agriculture

MDPI

Article

FPGA Implementation of a Convolutional Neural Network and Its Application for Pollen Detection upon Entrance to the Beehive

Tomyslav Sledevič, Artūras Serackis and Darius Plonis *

Department of Electronic Systems, Vilnius Gediminas Technical University, Saulėtekio Ave. 11, LT-10223 Vilnius, Lithuania

* Correspondence: darius.plonis@vilniustech.lt

Abstract: The condition of a bee colony can be predicted by monitoring bees upon hive entrance. The presence of pollen grains gives beekeepers significant information about the well-being of the bee colony in a non-invasive way. This paper presents a field-programmable-gate-array (FPGA)-based pollen detector from images obtained at the hive entrance. The image dataset was acquired at native entrance ramps from six different hives. To evaluate and demonstrate the performance of the system, various densities of convolutional neural networks (CNNs) were trained and tested to find those suitable for pollen grain detection at the chosen image resolution. We propose a new CNN accelerator architecture that places a pre-trained CNN on an SoC FPGA. The CNN accelerator was implemented on a cost-optimized Z-7020 FPGA with 16-bit fixed-point operations. The kernel binarization and merging with the batch normalization layer were applied to reduce the number of DSPs in the multi-channel convolutional core. The estimated average performance was 32 GOPS for a single convolutional core. We found that the CNN with four convolutional and two dense layers gave a 92% classification accuracy, and it matched those declared for state-of-the-art methods. It took 8.8 ms to classify a 512 × 128 px frame and 2.4 ms for a 256 × 64 px frame. The frame rate of the proposed method outperformed the speed of known pollen detectors. The developed pollen detector is cost effective and can be used as a real-time image classification module for hive status monitoring.

Keywords: convolutional neural network (CNN); field-programmable gate array (FPGA); pollen detection

check for updates

Citation: Sledevič, T.; Serackis, A.; Plonis, D. FPGA Implementation of a Convolutional Neural Network and Its Application for Pollen Detection upon Entrance to the Beehive. *Agriculture* **2022**, *12*, 1849. https://doi.org/10.3390/agriculture12111849

Academic Editor: Jiangbo Li

Received: 12 October 2022
Accepted: 2 November 2022
Published: 4 November 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2022 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

Convolutional neural networks (CNNs) have become wide spread in the last decade. They are most commonly used to solve machine vision tasks in applied science for electronic component classification [1], vehicle identification under rain conditions [2], in agricultural engineering for leaf disease classification [3], efficient beekeeping [4,5], or tree identification from unmanned aerial vehicles [6]. Currently, many workarounds have been proposed to speed-up the training stage and image classification procedure. This includes network training with normalized batches of data [7], the binarization of convolutional kernels [8–10], network compression using adaptive quantization schemes [11–14], or reducing the number of operations in the CNN model with neuromorphic techniques [15,16].

The primary application of CNNs is in various deep learning tasks related to image recognition [17]. The practical objective of embedded CNN implementations is an instant classification or object detection in 2D images where real-time decision-making is required [18]. FPGA-based CNN accelerators are used in various fields, e.g., manufacturing defect inspection [19], object detection in a video stream [20], real-time facial emotion recognition [21], or even embedded gesture classification [22]. Some studies propose a lightweight versions of the CNN architecture with optimized memory usage and computational complexity to solve a specific object identification task on FPGAs [19,21,22].

Tracking the condition of the beehive is a duty for the beekeepers to keep the bee colony healthy. The pollen foraging efficiency provides essential information for the behavioural research on honey bees. CNN-based image analysis methods are often applied for bee tracking and pollen detection in images [5]. Almost always, the dataset of bee images is collected using a contrast background (blue, green, or black) to improve the bee detectability [23–27]. In a related state-of-the-art imaging system, the pollen detection accuracy varied in the range of 89–99% and basically depended on the applied segmentation, classification methods, and dataset.

Ngo et al. [28] implemented an automated honey bee activity monitoring system. It was based on an observation box attached the hive with an integrated webcam for frame collection. A Kalman filter and the Hungarian algorithm were applied for tracking multiple bees. Recently, the authors extended their study with CNN-based bee detection [23]. A tiny YOLOv3 model was trained for the detection of multiple bees with pollen grains and non-pollen. A real-time 25 fps image processing speed was achieved on a GPU Jetson TX2 embedded system with a 94% classification accuracy. The training and testing datasets contained 3000 and 500 images, respectively.

Babic et al. [25] and Stojnic et al. [26] implemented pollen-bearing honey bee detection algorithms to classify bees upon hive entrance. The proposed approaches start from image segmentation. Then, the SIFT and VLAD descriptors are used for feature extraction on the segmented images. The classification was performed using the support vector machine (SVM) classifier. This achieved an 89% classification accuracy with 100 training images and took nearly 1 fps running the algorithm on a Raspberry Pi [25]. A training dataset of 800 images yielded a 92% classification accuracy and nearly a 6 fps rate on segmentation [26].

Yand and Collins [27] used faster RCNN with the VGG16 core network to detect pollen grains on individual bee images. The bee detection model was then combined with a bee tracking model based on the Kalman filter, so that each flying bee tracked in the successive video frames was identified as carrying pollen or not. The authors trained the network on 1000 images and obtained a 94% classification accuracy on 400 test images. The maximal achieved frame rate was not declared.

Rodrigues et al. [24] applied the VGG16, VGG19, ResNet50, and shallow CNN architectures for pollen grain bee recognition in images. The authors achieved a maximal classification accuracy of 96% on the proposed dataset (710 pollen and non-pollen bee images in total) of cropped single bee-centred images. Monteiro et al. [29] used the same dataset to train several known CNN models. The classification accuracy varied in the range 88–99%. The highest score was obtained with the DarkNet53 network. However, the relative distribution of the training/test images in the dataset is unknown.

This work partially continues our previous investigation of pollen grain detection in 100×100 px cropped and bee-centred images [30]. In this work, we collected the dataset of images captured above six beehive entrances with native ramps without any modifications. Images were labelled in two classes: with and without pollen grains. We investigated several configurations of a shallow CNN with different image resolutions and analysed the speed and accuracy of the proposed CNN accelerator implemented in a cost-optimized SoC FPGA.

The main idea to start this work was the implementation of a tool for CNN deployment into embedded systems that enables us to place a pre-trained network into low-cost edge devices. In general, the complexity of the CNN configuration varies depending on the requirement for its application. Therefore, the manual implementation of various densities of CNNs on edge devices is a time-consuming process, especially when CNNs with different configurations need to be implemented and evaluated repeatedly on system on chip (SoC) devices based on the processing system (PS) and programmable logic (PL). To reduce the time for the design and coding procedure of an FPGA circuit, it is preferably to have a reconfigurable core for the CNN. All the settings for that CNN core can be transferred from the PC to ARM and then from the ARM processing subsystem to programmable logic without reprogramming the FPGA, but only updating the parameters in the CNN's core.

Usually, recompiling a new program file for FPGA takes up to a few hours; therefore, it is preferable to have a ready convolutional kernel housed in the FPGA and reconfigure it from the processing subsystem by submitting new weight arrays for the convolutional kernels and dense layers.

In this paper, we present the hardware design steps of a convolutional neural network and its implementation on the ZynQ SoC FPGA. In the current research, the trained CNN was applied for pollen grain detection. In the next section, the conceptual architecture of the pollen grain detector is presented. In Section 3, the hardware implementation of the pollen grain detector is presented with the emphasis on the hardware design of the CNN. The performance of the FPGA-based CNN accelerator is investigated in Section 4. The image classification accuracy and speed are compared with known state-of-the-art methods.

To the best of our knowledge, this work presents a pollen detector that is the first one based on an FPGA platform. The known state-of-the-art methods use an images with a dark background to improve bee detectability. However, we collected a dataset of images from several beehives with native entrances and trained several CNNs to find a suitable one dependent on the image resolution. The top CNN structure was experimentally determined by tuning the number of layers, the kernels in the convolutional layers, and the number of neurons in the dense layer. We found a suitable CNN structure with the minimal number of layers for pollen detection. Then, the performance of the proposed system was checked. We evaluated the dependencies of the CNN's speed (time per frame) on its complexity. Instead of the manual implementation of each trained CNN in the FPGA by coding it in a hardware description language, we propose a hardware–software architecture for low-cost embedded systems, which allows users to place the customized configuration of the trained CNN to an SoC FPGA device without recompilation of the programming file, but uploading a new list of instructions and weights from the PC to a board.

