNNs, FC layers can be replaced by a pooling layer.

The computation of 1-bit MACs can be performed by XNOR followed by counting bits. Multipliers can be replaced by XNOR circuits since there are only two values for the input and weights. Multiple XNOR circuits can replace one MAC more energy efficiently with higher speed [30]. BNNs are well suited for small data sets. Accuracy remains almost unchanged with this approach. For large datasets, the accuracy is lower. A new version of BNN is XNOR-Net, where the accuracy can be improved. The TNN network uses two bits to represent the weights, which can improve the accuracy and make the calculation more efficient. Among all the methods, the binary network has the highest loss of precision and in return has the highest compression ratio. Ternarization exhibits better precision with a good compression ratio because zeros can be represented in this approach.

4.2. Memory Optimization Approaches in the Context of Implementation on FPGAs

CNNs can be developed on FPGAs using either high-level synthesis (HLS) or hardware description language (HDL). By using HLS, the software code can be put on the FPGA in a short time. Architectures based on HDL show better performance considering frequency, throughput, latency, power consumption, and resource utilization. Implementation using HDL requires advanced skills of the developer and takes more time [31]. HLS reduces the difficulty and required lead time of the design while ensuring high throughput and energy efficiency. Convolutions require many multiplications implemented on an FPGA with digital signal processing (DSP) blocks, where the synthesis tool can be forced to implement multiplications with logical elements instead of DSP blocks. This makes the implementation consist of AND gates and trees-of-half adders. Multiplications with zero are removed.

Latency is created by accessing the off-chip memory. A balance between on-chip and off-chip memory and on-chip memory and the number of PEs is sought to effectively utilize chip resources, alleviate DRAM bottlenecks, and maximize performance [32]. Due to limited BRAM resources, not all ifmaps and ofmaps can be stored on BRAM. Different

BRAM channels can be set up for the ifmaps of each layer. To save weights on on-chip memory, almost all BRAM resources are used. For layers with few parameters, all weights can be read in one loop. Otherwise, more loops are needed to fully read the weights [33].

Deep neural networks are not processed randomly. The given data flow can be adapted to the network and increase energy efficiency. The reuse of data should be maximized because the storage capacity is limited for small memories, but large memories consume more energy. Convolutions, FMs, and filters can be reused. When reusing convolutions, the same ifmap activations and filter weights are used within a channel in convolutional layers and in different combinations for different weighted sums. When reusing FMs, different filters can be applied to the same FM and ifmap activations are reused for convolution and FC layers. When reusing filters, different ifmaps are processed at once and filter weights are reused in convolution and FC layers if the batch is greater than 1. Data is stored in the local data hierarchy to reduce access to DRAM. Local memory can be used for subtotals. There are several approaches to handling data flow: stationary weights, stationary output, no local reuse, and a stationary row. Stationary weights minimize power consumption when reading the weights by maximizing the access of the weights from the RF to the PE. Each weight is read from DRAM into the RF from each PE and remains stationary for further access. Processing passes through as many MACs as possible that use the same weight as long as it is present in the RF. This maximizes the reuse of convolutions and filter weights. Ifmap activations are transmitted to all PEs and subtotals are accumulated. Stationary output minimizes energy consumption by reading and writing subtotals and accumulating subtotals for the same output activation value in the RF. With no local reuse, small RFs contribute to good energy efficiency. However, this approach is inefficient for area reduction and maximizing memory capacity and minimizing off-chip bandwidth.

No local memory is allocated to a PE. Instead, allocation to the global buffer takes place to increase capacity. For stationary rows, a 1D row convolution is processed in each PE. The row of filter weights is held stationary within the RF from the PE, allowing input activations to flow into the PE. As a result, only one memory location is used for the accumulation of the subtotals. The input activations can be reused. By processing a 1D convolution of a PE, multiple PEs can be combined into a complete 2D convolution. Other forms of reuse are enabled to reduce access to the global buffer.

The FPGA architecture is designed with a tile look-up table (TLUT) and a channel multiplexer (CMUX). The TLUT is designed to match the compressed weights with the input pixels. The CMUX is used for address localization and efficient on-chip memory access without conflicts [34]. Since convolutions affect performance, a combination of four optimization strategies can be used, including: intra-layer optimization, inter-channel parallelism, memory structure optimization, and designs in the form of a processing chain such as ping-pong buffer memories and FIFOs [35].

5. Proposed Energy Efficient Embedded System for Organic Weeding Robots and Discussion

This section presents hardware solutions for FPGA acceleration as well as software solutions for the implementation of energy efficient artificial neural networks for organic weeding robots. The current software and hardware configurations of the developed weeding robot used by our cooperation partner Westhof Bio GmbH in Germany are presented, followed by an examination of the possible implementation of the networks on FPGAs. Special attention is given to the YOLO network, since this network architecture is currently used for the detection of weeds [34]. This section closes with a discussion.

