electronics

MDPI

Article

FPGA-Based CNN for Eye Detection in an Iris Recognition at a Distance System

Camilo A. Ruiz-Beltrán, Adrián Romero-Garcés, Martín González-García, Rebeca Marfil and Antonio Bandera *

Department of Electronic Technology, University of Málaga, 29071 Málaga, Spain; camilo@uma.es (C.A.R.-B.); argarces@uma.es (A.R.-G.); martin@uma.es (M.G.-G.); rebeca@uma.es (R.M.)
* Correspondence: ajbandera@uma.es

Abstract: Neural networks are the state-of-the-art solution to image-processing tasks. Some of these neural networks are relatively simple, but the popular convolutional neural networks (CNNs) can consist of hundreds of layers. Unfortunately, the excellent recognition accuracy of CNNs comes at the cost of very high computational complexity, and one of the current challenges is managing the power, delay and physical size limitations of hardware solutions dedicated to accelerating their inference process. In this paper, we describe the embedding of an eye detection system on a Zynq XCZU4EV UltraScale+ multiprocessor system-on-chip (MPSoC). This eye detector is used in the application framework of a remote iris recognition system, which requires high resolution images captured at high speed as input. Given the high rate of eye regions detected per second, it is also important that the detector only provides as output images eyes that are in focus, discarding all those seriously affected by defocus blur. In this proposal, the network will be trained only with correctly focused eye images to assess whether it can differentiate this pattern from that associated with the out-of-focus eye image. Exploiting the neural network's advantage of being able to work with multi-channel input, the inputs to the CNN will be the grey level image and a high-pass filtered version, typically used to determine whether the iris is in focus or not. The complete system synthetises other cores and implements CNN using the so-called Deep Learning Processor Unit (DPU), the intellectual property (IP) block released by AMD/Xilinx. Compared to previous hardware designs for implementing FPGA-based CNNs, the DPU IP supports extensive deep learning core functions, and developers can leverage DPUs to conveniently accelerate CNN inference. Experimental validation has been successfully addressed in a real-world scenario working with walking subjects, demonstrating that it is possible to detect only eye images that are in focus. This prototype module includes a CMOS digital image sensor that provides 16 Mpixel images, and outputs a stream of detected eyes as 640 × 480 images. The module correctly discards up to 95% of the eyes present in the input images as not being correctly focused.

Keywords: eye detection; convolutional neural networks; multiprocessor system-on-chip (MPSoC); Deep Learning Processor Unit (DPU)

check for updates

Citation: Ruiz-Beltrán, C.A.; Romero-Garcés, A.; González-García, M.; Marfil, R.; Bandera, A. FPGA-Based CNN for Eye Detection in an Iris Recognition at a Distance System. *Electronics* **2023**, *12*, 4713. https://doi.org/10.3390/electronics12224713

Academic Editors: Valeri Mladenov and D. J. Lee

Received: 13 September 2023
Revised: 10 November 2023
Accepted: 17 November 2023
Published: 20 November 2023

Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Due to their exceptional performance, convolutional neural networks (CNNs) have emerged as the state of the art in recent years for image recognition, object detection, image segmentation and many other applications, rapidly replacing traditional computer vision methods [1]. In brief, a CNN is a type of feedforward neural network that is distinguished by its ability to extract the most relevant features for the task to be solved from the data utilising convolutional structures [1]. Due to the advantages they provide, such as local connection, weight sharing and dimensionality reduction by sampling, these types of networks have become extremely popular in research and industry scenarios. Nonetheless, despite their excellent performance, the high computational complexity

associated with CNNs may pose a hindrance to their use in applications where power consumption or device size/weight are limiting factors. For example, for the inference of a small image (244 × 244), a large CNN may require around 40G of multiplication or addition operations [2]. The most popular solution for handling such complexity is to employ graphics processing units (GPUs), and make use of the degree of parallelism they exhibit. GPUs have been widely employed for CNN training and inference, but they are power-hungry engines, which has led to many CNN-based applications being deployed on external servers (cloud computing).

As previously mentioned, there exists a significant niche of applications that necessitate edge computing. These applications typically rely on battery power (for instance, unmanned aerial vehicles, intelligent robotics) and/or the device's size and weight are key factors (Internet of Things, implanted biomedical devices). In these scenarios, it is possible that GPUs may not be the best solution [3]; therefore, other possibilities have been investigated, such as mobile GPUs [4], tensor processing units (TPUs) [5], field programmable gate arrays (FPGAs) [6,7], in-memory computing (iMC) [8] and application specific integrated circuit (ASIC) [9]. Similar to the GPU, TPU power consumption can be excessive for edge computing, with iMCs and ASICs being the most energy efficient alternatives. In contrast, due to their lack of reprogrammability, these two options present poor compatibility and scalability with CNN models, characterised by rapid evolution. FPGA offers reprogrammability and, when its hardware architecture is well designed, provides adequate energy efficiency [10,11]. FPGA can satisfy power and size constraints, emerging as a design alternative [12]. It has both pipeline parallelism and data parallelism, so it provides lower latency for processing tasks, becoming a high-performance and flexible accelerator for CNN inference [13]. Moreover, FPGA has nowadays been integrated into multi-processor system-on-chips (MPSoCs), in which computer and embedded logic elements are integrated (such as the Zynq-7000 SoC and Zynq Ultrascale+ MPSoC series of FPGA boards). In this way, these MPSoCs provide the FPGA with the ability to accelerate CNN inference and the computing capability to work as a stand-alone system without the need for interfacing to an external computer/controller.

Biometric identification over long distances and with people on the move is a complex challenge. In particular, the situation is complicated when the element used for identification is small, such as the iris, which is the highly textured, ring-shaped, externally visible tissue present in the eye. Each iris has a unique texture pattern that allows for unique identification of a person. This characteristic renders iris recognition identification a popular and widely applicable solution, especially in its version of iris identification at a distance (IAAD), which is utilised in various fields such as border control, surveillance, law enforcement, etc. [14]. In our case, the idea is that the system can capture the iris of a moving subject about two metres away from the camera. Like previous systems, our system will be deployed to cover a partially controlled access point, where subjects must walk along a guideline and look directly into the camera. People will walk through this gateway at a normal pace and must avoid any behaviour that would impede the acquisition of the iris image. In this scenario, the camera position will be fixed and, to cover a field of view of a certain size, a 16 Mpixel sensor is used. There will be a single camera, which will capture the irises with a resolution of about 190 pixels per centimetre. With these systems, one of the big problems is that the depth of field is very limited (only about ten centimetres), so in order to capture at least a couple of images of the irises with the necessary quality of focus, it will be necessary to process many images per second (in our case, the maximum provided by the sensor (47 fps)). The problem is that, as the system has about 2–3 s of recording per user in which the eyes can be detected, the number of eye images that will have to be sent to the external computer for processing can exceed 250 images. As a normal case, the external computer cannot process this volume of information before the user has left the access point. One solution to this problem is to filter out the large number of images in which the iris is not in focus. This means discarding almost 97% of the eye images captured by the system [15]. In this paper, we focus on one of the first steps of the iris recognition

identification system: the detection of the eye/iris image. As reviewed in Section 2, deep models have been intensively used for solving the eye detection task. To speed up the hardware description process, the CNN can be deployed on the FPGA using the Deep Learning Unit (DPU) proposed by AMD/Xilinx. DPU is a hardware engine dedicated to neural networks. Its strength lies in its ability to accelerate convolutional computation, maximum pooling, full connection and activation function calculation, with configurable parameters. DPU supports classical CNN structures, such as YOLO (You Only Look Once), VGG (Visual Geometry Group), ResNet or MobileNet. In this case, a CNN hardware and software platform for eye detection based on DPU accelerator is synthesised in a Zynq UltraScale+ MPSoC XCZU4EV. Tiny YOLO-v3 eye detection is deployed in the FPGA using DPU. The computational performance and acceleration effect of DPU are analysed. To ensure that only focused eyes are detected, use will be made of the CNN's ability to support multi-channel inputs. Thus, in addition to the grey level image captured by the NIR sensor, a high-pass filtered image is introduced as a second input channel. The filter used is the one originally proposed by J. Daugman to evaluate whether or not a detected iris was affected by defocus blur [16] With this input, the trained network discards practically all defocused eyes, which are not detected at all. In our scenario, this means that only 8–9 eyes per person are detected. The implemented system includes frame grabber and eye region detection on a single platform, and can process 87 fps, providing as output only regions containing focused eyes.

The rest of the paper is organised as follows: The state of the art in the topic of eye detection with deep models and of real-time object detection on FPGA-based devices is briefly revised in Section 2. Section 3 provides an overview of the whole proposed framework for image capture and eye detection and details about the DPU-based implementation. Experimental results are presented in Section 4. Finally, conclusions and future work are presented in Section 5.

2. Related Work

As in numerous other fields of signal processing, many of the basic tasks in computer vision and image processing are currently dominated by the use of deep CNNs. Data condensation and deep training aimed at enhancing the really essential information allow CNNs to extract abstract features, which then achieve excellent results in the classification phase [17]. This is why, in the specific field of object detection, CNN has become the state of the art.

However, previous work has already glimpsed the potential of neural networks to solve this problem. In the early 1990s, Waite and Vincent [18] designed a neural network for localising eyes. Briefly, they suggested that there exist eye micro-features (e.g., top and bottom eyelid, and left and right corners) that are invariant, and, therefore, micro-features near image regions can be used to generate a neural network separately. Based on this same scheme, Reinders et al. [19] proposed the detection of facial features, such as eyes, nose or mouth, in a sequence of images using the magnitude and orientation of the gradient as inputs to the neural network. The proposal searches a target area using different neural networks to combine the results and locate the features. The final output of the neural network is post-processed using a probabilistic method that takes into account the geometric information of the micro-features.