2. Architecture of Pollen Grain Detector and Dataset Collection

The pollen grain detector consisted of an FPGA board, camera, and personal computer (Figure 1). For concept validation, the images were transferred from the computer to the FPGA, and then, the classification results from the board went back to the computer. This approach enabled us to check the presence of pollen grains on a larger dataset of images in a shorter time than capturing images from a camera above the beehive. The pollen detector can be applied as a real-time visual classifier when the image source is switched to a camera input on an FPGA board. With the camera interface, the FPGA reads the stream of frames and stores it to RAM. In a debugging stage when processing the image dataset, the responses from the CNN output are passed back to the computer. When the pollen detector is used as a real-time classifier of images captured with a camera, then the results of pollen presence in the frames are periodically stored in the QSPI flash memory chip on the FPGA board. If the PC is connected to the board, then the classification results are transferred through the serial port in real-time. All the results logged in the flash memory can be transferred to the PC through the Ethernet port for further analysis.

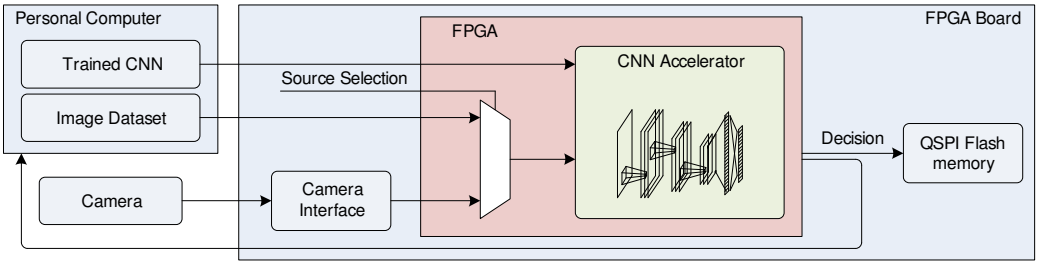


Figure 1. Conceptual illustration of pollen grain detection in images.

The training dataset was collected from 6 beehives with native ramps upon entrance to the beehive; see Figure 2a–f. The raw images were captured at a 3 fps rate 0.4 m above of the ramp with 1920 × 1080 px resolution. Yellow rectangles in Figure 2 mark a region

of interest (ROI) where we expect to detect a bee with pollen grains. The size of the ROI in the raw image was 1024 × 256 px. It was the largest resolution of the images in the input to the CNN that we tested during the experiments. All the images were labelled into two classes: images with (class I) and without (class II) pollen grains in the ROI. If at least one bee appears with pollen grains in the ROI, then that image is assigned to class I. If there are no grains in an image, then it is labelled as class II. The final dataset contained 1024 × 256 px resolution RGB images: 6000 in class I and 6000 in class II (2000 per beehive). The bees with pollen grains are pointed out with red arrows in Figure 2.

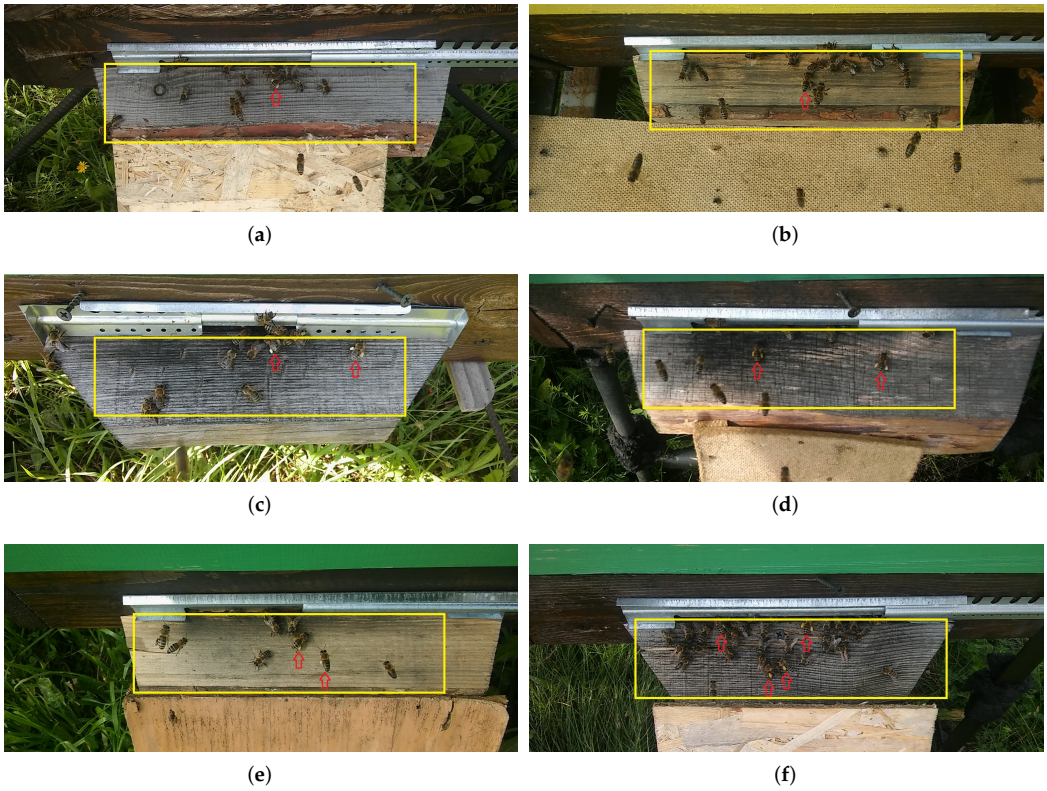


Figure 2. Bees with pollen grains (one in (**a**,**b**); two in (**c**–**e**); four in (**f**)) in the region of interest (yellow rectangles 1024 × 256 px in raw images) upon entrance to six different beehives.

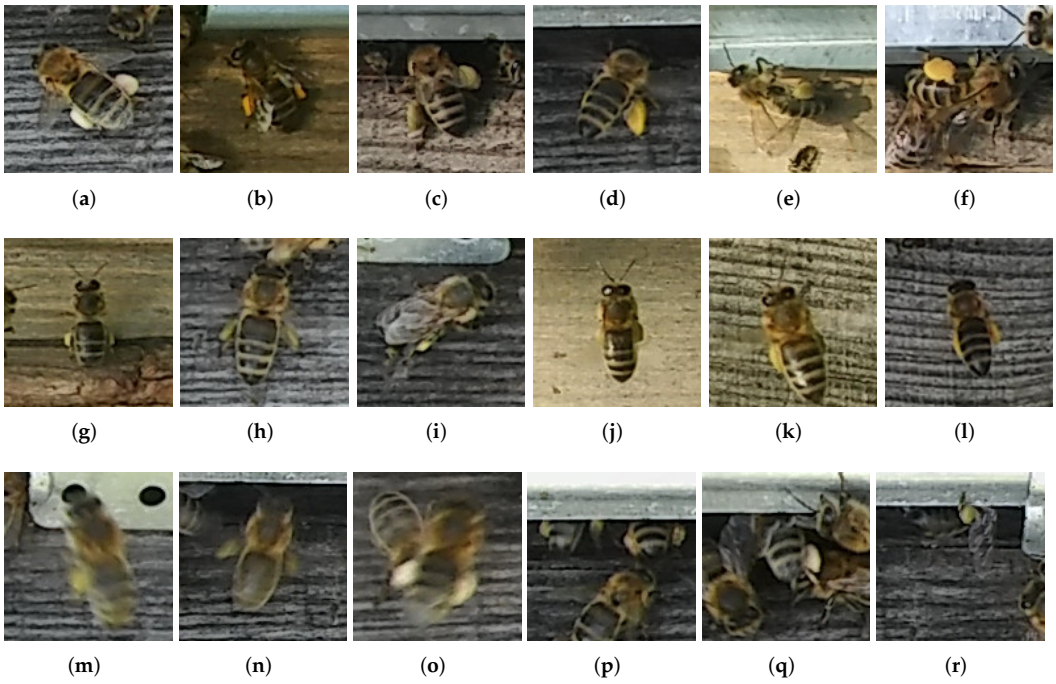
In the experimentation stage, we needed to evaluate the influence of the image resolution on classification accuracy. The lower the image resolution, the faster the classifier will work. However, the image resolution needs to be sufficient to keep the details of the pollen grains, especially knowing that the shape of the grains is not uniform. Figure 3a presents the quality of the image of the bee with pollen grains in a 1024 × 256 px raw image. The next three images (Figure 3b–d) were downsampled 2, 4, and 8 times, respectively.

