






Article

Implementation of a DPU-Based Intelligent Thermal Imaging Hardware Accelerator on FPGA

Abdelrahman S. Hussein ^{1,*} , Ahmed Anwar ¹, Yasmine Fahmy ¹ , Hassan Mostafa ^{1,2} ,
Khaled Nabil Salama ³  and Mai Kafafy ¹ 

¹ Faculty of Engineering, Cairo University, Giza 12613, Egypt; ahmed.anwar.saeed@gmail.com (A.A.); yASFahmy@eng.cu.edu.eg (Y.F.); hmostafa@staff.cu.edu.eg (H.M.); mai.b.s.ali@eng.cu.edu.eg (M.K.)

² Nanotechnology and Nanoelectronics Program, Zewail City of Science and Technology, University of Science and Technology, Giza 12578, Egypt

³ Electrical Engineering Program, King Abdullah University of Science and Technology, Thuwal 23955-6900, Saudi Arabia; khaled.salama@kaust.edu.sa

* Correspondence: abdelrahman_hussein@ieee.org

Abstract: Thermal imaging has many applications that all leverage from the heat map that can be constructed using this type of imaging. It can be used in Internet of Things (IoT) applications to detect the features of surroundings. In such a case, Deep Neural Networks (DNNs) can be used to carry out many visual analysis tasks which can provide the system with the capacity to make decisions. However, due to their huge computational cost, such networks are recommended to exploit custom hardware platforms to accelerate their inference as well as reduce the overall energy consumption of the system. In this work, an energy adaptive system is proposed, which can intelligently configure itself based on the battery energy level. Besides achieving a maximum speed increase that equals $6.38\times$, the proposed system achieves significant energy that is reduced by 97.81% compared to a conventional general-purpose CPU.

Keywords: Convolutional Neural Network (CNN); Deep Learning Processing Unit (DPU); Field Programmable Gate Array (FPGA)



Citation: Hussein, A.S.; Anwar, A.; Fahmy, Y.; Mostafa, H.; Salama, K.N.; Kafafy, M. Implementation of a DPU-Based Intelligent Thermal Imaging Hardware Accelerator on FPGA. *Electronics* **2022**, *11*, 105. <https://doi.org/10.3390/electronics11010105>

Academic Editor: Nunzio Cennamo

Received: 31 October 2021

Accepted: 13 December 2021

Published: 29 December 2021

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



Copyright: © 2021 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

Thermal imaging, also known as thermography, is constructing a heat map of the surrounding. This is done by detecting the heat signature of the different objects using thermal cameras that operate in the infra-red range (9–14 μm). Although thermography started as a military application to detect enemy forces, it has recently found its way to many more applications thanks to its numerous advantages. For example, the dependence on heat allows thermography to operate well in a non-lit environment, and consequently, it is suitable for surveillance applications [1] and wildlife monitoring [2]. Thermal imaging can be used to monitor the welfare of the elderly as it only captures the human body temperature and, therefore, does not violate their privacy [3]. Thermography helps to detect early diseases in humans and plants besides early detection of thermal discomfort among farm animals.

Thermography applications, especially monitoring, produce long sequences of thermal images (or video frames). Different frames are expected to have different significance. For example, while some frames might be safely ignored, others might be essential. An intelligent edge device should not only decide the significance of the frame but also adjust its behavior accordingly. As suggested in [4], edge devices should adjust their compression rate and activity rate according to the image importance, the battery status, and the remaining operation time.

In this paper, an edge device with a thermal camera, a transmitter, and an Intelligent Thermal Image Processing Unit (ITIPU) is proposed. Detailed design of the ITIPU is

provided in Section 3. Generally, the ITIPU consists of an image classifier, encoder, and controller. The classifier is used to determine the importance of the captured image, the encoder compresses the image before transmission, and the controller makes decisions based on the classifier output. These decisions include deciding the compression rate of the image and the frame rate of the thermal camera. The classification and the compression are both designed based on Convolutional Neural Networks (CNN) architectures with different mathematical operations and different layers. It should be noted that despite the dominant accuracy achieved by the state-of-the-art deep neural networks, this comes with a huge price represented in the computational cost [5]. This huge computational cost can be observed for example in a simple model such as the LeNet-5 model, which requires about 680×10^3 arithmetic operations (addition or multiplication) per classification (op/cl) to carry out a simple MNIST handwritten digits classification, whereas the VGG16 requires about 31G op/cl handling the 1000-class ImageNet task [5]. As the aim of this work is to design and implement a general-purpose ITIPU, FPGA is an excellent candidate to be used to accelerate our Deep Neural Network (DNN) model besides offering capacity for customization. FPGA is also exploited to offer significant reconfigurability and adaptability during run-time which can be used to reconfigure the programmable logic based on specific pre-defined parameters, such as the remaining battery energy level or data transfer speed. In order to deploy a CNN on the FPGA, long design time might be needed which is a challenging problem, and correspondingly, a great effort is then required to deploy a CNN architecture to an FPGA. Xilinx Inc. has addressed this problem by releasing an Intellectual Property core (IP core) called Deep Learning Processing Unit (DPU). The DPU provides an efficient implementation of different CNN architectures on the FPGA by supporting various deep learning operations. The DPU offers flexibility in the implementation of CNNs on the FPGA as well as efficiency in terms of, design time, energy consumption, and latency.