5.1. Current Soft- and Hardware Configuration of the Developed Weeding Robot

In [1], a method for detecting multiple weed species in organic farming using a modified YOLO approach was presented. The proposed method was able to detect the weeds in real time at up to 56 FPS, which is an important feature for the development of modern smart farming applications. A lower precision was accepted in favor of a higher

calculation rate of about 56 FPS. The best average precision of 75.07% was achieved at 18.65 FPS and an input size of 832 × 832 pixels. The detection speed can be adjusted to the application's accuracy requirement to maximize the detection speed. The proposed method shows that it is flexible and robust. Only a small dataset consisting of 50 images is necessary to achieve acceptable results.

In [36] we already evaluated hardware accelerators for AI applications in organic farming. In this approach, a Coral USB Accelerator (Google LLC, Googleplex, CA, USA) with an Edge-TPU and a Raspberry Pi 4 (Raspberry Pi Ltd, Cambridge, UK) Trading Model B was used. The MobileNetV2-SSD was chosen for this application because of its ability to run on embedded systems. The values achieved by this approach in terms of energy consumption, accuracy, and speed were used for comparison with [34] to improve the current weed control system. In terms of speed, the tiny-YOLO accelerated on a Jetson TX2 (Nvidia Corporation, Santa Clara, CA, USA) was surpassed. However, the accuracy of the tiny-YOLO could not be achieved. As a lesson learned, it should be noted that an improvement in training must be made. During the first training attempts, the reduction of the learning rate was set to a step size that was too large, so that the learning rate was too high in the later course of training. Due to the excessively high learning rate, the training results became increasingly worse. The energy consumption of the Raspberry Pi 4 with the Coral USB-Accelerator is significantly lower than that of the Jetson TX2. The acquisition costs of the Raspberry Pi and the Coral USB accelerator are also lower than those of the Jetson TX2.

5.2. FPGA Implementation of the YOLO Network

YOLO implements classification and regression in object detection so that the object is assigned to a class and information about the object's coordinates is revealed [37]. YOLO has a similar structure to GoogLeNet [38]. The you-only-look-once (YOLO) network is faster and performs better than comparable networks such as single shot multibox detection (SSD) and Faster R-CNN. Other similar networks include SqueezeNet, MobileNet, and ShiftNet [39].

In the following, we particularly consider YOLOv2. YOLOv2 shows better precision than simplified YOLO networks such as Tiny Yolov2. YOLOv2 is an object detection model published in 2017 by Joseph Redmon and Ali Farhadi. This CNN model is suitable for the simultaneous prediction of bounding box localization and categorization [40]. YOLOv2 is improved by Darknet-19. It leads to 19 convolutional layers, one average pooling layer, five maximum pooling layers, one softmax layer, and 3 × 3 and 1 × 1 convolutional filters. YOLOv3 has a more complex network structure and is slower than YOLOv2.

To implement YOLOv2 on FPGAs, the first step is to apply various compression techniques to the network. BNN and XNOR networks replace classical floating point operations with binary operations, as explained earlier, which leads to a significant reduction in memory bandwidth and memory requirements. The DSP blocks are not fully utilized. REQ -YOLO can be used to implement quantization. REQ-YOLO compresses the YOLO network with a small loss of accuracy. An alternating direction method of multipliers (ADMM) is used [41]. Binary operations can be implemented using look-up tables.

A streaming design can be used to implement YOLO. Re-training and quantization are applied to the parameters of the YOLO network. Binary weights are used, allowing the entire network to be stored on the FPGA's BRAMs to reduce the necessary off-chip access and improve performance. Convolutional layers are built as a processing chain. The input image is delivered row by row to the accelerators. The output from the previous layer is delivered row by row. Intermediate data is reused at the layers to reduce access to external memory. Reduced access to DRAM reduces power consumption. Each convolutional layer is allocated to a hardware block. Quantization enables a fast and energy-efficient CNN accelerator. The entire quantized CNN model can be accommodated on on-chip BRAMs [21]. Another approach is to implement 3D convolutions as GEMM (general matrix multiplication) [42]. In [43] the authors introduce an FPGA-based deep learning

acceleration core architecture, specifically targeting image detection, with a focus on the YOLO model. The architecture introduces a streamlined parallel acceleration scheme that addresses arithmetic power, speed, and resource consistency issues. Leveraging a three-level data cache architecture and optimized bus accessing strategies, the design achieves significant performance enhancements. Notably, it achieves a 14 FPS inference for the Tiny Yolo model, utilizing less than 25% of the FPGA resource. The study also emphasizes the need for further research on acceleration algorithms for other models, such as RNN-based or GAN-based architectures.