During image labelling, we noticed several attributes regarding visually recognizable pollen grains. All the observed grains came in shades of yellow and white; this really depends on where the bees have foraged. In cases when the bee landed on the ramp, then the captured images contained clearly visible grains on both hind legs, as shown in Figure 4a–c. In some cases, only one grain was visible to the camera due to partial overlap with another bee, or the grain was obscured by others, or the bee only had one pollen grain (Figure 4d–f). Some times, bees appeared with tiny grains, as shown in Figure 4g–i. Flying bees kept the grains closer to their body(Figure 4j–l). A blurred bee appeared when it moved fast, as shown in Figure 4m–o. In the case when the ROI overlapped with the gate to the beehive, then some beeswere captured partially (Figure 4p–r). All the images with the above-mentioned cases were assigned to class I. If there was no pollen grain in an image, then it was assigned class II.



(**a**) (**b**) (**c**) (**d**)

Figure 3. Pollen identifiability in raw image (**a**) and after image downsampling by ×2 (**b**), ×4 (**c**), and ×8 (**d**) on the input to the CNN and keeping the same ROI as in the raw image (yellow rectangles in Figure 2).



(**a**) (**b**) (**c**) (**d**) (**e**) (**f**)

(**g**) (**h**) (**i**) (**j**) (**k**) (**l**)

(**m**) (**n**) (**o**) (**p**) (**q**) (**r**)

Figure 4. Bees at the entrance of the beehive: normal case with a grain pair (**a**–**c**), a single pollen grain (**d**–**f**), tiny grains (**g**–**i**), nestled grains (**j**–**l**), blurred grains (**m**–**o**), and partially obscured by the gate (**p**–**r**).

The image dataset was labelled manually by the authors. First, we cropped a 1024×256 px ROI image form a raw 1920 × 1080 px frame. Second, we labelled the dataset into two classes by saving the images in different folders and repeating that procedure for the six beehives and keeping the dataset separate for each of the six beehives. We used 80% of the images to train the CNNs and the remaining 20% to test the accuracy. To test accuracy on the full dataset, 4800 images were used to train the CNNs and 1200 to test the classifier. To test the accuracy on a single beehive dataset, 800 images were used for training and 200 for testing. The training procedure used the stochastic gradient descent algorithm with 0.9 momentum and a mini batch size of 64 images. The CNNs were trained until they reached the minimal loss point on the training curve (usually in an hour for a CNN with 4 convolutional layers and 16 kernels in each layer). Training was performed in Matlab R2020b on an RTX2060 GPU.

3. Hardware Implementation of the Convolutional Neural Network

In this section, we present the hardware implementation of the image classification core that was applied to pollen detection. The FPGA implementation of the CNN core was presented recently in detail [31]. In the current work, we extended the description of the hardware implementation of the feed-forward part with emphasis on the convolutional

core. The CNN was trained with the Neural Network Toolbox in Matlab, and then, the parameters of the CNN were transferred to the external memory on the FPGA board. The CNN accelerator was implemented on the FPGA as the configurable IP core. The CNN accelerator requires a master that handles core reconfiguration and data feeding. We used a ZynQ platform to implement a master on the ARM processor and a slave core on the FPGA. From a top-level architectural view, the ARM works as a bridge and data-flow controller between the personal computer, hardware accelerator, and on-board memory. In the current version of the CNN accelerator, the ARM processor was intended to serve the hardware core with input images, the CNN parameters, and a stream of feature maps in either the FPGA and RAM directions.

3.1. Translation of the CNN Model to the SoC FPGA

The model of the CNN together with the training procedure was implemented on the PC and then mapped to the SoC FPGA. The translation steps of the CNN model to the SoC FPGA were as follows:

- High-level description of the CNN model;
- Kernel binarization and training of the CNN;
- Conversion to fixed-point precision;
- Restructuring of the parameters in the convolutional and dense layers;
- Memory allocation;
- Instruction generation.

The current model of the CNN was restricted to a set of layers: convolutional, batch normalization, activation, max pooling, and dense. All the convolutional kernels were limited to a size of 3×3 pixels. The stride of all convolutional kernels was fixed to 1×1 pixels. The size of the max pooling kernel was fixed to 2×2 pixels. Up to 4096 neurons were allowed in each dense layer. The total number of dense layers was limited to 3. During a high-level description of the CNN model, the standard *Matlab* syntax can be used for the initialization of the CNN layers. The main configurable parameters were: the resolution of the input image, the number of kernels in the convolutional layers, the number of convolutional layers, the number of dense layers, and the number of neurons in each dense layer.

The binarization of the convolutional kernels and their merging with batch normalization were presented recently in detail [32]. The binarization of the convolutional kernel was joined with the training of the CNN. To train the pseudo-binary kernels, we applied a double training on the initialized layers of the CNN. The training–binarization–training procedure allowed us to use the standard Matlab training function and obtain pseudo-binary kernels in the final trained model. The first stage of training used double-precision floating-point numbers. The product of the input feature map with the convolutional kernel needed a multiplication operation to be applied 9 times for the 3×3 size kernel. This yielded 9 DSP operations per kernel. Therefore, binary kernels are usually applied to reduce the demand of the DSP blocks [8,9]. After the first training of the CNN came the binarization of all the kernels in all convolutional layers. The parameters in a 3×3 size w kernel are approximated by:

$$\begin{bmatrix} w(0,0) & w(0,1) & w(0,2) \\ w(1,0) & w(1,1) & w(1,2) \\ w(2,0) & w(2,1) & w(2,2) \end{bmatrix} \approx A \begin{bmatrix} B(0,0) & B(0,1) & B(0,2) \\ B(1,0) & B(1,1) & B(1,2) \\ B(2,0) & B(2,1) & B(2,2) \end{bmatrix}; \quad (1)$$

here, $i \in [0,1,2]$ and $j \in [0,1,2]$ are the vertical and horizontal indices in the convolutional kernel; A is a positive scaling factor estimated as the mean of the absolute weights w in a kernel:

$$A = \frac{1}{9}||w||. \quad (2)$$

$B(i,j)$ is an approximated binary kernel, which takes the sign of the weight array $w(i,j)$ [8]:

$$B(i,j) = \text{sign}(w(i,j)); \tag{3}$$

here, $B(i,j) = 1$ if $w(i,j) \geq 0$ and $B(i,j) = -1$ if $w(i,j) < 0$.

According to (1)–(3), the kernel weights $w^{i,j}$ are pseudo binarized as follows:

$$\begin{bmatrix} -0.1 & 0 & 0.3 \\ 0.3 & -0.2 & 0.4 \\ 0.2 & 0.3 & -0.3 \end{bmatrix} \approx 0.233 \begin{bmatrix} -1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & -1 \end{bmatrix}. \tag{4}$$

Before the second training, all the kernels in the convolutional layers were replaced according to (5) applying the binary kernel and then multiplication with scaling factor A. The approximated kernels consequently brought in an error to the convolutional product. Therefore, the second call of the network training function was required to adapt the parameters in batch normalization and the dense layers and compensate the impact of the pseudo binarization on the accuracy of the CNN. The convolutional kernels were not adapted during the second training stage by setting the learning rate factors to zero for the weights and biases in all already binarized convolutional layers. We call the process pseudo binarization because it does not give a pure binary kernel. However, the response from the binary kernel was scaled further by a factor of A—the mean of the absolute weights to keep the amplitude of the kernel response on the same level as before binarization, like filtering with the primary floating-point kernel that was obtained after the first training stage.

The proposed CNN core operates with 16-bit signals, and therefore, the trained model needed to be converted to fixed-point precision. While solving the issues related to the saturation and rounding of the signals, we tuned the number of bits given for the integer and fractional parts of the parameters not exceeding the 16-bit limit. To efficiently utilize the throughput of an Ethernet connection and the on-board FPGA RAM, the compression of two neighbour parameters was applied to merge them into one 32-bit pair as high 16 bits and low 16 bits. The parameters of the convolutional and dense layers had a specific format and order used for fast CNN core configuration and data loading from the DDR RAM. Therefore, the restructuring procedure was applied separately to the lists of parameters in the convolutional and dense layers.