In the iTracker [20], an end-to-end CNN is trained for eye tracking. They use as input to the model the image of the face and its location in the image (face grid), and the images of the eyes. The model then infers the head pose (relative to the camera) and the eye poses (relative to the head). The CNN is similar to the popular AlexNet [21]. For real-time inference, additional knowledge is added (face and facial landmark eye detection). Thus, the model can run on an iPhone at 10–15 fps. Sun et al. [22] described a facial landmark detector using a three-stage cascaded CNN. The first stage of the model provides a raw estimate of facial points, which are refined in the next two stages. Similarly, the model proposed by Huang et al. [23] uses two-stage cascaded CNNs. The first stage performs

eye landmark detection and eye state estimation. Multitask learning is employed to obtain good initial eye positions. The second stage refines the eye positions. The model proposed by Nsaif et al. [17] also follows a multi-mission learning scheme, where Faster R-CNN (region-based CNN) is used to detect the initial eye regions, and Gabor filters and a naïve Bayes model is employed for fine-tuning these positions. The R-CNN was proposed as an accurate object detector by Girschick et al. [24]. In summary, the R-CNN takes an input image and firstly obtains bottom-up region proposals via selective search (SS). The image region in each window is warped to a fixed size, and a pre-trained deep network is used to compute features for each window. Then, it classifies these regions using class-specific linear SVMs (Support Vector Machines) trained on these features. As a major disadvantage, it applies the deep CNN for feature extraction repeatedly to all windows in the image, which is a hard timing bottleneck [17]. To reduce the cost of the R-CNN, convolutions can be shared across region proposals. For instance, adaptively sized pooling (Spatial Pyramid Pooling network) SPP-Net [25] extracts the feature maps from the whole image only once, and then it applies the spatial pyramid pooling on each region proposal (of the feature maps) to pool a fixed-length representation of this proposal. Thus, the SSP-Net extracts features from regions of the feature maps and not directly from image regions. It only applies the time-consuming convolutions once. The Fast R-CNN conducts end-to-end training on shared convolutional features [26]. It takes as input the whole image and a set of object proposals. The entire image is then processed with several convolutional and max-pooling layers to obtain the feature map. For each region proposal, a fixed-length feature vector is extracted from the feature map. These feature vectors are fed into a set of fully connected layers that finally branch into two output layers: one of them provides the probability estimate over the set of object classes, and the other one provides the refined bounding-box of the object. Surprisingly, these methods strive to speed up the feature extraction process but do not consider the problem of the initial proposal of regions, which is completely detached from the CNN. This causes the bottleneck to be in this initial phase. To avoid this problem, Faster R-CNN is proposed [27]. Faster R-CNN consists of two modules: a deep CNN to propose regions (the Region Proposal Network, RPN), and a Fast R-CNN that uses these regions. Both modules are not independent, but are intimately tied together, as the whole system is a unified network for object detection. Faster R-CNN was implemented on AMD/Xilinx® Zynq®-based FPGA using Python productivity for Zynq (PYNQ) [28]. The pre-trained model (Faster RCNN + Inception v2) was implemented using TensorFlow Application Programming Interface (API). Specifically, the model was implemented in an AMD/Xilinx® Zynq® ZCU104 FPGA (XCZU7EV-2FFVC1156 MPSoC), and the inference time was 58 ms (17 fps). The authors argue that this inference time is comparable to the one provided when testing in an i7 Intel® core CPU [28].

Other popular CNN-based object detectors do not consider the initial existence of a set of region proposals. In 2016, YOLO was proposed by Redmon et al. [29]. The goal is to accomplish the detection procedure as well as to estimate class probabilities and bounding boxes from full images in only one evaluation. YOLO uses a straightforward scheme based on regression to predict the detection outputs [29]. Since its creation, YOLO has evolved through many iterations (YOLOv1, YOLO9000, YOLOv3, PP-YOLO, etc.). The Single Shot MultiBox Detector (SSD) [30] and the CornerNet [31] are other examples of regression-based approaches. This single-shot scheme has been preferred for FPGA implementation. To reduce off-chip accesses, Nguyen et al. [32] proposed to retrain and quantise the parameters of the YOLO CNN using binary weight and flexible low-bit activation. Using a Virtex-7 VC707 FPGA device and 416 × 416 size images, they are able to process 109.3 fps. Nakahara et al. [33] proposed a light-weight YOLOv2, which includes a binarised CNN for feature extraction and parallel support vector regression (SVR) for predicting the detection outputs (class estimation and bounding box prediction). To reduce the computational complexity of SSD, Sandler et al. [34] proposed to use as base network of SSD the MobileNetV2 mobile architecture, as well as to replace convolutions by lightweight depthwise separable convolutions. The so-called SSDLite was implemented on FPGA by

Kim et al. [35], achieving a relevant frame rate of 84.8 fps when implemented on Intel Arria 10 FPGA. Building on the Tiny YOLO, the proposal by Preußer et al. [36] makes use of the computational opportunities provided by the Zynq UltraScale+ platform in the Programmable Logic (quantisation of the input and output layers to eight-bit fixed-point values and binarised input layers), but also in the Processing System (multi-threading and NEON vectorisation). Wang et al. [37] designed and implemented a hardware accelerator based on the YOLOv3 network model. They make intensive use of Vitis AI to reduce the network scale and decrease the access time to off-chip memory. Using a Xilinx Zynq ZCU104, they reported a maximum frame rate of 206.37 fps (but they do not work with a video stream, but first read a large-size image into the memory). But one of the most widely used versions in its transfer to FPGA has been the Tiny-YOLOv3 [38]. Li et al. [39] improve this algorithm by increasing from two-scale to three-scale detection, and merging the batch normalisation layer with the convolution layer. Zhang et al. [40] mapped the Tiny-YOLOv3 structure to the FPGA and optimised the accelerator architecture for Zedboard to make it run with very limited resources. To reduce the memory size and improve the speed, YOLOv3-tiny was compressed via a model quantisation method [41]. Specifically, the trained model was quantised with 8-bit fixed-point and implemented on a Xilinx Ultrascale+MPSoC ZCU102 board. They showed that the memory size of the model decreased from 33.1 MB to 8.27 MB. The process can distinguish between two traffic signals in real time (104.17 fps). Velicheti et al. [42] also evaluated multiple precision of Tiny-YOLOv3 (FIXED-8, FIXED-16, FLOAT32). As a comparison with the standard version of YOLOv3, Esen et al. [43] reported a frame rate of 41.1 fps when running the model with images of size 224 $\times$ 224 on the ZCU102 board.

Instead of modifying the parameters/algorithm of the network models, another option is to codesign the model and hardware architecture together. Ma et al. [44] proposed a hardware implementation of the SSD300 algorithm based on replacing dilated convolutions and convolutions associated with a stride of 2 with convolutions with a stride of 1, using fixed-point arithmetic, and employing dynamic quantisation to retain the detection accuracy of floating-point representation. Hardware/software codesign was employed to accelerate Tiny-YOLOv3 by designing a hardware accelerator for convolution [45]. Suh et al. [46] presented an energy-efficient accelerator on a resource-constrained FPGA. They trained the VGG-based SSD using a uniform/unified quantisation scheme (UniPOT) and optimised the DSP/memory utilisation employed dual-data rate DSP design to double the throughput, and reduced DDR latency aided by DMA descriptor buffer design.

3. Methodology

The proposed system for person identification via iris recognition consists of an iris image capture and processing unit that is located on a pole, facing an arch or access point. The optics and sensor employed by the system allow the capture of iris images focused at about 1.7 m from the pose location. The exposure time is very low and allows motion blur to be ruled out when the person walks at a normal speed (1–2 m per second). However, the depth of field is very shallow (only about 10–15 cm), so to ensure that a focused image is captured the system is forced to work at the maximum speed of the image sensor used (a Teledyne e2v EMERALD 16MP, which provides up to 47 fps). Figure 1 (Left) illustrates the layout of the application scenario. The pole includes both the sensor and the capture and processing system, as well as the lighting (51 W using high-power LEDs). If the distance between the sensor and the subject is increased, or decreased, the optical parameters of the system (mainly the focal length of the lens) must be adjusted to maintain the capture resolution (a minimum of 200–250 pixels/centimetre) and to increase, as far as possible, the depth of field. The capture and processing system is connected by cable to a computer with an i9 Intel® processor, which is responsible for extracting the normalised iris pattern and comparing it with the database to close the identification. These tasks are outside the scope of this article.

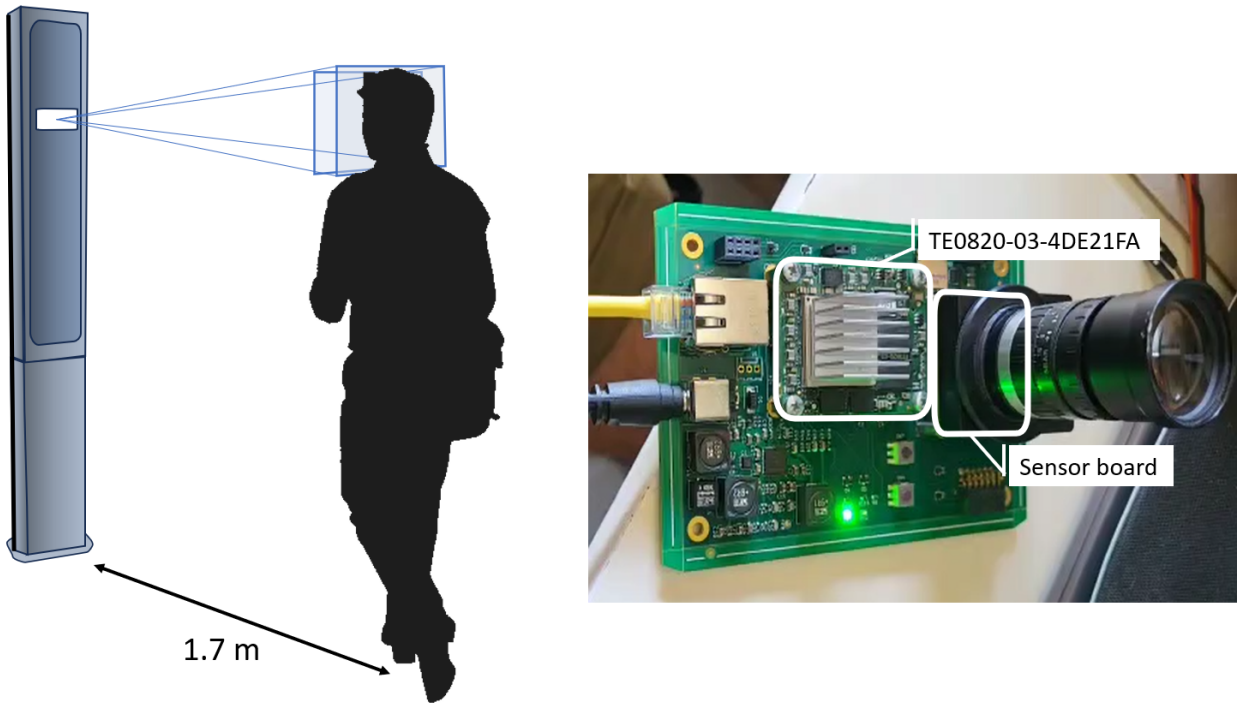

Figure 1. (**Left**) Layout of the application scenario; and (**right**) capture and processing unit. It includes the AMD/Xilinx UltraScale+, image sensor EMERALD 16MP, optics and two additional boards (see text for details).