A configurable CNN accelerator is based on an architecture consisting of an ARM and an FPGA, allowing a processing chain to be realized using HLS. The input buffer memory and the weight buffer memory are used for computation. The output buffer memory for storing intermediate results is designed as a ping-pong model. A dynamic fixed-point number quantization strategy is used to improve the efficiency of convolutions and reduce the access time to off-chip memory. Instead of using floating-point numbers for weights, 16-bit fixed-point numbers are used in this approach. Another method is to use a Winograd algorithm.

The Winograd transform is a fast convolution algorithm that can simplify MAC operations by replacing multiplications with additions. Performance can be improved by reducing multiplications and lower loop iterations. A Winograd algorithm is applied to efficiently implement convolution to solve the unaligned global memory access problem with an alignment stream buffer memory (AS buffer memory). The available memory access bandwidth is fully utilized as well as all available DSP resources to achieve the highest possible parallelization. This approach results in 10 ms per frame. Latency is reduced compared to using GPUs. The Winograd algorithm reduces the computation for convolutional layers compared to using Verilog/VHDL. All layers are supported by an iterative Conv-ReLU pooling processing chain. The AS buffer is intended to cache properties or features, and the weight buffer caches weights and biases to increase throughput. FMs and filters as well as ofmaps are stored in global memory. Feature data is read from DRAM to the on-chip AS buffer memory cache. Once the execution of the convolution operation starts, a set of filters is read from DRAM to the input feature column buffer memory. The convolution, ReLU, and pooling modules operate in the form of a processing chain. The ofmap is written back to the DRAM. Each convolution layer is followed by a batch normalization (BN) layer and a scale layer. Logic, DSP resources, and latency are reduced by unifying the three layers.

According to the research of [44], they presented an FPGA-based implementation of YOLOv2 for object detection tasks on resource-constrained computer vision-based IoT edge devices and remote control vehicles equipped with cameras. Their approach focuses on resolving issues related to data reloading and off-chip memory access, employing an efficient dataflow strategy and multi-level buffers that maximize on-chip data transfer and minimize external memory access. The authors were able to achieve remarkable results with low memory resource utilization and low-power consumption, showcasing the potential for implementing complex models on edge devices.

Various techniques are used to compress YOLOv2, such as 8-bit quantization, structural pruning, and joining the convolutional, BN, and scale layers. The finetuning of the parameters is enabled by re-training, which prevents a drop in accuracy. Adaptive logic modules (ALM s) are used to implement convolutional operations. Fixed-point numbers reduce computational complexity and resource consumption. To optimize memory, a buffer pipeline method can be used to increase efficiency. By combining both methods, the FPGA's resource consumption is optimized, and the power consumption is reduced up to 2.7W. The accuracy loss is 3% at most. Weights with 16-bit fixed-point numbers are used. Weights with 32-bits are used for training. To solve the memory optimization problem of the FPGA accelerator, a buffer pipeline method can be applied. When using an SoC, data in the logical part interacts with the CPU via an external memory. This is controlled by an advanced extensible interface (AXI) bus. When data is exchanged with the accelerator, to

balance the timing coordination requirements with the data flow, a first-in-first-out (FIFO) interface is added to the AXI bus. This allows input and output data from the accelerator to be transferred efficiently. The average accuracy is 78.25%. HLS can be inefficient in terms of hardware resources and performance. PEs are crucial for the implementation of convolutional operations. To reduce hardware resources and power consumption, a new PE was designed. For this purpose, modified-booth-encoding (MBE)-multipliers and Wallace tree adders can be used to replace MACs and typical adder trees [45].

BNN Pynq is a project from Xilinx that runs a BNN network on a Pynq board. The Pynq board is a low-end Zynq board equipped with an ARM CPU and a FPGA [46]. Binarized modified convolutions for object detection based on Darknet and modified YOLO can be implemented on a Pynq Z1 board [47]. This achieves an accuracy of 72.3% and an efficiency of 17.54 FPS. The implementation of a YOLOv2 network modified with Winograd on a Pync board is supported, e.g., by the Vivado HLS tool (Advanced Micro Devices Inc., Santa Clara, CA, USA). On a Pync Z2 board, this can achieve an accuracy of 78.25% for input images of size 416 × 416 with a power consumption of 2.7 W and a frequency of 125 MHz. The Pync Z2 is equipped with 153 DSPs.

A YOLOv2 network modified using the Winograd algorithm, pruning, and quantization can be implemented on an Arria 10 FPGA with 2 GB DDR4 DRAM at a frequency of 212 MHz. Implementing a YOLO network optimized with Winograd on a Virtex UltraScale+ VC707 (Xilinx Inc., San Jose, CA, USA), a throughput of 1.877 TOP/s is achieved at a frequency of 200 MHz and an on-chip energy consumption of 18.29 W. The average mean accuracy is 64.16%. By combining the convolutional and pooling operations of the YOLOv2 network, the time required to access off-chip memory can be reduced.