To map a trained CNN model to the SoC FPGA, we employed a Python-written converter, which generates a list of instructions for the CNN accelerator and uploads it to the FPGA board. First, the converter takes a set of $*.csv$ files where the CNN parameters were exported from Matlab and converts them to a CNN-core-supportable format. Next, the converter allocates the memory addresses in an external RAM, then generates a set of instructions for the CNN core according to the CNN architecture defined in Matlab. Finally, the converter schedules the instructions and transfers them to the processing system, which uploads the instructions to the on-board RAM.

3.2. Top-Level Architecture

The top-level architectural diagram of the proposed CNN accelerator is presented in Figure 5. It is divided into two main functional blocks. The first block is the PS. The main purpose of the PS is the management of the data exchange between the personal computer (PC), external memory, and hardware accelerator. The second block is the programmable logic. It contains the hardware implementation of the CNN accelerator.

The images to be classified together with the configuration parameters of the CNN are transmitted from the PC to the PS through an Ethernet connection. The PS runs the IP echo server, receives the data from the PC, and places the images, instructions, and parameters of the CNN in the external memory. The convolution and dense cores are located on the PL. The configurations for the convolutional core are streamed to the core configuration memory before the start of the image processing on the CNN accelerator. The core configuration memory contains the values of the kernels and biases. The CNN

instructions set the convolutional core to work in the desired mode and contain information about the quantity and location of the data to be transferred between the memory and PL.

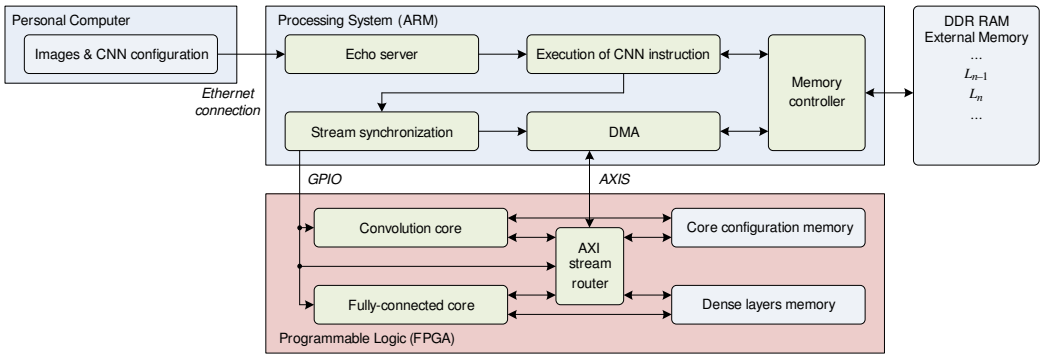


Figure 5. Top-level architecture of the CNN accelerator.

The application program on the PS manages all the GPIO and AXI data streams between the PL and external DDR RAM. The begin of the CNN execution is initialized by a start command received from the PC. Then, the PS configures four synchronous DMA streams through the high-performance (HP) ports for the data exchange between the external memory and CNN accelerator (Figure 6). The PS initializes two DMA streams from the PS to the PL to transfer the multi-channel data to the inputs of the convolutional core and two DMA steams to store the multi-channel responses from the convolutional core to the external memory. The AXI stream router manages the DMA transfers on the PL between the external memory and convolution, the dense cores, as well as the on-chip block-RAM-based core configuration and dense layer memories.

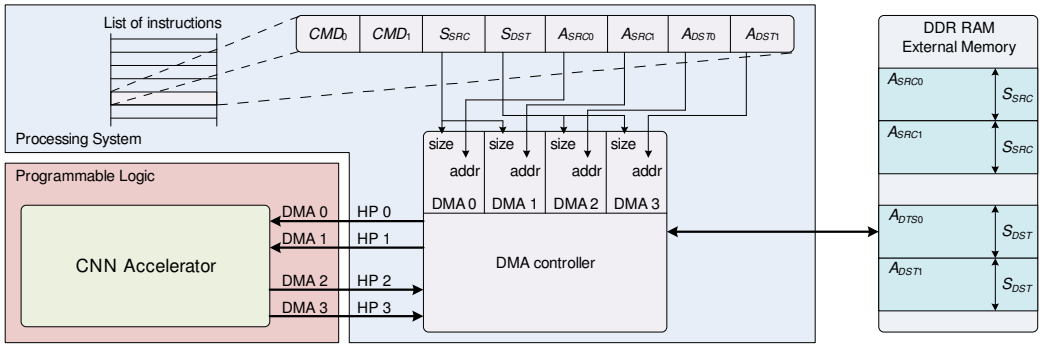


Figure 6. Execution of the instruction on the PS and data exchange between the external memory and CNN accelerator on the PL.

In the case when the convolution is requested, the weights are delivered from the core configuration memory to the kernels in a convolutional core. The input images are streamed directly from the external memory to the core. The responses from the first convolutional layer are streamed back to the memory. To compute each next convolutional layer L_n, the outputs of the previous layer L_{n-1} are transferred from the L_{n-1} location in the external memory to the convolutional core, and the responses are streamed back to the L_n location. In the case when the computation of the dense layer is requested, the outputs of the last convolutional core are transferred to the on-chip dense layer memory, and the responses from each next dense layer are stored in the on-chip block RAM. The weights of the neural synapses are streamed directly from the external memory to the dense layer core. Only the responses from the last dense layer are streamed to the DDR RAM and then to the PC as the output of the final CNN layer.

The program on the PS reads instruction by instruction from a list and adjusts the size and address values for the DMA controller according to the currently requested size of the source S_{SRC} and destination S_{DST} streams (Figure 6). The DMA controller knows from

where it reads an image or feature map according to the source A_{SRC0}, A_{SRC1} addresses. A feature map currently computed by a convolutional core or responses from the dense layer are stored to the memory at the destination addresses A_{DST0}, A_{DST1}. With a single address, the processor accesses 32 bits of data in the external memory. The CNN accelerator uses 16-bit precision per parameter. Therefore, two features from two neighbour channels are stored at the same address (Figure 7). The DMA controller works in a 100 MHz clock domain. It has two DMA channels to the DDR memory (S2MM—stream to memory map) and two channels to the CNN accelerator (MM2S—memory map to stream). The convolutional core works in a 50 MHz clock domain and receives an input stream of 8 feature maps. Synchronous streaming of 100 MHz $\times$ 2 DMA $\times$ 32-bit data is interleaved to 50 MHz $\times$ 8 channels $\times$ 16 bits of features and delivered to the input of the convolutional core. That core generates an output stream of 8 feature maps at 50 MHz, which is composed back to 2 synchronous DMA streams at 100 MHz.

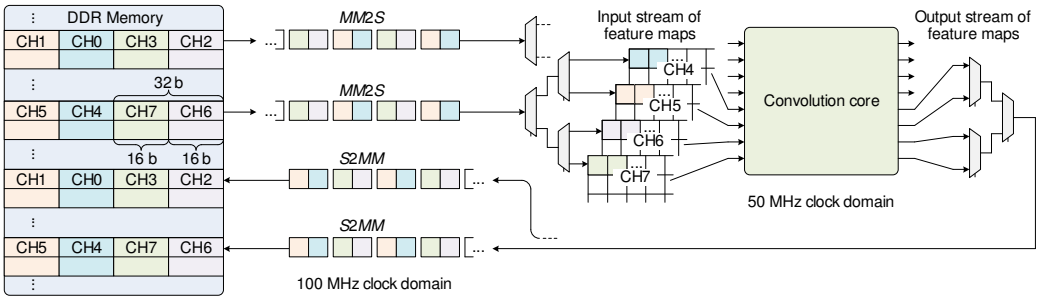


Figure 7. The stream of the input and output feature maps to/from the convolutional core on the PL.

3.3. Convolutional Core

The convolutional core contains 64 binary kernels that form a computational unit with 8 input and 8 output channels. Every input channel has 2 line buffers and a 3 $\times$ 3 sliding window through which the binary kernels access the pixels of the input feature maps (Figure 8). Before the processing of the stream of feature maps, the binary kernels receive a weight vectors from the core configuration memory. The binary weights toggle the multiplexers, which pass the direct value of a pixel in the feature map or negate that pixel. The binary kernel may be set to execute the convolution or work like a channel adder.