The design attempts to make the image available for processing as quickly as possible. Since the image capture is done with a custom frame grabber implemented in the logic part of the FPGA, the most optimal way to implement the image processing is for it to be carried out in the FPGA itself. Thus, as the image pixels are read, they can be processed. In cases in which a GPU is used for processing, a communication channel with the frame grabber would have to be defined to reduce latency. Options such as AMD's DirectGMA or NVIDIA's GPUDirect require a PCIe bus to which both devices are connected. In our case, the embedded device does not require a CPU or the existence of such a bus. Embedding the processing in the FPGA reduces weight and power consumption, and the computational power of the FPGA is comparable to that of the GPU as previous work has demonstrated [33,47,48]. Figure 1 (Right) shows the capture and processing unit deployed on the pole. The purpose of this unit is to capture a video stream of the person passing the access point from which the focused images are filtered and the regions of interest (ROIs) are detected and cropped from the input image, those ROIs contain the eye images that will be sent to the external PC to be processed. The unit is built around a commercial AMD/Xilinx Ultrascale+ module (the TE0820-03-4DE21FA micromodule from Trenz) and the EMERALD 16MP from Teledyne e2v. The interface between these two modules is provided by two carrier boards. The main features of the TE0820-03-4DE21FA are summarised in Table 1. The TE0820-03-4DE21FA is an industrial-grade MPSoC module integrating an AMD/Xilinx Zynq™ UltraScale+™ ZU4EV, 2 GByte DDR4 SDRAM, and 128 MByte Flash memory for configuration and operation.

The EMERALD 16MP features a small true global shutter pixel (2.8 µm) and a reduced DSNU (Dark Signal Non-Uniformity) value. Both features are adequate for our application scenario (low-light context due to a small exposure time and a reduced aperture). Table 2 summarises the relevant features of the EMERALD 16MP sensor.

Table 1. Key features of the TE0820-03-4DE21FA module (https://docs.xilinx.com/v/u/en-US/ds891-zynq-ultrascale-plus-overview (accessed on 3 October 2023)).

AMD/Xilinx Zynq™ UltraScale+™	**XCZU4EV-1SFVC784I**
Logic Cells	192 K
Look Up Tables (LUTs)	88 K
DSP Slices	728 (18 × 25 MACCs)
CLB Flip-Flops	176 K
Block RAM	4.5 Mb
EMMC FLASH	8 GB
QSPI FLASH	256 Mbit
HP I/O	96
HD I/O	84
Application Processor	Quad-Core ARM Cortex-A53 MPCore
Real-Time Processor	Dual-core ARM Cortex-R5 MPCore
Graphics Processor	Mali-400 MP2

Table 2. Key features of the EMERALD 16MP sensor (https://imaging.teledyne-e2v.com/content/uploads/2023/06/2023-05-24_TDY-e2v_Standard-CMOS-Image-Sensors-Guide_web.pdf (accessed on 3 October 2023)).

Resolution (pixels)	16 M
Format (H × V)	4096 × 4096
Pixel pitch (µm)	2.8
Shutter Type	Global
Frame Rate (at 10 bits)	47
Output format	LVDS from 8b to 12b

Previous work has solved the problem of real-time eye detection using a modified version of the popular algorithm proposed by Viola and Jones [49]. This implementation is very fast, and allows the eye images to be obtained at the required speed [50]. The problem is that the system provides many false positives, which saturate the subsequent recognition module. Many of these false positives contain out-of-focus eyes, which could be discarded by a blur estimation module [15], but others are associated with regions that do not contain eyes. The biggest problem with this approach lies in scalability and adaptability. Any change involves rewriting the FPGA cores, which is time-consuming and unsuitable for scalability. On the other hand, the use of DPUs allows for faster development since, once the DPU is implemented in the design, it is easy to implement different models of CNNs. Furthermore, the system can be reused to detect different objects besides the eye, e.g., detecting the mouth to detect a yawn in a driver drowsiness detection scenario. It can be easily retrained if problems are detected at a late stage of development, and allows for model comparison, which helps to optimise and find the best solution.

Figure 2 shows the architecture for deploying the CNN-based eye detection flow on an MPSoC-based platform. The input images come from the EMERALD 16M sensor. Although the images are captured, and stored in DDR memory, in their original size, for processing they are reduced to a size of 256 × 256 pixels. Previous work has shown that eyes are accurately detected despite this reduction in resolution [15,50]. These scaled images are made available to the DPU using VDMA. But the system will receive a multi-channel input, where in addition to the input image in grey levels, a second channel will include a high-pass filtered version. Figure 3 provides an example of what these filtered versions look like, both for a focused image and an out-of-focus image. The grey level images in the top row show little difference, so if you train the network with them, you will have an in-focus or out-of-focus eye detector. The purpose of adding the high-pass filtering features is to verify whether they allow out-of-focus eyes to go undetected. The DPU is in charge of deep learning acceleration. The DPU requires instructions to implement the

network, which are prepared using the Deep Neural Network Compiler (DNNC) and Deep Neural Network Assembler (DNNAS) tools. The scaled image can be monitored via HDMI for verification. The detected eye regions are managed via the Processing System (PS) of the MPSoC. Briefly, the system must rescale the coordinates of the detected eyes to crop high-resolution versions of the original 4096 × 4096 pixel images.

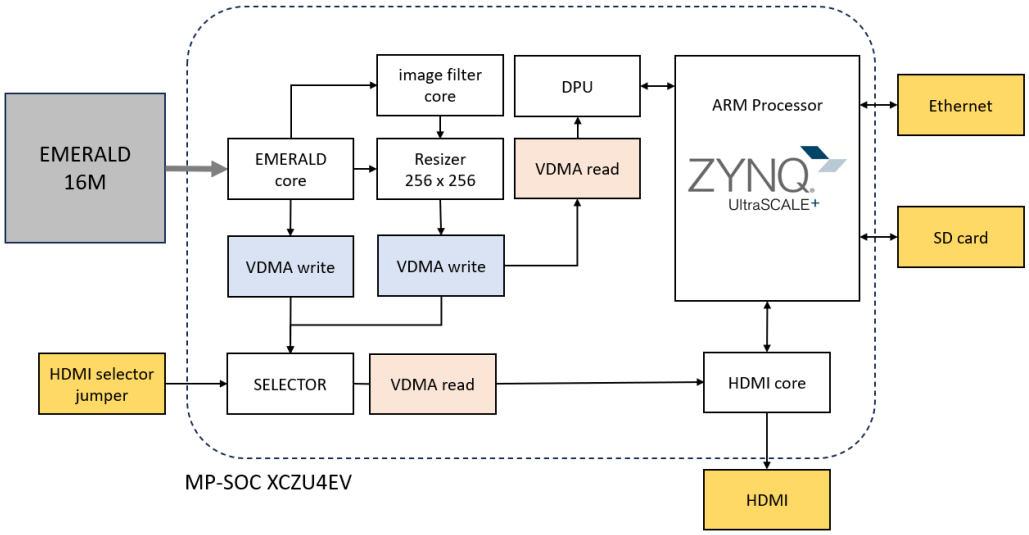

Figure 2. Onboard implementation for CNN-based eye detection.

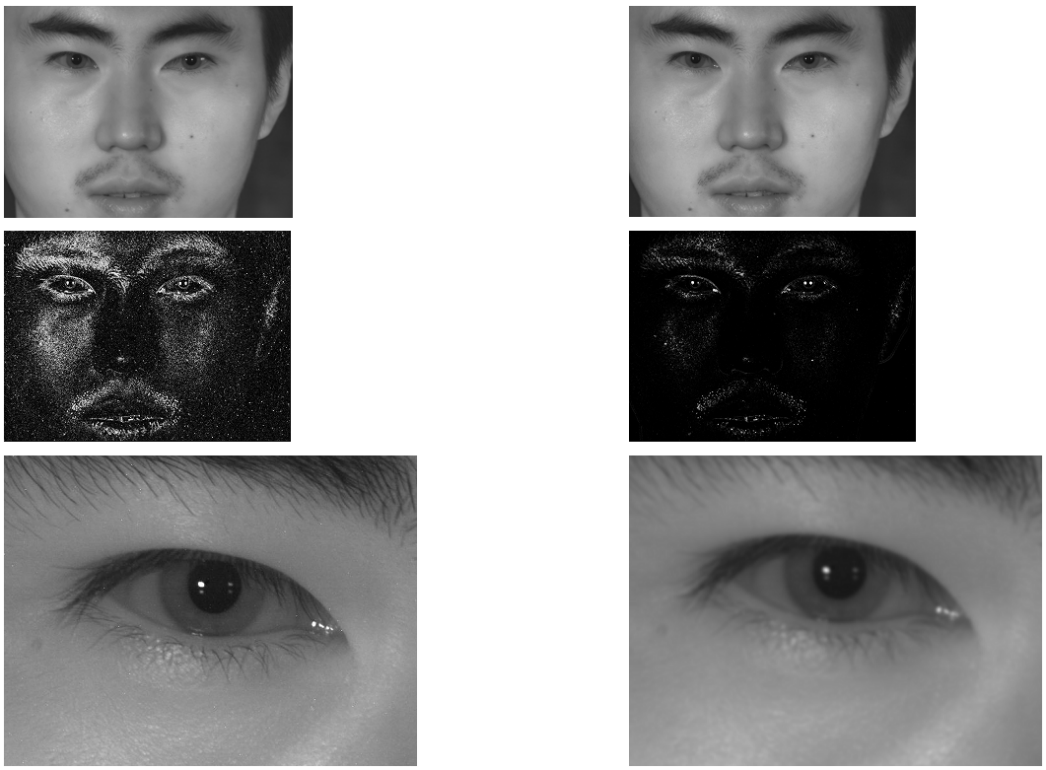

Figure 3. (**Left**) Focused image and associated high-pass filtered image, and (**right**) out-of-focus image and associated high-pass filtered image. In the bottom row, a zoomed-in image of the eye region is provided (see text for details).