Using Artix 7 TSBG484 and 16-bit fixed-point number precision, the throughput is 22 GOP/s at a frequency of 100 MHz. The power consumption is 7.53 W. When using a Virtex 7 VX485T and a 32-bit floating-point number precision, the throughput is 61.62 GOP/s at a frequency of 100 MHz and the power consumption is 3W. Using a Stratix V, a throughput of 136.5 GOPs is achieved at a frequency of 120 MHz and a power consumption of 19.1W. With the ZCU 102 Zynq UltraScale+, a peak throughput of 289 GOP/s and an average throughput of 102 GOP/s are achieved at a 16-bit fixed-point number precision and a frequency of 300 MHz. At the same time, the power consumption is 11.8 W. YOLOv2 achieves 35.71 FPS (frames per second) on the Xilinx ZCU 102 board with a Zynq UltraScale+ MPSoC when all weights are stored on off-chip DDR memory and a buffer is used for binary weights [30]. Using Wallace adder trees to optimize Tiny YOLOv2, a Zynq 706 achieves a throughput of 87.03 GOP/s and an efficiency of 61.64 GOP/s/W [48]. To summarize Table 4 compares various FPGA-based implementations of the YOLO network. These implementations employ diverse strategies, including compression techniques, configurable CNN accelerators, Winograd algorithms, and quantization methods, to optimize performance and resource utilization. The comparison sheds light on the nuanced trade-offs between processing speed, energy consumption, and detection accuracy.

Table 4. Comparative Analysis of FPGA Implementations for YOLOv2 Network, Including Time Consumption, Energy Efficiency and Accuracy.

FPGA Implementation	Inference Time (ms)	Energy Efficiency (W)	Accuracy (%)
YOLOv2 with Configurable CNN Accelerator and Winograd Algorithm	10	2.7	78.25
YOLOv2 with BNN Pynq Implementation	15	5.1	72.3
YOLOv2 on Artix 7 FPGA	18	7.53	68.9
YOLOv2 on Stratix V FPGA	20	19.1	-
Tiny YOLOv2 with Wallace Adder Trees on Zynq 706 FPGA	16	6.2	-

5.3. Discussion

Figure 6 shows a comparison of different selected FPGA accelerators for the YOLO network, whose testing has already been described in the literature. Different components

are compared with each other. It can be seen that the costs increase as the complexity of the SoCs increases. In order to enable a particularly efficient implementation in terms of both technical implementation and monetary aspects, an adaptation of the CNN network should be aimed for. Implementations of the YOLO network using an FPGA accelerator from the literature achieve good results by taking this approach. A low-cost implementation is possible by using the Pynq Z1. Testing is outlined in various sources.

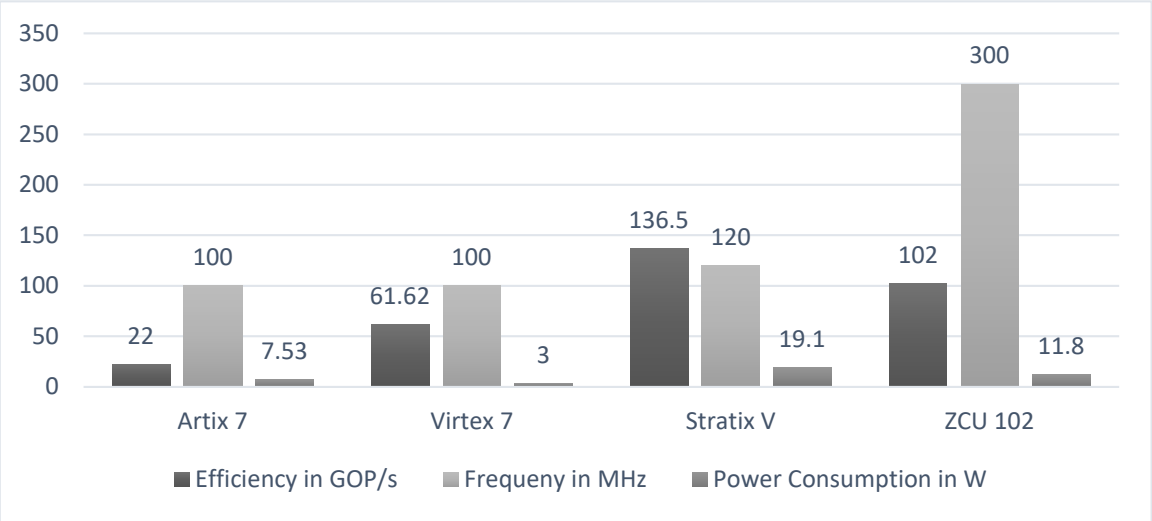

Figure 6. FPGAs for YOLO acceleration in comparison.