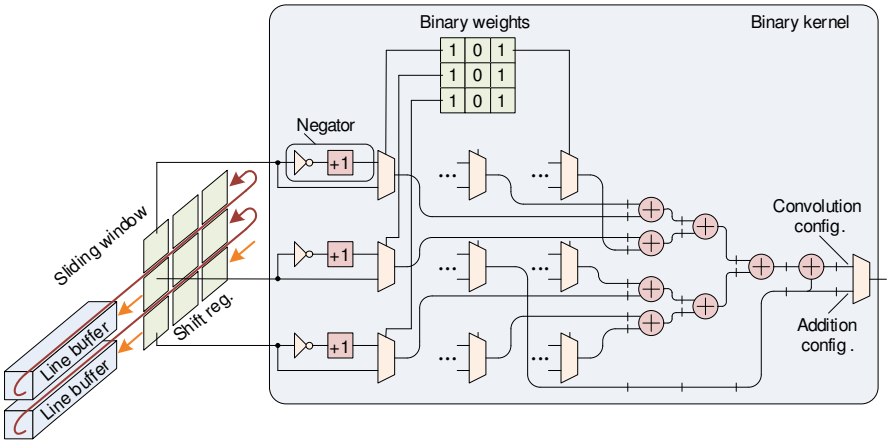


Figure 8. The binary kernel with a 3 $\times$ 3 sliding window for convolution/addition core configuration.

The product of the binary kernel goes to the modified batch normalization block (Figure 9). The parameter $K_c = kA_c$ contains a scaling factor A_c of the c^{th} kernel (2), and k includes all the parameters related to batch normalization [31]. Parameter b adds an offset on the output channel of the binary kernel. The batch normalized product goes to the ReLU unit and then to the 2 $\times$ 2 max polling block (Figure 10). If the core is configured to summation, then it skips the ReLU and max polling units. If the convolutional layer is not

followed by ReLU or max polling, then the batch-normalized product goes to the output of the CNN core.

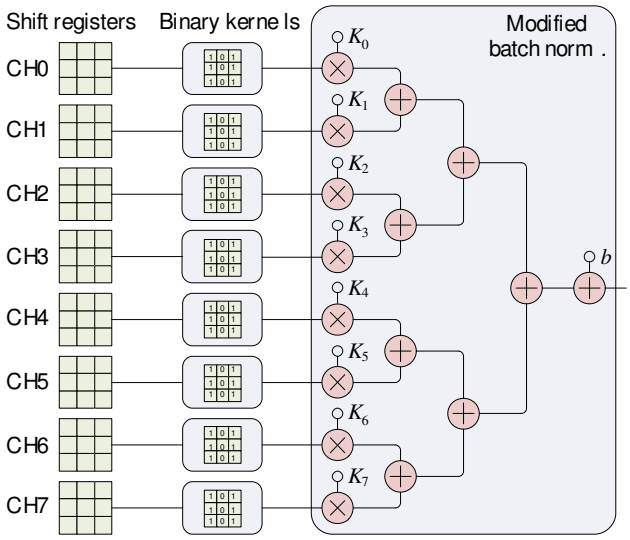


Figure 9. The 8-channel binary convolution and modified batch normalization block.

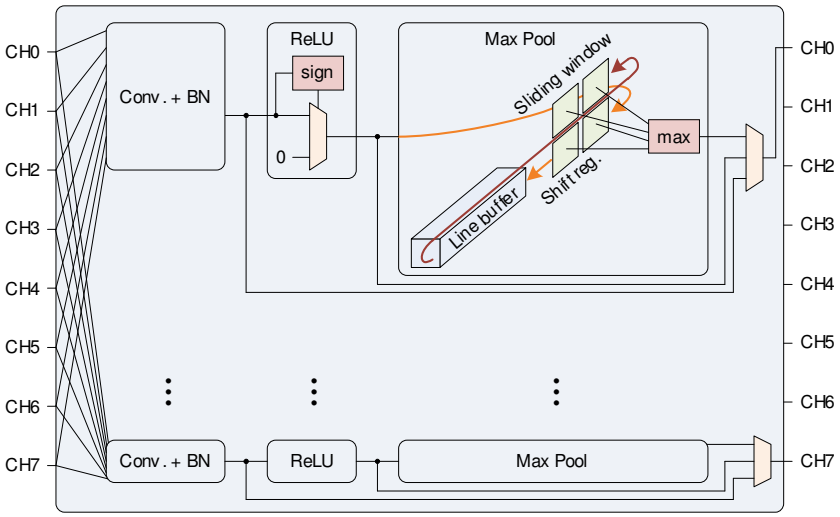


Figure 10. A core with 8 input and 8 output channels for convolution, batch normalization, ReLU, and max pooling computation.

In most cases, when the number of kernels in a convolutional layer is larger than 8, then the convolutional core needs to be accessed multiple times. Figure 11 presents an example of a convolutional layer that receives 24 feature maps from the previous layer and forms 8-channel feature maps for the next layer. For every extra 8 input channels, there will be scheduled one convolution instruction and two summation instructions. For every extra 8 output channels, the core configuration plan (Figure 11) remains the same; only a new setting from the core configuration memory will be loaded to the convolutional core, and a new list of instructions will be executed with new source/destination addresses. One record in a list of instructions (Figure 6) corresponds to one convolutional core access. The indices under the blocks in Figure 11 mark the order of instruction execution. The sequence of instructions for a layer with more than 16 feature maps on the input is as follows. At the beginning, there are two convolution and two addition operations, then repeatedly a single convolution with two additions, and so on. If instead of 8 feature maps on the output, only 4 maps need to be formed, then the number of addition cores will be reduced twice. If the number of input feature maps is 17 (instead of 24), then the structure of the convolutional

layer remains same, as shown in Figure 11. The unused input channels will obtain zeros, and the execution time will be still the same as having 24 feature maps. The design of the dense layers was presented in details in a recent research work [31].

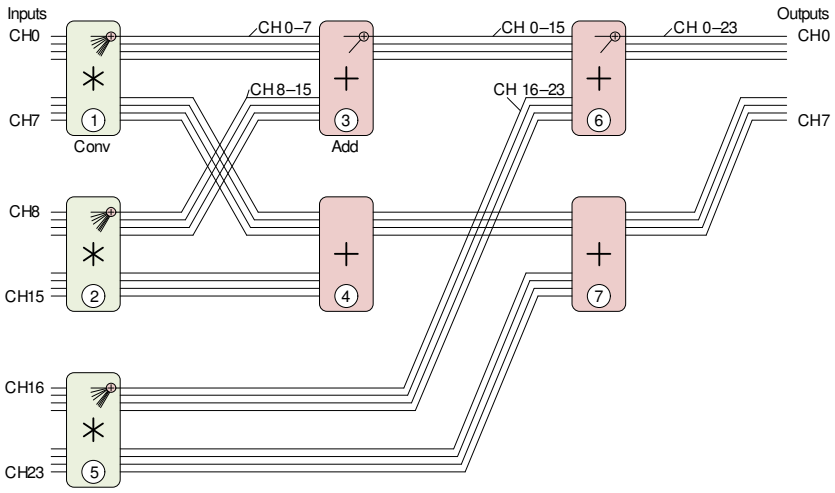


Figure 11. Core reconfiguration plan to process 24 inputs (channels CH0 to CH23) and obtain 8 output channels (CH0 to CH7). Green block—convolutional core, red block—same convolutional core configured for the sum operation.

4. Experimental Evaluation and Results

To validate the proposed FPGA-based CNN accelerator for pollen grain detection, we used the ZynQ SoC XC7Z020 chip on a ZedBoard [33] for the performance assessment. The CNN accelerator was implemented in VHDL, and the software code for the ARM processor was written in the C language using the Xilinx Vivado and SDK design tools, respectively. The post-implementation report (Table 1) showed that 67% of the logic resources were utilised and 34% of the on-chip memory was utilised. The convolutional and dense layer cores utilised 64 and 4 DSPs, respectively, and that yielded 31% of all on-chip DSPs.

Table 1. Post-implementation report of the CNN accelerator on the Z-7020 FPGA.

Resource	Available	Utilisation	Utilisation (%)
LUT	53,200	35,521	67
Flip-Flops	106,400	43,405	41
4kB BRAM	140	48	34
DSP	220	68	31

Table 2 provides a summary of the proposed CNN implementation in comparison with other CNN accelerators on the same Z-7020 device. The compared implementations come with 8- and 16 bit fixed-point precision. The convolutional kernels in our proposed implementation were computed on a relatively low 50 MHz clock in comparison to the others. The 50 MHz frequency for the convolutional core was selected purposely to process two synchronous DMA streams of feature maps on the input to the core and generate two DMA streams from the core to the memory with a 100% interface utilisation rate in both directions (Figure 7). The convolutional GOPS of the proposed implementation was similar to DnnWeaver [34], but lower than fpgaConvNet [35] and Angel-Eye [36], due to the at least 2.5-times higher clock frequency and three-times higher DSP occupation by the fpgaConvNet and Angel-Eye accelerators.