3.1. The DPU IP Core

Parallelism and ease of programming are possibly the features that justify the use of FPGA to accelerate CNN inference [51]. To synthesise the CNN on the FPGA, two options

can be used: describe the hardware circuit directly using Hardware Description Language (HDL), or describe it using a high-level language (such as C/C++, System C or Matlab) and use the High Level Synthesis (HLS) tool to generate, from this high-level code, the HDL version. Possibly, the first option is more suitable, but it is also more time-consuming [52]. HLS can help the designer, typically more familiar with the use of high-level languages, by allowing to speed up the writing of the code. Furthermore, apart from aiding development from a design, HLS allows the verification of this design to be more efficient as well, as it can be carried out with a high-level language simulator, rather than an HDL simulator. Unlike an HDL simulation, the high-level verification does not have to simulate every clock edge, so it is much faster. If the high-level verification is successful, the HLS tool ensures that the generated HDL version corresponds to this description and therefore its functionality will be equivalent. This tool has been used extensively in the design of the cores that make up our proposal.

Despite attempts to synthetise the functional layers of the CNNs into pure FPGAs, most current work advocates the design of heterogeneous architectures, which make use of the FPGA but also the CPU. These platforms, such as the Xilinx Zynq SoC, allow a balance to be struck between performance and the flexibility needed to handle the different layers of the CNN. In our case, our aim is to implement the CNN in the programmable logic (PL) of a XCZU4EV AMD/Xilinx Zynq UltraScale+ MPSoC. To further accelerate the process of designing and synthesising a CNN in the logic part of the MPSoC, use will be made of the AMD Deep Learning Processor Unit (DPU) [11]. The DPU is a programmable engine, offered by AMD for Zynq-7000 SoC and Zynq Ultrascale+ MPSoC devices. It consists of a Computing Engine, an Instruction Scheduler, and an On-Chip Buffer Controller (see Figure 4). After start-up, the DPU fetches instructions and decoders via the Fetcher and Decoder modules from the off-chip memory to control, using the Dispatcher, the operation of the Computing Engine [51]. There is a specialised instruction set for DPU generated via the Vitis™ AI compiler, which enables DPU to work efficiently for many CNNs (VGG, ResNet, GoogLeNet, YOLO, SSD, MobileNet, FPN, etc.). The On-Chip Buffer Controller manages the on-chip memory to store data (buffer input activations, intermediate feature maps, and output meta-data). To reduce off-chip memory bandwidth, it reuses data as much as possible. The Computing Engine implements a deep pipelined design for convolution computation, where the PEs (Processing Elements) take full advantage of fine-grained building blocks in the AMD/Xilinx device for constructing multipliers, adders, etc. The DPU is implemented in the programmable logic (PL) and is integrated with the processing system (PS) using an AXI interconnect.

In our case, the DPU targeted reference design provided by Vitis AI 3.0 was implemented with AMD Vitis for the TE0820-03-4DE21FA board. The B1600 configuration of the DPU was synthetised with default settings (Low RAM Usage, Channel augmentation disabled, Save argmax enabled). Table 3 shows the resource utilisation. The operating frequency of the DPU was 150 MHz. Petalinux 2021.2 was used to generate a Linux OS image for the TE0820-03-4DE21FA. The CNN application running on the DPU was generated as described in Section 3.2.2.

Table 3. Resource utilisation of the DPU targeted reference design.

Resource	Utilisation	Available	Utilisation (%)
BRAM_18K	182	256	71
DSP48E	335	728	46
FF	94,867	175,680	54
LUT	66,758	87,840	76
URAM	24	48	50
MMCM	1	4	25

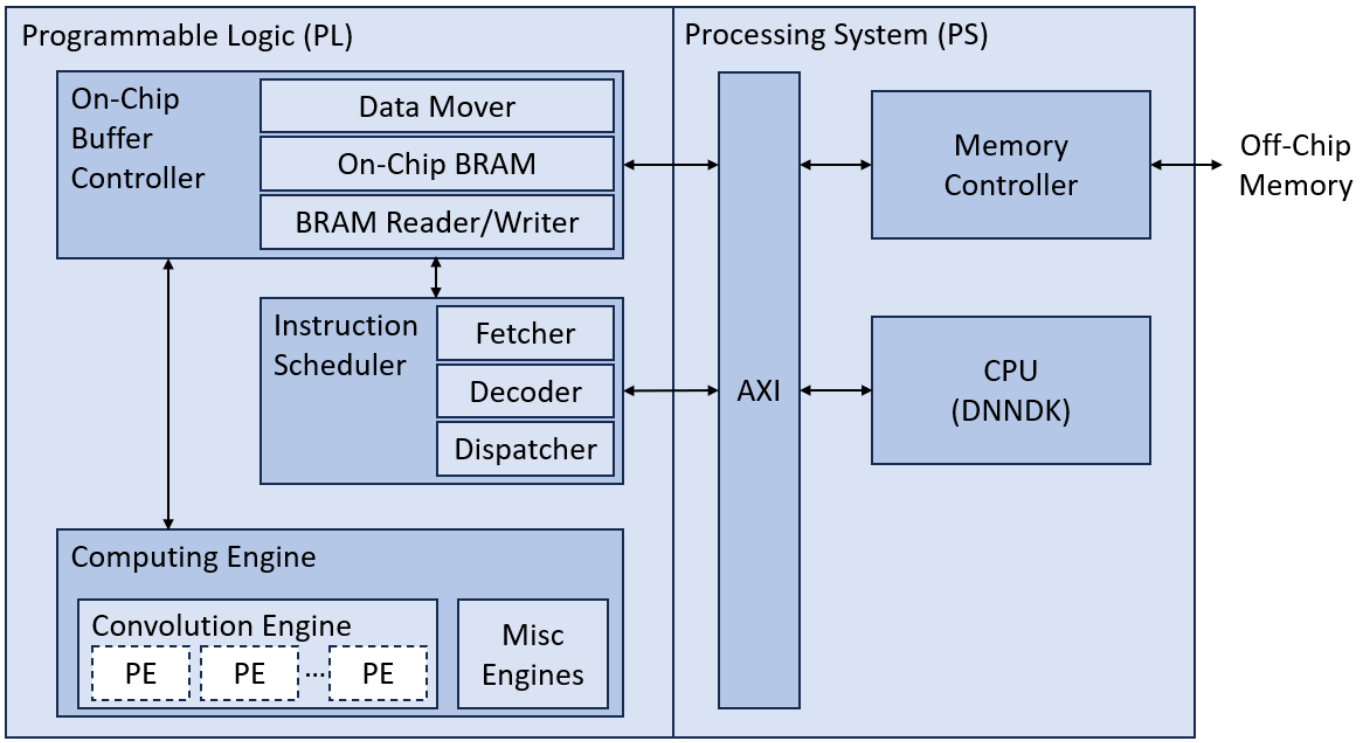

Figure 4. Basic structure of AMD DPU.

3.2. Generation of the CNN Model

3.2.1. YOLO Version 3 Tiny

The Tiny YOLO-v3 was proposed by Joseph Redmon to reduce the complex network (107 layers) of YOLO-v3 [53]. The aim is to increase the detection speed, providing a network that can be executed in real-time when running in an embedded board. Briefly, the Tiny YOLO-v3 divides up the image into a grid. Then, it predicts a three-dimensional tensor containing objectness score, bounding box and class prediction at two different scales (dividing the image into a 13 × 13 grid and a 26 × 26 grid). Bounding boxes without the best objectness scores are ignored for final detections.

Figure 5 shows the complete architecture of the network. Instead of using Darknet-53 as YOLO-v3 does [53], the architecture is based on a seven-layer standard convolution structure. In its original version, the input image is 416 × 416 pixels in size. After ten convolutions and six max-pooling operations, the output feature maps have a size of 13 × 13 units. In addition, the feature map after the eighth convolution is convolved and upsampled to obtain a size of 26 × 26 units. This map is concatenated with the result of the fifth convolution and convolved to obtain a second output of size 26 × 26 units. Both output feature maps, at two different scales, contain the prediction information about objects. The reduction in FLOPS and model size with respect to YOLO-v3 is very significant [40,41,54], allowing it to run on embedded devices. Although some authors point out that this structure cannot extract higher-level semantic features, so its accuracy is lower [41], in our application framework, in which only complete eyes of a certain size are to be detected under the same lighting conditions, the network has proven to be efficient and accurate.

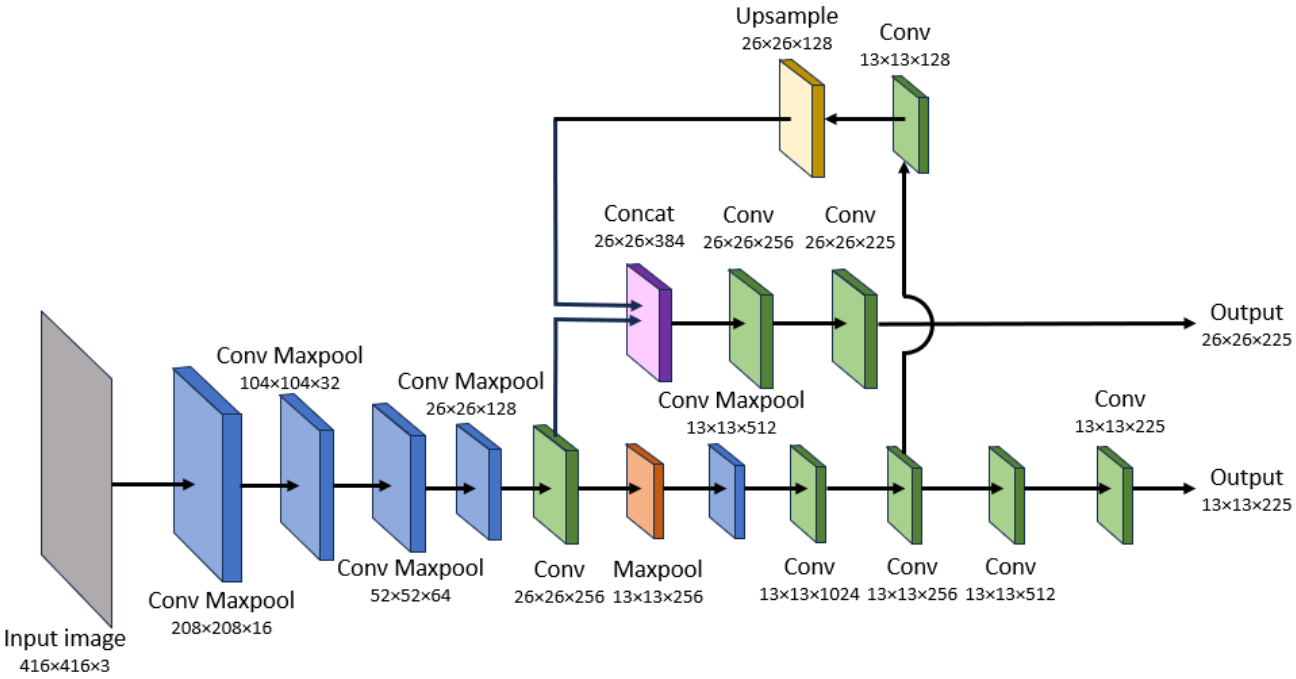

Figure 5. The architecture of Tiny YOLO-v3. Conv Maxpool refers to convolution and max-pooling, Conv to convolution and Concat to concatenate. The numbers indicate the size of the output of the layer. See [54] for a detailed view of the architecture.