Different approaches to implementing YOLOv2 on a Pynq board achieve about 70-80% accuracy, with modified networks able to achieve 15–16 FPS efficiency, while a direct YOLOv2 implementation achieves 0.16 FPS efficiency. Particularly good results are obtained with the ZCU 102. It can achieve the same 85.2% accuracy as a 256-core Pascal GPU when implementing a modified YOLO with binary weights. In addition, the FPS of 35.71 for the ZCU 102 board is higher than the GPU efficiency of 1.39 FPS. When the efficiency in FPS/W is compared between ZCU 102 and the Pynq board when the modified YOLOv2 network is applied, the Pynq board can achieve a higher efficiency of 10.44–12.1 FPS/W compared to 7.93 FPS/W of the ZCU 102 due to the lower power consumption.

Concluding our review of the energy efficient embedded system acceleration of CNNs, we can summarize that compared to the current solution used for our developed weeding robot, neither the Pynq nor the ZCU 102 represent a degradation of the system in terms of performance or efficiency. Both devices can improve the system due to the nature of the hardware accelerators and when YOLO is appropriately modified. The ZCU 102 offers the possibility of additional extensions due to its large memory capacity.

Conceivable extensions are additional classes, e.g., beet or spinach. Since FPGAs reduce the space requirement compared to a GPU, a use of further applications of artificial neural networks in the field of autonomous smart or precision farming is conceivable, e.g., by using drones [49–53]. The energy efficiency offered by FPGAs can additionally optimize the weeding robot. FPGAs lend themselves to battery operation or operation by means of solar modules.

6. Conclusions

The cultivation of organic vegetables has long been challenged by the persistent issue of weed control, exacerbated by the continuous increase in chemical and synthetic pesticide use. To address this problem, the development of an automated weeding robot has been proposed, although its energy demands, particularly during the GPU-driven inference phase, present significant obstacles to achieving a sustainable operation. The comprehen-

sive evaluation of various FPGA implementations for the YOLO network underscores the versatility and efficiency of FPGA-based solutions in enhancing the performance of embedded systems, particularly in the domain of organic weeding robots. The comparison highlights the nuanced trade-offs between processing speed, energy consumption, and detection accuracy. Different FPGA architectures, including Artix 7, Stratix V, and ZCU 102, have demonstrated their capabilities in achieving varying levels of performance metrics, with each approach exhibiting unique strengths in specific contexts. Notably, the examination of the Pynq board and the ZCU 102 reveals that both systems offer viable options for enhancing the weeding robot's functionality, without compromising its efficiency or performance. Moreover, the energy efficiency provided by FPGA architectures presents an opportunity for optimizing the weeding robot, making it conducive for potential applications in autonomous and precision farming, including the use of drones. With the compact form factor and enhanced energy efficiency, FPGA-based solutions enable the possibility of extended operation through battery or solar-powered configurations, thereby showcasing their potential as a promising avenue for further advancements in the domain of smart farming and agricultural automation.

Author Contributions: Conceptualization, V.C., C.K., L.O.H. and S.H.; methodology, V.C. and C.K.; software, V.C. and L.O.H.; validation, V.C. and C.K.; formal analysis, V.C. and C.K.; investigation, V.C. and C.K.; resources, S.H.; data curation, V.C. and C.K.; writing—original draft preparation, V.C., C.K. and S.H.; writing—review and editing, V.C., C.K., L.O.H. and S.H.; visualization, V.C. and C.K.; supervision, S.H.; project administration, S.H.; funding acquisition, S.H. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the European Innovation Partnership (EIP) and the Federal State of Schleswig-Holstein, measure 16.1 "EIP agricultural production and sustainability in agriculture" pursuant to article 55 of regulation No. 1305/2013.

Institutional Review Board Statement: Not applicable.

Data Availability Statement: The data are not publicly available due to our laboratory's privacy and data protection policy.

Acknowledgments: We would like to thank our cooperation partner Westhof Bio GmbH for their many years of support. We also acknowledge the financial support provided by the Federal State of Schleswig-Holstein within the funding program *Open Access Publication Funds*.