Table 2. Comparison of the proposed CNN accelerator to the state-of-the-art fpgaConvNet [35], DnnWeaver [34], and Angel-Eye [36] implementations on the same Z-7020 device.

Implementation	Proposed	fpgaConvNet	DnnWeaver	Angel-Eye
Clock, MHz (conv.)	50	125	150	214
LUTs	36K	–	35K	30K
DSPs	68	220	140	190
4kB BRAMs	48	–	140	86
GOPS (conv.)	32	48.53	31.35	84.3
Precision	16 b	16 b	16 b	8 b

During the experimental evaluation, we needed to discover how the image resolution affected recognition accuracy. Next, we found the cost-efficient configuration of the CNN that was trained on a full dataset and a partial dataset collected from a single beehive. Table 3 provides the classification accuracy, speed, and memory consumption of the CNN accelerator applied for the classification of images with and without pollen. Classification accuracy is computed as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}; \quad (5)$$

here, TP, TN, FP, and FN denote true positives, true negatives, false positives, and false negatives, respectively. All values were taken from a confusion matrix.

Every accuracy record in Table 3 is the mean value of five training and test procedures. The images for training and testing were randomly selected from a full dataset with a proportion of 4 to 1 in a way that exactly 80% and 20% of the images from every beehive were allocated in the training and test datasets, respectively.

During the experimentation, we trained dozens of networks with different resolutions of the input images. At the beginning, we trained the CNN on a full dataset and raw 1024 × 256 px images. Then, we downsampled the raw images 2, 4, and 8 times and followed the classification accuracy. The optimal number of feature maps in every convolutional layer needed to be a multiple of eight due to the specifics of the architecture of the proposed convolutional core. In Table 3, we use a simplified abbreviation to describe an investigated structure of the CNN as: [C_1-C_2-…-C_n:D_1-D_2], where C_n marks the number of convolutional kernels in the n-th convolutional layer. All the kernels were 3 × 3 px. Here, every convolutional layer was followed by batch normalization, ReLU, and max polling. D_1 and D_2 mark the number of neurons in the first and second dense layers. The number of neurons in the second dense layer D_2 was fixed to two due to the two classes of images with and without pollen. The number of neurons in the first dense layer D_1 varied from 8 to 64. We observed that the optimal number of neurons in D_1 was 32. Increasing D_1 did not improve the classification accuracy, but slowed down the training procedure.

We noticed that five convolutional layers were optimal to classify the raw 1024 × 256 px images with a 91% accuracy. A further increase in the amount of convolutional layers did not improve the classification accuracy of the raw images. For 2- and 4-times downsampled images (512 × 128 px and 256 × 64 px), it is recommended to set four convolutional layers to achieve a 92% accuracy. The increase in the amount of kernels to 32 per convolutional layer did not improve the classification accuracy and remained the same as for 16 kernels per layer. The CNN with two or three convolutional layers gave an 82% classification accuracy on the eight-times downsampled (128 × 32 px) images. Therefore, the 128 × 32 px resolution was too low to detect pollen grains in the images. Images with 512 × 128 px or 256 × 64 px resolution that covered the entire entrance ramp are recommended for practical applications with the proposed CNN accelerator, which was configured to process a [16-16-16-16:32-2] CNN structure. It took 8.8 ms to classify one 512 × 128 px frame and 2.4 ms for a 256 × 64 px frame, respectively.

Table 3. Classification accuracy, time per frame, and memory consumption depending on the image resolution and CNN structure.

Image Resolution	CNN Structure	Accuracy (%)	Time (ms/frame)	RAM (MB)
Full dataset				
1024 × 256	16-16:32-2	55	49.4	20.5
	16-16-16:32-2	88	37.1	8.6
	16-16-16-16:32-2	89	34.1	5.7
	16-16-16-16:8-2	88	33.1	4.9
	16-16-16-16-16:32-2	91	33.4	4.9
	16-16-16-16-16:64-2	91	33.7	5.2
	16-16-16-16-16:16-2	90	33.2	4.8
	16-16-16-16-16:8-2	90	33.1	4.8
	16-8-8-8-8:8-2	85	20.7	3.8
	16-8-8-8-8:32-2	90	20.8	3.9
	16-16-16-16-16-16:32-2	91	33.3	4.8
512 × 128	16-16:32-2	86	12.5	5.1
	16-16-16:32-2	87	9.5	2.2
	16-16-8-8:32-2	88	7.8	1.1
	16-16-16-16:32-2	92	8.8	1.4
	32-16-16-16:32-2	92	18.8	2.5
	32-32-32-32:32-2	92	36.1	2.9
	16-16-16-16-16:32-2	92	8.7	1.3
	16-16-16-16-16:8-2	91	8.6	1.2
	16-8-8-8-8:8-2	89	5.3	1.0
256 × 64	32-16:32-2	87	5.9	1.6
	16-16:32-2	87	3.2	1.3
	16-16:64-2	88	4.6	2.3
	16-16-16:32-2	89	2.6	0.6
	16-16-16-16:32-2	92	2.4	0.4
	16-16-16-16:16-2	91	2.4	0.4
	16-16-16-16:8-2	90	2.4	0.3
128 × 32	16:32-2	64	1.6	1.1
	16-16:32-2	82	0.9	0.4
	16-16-16:32-2	82	0.8	0.2
	16-8:32-2	81	0.6	0.2
	8-8:32-2	80	0.4	0.2
Single beehive dataset				
256 × 64	8-8:16-2	60	0.9	0.4
	8-8:32-2	95	1.2	0.6
	8-8-8:32-2	95	0.8	0.3
	16-16-16:32-2	95	2.6	0.6
128 × 32	8:32-2	92	0.8	0.5
	16:32-2	93	1.6	1.1

If the CNN was trained on a dataset collected from a single beehive, then the CNN with two convolutional layers [8-8:32-2] yielded about a 95% classification accuracy for 256 × 64 px frames. Even the CNN with a single convolutional layer and 16 kernels gave a 93% accuracy for pollen detection in 128 × 32 px images.

To prove the effectiveness of the proposed approach, our implementation was compared with other related works, as presented in Table 4. The comparison showed that the proposed system outperformed the rest in terms of the frame rate and can provide real-time pollen presence detection. The classification accuracy was similar to other works. If the CNN model was trained on a dataset collected from an entrance of a single beehive and the trained model was further used to classify frames from that beehive, then the classification accuracy was as high as was achieved by Ngo et al. [23] and Rodriguez et al. [24].

Ngo et al. [23] used a GPU-based Jetson TX2 embedded system; Babic et al. [25] implemented their classification algorithm on an Intel i3 CPU and then accelerated it with a Raspberry Pi. All the other approaches were implemented on a PC and were based on CPU and GPU cores.

Table 4. Comparative evaluation of the proposed implementation with state-of-the-art methods.

Authors	Implementation	Dataset	Resolution	FPS	Accuracy
Yang and Collins [27]	Faster RCNN on PC	1000/400	200 × 200	-	93
Rodriguez et al. [24]	CNN on PC	497/213	180 × 300	-	96
Monteiro et al. [29]	DarkNet53 on PC	710	180 × 300	-	99
Sledevic [30]	CNN on PC	1600/400	100 × 100	-	94
Ngo et al. [23]	YOLOv3 on GPU	3000/500	416 × 416	25	94
Babic et al. [25]	SVM on Raspberry Pi	100/50	1280 × 360	1	89
Stojnic et al. [26]	SVM on PC	800/200	1280 × 720	6	92
This work	CNN on FPGA	4800/1200	512 × 128	113	92
		4800/1200	256 × 64	416	92
		800/200	256 × 64	1250	95

In Table 4, the Dataset row marks the number of images used to train/test the classifier. It is worth mentioning that the compared works used slightly different approaches to solve the pollen grain detection task. The first four works [24,27,29,30] detected pollen grains in cropped images where the bee was already centred. They used a relatively lower resolution in comparison to the next three works and did not investigate the time required to process one frame, while the works [23,25,26] solved the real-time multiple bee detection and localization problem upon entrance to the beehive along with pollen classification. In all the mentioned approaches (Table 4), the authors used monotone coloured background boards (usually black, dark blue, dark green) at the entrance to the hive to improve the contrast and enhance the detectability of the bees, while our approach solved the pollen grain presence detection task without any modifications to the hive entrance ramps.