3.2.2. Neural Network Training

Darknet [53] is an open source framework created by Joseph Redmon, used primarily for the implementation and creation of convolutional neural networks. Darknet makes use of CUDA to take full advantage of the power of GPUs to accelerate the required computations. The CUDA architecture allows C/C++ developers to interface directly with Nvidia GPUs. This allows them to take advantage of the massive parallelism and computational power of these platforms, performing complex tasks in less time and processing large volumes of data more efficiently. In addition, this framework is responsible for doing all the image pre-processing, rescaling the images to the size of the neural network input and normalising the pixels to have a smaller scale ([0, 1] instead of [0, 255]) and thus achieving greater numerical stability that will lead to greater convergence. Once Darknet is installed, the weights of the Tiny YOLO-v3 neural network are downloaded, the configuration file is modified so that it can be trained with Darknet and the network is trained using a training and validation dataset.

CASIA-Iris-Distance v4 images were used to train the network. These images show a frontal image of the person's face with lighting conditions that are very similar to those in our scenario (see Figure 6). Images were captured indoors, with a distance close to 2 m, and using a self-developed long-range multi-modal biometric image acquisition and recognition system (LMBS). The illumination used is near-infrared. The eyes present in the 2567 images were manually labelled. All these eyes are correctly focused, in order to condition the subsequent operation of the detection process, and not to detect unfocused eyes.

The neural network was trained over 6000 iterations. This value is the minimum set by the YOLO-v3 guidance since there is only one class to be detected (2000 iterations for each type of object to be detected with a minimum total of 6000). The training loss curve is shown in Figure 7.

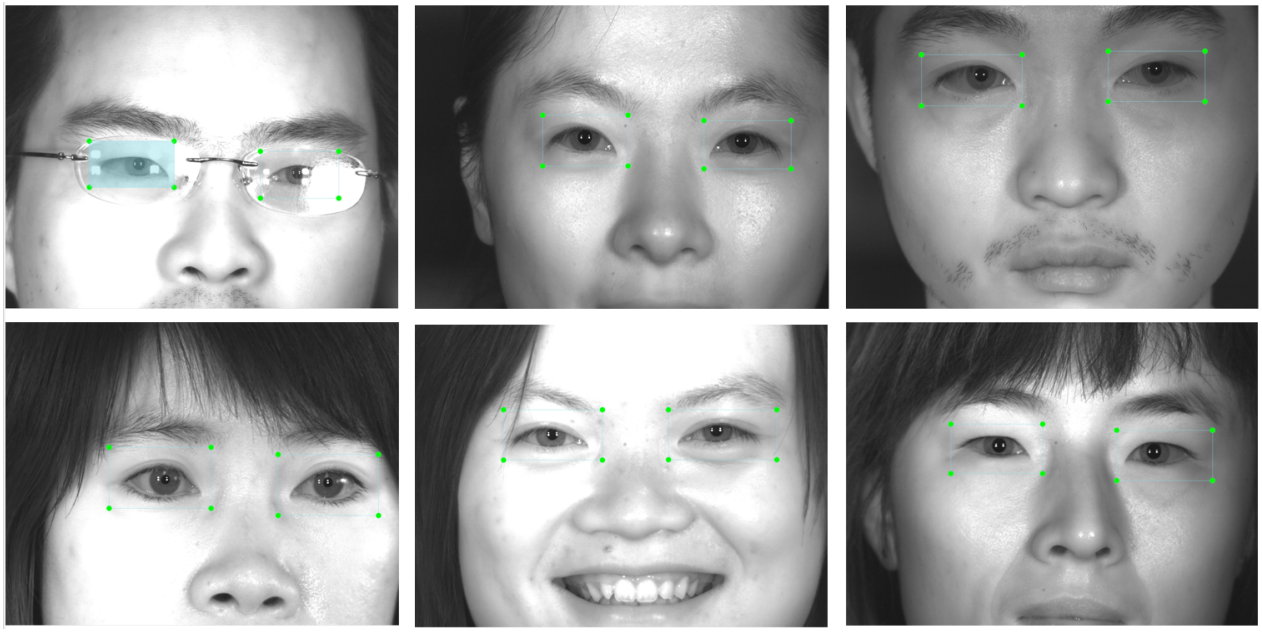

Figure 6. Images from the CASIA-Iris-Distance V4 database labelled using the labelImg framework (https://labelstud.io/ (accessed on 9 September 2023)). The small dots in the images are the corners of the bounding boxes that delimit eye regions.

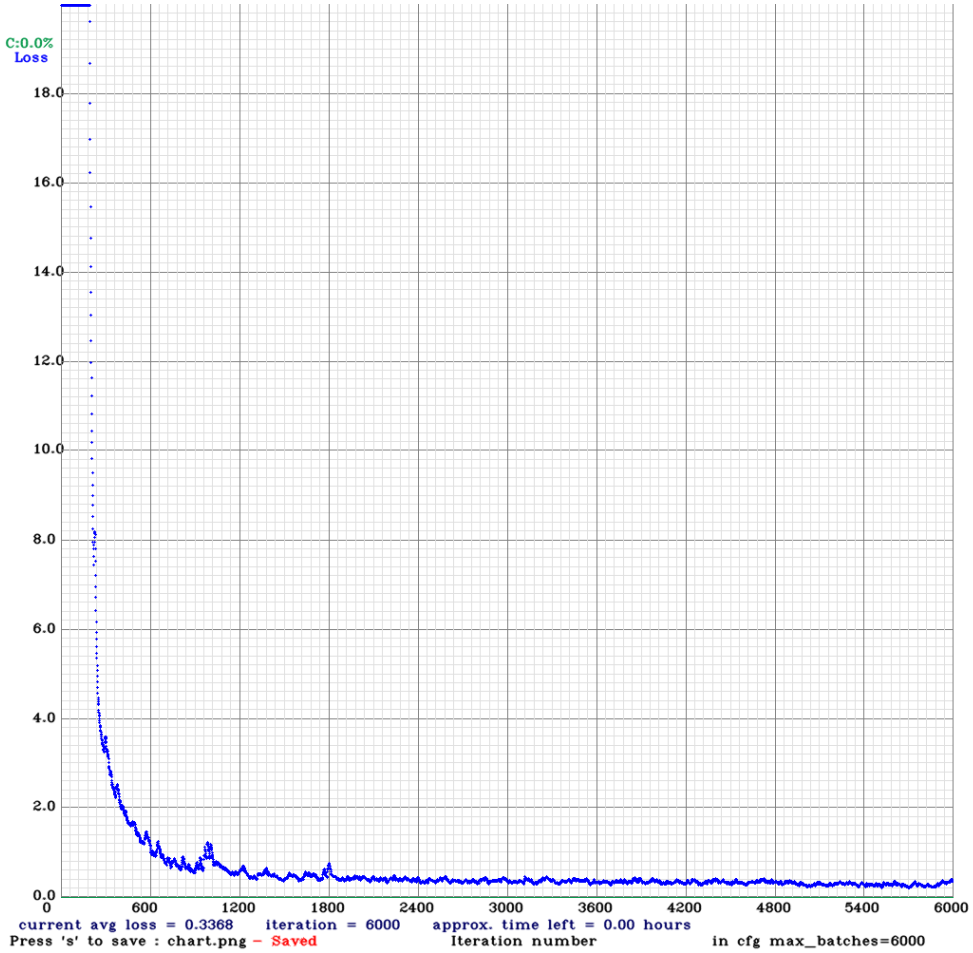

Figure 7. Tiny YOLO-v3 Training Loss curve.

Once the network was trained, several inference tests were performed on images that were not taken into account during training. In all cases, eyes that were in focus were correctly detected. This point is significant: the network does not detect out-of-focus eyes. As intended by this proposal, this allows it not only to detect eyes, but also to discard those that are not in focus. In a system that works with moving people, where up to 250 eye images can be captured in 2–3 s, being able to discard invalid eye images is really necessary [15]. In this implementation, it will therefore not be necessary to include a defocus blur estimation module in the system.

Because the Vitis AI toolchain only supports TensorFlow and PyTorch models, once the network had been trained, the set of weights was converted to TensorFlow files using the keras-YOLOv3-model-set software (https://github.com/david8862/keras-YOLOv3-model-set (accessed on 3 September 2023)). This converter takes the Darknet weights and configuration file to create the equivalent TensorFlow network from the provided data. Finally, once we have the weights in Tensorflow format we must define the network graph in a single file (freeze_graph, https://github.com/tensorflow/tensorflow/tree/master (accessed on 10 September 2023)). The model is not ready yet since it uses float values which are not suitable for the DPU; therefore, a quantisation process is carried out to convert those 32-bit floating-point arithmetic values with 8-bit fixed-point; this is achieved by means of the vitis AI tools via a quantisation process.