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

References

1. Czymmek, V.; Harders, L.O.; Knoll, F.J.; Hussmann, S. Vision-Based Deep Learning Approach for Real-Time Detection of Weeds in Organic Farming. In Proceedings of the 2019 IEEE International Instrumentation and Measurement Technology Conference (I2MTC), Auckland, New Zealand, 20–23 May 2019; pp. 585–589. [CrossRef]
2. Miron, R. Maschinell lernende, neuronale Netzwerke als Intelligenzgeber. Special Feature Digi-Key. 2016. pp. 54–56. Available online: https://blog.iao.fraunhofer.de/spielarten-der-kuenstlichen-intelligenz-maschinelles-lernen-und-kuenstliche-neuronale-netze (accessed on 4 October 2023).
3. Sze, V.; Chen, Y.-H.; Yang, T.-J.; Emer, J.S. Efficient Processing of Deep Neural Networks: A Tutorial and Survey. *Proc. IEEE* **2017**, *105*, 2295–2329. [CrossRef]
4. Brübach, A. Complement and alternative to established machine vision—Deep Learning at Vision. *IEEE Ind. Eng. Effic.* **2018**, *9*, 28–30.
5. Wang, E.; Davis, J.J.; Zhao, R.; Ng, H.; Niu, H.; Cheung, P.; Constantinides, G.A. Deep Neural Network Approximation for Custom Hardware: Where We´ve Been, Where We´re Going. *ACM Comput. Surv.* **2019**, *52*, 1–39. [CrossRef]
6. Abdelouahab, K.; Pelcat, M.; Sérot, J.; Berry, F. Accelerating CNN inference on FPGAs: A Survey. *arXiv* **2018**, arXiv:1806.01683. [CrossRef]
7. 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]
8. Phu, H.V.; Tan, T.M.; Van Men, P.; Van Hieu, N.; Van Cuong, T. Design and Implementation of Configurable Convolutional Neural Network on FPGA. In Proceedings of the IEEE 6th NAFOSTED Conference on Information and Computer Science (NICS), Hanoi, Vietnam, 12–13 December 2019; pp. 298–302. [CrossRef]

9. Nurvitadhi, E.; Sheffield, D.; Sim, J.; Mishra, A.; Venkatesh, G.; Marr, D. Accelerating Binarized Neural Networks: Comparison of FPGA, CPU, GPU, and ASIC. In Proceedings of the 2016 International Conference on Field-Programmable Technology (FPT), Xi'an, China, 7–9 December 2016; pp. 77–84. [CrossRef]
10. 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; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), Zhangjiajie, China, 10–12 August 2019; pp. 1674–1681. [CrossRef]
11. Hareth, S.; Mostafa, H.; Shehata, K.A. Low power CNN hardware FPGA implementation. In Proceedings of the 2019 31st International Conference on Microelectronics (ICM), Cairo, Egypt, 15–18 December 2019; pp. 162–165. [CrossRef]
12. He, D.; Wang, Z.; Liu, J. A Survey to Predict the Trend of AI-able Server Evolution in the Cloud. *IEEE Access* **2018**, *6*, 10591–10602. [CrossRef]
13. Wei, G.; Hou, Y.; Cui, Q.; Deng, G.; Tao, X.; Yao, Y. YOLO Acceleration using FPGA Architecture. In Proceedings of the 2018 IEEE/CIC International Conference on Communications in China (ICCC), Beijing, China, 16–18 August 2018; pp. 734–735. [CrossRef]
14. Shahshahani, M.; Goswami, P.; Bhatia, D. Memory Optimization Techniques for FPGA based CNN Implementations. In Proceedings of the 2018 IEEE 13th Dallas Circuits and Systems Conference (DCAS), Dallas, TX, USA, 12 November 2018; pp. 1–6. [CrossRef]
15. Nurvitadhi, E.; Venkatesh, G.; Sim, J.; Marr, D.; Huang, R.; Ong Gee Hock, J.; Liew, Y.T.; Srivatsan, K.; Moss, D.; Subhaschandra, S.; et al. Can FPGAs Beat GPUs in Accelerating Next-Generation Deep Neural Networks? In Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterrey, CA, USA, 22 February 2017; pp. 5–14. [CrossRef]
16. Müller, P. KI offline und am Edge—Künstliche Intelligenz mit FPGAs: So gelingt der Einstieg. *Elektron. Ind.* **2020**, *12*, 18–19.