5. Discussion

Table 3 shows that the higher the resolution of the image, the more convolutional layers need to be used in the CNN to detect the presence of pollen grains. The input image needs to pass a few convolutional and subsampling layers to clarify the shape of the grains, especially if a convolutional kernel covers only a 3 × 3 pixel area, which is smaller than the size of a grain. The visual presentation of the results in the hidden layers helps to understand what kind of features the CNN is trying to extract from the images. Figure 12 presents a few samples of the feature maps (activators) extracted by the CNN in the image with two bees with pollen. The resolution of the input image (Figure 12a) was 1024 × 256 px, and the CNN had a [16-16-16-16-16:32-2] structure with five convolutional layers and 16 kernels in each layer. The most explicit maps (one map per convolutional layer) are presented in Figure 12b–f. The activators showed how the CNN saw an image in the inputs to the 2nd, 3rd, 4th, and 5th convolution and 1st dense layer. The pixels with a high amplitude in the maps mark the location of pollen grains detected in an image. Our proposed method did not localize or even count the pollen grains; however, it showed in general whether the pollen grains were present in a frame or not. Single or more pollen grains gave same positive responses. The future work will be to train more advanced models of the CNN for bee localization and tracking.

The full dataset consisted of images collected from six beehives with different entrance ramps. If the CNN accelerator were planned to use only on a single beehive, then it would be preferable to train the CNN on the dataset for that target beehive. This would yield faster processing and a higher classification rate (95%) with fewer parameters for CNN. A single 256 × 64 px resolution frame was classified in 2.4 ms using the [16-16-16-16:32-2] CNN structure. This yielded more than 400 fps. Therefore, it is recommended to employ a

CNN accelerator as an image classification server and send the frames to it from cameras mounted on multiple beehives.

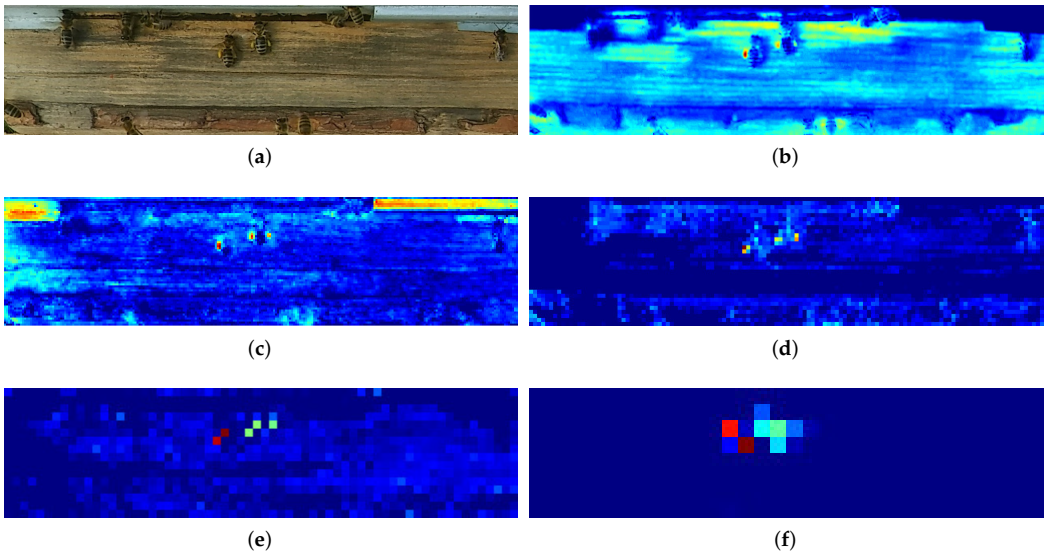


Figure 12. Image of the entrance to a hive with two bees with pollen in the centre (**a**). The 512 × 128 px feature map from the input to the 2nd convolutional layer (**b**), the 256 × 64 px feature map from the input the to 3rd convolutional layer (**c**), the 128 × 32 px feature map from the input to the 4th convolutional layer (**d**), the 64 × 16 px feature map from the input to the 5th convolutional layer (**e**), and 32 × 8 px feature map from the input to the 1st dense layer (**f**).

In all the mentioned state-of-the-art methods, the background colour at the hive entrance was intentionally selected as monotone and dark to increase the gradient between the bee and the background. Therefore, edge detection algorithms [37] can be successfully employed not only to extract the shape of the bee, but also to locate multiple bees that appear in an image.

In this work, we used the Z-7020 SoC FPGA. According to the utilisation results (Table 1), the design of the CNN accelerator can fit even more cost-optimised devices such as the Z-7015 and Z-7014S [38]. The CNN accelerator utilised 34% of the BRAM on the Z-7020 chip. Therefore, 0.36 MB of on-chip memory was free. In the case when we plan to use a CNN structure with a memory requirement less than 0.36 MB, it is preferable to use the BRAM on the Z-7020 instead of the DMA stream interfaces to access the external memory. Because the bottleneck of the proposed design is the limited speed of the feature map transaction between the convolutional core on FPGA and the external memory, the temporary storage of the data in the BRAM would speed-up access to the feature maps and enable us to run a convolutional core at a higher clock frequency.

The main reason for the relatively low 50 MHz clock frequency of the convolutional core was the synchronization of the core with the AXI stream of data. Two AXI channels give 64 bits at 100 MHz. This results in four parallel streams of 16-bit interleaved sequences of samples. From those streams, the convolutional core extracts eight channel feature maps, and therefore, it should run at 50 MHz (4 CH ×16 b ×100 MHz = 8 CH ×16 b ×50 MHz). If the internal buffers are used to store the input/output feature maps in separate BRAMs, then the clock frequency can be increased at least twice because the feature maps will not go outside the FPGA and the processing speed will be not limited by the throughput between the FPGA and external RAM.

6. Conclusions

In this work, we proposed a hardware–software architecture of a CNN for implementation on SoC FPGA-based embedded systems. A dozen CNN configurations were trained for pollen detection in the images. The [16-16-16-16:32-2] CNN structure with four

convolutional and two dense layers gave the highest pollen grain classification accuracy for 512 × 128 px and 256 × 64 px resolution frames and took 8.8 ms and 2.4 ms per single frame classification, respectively. The accuracy and speed make it suitable for practical application and pollen detection upon entrance to the beehive. The CNN accelerator was implemented on a cost-optimized Z-7020 FPGA with 16-bit fixed-point operations, and it utilized 31% of DSP, 67% of logic, and 34% of BRAM resources. The performance of 32 GOPS on a single convolutional core was similar to the performance obtained on existing FPGA-based CNN accelerators [34,35].

The results showed that the proposed CNN architecture is fast enough to classify bees with or without pollen in real-time, even if implementing the classifier on low-cost FPGA boards based on the Z-7015 and Z-7014S chips. Therefore, we plan to extend the functionality of the design for real-time multiple bee localization and tracking. Therefore, the complexity of the CNN needs to be increased by implementing a tiny YOLO or a custom CNN. Future research is also directed at a speed up of the CNN accelerator. Higher throughput can be achieved by the duplication of the convolutional cores and the dedication of the on-chip BRAM for temporary storage of the layer outputs and reduction of the AXI streams.

Author Contributions: Conceptualization, T.S.; methodology, T.S. and A.S.; software, T.S.; validation, A.S. and D.P.; investigation, T.S., D.P. and A.S.; writing—original draft preparation, T.S., D.P. and A.S.; visualization, T.S. and D.P. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: The authors would like to thank the Editors and Reviewers for many constructive suggestions and comments, which helped to improve the quality of the paper.