The DPU cores are flexible with the implementation; there are many options such as the multiply–accumulate operations, memory usage, activation functions and softmax core, among others. This means that the model must be compiled taking into account the implementation details of the deployed DPU core. In Vitis AI, both the architecture of the machine learning model and the weights and parameters needed to perform inference on the hardware platform are encapsulated in an xmodel file extension. To generate it, the steps of the Vitis AI workflow must be followed. Specifically, the deployed DPU has an identification code denominated fingerprint in the form of a file arch.json, which is generated via the Vivado synthesising tool. This file is used to encode all the properties of the deployed DPU so the compile model can make use of all the available resources within the core, resulting in a highly optimised DPU instruction sequence and any operation that is not available on the DPU is carried out by the CPU.

4. Inference Results

The entire framework was validated with images captured in a real-life scenario in order to assess its ability to detect focused eye images. When the input image size of 416×416 is maintained, the system detects eye regions in focus, discarding most of the regions containing out-of-focus eyes (at a rate of close to 100%). However, the processing rate is relatively low (29 fps). In any case, it is lower than the fps provided by of the EMERALD sensor (47 fps). To increase this frame rate, the size of the input image of the YOLO network can be decreased. For a size of 352×352 the speed increases up to 39 fps. If we decrease this size to 320×320, the system is able to process 44 fps, a value very close to the frames per second provided by the sensor. In previous work, the sensor input image was rescaled to a size of 256×256 pixels [15,50]. With this size, for example, these previous implementations using a modified Viola Jones classifier achieved 100% positive detections in the CASIA-Iris-Distance v4 database. When the size of the input image is rescaled to this size, the YOLO processing speed increases to 87 fps. The system correctly detects regions with in-focus eyes, but discards regions with out-of-focus eyes to a much lesser extent (see Figure 8). It must be noted that, in order to compare the results obtained when using one or the other input image size, the sequences were recorded and then all the frames were processed one by one to obtain the detections.

Figure 8. Frames of a sequence of a person passing in front of the sensor (boxes mark detected eyes): (**Top**) eyes detected using a 256 × 256 input image; (**Middle**) eyes detected using a 352 × 352 input image; and (**Bottom**) eyes detected using a 416 × 416 input image.

A measure of Mean Average Precision (mAP) can be used to evaluate the neural network. The mAP provides a quantitative measure of the detection accuracy, considering both the localisation accuracy and the classification accuracy of the detected objects. In our case, using frontal face images with correctly focused eye regions as a validation database, the mAP is 100% for input images of 256 × 256 pixels (Figure 9). That is, in these images, all eyes are correctly detected. It is important to note that we only have one class, that the correctly focused eyes in our scenario are always the same size, and that the lighting conditions are always the same. For input images of 416 × 416 pixels, the mAP is reduced to 95.31%.

Figure 9. Eyes detected using a 256 × 256 input image (boxes mark detected eyes).

Finally, the proposal's ability to handle faces of individuals of diverse races and ages was evaluated using the Flickr database [55]. The detected eyes are not valid for recognition, and the images are not NIR but have been captured in visible light. However, the method is able to detect the eyes without problem. Figure 10 shows several eye detections in images from this database. It is important to note that the proposed method has problems detecting

eyes in non-frontal faces, or in faces where hair partially covers the iris. In both cases, even if the eyes were detected, it would be difficult to use them for recognition.

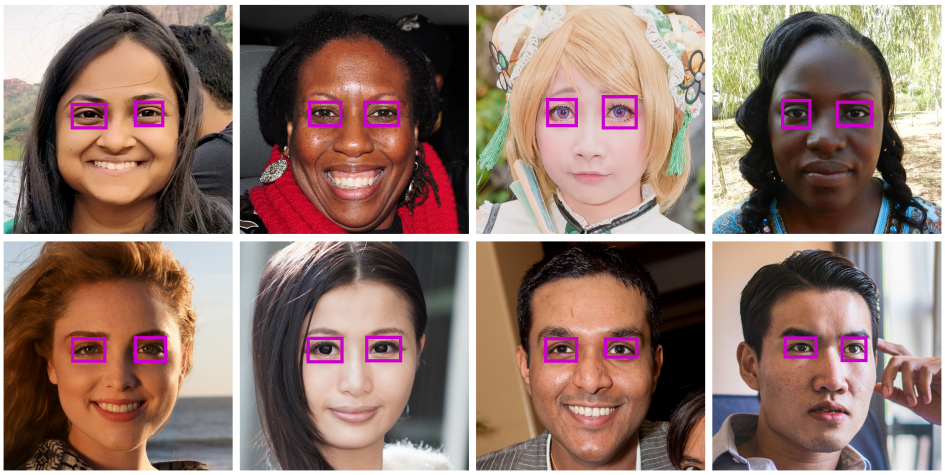

Figure 10. Eyes detected using 352 × 352 input images (boxes mark detected eyes). Original images are part of the Flickr database [55].

Comparison with Prior Work on Eye Detection

In order to compare the results provided by this proposal with respect to other approaches in the literature, only 20% of the CASIA-Iris-Distance v4.0 database has been used to retrain the network and the remaining 80% as a test set. The scheme is similar to that used in previous proposals, which allows for comparison of the results. Table 4 shows the results obtained via different methods using this same database for training and evaluation. Except for our previous proposal [50] (which also runs on a Ultrascale+ XCZU4EV micromodule), all these proposals run on an Intel i7-7700 CPU at 3.6 GHz, using an NVIDIA GeForce GTX 1070 (1920 CUDA cores and 8 GB of memory) and 16 GB of memory [17] In the table, we compare methods that are not based on Deep Learning techniques, such as Uhl and Wild's proposal [56], which detects faces and eyes, and Ruiz-Beltran et al. [50], which focuses only on eye detection, both based on the original proposals by Viola and Jones [49] and Lienhart and Maydt [57]. The other three methods in Table 4 are CNN-based proposals. As described in Section 2, these approaches implement algorithms that propose initial regions in which to locate the eyes. Faster R-CNN [27] introduces a Region Proposal Network (RPN) that shares the convolutional features of the full image with the detection network, allowing them to perform region proposals at almost no cost. Briefly, this RPN component informs the network where to look. This attention scheme is also found in FR-CNN-NB and FR-CNN-GNB [17]. The FR-CNNN-NB employs Faster R-CNN to extract the features and obtain a first estimate of the position of the eyes, which is then enhanced using a Bayesian model. The FR-CNN-GNB complements the previous proposal with the use of Gaussian filters (FR-CNNN-GNB).

Table 4. Comparison of the accuracy and time of eye detection of the proposed approach with other algorithms on the CASIA v4 Distance database.

Approach	Success Rate	Frames per Second (fps)
Uhl and Wild [56]	96.4%	0.78
Ruiz-Beltran et al. [50]	100%	88
Faster R-CNN [27]	98.21%	1.69
FR-CNN-NB [17]	99.1%	1.69
FR-CNN-GNB [17]	100%	1.69
Proposed (256 × 256 input images)	100%	87

To conduct these tests, our design must read the images from the SD card present in the hardware design. As described in Ruiz-Beltran et al. [50], the loading of the image into the frame buffer is carried out by the ARM (PS part), which is also responsible for managing the execution. The frame rate that can be processed per second is computed by adding specific video analysis cores in the design. As proposed by Ruiz-Beltran et al., the video channel rescales the image to a size of 256 × 256 pixels. It is important to note that all other methods work with the original size of the input images (2352 × 1728 pixels).

The results obtained (100% success rate) may indicate that the model is too complex for the problem we are solving (overfitting). The success rate in the CASIA v4 Distance dataset is very high, but it should be noted that it is the same as that obtained in the real scenario. In the dataset images, all the eyes present are correctly focused and therefore detected. In the tests carried out with the system deployed in the real environment, not a single focused eye image was not detected. However, the system has an error rate associated with the detection of eye images that, because they are out of focus, are not useful for recognition. This is a low rate (5–6 eye images per person passing through the system) and totally acceptable. As mentioned above, in this system, it is not possible to fail to detect eye images that are in focus. Significantly, the results are very similar to those obtained by Ruiz-Beltrán et al. [50], with the advantage that the use of current deep learning techniques allows for greater ease of training and the possibility of extending the system, if necessary, to detect facial features other than the eyes. In addition, the eye detection process is more robust. Figure 11 shows in its top row the detections using the first method. False detections, caused in part by artefacts created by the normalisation in brightness, can be seen, which should be discarded by the iris extraction system. The method detects all eyes that appear in the input sequence. The bottom row shows the results obtained via the proposed method. The eyes are correctly detected and there are no false positives.

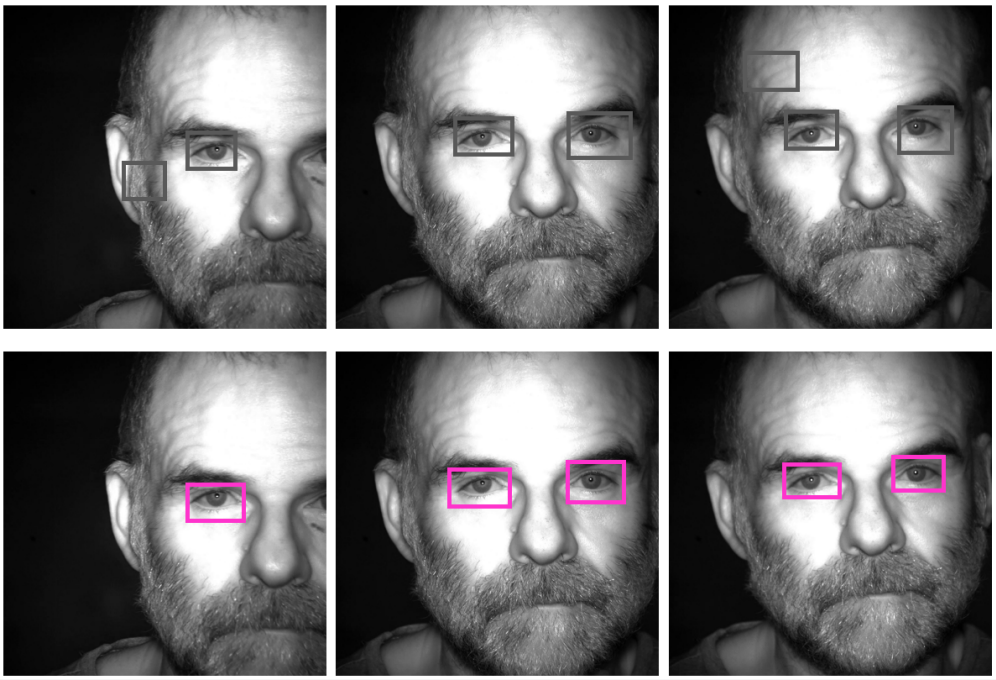

Figure 11. (**Top**) Eyes detected using the modified Viola-Jones approach [50]; and (**Bottom**) eyes detected using the proposed approach (256 × 256 input images). Rectangular boxes mark detected eyes.