17. Fowers, J.; Brown, G.; Wernsing, J.; Stitt, G. A performance and energy comparison of convolution on GPUs, FPGAs, and multicore processors. *ACM Trans. Archit. Code Optim.* **2013**, *9*, 25. [CrossRef]
18. Wu, D.; Zhang, Y.; Jia, X.; Tian, L.; Li, T.; Sui, L.; Xie, D.; Shan, Y. A High-Performance CNN Processor Based on FPGA for MobileNets. In Proceedings of the 2019 29th International Conference on Field Programmable Logic and Applications (FPL), Barcelona, Spain, 8–12 September 2019; pp. 136–143. [CrossRef]
19. Zhu, J.; Wang, L.; Liu, H.; Tian, S.; Deng, Q.; Li, J. An Efficient Task Assignment Framework to Accelerate DPU-Based Convolutional Neural Network Inference on FPGAs. *IEEE Access* **2020**, *8*, 83224–83237. [CrossRef]
20. Khabbazan, B.; Mirzakuchaki, S. Design and Implementation of a Low-Power, Embedded CNN Accelerator on a Low-end FPGA. In Proceedings of the 2019 22nd Euromicro Conference on Digital System Design (DSD), Kallithea, Greece, 28–30 August 2019; pp. 647–650. [CrossRef]
21. Nguyen, D.T.; Nguyen, T.N.; Kim, H.; Lee, H.-J. A High-Throughput and Power-Efficient FPGA Implementation of YOLO CNN for Object Detection. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2019**, *27*, 1861–1873. [CrossRef]
22. Kljucaric, L.; George, A.D. Deep-Learning Inferencing with High-Performance Hardware Accelerators. In Proceedings of the 2019 IEEE High Performance Extreme Computing Conference (HPEC), Waltham, MA, USA, 24–26 September 2019; pp. 1–7. [CrossRef]
23. Nurvitadhi, E.; Sim, J.; Sheffield, D.; Mishra, A.; Krishnan, S.; Marr, D. Accelerating recurrent neural networks in analytics servers: Comparison of FPGA, CPU, GPU, and ASIC. In Proceedings of the 2016 26th International Conference on Field Programmable Logic and Applications (FPL), Lausanne, Switzerland, 29 August 2016–2 September 2016; pp. 1–4. [CrossRef]
24. Werner, H. Intelligentere Edge-Endgeräte durch KI-Einsatz in FPGAs. *Elektronikpraxis* **2019**, *7*, 30–32.
25. Vineetha, K.V.; Mohit, M.; Reddy, S.K.; Ramesh, C.; Kurup, G.D. An efficient design methodology to speed up the FPGA implementation of artificial neural networks. *Eng. Sci. Technol. Int. J.* **2023**, *47*, 101542. [CrossRef]
26. Wu, R.; Guo, X.; Du, J.; Li, J. Accelerating Neural Network Inference on FPGA-Based Platforms—A Survey. *Electronics* **2021**, *10*, 1025. [CrossRef]
27. Gupta, N. Tiefe neuronale Netze auf FPGAs. *Markt&Technik* **2015**, *4*, 32–34.
28. Alawad, M.; Lin, M. Scalable FPGA Accelerator for Deep Convolutional Neural Networks with Stochastic Streaming. *IEEE Trans. Multi-Scale Comput. Syst.* **2018**, *4*, 888–899. [CrossRef]
29. Yang, T.-J.; Chen, Y.-H.; Sze, V. Designing Energy-Efficient Convolutional Neural Networks Using Energy-Aware Pruning. In Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21–26 July 2017; pp. 6071–6079. [CrossRef]
30. Nakahara, H.; Fujii, T.; Sato, S. A fully connected layer elimination for a binarized convolutional neural network on an FPGA. In Proceedings of the 2017 27th International Conference on Field Programmable Logic and Applications (FPL), Ghent, Belgium, 4–8 September 2017; pp. 1–4. [CrossRef]
31. Kyriakos, A.; Kitsakis, V.; Louropoulos, A.; Papatheofanous, E.-A.; Patronas, I.; Reisis, D. High Performance Accelerator for CNN Applications. In Proceedings of the 2019 29th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS), Rhodes, Greece, 1–3 July 2019; pp. 135–140. [CrossRef]
32. Chiu, G.R.; Ling, A.C.; Capalija, D.; Bitar, A.; Abdelfattah, M.S. Flexibility: FPGAs and CAD in Deep Learning Acceleration. In Proceedings of the ISPD 2018 International Symposium on Physical Design, Monterey, CA, USA, 25–28 March 2018; pp. 34–41. [CrossRef]