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

References

1. Varna, D.; Abromavičius, V. A System for a Real-Time Electronic Component Detection and Classification on a Conveyor Belt. *Appl. Sci.* **2022**, *12*, 5608. [CrossRef]
2. Hu, M.; Wang, C.; Yang, J.; Wu, Y.; Fan, J.; Jing, B. Rain rendering and construction of rain vehicle color-24 dataset. *Mathematics* **2022**, *10*, 3210.
3. Karlekar, A.; Seal, A. SoyNet: Soybean leaf diseases classification. *Comput. Electron. Agric.* **2020**, *172*, 105342.
4. Odemer, R. Approaches, challenges and recent advances in automated bee counting devices: A review. *Ann. Appl. Biol.* **2022**, *180*, 73–89. [CrossRef]
5. Hadjur, H.; Ammar, D.; Lefèvre, L. Toward an intelligent and efficient beehive: A survey of precision beekeeping systems and services. *Comput. Electron. Agric.* **2022**, *192*, 106604. [CrossRef]
6. Csillik, O.; Cherbini, J.; Johnson, R.; Lyons, A.; Kelly, M. Identification of citrus trees from unmanned aerial vehicle imagery using convolutional neural networks. *Drones* **2018**, *2*, 39. [CrossRef]
7. Ioffe, S.; Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Proceedings of the International Conference on Machine Learning, PMLR, Lille, France, 7–9 July 2015; pp. 448–456.
8. Rastegari, M.; Ordonez, V.; Redmon, J.; Farhadi, A. Xnor-net: Imagenet classification using binary convolutional neural networks. In Proceedings of the European Conference on Computer Vision, Amsterdam, The Netherlands, 11–14 October 2016; pp. 525–542.
9. Courbariaux, M.; Hubara, I.; Soudry, D.; El-Yaniv, R.; Bengio, Y. Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or −1. *arXiv* **2016**, arXiv:1602.02830.
10. Liang, S.; Yin, S.; Liu, L.; Luk, W.; Wei, S. FP-BNN: Binarized neural network on FPGA. *Neurocomputing* **2018**, *275*, 1072–1086. [CrossRef]
11. Chen, J.; Liu, L.; Liu, Y.; Zeng, X. A learning framework for n-bit quantized neural networks toward FPGAs. *IEEE Trans. Neural Netw. Learn. Syst.* **2020**, *32*, 1067–1081. [CrossRef]
12. Wang, J.; Lin, J.; Wang, Z. Efficient hardware architectures for deep convolutional neural network. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2017**, *65*, 1941–1953. [CrossRef]

13. Lian, X.; Liu, Z.; Song, Z.; Dai, J.; Zhou, W.; Ji, X. High-performance fpga-based cnn accelerator with block-floating-point arithmetic. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2019**, *27*, 1874–1885. [CrossRef]
14. Guo, K.; Han, S.; Yao, S.; Wang, Y.; Xie, Y.; Yang, H. Software-hardware codesign for efficient neural network acceleration. *IEEE Micro* **2017**, *37*, 18–25. [CrossRef]
15. Tapiador-Morales, R.; Linares-Barranco, A.; Jimenez-Fernandez, A.; Jimenez-Moreno, G. Neuromorphic LIF row-by-row multiconvolution processor for FPGA. *IEEE Trans. Biomed. Circuits Syst.* **2018**, *13*, 159–169. [PubMed]
16. Chung, J.; Shin, T.; Yang, J.S. Simplifying deep neural networks for FPGA-like neuromorphic systems. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2018**, *38*, 2032–2042. [CrossRef]
17. Alzubaidi, L.; Zhang, J.; Humaidi, A.J.; Al-Dujaili, A.; Duan, Y.; Al-Shamma, O.; Santamaría, J.; Fadhel, M.A.; Al-Amidie, M.; Farhan, L. Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions. *J. Big Data* **2021**, *8*, 53. [CrossRef]
18. Shawahna, A.; Sait, S.M.; El-Maleh, A. FPGA-based accelerators of deep learning networks for learning and classification: A review. *IEEE Access* **2018**, *7*, 7823–7859. [CrossRef]
19. Luo, Y.; Chen, Y. FPGA-Based Acceleration on Additive Manufacturing Defects Inspection. *Sensors* **2021**, *21*, 2123. [CrossRef]
20. 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]
21. Kim, J.; Kang, J.K.; Kim, Y. A Resource Efficient Integer-Arithmetic-Only FPGA-based CNN Accelerator for Real-Time Facial Emotion Recognition. *IEEE Access* **2021**, *9*, 104367–104381. [CrossRef]
22. Wang, C.C.; Ding, Y.C.; Chiu, C.T.; Huang, C.T.; Cheng, Y.Y.; Sun, S.Y.; Cheng, C.H.; Kuo, H.K. Real-Time Block-Based Embedded CNN for Gesture Classification on an FPGA. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2021**, *68*, 4182–4193. [CrossRef]
23. Ngo, T.N.; Rustia, D.J.A.; Yang, E.C.; Lin, T.T. Automated monitoring and analyses of honey bee pollen foraging behavior using a deep learning-based imaging system. *Comput. Electron. Agric.* **2021**, *187*, 106239. [CrossRef]
24. Rodriguez, I.F.; Megret, R.; Acuna, E.; Agosto-Rivera, J.L.; Giray, T. Recognition of Pollen-Bearing Bees from Video Using Convolutional Neural Network. In Proceedings of the 2018 IEEE Winter Conference on Applications of Computer Vision (WACV), Lake Tahoe, NV, USA, 2–15 March 2018; pp. 314–322. doi: 10.1109/WACV.2018.00041. [CrossRef]
25. Babic, Z.; Pilipovic, R.; Risojevic, V.; Mirjanic, G. Pollen bearing honey bee detection in hive entrance video recorded by remote embedded system for pollination monitoring. *ISPRS Ann. Photogram. Remote Sens. Spat. Inf. Sci.* **2016**, *3*, 51–57.
26. Stojnić, V.; Risojević, V.; Pilipović, R. Detection of pollen bearing honey bees in hive entrance images. In Proceedings of the 2018 17th International Symposium INFOTEH-JAHORINA (INFOTEH), East Sarajevo, Bosnia and Herzegovina, 21–23 March 2018; pp. 1–4.
27. Yang, C.; Collins, J. Deep learning for pollen sac detection and measurement on honeybee monitoring video. In Proceedings of the 2019 International Conference on Image and Vision Computing New Zealand (IVCNZ), Dunedin, New Zealand, 2–4 December 2019; pp. 1–6.
28. Ngo, T.N.; Wu, K.C.; Yang, E.C.; Lin, T.T. A real-time imaging system for multiple honey bee tracking and activity monitoring. *Comput. Electron. Agric.* **2019**, *163*, 104841. [CrossRef]
29. Monteiro, F.C.; Pinto, C.M.; Rufino, J. Towards Precise Recognition of Pollen Bearing Bees by Convolutional Neural Networks. In Proceedings of the Iberoamerican Congress on Pattern Recognition, Porto, Portugal, 10–13 May 2021; pp. 217–226.
30. Sledevič, T. The application of convolutional neural network for pollen bearing bee classification. In Proceedings of the 2018 IEEE 6th Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE), Vilnius, Lithuania, 8–10 November 2018; pp. 1–4.
31. Sledevič, T.; Serackis, A. mNet2FPGA: A Design Flow for Mapping a Fixed-Point CNN to Zynq SoC FPGA. *Electronics* **2020**, *9*, 1823. [CrossRef]
32. Sledevič, T. Adaptation of convolution and batch normalization layer for CNN implementation on FPGA. In Proceedings of the 2019 Open Conference of Electrical, Electronic and Information Sciences (eStream), Vilnius, Lithuania, 25 April 2019; pp. 1–4.
33. Xilinx Incorporated. ZedBoard—Zynq SoC Development Board. Available online: https://www.xilinx.com/products/boards-and-kits/1-8dyf-11.html (accessed on 28 March 2022).
34. Sharma, H.; Park, J.; Mahajan, D.; Amaro, E.; Kim, J.K.; Shao, C.; Mishra, A.; Esmaeilzadeh, H. From high-level deep neural models to FPGAs. In Proceedings of the 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Taipei, Taiwan, 15–19 October 2016; pp. 1–12.
35. Venieris, S.I.; Kouris, A.; Bouganis, C.S. Toolflows for Mapping Convolutional Neural Networks on FPGAs: A Survey and Future Directions. *ACM Comput. Surv. (CSUR)* **2018**, *51*, 1–36.
36. 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.* **2017**, *37*, 35–47.
37. Nausheen, N.; Seal, A.; Khanna, P.; Halder, S. A FPGA based implementation of Sobel edge detection. *Microprocess. Microsyst.* **2018**, *56*, 84–91. [CrossRef]
38. Xilinx Incorporated. Zynq-7000 SoC Family Product Selection Guide. Available online: https://www.xilinx.com/content/dam/xilinx/support/documents/selection-guides/zynq-7000-product-selection-guide.pdf (accessed on 2 September 2022).