5. Conclusions and Future Work

To facilitate the deployment of a remote identification system based on iris recognition, it is important to minimise the weight and power consumption of the devices, while preserving efficiency in terms of processing speed and performance. Furthermore, given

the large number of eyes that can be detected, and in order not to saturate later stages of the system, it is important that the device only detects, as far as possible, those images that contain focused eyes. This paper describes the implementation of a complete framework for detecting correctly focused eyes, synthesised in an MPSoC and designed to meet the constraints of remote iris recognition. The framework includes all hardware cores for image resizing, eye detection based on the Tiny-YOLO v3 network and final cropping of the eye images from the original input image. In order for the system to discard images of eyes that are not correctly focused, a simple but effective scheme has been proposed. Basically, the idea is to use as input to the system not only the captured image, but also a high-pass filtered version of it. As pointed out by J. Daugman [16], the effect of defocusing is mainly to attenuate the higher frequencies of the image. Tests show that, when this information is considered in the training and classification steps, the designed system detects only focused eye images. Like the rest of the system, the proposal has been correctly integrated into the hardware synthesised in the MPSoC.

The proposal has been extensively evaluated in a real working environment, with hardware selected (or custom designed) for this scenario. The system has demonstrated its ability to capture at a distance the eye images of moving people, discarding those affected by defocus blur. In the original input images captured by the sensor, the eye is relatively large, allowing the system to behave correctly when the input image is reduced to 256 $\times$ 256 pixel size. In fact, when both accuracy and speed are used as key factors in determining system parameters, the 256 $\times$ 256 input image size provides the best results. Thus, this size provides an accuracy of 100 percent on the validation database (images captured by the system itself) and a processing speed of 87 fps. While it is true that the system occasionally detects eyes that are not correctly focused, the set of unfocused images that are discarded exceeds 95% of the eyes that appear in the input images. Since not a single focused eye should be missed, it is preferable that the system admits a certain amount of false positives (unfocused eyes) but maintains the referred 100% detection of true positives (focused eyes). In summary, the proposed device functions as a smart camera, returning as output images of 640 $\times$ 480 pixels that include captures of the eyes in correct focus.

Future work aims to embed the later stages of the iris recognition system (iris segmentation and normalisation) in the MPSoC. Regarding the eye region detection system, the next steps will involve training the network using images captured by the system itself, and evaluating the possibility of using a DPU that processes more multiply–accumulate operations (MACs) per clock cycle (B2304 or B3136), for a faster network, which could work with image sensors that provide more images per second than the one currently deployed in the system. In addition, we are currently addressing the problem of detecting the presence of textured contact lenses using binarised statistical image features (BSIFs) and a set of three support vector machine (SVM)-based classifiers [58]. For integration into the MPSoC, we are testing the possibility of estimating the three required BSIF features for the entire image (we can achieve this simultaneously with image capture) and then implementing the SVM classification on the dual-core ARM Cortex-R5 MPCore available on the XCZU4EV-1SFVC784I.

Author Contributions: Conceptualization, C.A.R.-B.; Methodology, C.A.R.-B., A.R.-G. and M.G.-G.; Software, C.A.R.-B.; Validation, C.A.R.-B. and A.B.; Formal analysis, A.R.-G. and M.G.-G.; Investigation, C.A.R.-B. and M.G.-G.; Data curation, C.A.R.-B.; Writing—original draft, C.A.R.-B. and A.B.; Supervision, A.B.; Funding acquisition, R.M. and A.B. All authors have read and agreed to the published version of the manuscript.

Funding: This work has been supported by grants CPP2021-008931, PDC2022-133597-C42, TED2021-131739B-C21 and PID2022-137344OB-C32, funded by MCIN/AEI/10.13039/501100011033 and by the European Union NextGenerationEU/PRTR (for the first three grants), and "ERDF A way of making Europe" (for the fourth grant).

Data Availability Statement: Publicly available datasets were analyzed in this study. The Flickr database can be found here: https://github.com/NVlabs/ffhq-dataset. The CASIA-Iris Distance V4 datase can be found here: https://hycasia.github.io/dataset/casia-irisv4/. A third dataset was captured with the proposed system in the real scenario. These data are not publicly available because we are currently working to increase the volume of data and organise them properly. These data can be obtained on request from the corresponding author.

Acknowledgments: Portions of the research in this paper use the CASIA-Iris V4 collected by the Chinese Academy of Sciences—Institute of Automation (CASIA).

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

References

1. Zhang, X.; Ma, Y.; Xiong, J.; Hwu, W.M.W.; Kindratenko, V.; Chen, D. Exploring HW/SW Co-Design for Video Analysis on CPU-FPGA Heterogeneous Systems. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2022**, *41*, 1606–1619. [CrossRef]
2. Guo, K.; Sui, L.; Qiu, J.; Yu, J.; Wang, J.; Yao, S.; Han, S.; Wang, Y.; Yang, H. Angel-Eye: A Complete Design Flow for Mapping CNN onto Embedded FPGA. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2018**, *37*, 35–47. [CrossRef]
3. Zhang, X.; Li, Y.; Pan, J.; Chen, D. Algorithm/Accelerator Co-Design and Co-Search for Edge AI. *IEEE Trans. Circuits Syst. II Express Briefs* **2022**, *69*, 3064–3070. [CrossRef]
4. Latifi Oskouei, S.S.; Golestani, H.; Hashemi, M.; Ghiasi, S. CNNdroid: GPU-Accelerated Execution of Trained Deep Convolutional Neural Networks on Android. In Proceedings of the 24th ACM International Conference on Multimedia, Amsterdam, The Netherlands, 15–19 October 2016 ; Association for Computing Machinery: New York, NY, USA, 2016; MM '16, pp. 1201–1205. [CrossRef]
5. Jouppi, N.P.; Young, C.; Patil, N.; Patterson, D.; Agrawal, G.; Bajwa, R.; Bates, S.; Bhatia, S.; Boden, N.; Borchers, A.; et al. In-datacenter performance analysis of a tensor processing unit. In Proceedings of the 2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA), Toronto, ON, Canada, 24–28 June 2017; pp. 1–12. [CrossRef]
6. Novickis, R.; Justs, D.J.; Ozols, K.; Greitāns, M. An Approach of Feed-Forward Neural Network Throughput-Optimized Implementation in FPGA. *Electronics* **2020**, *9*, 2193. [CrossRef]
7. Chen, Y.X.; Ruan, S.J. A Throughput-Optimized Channel-Oriented Processing Element Array for Convolutional Neural Networks. *IEEE Trans. Circuits Syst. II Express Briefs* **2021**, *68*, 752–756. [CrossRef]
8. Jia, H.; Valavi, H.; Tang, Y.; Zhang, J.; Verma, N. A Programmable Heterogeneous Microprocessor Based on Bit-Scalable In-Memory Computing. *IEEE J. Solid-State Circuits* **2020**, *55*, 2609–2621. [CrossRef]
9. Chen, Y.H.; Krishna, T.; Emer, J.S.; Sze, V. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. *IEEE J. Solid-State Circuits* **2017**, *52*, 127–138. [CrossRef]
10. Li, J.; Un, K.F.; Yu, W.H.; Mak, P.I.; Martins, R.P. An FPGA-Based Energy-Efficient Reconfigurable Convolutional Neural Network Accelerator for Object Recognition Applications. *IEEE Trans. Circuits Syst. II Express Briefs* **2021**, *68*, 3143–3147. [CrossRef]
11. 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]
12. 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]
13. Wang, Z.; Li, H.; Yue, X.; Meng, L. Briefly Analysis about CNN Accelerator based on FPGA. *Procedia Comput. Sci.* **2022**, *202*, 277–282. [CrossRef]
14. Nguyen, K.; Fookes, C.; Jillela, R.; Sridharan, S.; Ross, A. Long range iris recognition: A survey. *Pattern Recognit.* **2017**, *72*, 123–143. [CrossRef]
15. Ruiz-Beltrán, C.A.; Romero-Garcés, A.; González-García, M.; Marfil, R.; Bandera, A. Real-Time Embedded Eye Image Defocus Estimation for Iris Biometrics. *Sensors* **2023**, *23*, 7491. [CrossRef]
16. Daugman, J. How iris recognition works. *IEEE Trans. Circuits Syst. Video Technol.* **2004**, *14*, 21–30. [CrossRef]
17. Nsaif, A.K.; Ali, S.H.M.; Jassim, K.N.; Nseaf, A.K.; Sulaiman, R.; Al-Qaraghuli, A.; Wahdan, O.; Nayan, N.A. FRCNN-GNB: Cascade Faster R-CNN with Gabor Filters and Naïve Bayes for Enhanced Eye Detection. *IEEE Access* **2021**, *9*, 15708–15719. [CrossRef]
18. Waite, J.; Vincent, J. A probabilistic framework for neural network facial feature location. *Br. Telecom Technol. J.* **1992**, *10*, 20–29.
19. Reinders, M.; Koch, R.; Gerbrands, J. Locating facial features in image sequences using neural networks. In Proceedings of the Second International Conference on Automatic Face and Gesture Recognition, Killington, VT, USA, 14–16 October 1996; pp. 230–235. [CrossRef]
20. Krafka, K.; Khosla, A.; Kellnhofer, P.; Kannan, H.; Bhandarkar, S.; Matusik, W.; Torralba, A. Eye Tracking for Everyone. In Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016; pp. 2176–2184. [CrossRef]
21. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. ImageNet Classification with Deep Convolutional Neural Networks. In Proceedings of the 25th International Conference on Neural Information Processing Systems, Lake Tahoe, NV, USA, 3–6 December 2012; Curran Associates Inc.: Red Hook, NY, USA, 2012; NIPS'12, Volume 1, pp. 1097–1105.