33. Chang, X.; Pan, H.; Zhang, D.; Sun, Q.; Lin, W. A Memory-Optimized and Energy-Efficient CNN Acceleration Architecture Based on FPGA. In Proceedings of the 2019 IEEE 28th International Symposium on Industrial Electronics (ISIE), Vancouver, BC, Canada, 12–14 June 2019; pp. 2137–2141. [CrossRef]
34. Lu, L.; Xie, J.; Huang, R.; Zhang, J.; Lin, W.; Liang, Y. An Efficient Hardware Accelerator for Sparse Convolutional Neural Networks on FPGAs. In Proceedings of the 2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), San Diego, CA, USA, 28 April–1 May 2019; pp. 17–25. [CrossRef]
35. Huang, C.; Ni, S.; Chen, G. A layer-based structured design of CNN on FPGA. In Proceedings of the 2017 IEEE 12th International Conference on ASIC (ASICON), Guiyang, China, 25–28 October 2017; pp. 1037–1040. [CrossRef]
36. Czymmek, V.; Möller, C.; Harders, L.O.; Hussmann, S. Deep Learning Approach for high Energy efficient Real-Time Detection of Weeds in Organic Farming. In Proceedings of the 2021 IEEE International Instrumentation and Measurement Technology Conference (I2MTC), Glasgow, UK, 17–20 May 2021; pp. 1–6. [CrossRef]
37. 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]
38. Bao, C.; Xie, T.; Feng, W.; Chang, L.; Yu, C. A Power-Efficient Optimizing Framework FPGA Accelerator Based on Winograd for YOLO. *IEEE Access* **2020**, *8*, 94307–94317. [CrossRef]
39. Zhang, S.; Cao, J.; Zhang, Q.; Zhang, Q.; Zhang, Y.; Wang, Y. An FPGA-Based Reconfigurable CNN Accelerator for YOLO. In Proceedings of the 2020 IEEE 3rd International Conference on Electronics Technology (ICET), Chengdu, China, 8–12 May 2020; pp. 74–78. [CrossRef]
40. Yang, A.; Li, Y.; Shu, H.; Deng, J.; Ma, C.; Li, Z.; Wang, Q. An OpenCL-Based FPGA Accelerator for Compressed YOLOv2. In Proceedings of the 2019 International Conference on Field-Programmable Technology (ICFPT), Tianjin, China, 9–13 December 2019; pp. 235–238. [CrossRef]
41. Ding, C.; Wang, S.; Liu, N.; Xu, K.; Wang, Y.; Liang, Y. REQ-YOLO: A Resource-Aware, Efficient Quantization Framework for Object Detection on FPGAs. In Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Seaside, CA, USA, 24–26 February 2019; pp. 33–42. [CrossRef]
42. Wai, Y.J.; Yussof, Z.B.M.; Salim, S.I.B.; Chuan, L.K. Fixed Point Implementation of Tiny-Yolo-v2 using OpenCL on FPGA. *IJACSA Int. J. Adv. Comput. Sci. Appl.* **2018**, *9*, 506–512. [CrossRef]
43. Yang, X.; Zhuang, C.; Feng, W.; Yang, Z.; Wang, Q. FPGA Implementation of a Deep Learning Acceleration Core Architecture for Image Target Detection. *Appl. Sci.* **2023**, *13*, 4144. [CrossRef]
44. Zhang, Z.; Mahmud, M.A.P. Resource-constrained FPGA implementation of YOLOv2. *Neural Comput. Appl.* **2022**, *34*, 16989–17006. [CrossRef]
45. Farrukh, F.U.D.; Xie, T.; Zhang, C.; Wang, Z. Optimization for Efficient Hardware Implementation of CNN on FPGA. In Proceedings of the 2018 IEEE International Conference on Integrated Circuits, Technologies and Applications (ICTA), Beijing, China, 21–23 November 2018; pp. 88–89. [CrossRef]
46. Yoshimoto, Y.; Shuto, D.; Tamukoh, H. FPGA-enabled Binarized Convolutional Neural Networks toward Real-time Embedded Object Recognition System for Service Robots. In Proceedings of the 2019 IEEE International Circuits and Systems Symposium (ICSyS), Kuantan, Malaysia, 18–19 September 2019; pp. 1–5. [CrossRef]
47. Kim, H.; Choi, K. Low Power FPGA-SoC Design Techniques for CNN-based Object Detection Accelerator. In Proceedings of the 2019 IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON), New York, NY, USA, 10–12 October 2019; pp. 1130–1134. [CrossRef]
48. Farrukh, F.U.D.; Zhang, C.; Jiang, Y.; Zhang, Z.; Wang, Z.; Wang, Z.; Jiang, H. Power Efficient Tiny Yolo CNN Using Reduced Hardware Resources Based on Booth Multiplier and WALLACE Tree Adders. *IEEE Open J. Circuits Syst.* **2020**, *1*, 76–87. [CrossRef]
49. Czymmek, V.; Schramm, R.; Hussmann, S. Vision Based Crop Row Detection for Low Cost UAV Imagery in Organic Agriculture. In Proceedings of the 2020 IEEE International Instrumentation and Measurement Technology Conference (I2MTC), Dubrovnik, Croatia, 25–28 May 2020; pp. 1–6. [CrossRef]
50. Harders, L.O.; Czymmek, V.; Wrede, A.; Ufer, T.; Hussmann, S. Deep learning approach for UAV-based weed detection in horticulture using edge processing. *Appl. Mach. Learn.* **2022**, *12227*, 122270R. [CrossRef]
51. Harders, L.O.; Czymmek, V.; Wrede, A.; Ufer, T.; Hussmann, S. UAV-based real-time weed detection in horticulture using edge processing. *SPIE J. Electron. Imaging* **2023**, *32*, 052405. [CrossRef]
52. Hussmann, S.; Clausen, K.; Harders, L.O. Vision-based crop row detection system for UAV-based weed detection in arboriculture. In Proceedings of the Optical Technology and Measurement for Industrial Applications Conference, Yokohama, Japan, 17–21 April 2023; p. 1260707. [CrossRef]
53. Czymmek, V.; Moeller, C.; Schacht, E.; Harders, L.O.; Hussmann, S. Autonomous fawn tracking system based on drone images and CNNs. In Proceedings of the Optical Technology and Measurement for Industrial Applications Conference, Yokohama, Japan, 17–21 April 2023. [CrossRef]