22. Sun, Y.; Wang, X.; Tang, X. Deep Convolutional Network Cascade for Facial Point Detection. In Proceedings of the 2013 IEEE Conference on Computer Vision and Pattern Recognition, Portland, OR, USA, 23–28 June 2013; pp. 3476–3483. [CrossRef]
23. Huang, B.; Chen, R.; Zhou, Q.; Yu, X. Eye landmarks detection via two-level cascaded CNNs with multi-task learning. *Signal Process. Image Commun.* **2018**, *63*, 63–71. [CrossRef]
24. Girshick, R.; Donahue, J.; Darrell, T.; Malik, J. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, USA, 23–28 June 2014; pp. 580–587. [CrossRef]
25. He, K.; Zhang, X.; Ren, S.; Sun, J. Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. *IEEE Trans. Pattern Anal. Mach. Intell.* **2015**, *37*, 1904–1916. [CrossRef]
26. Girshick, R. Fast R-CNN. In Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV), Santiago, Chile, 7–13 December 2015; pp. 1440–1448. [CrossRef]
27. Ren, S.; He, K.; Girshick, R.; Sun, J. Faster R-CNN: Towards real-time object detection with region proposal networks. *Proc. Adv. Neural Inf. Process. Syst.* **2015**, *28*, 91–99. [CrossRef]
28. Sharma, A.; Singh, V.; Rani, A. Implementation of CNN on Zynq based FPGA for Real-time Object Detection. In Proceedings of the 2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT), Kanpur, India, 6–8 July 2019; pp. 1–7. [CrossRef]
29. Redmon, J.; Divvala, S.; Girshick, R.; Farhadi, A. You Only Look Once: Unified, Real-Time Object Detection. In Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016; pp. 779–788. [CrossRef]
30. Liu, W.; Anguelov, D.; Erhan, D.; Szegedy, C.; Reed, S.; Fu, C.Y.; Berg, A.C. SSD: Single shot multibox detector. In Proceedings of the Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, 11–14 October 2016; Proceedings, Part I 14; Springer: Berlin/Heidelberg, Germany, 2016; pp. 21–37.
31. Law, H.; Deng, J. Cornernet: Detecting objects as paired keypoints. In Proceedings of the European Conference on Computer Vision (ECCV), Munich, Germany, 8–14 September 2018; pp. 734–750.
32. 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. Syst.* **2019**, *27*, 1861–1873. [CrossRef]
33. Nakahara, H.; Yonekawa, H.; Fujii, T.; Sato, S. A Lightweight YOLOv2: A Binarized CNN with A Parallel Support Vector Regression for an FPGA. In Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 25–27 February 2018; Association for Computing Machinery: New York, NY, USA, 2018; FPGA'18, pp. 31–40. [CrossRef]
34. Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; Chen, L.C. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In Proceedings of the 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 4510–4520. [CrossRef]
35. Kim, S.; Na, S.; Kong, B.Y.; Choi, J.; Park, I.C. Real-Time SSDLite Object Detection on FPGA. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2021**, *29*, 1192–1205. [CrossRef]
36. Preußer, T.B.; Gambardella, G.; Fraser, N.; Blott, M. Inference of quantized neural networks on heterogeneous all-programmable devices. In Proceedings of the 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, Germany, 19–23 March 2018; pp. 833–838. [CrossRef]
37. Wang, J.; Gu, S. FPGA Implementation of Object Detection Accelerator Based on Vitis-AI. In Proceedings of the 2021 11th International Conference on Information Science and Technology (ICIST), Chengdu, China, 21–23 May 2021; pp. 571–577. [CrossRef]
38. Redmon, J.; Farhadi, A. YOLOv3: An Incremental Improvement. *arXiv* **2018**, arXiv:1804.02767.
39. Li, W.; Zhang, L.; Lv, S. An improved Tiny-YOLOv3 and its implementation with FPGA. In Proceedings of the 2022 IEEE 10th Joint International Information Technology and Artificial Intelligence Conference (ITAIC), Chongqing, China, 17–19 June 2022; Volume 10, pp. 735–739. [CrossRef]
40. Zhang, H.; Jiang, J.; Fu, Y.; Chang, Y. Yolov3-tiny Object Detection SoC Based on FPGA Platform. In Proceedings of the 2021 6th International Conference on Integrated Circuits and Microsystems (ICICM), Nanjing, China, 22–24 October 2021; pp. 291–294. [CrossRef]
41. Oh, S.; You, J.H.; Kim, Y.K. Implementation of Compressed YOLOv3-tiny on FPGA-SoC. In Proceedings of the 2020 IEEE International Conference on Consumer Electronics—Asia (ICCE-Asia), Seoul, Republic of Korea, 1–3 November 2020; pp. 1–4. [CrossRef]
42. Velicheti, P.; Pentapati, S.; Purini, S. Systolic Array based FPGA accelerator for Yolov3-tiny. In Proceedings of the 2022 IEEE High Performance Extreme Computing Conference (HPEC), Waltham, MA, USA, 19–23 September 2022; pp. 1–2. [CrossRef]
43. Esen, F.; Degirmenci, A.; Karal, O. Implementation of the Object Detection Algorithm (YOLOV3) on FPGA. In Proceedings of the 2021 Innovations in Intelligent Systems and Applications Conference (ASYU), Elazig, Turkey, 6–8 October 2021; pp. 1–6. [CrossRef]
44. Ma, Y.; Zheng, T.; Cao, Y.; Vrudhula, S.; Seo, J.s. Algorithm-Hardware Co-Design of Single Shot Detector for Fast Object Detection on FPGAs. In Proceedings of the 2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), San Diego, CA, USA, 5–8 November 2018; pp. 1–8. [CrossRef]

45. Ahmad, A.; Pasha, M.A.; Raza, G.J. Accelerating Tiny YOLOv3 using FPGA-Based Hardware/Software Co-Design. In Proceedings of the 2020 IEEE International Symposium on Circuits and Systems (ISCAS), Seville, Spain, 12–14 October 2020; pp. 1–5. [CrossRef]
46. Suh, H.s.; Meng, J.; Nguyen, T.; Venkataramanaiah, S.K.; Kumar, V.; Cao, Y.; Seo, J.s. Algorithm-Hardware Co-Optimization for Energy-Efficient Drone Detection on Resource-Constrained FPGA. In Proceedings of the 2021 International Conference on Field-Programmable Technology (ICFPT), Auckland, New Zealand, 6–10 December 2021; pp. 1–9. [CrossRef]
47. Cong, J.; Fang, Z.; Lo, M.; Wang, H.; Xu, J.; Zhang, S. Understanding Performance Differences of FPGAs and GPUs: (Abtract Only). In Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 25–27 February 2018; Association for Computing Machinery: New York, NY, USA, 2018; FPGA'18, p. 288. [CrossRef]
48. Qasaimeh, M.; Denolf, K.; Lo, J.; Vissers, K.; Zambreno, J.; Jones, P.H. Comparing Energy Efficiency of CPU, GPU and FPGA Implementations for Vision Kernels. In Proceedings of the 2019 IEEE International Conference on Embedded Software and Systems (ICESS), Las Vegas, NV, USA, 2–3 June 2019; pp. 1–8. [CrossRef]
49. Viola, P.; Jones, M. Rapid object detection using a boosted cascade of simple features. In Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001, Kauai, HI, USA, 8–14 December 2001; Volume 1, p. I. [CrossRef]
50. Ruiz-Beltrán, C.A.; Romero-Garcés, A.; González, M.; Pedraza, A.S.; Rodríguez-Fernández, J.A.; Bandera, A. Real-time embedded eye detection system. *Expert Syst. Appl.* **2022**, *194*, 116505. [CrossRef]
51. Liu, W.; Tan, K. Face Landmark Detection Based on Deep Learning Processor Unit on ZYNQ MPSoC. In Proceedings of the 2022 7th International Conference on Intelligent Computing and Signal Processing (ICSP), Xi'an, China, 15–17 April 2022; pp. 415–419. [CrossRef]
52. Zhang, X.; Wang, J.; Zhu, C.; Lin, Y.; Xiong, J.; Hwu, W.m.; Chen, D. DNNBuilder: An Automated Tool for Building High-Performance DNN Hardware Accelerators for FPGAs. In Proceedings of the 2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), San Diego, CA, USA, 5–8 November 2018; pp. 1–8. [CrossRef]
53. Redmon, J. Darknet: Open Source Neural Networks in C. 2013–2016. Available online: http://pjreddie.com/darknet/ (accessed on 16 November 2023).
54. Maya-Martínez, S.; Argüelles-Cruz, A.; Guzmán-Zavaleta, Z.; Ramírez-Cadena, M. Pedestrian detection model based on Tiny-Yolov3 architecture for wearable devices to visually impaired assistance. *Front. Robot AI* **2023**, *10*, 1052509. [CrossRef]
55. Karras, T.; Laine, S.; Aila, T. A Style-Based Generator Architecture for Generative Adversarial Networks. *arXiv* **2018**, arXiv:1812.04948.
56. Uhl, A.; Wild, P. Combining Face with Face-Part Detectors under Gaussian Assumption. In Proceedings of the Image Analysis and Recognition, Aveiro, Portugal, 25–27 June 2012; Campilho, A., Kamel, M., Eds.; Springer: Berlin/Heidelberg, Germany, 2012; pp. 80–89.
57. Lienhart, R.; Liang, L.; Kuranov, A. A detector tree of boosted classifiers for real-time object detection and tracking. In Proceedings of the 2003 International Conference on Multimedia and Expo. ICME'03, Baltimore, MD, USA, 6–9 July 2003; Proceedings (Cat. No. 03TH8698), Volume 2, pp. 277–280. [CrossRef]
58. Romero-Garcés, A.; Ruiz-Beltrán, C.; Marfil, R.; Bandera, A. Lightweight Cosmetic Contact Lens Detection System for Iris Recognition at a Distance. In Proceedings of the 18th International Conference on Soft Computing Models in Industrial and Environmental Applications (SOCO 2023), Salamanca, Spain, 5–7 September 2023; García Bringas, P., Pérez García, H., Martínez de Pisón, F.J., Martínez Álvarez, F., Troncoso Lora, A., Herrero, Á., Calvo Rolle, J.L., Quintián, H., Corchado, E., Eds.; Springer: Cham, Switzerland , 2023; pp. 246–255.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.