While this work proposes a compression engine besides a set of classifiers that differ in performance and accuracy, these differences should be leveraged to expand both the adaptive and reconfigurable features of the system in order to address the limitations of the IoT applications and enhance the computational processing capacity. Accordingly, the proposed system should demonstrate a set of crucial specifications and features that would overcome those limitations. Such crucial features are considered to be parallel processing on different levels and reconfigurability. While the former should be realized in executing different tasks (i.e., classification, and compression) simultaneously as well as parallelism on the scale of each task, the latter can be reflected on reconfiguring the system based on a specific sensitivity list.

In this work, an ITIPU based on a unified deep-learning design is proposed. The design is implemented by a DPU on the programmable logic of the Xilinx Zynq UltraScale+ MPSoC ZCU104 FPGA Evaluation Kit. The main aim of the design is to achieve a programmable flexible design with a high image processing throughput and minimum energy consumption. The main contributions of this work are:

- An adaptive system is proposed to carry out the image classification and compression. The system is reconfigured based on several parameters such as the battery energy level and the data transfer rate.
- To best of the authors' knowledge, this is the first time such an IP is used in an adaptive scheme in order to maintain the system operation in different scenarios. This promotes its feasibility for IoT applications as it results in lengthening the battery life, especially, in applications with a limited energy budget.
- Additionally, the proposed system is capable of executing both classification and compression tasks simultaneously, which boosts the overall system performance.
- The experimental results of this work provide useful design recommendations and insights to hardware accelerators designers to reduce the CNN accelerators design time with high throughput and low energy consumption.

The rest of this paper is organized as follows: Section 3 shows the system-level and Section 4 shows the proposed designs of deep learning-based image classification and compression. DPU design flow is presented in Section 5, and the implementation is detailed in Section 6. Finally, Section 7 concludes the paper.

2. Related Work

Recently, the authors in [6] evaluated the performance of the DPU, among other architectures, using different vision kernels and neural networks as benchmarks. The outcomes showed that the DPU-Based accelerator on FPGA outperformed both ARM Cortex A57 CPU with 32-bit floating-point precision and Nvidia Jetson TX2 with 16-bit floating-point precision in terms of the throughput of MobileNet-V2. Furthermore, the accuracy loss observed due to the model compression was negligible (less than 1.1%). With regard to object detection applications, Wang et al. [7] used the DPU to accelerate YOLOv3 [8] and compare its performance with its counterpart on Nvidia GeForce GTX1080. While the numerical precision was significantly reduced on the DPU compared to the GPU (8-bit fixed precision and 32-bit floating-point, respectively), the DPU demonstrated more than $2\times$ for single-threaded and $6\times$ while leveraging multi-threading higher throughput than the GPU. In terms of energy efficiency, the DPU processed 3.38 FPS/W whereas the GPU only maintained only 0.26 FPS/W. Similarly, Chan et al. [9] deployed the same system for robots to detect agricultural crops where the DPU also outperformed both CPU and GPU in terms of power efficiency. From the perspective of the software stack, Zhu et al. [10] addressed the challenge of optimizing the task scheduler by building a task assignment that uses the Vitis AI software stack. This led to significant improvement in terms of performance. One main problem with the aforementioned work is that the systems demonstrated constant energy consumption that did not change according to certain sensitivities (e.g., the battery lifetime). This may present a challenge for IoT applications to adopt these systems. Accordingly, this work addresses this problem by proposing an energy-adaptive system that features a shorter development time, higher flexibility that covers a wide range of CNN architectures. The other observation is that the prior work evaluated the DPU performance using standardized CNN architectures (e.g., VGG16, ResNet-50, etc.). Thus, the proposed work also addresses this observation by proposing novel CNN architectures for classification and compression.

3. System Design

This section describes the system-level design of an intelligent thermal image processing unit (ITIPU). The unit consists of an image classifier, an image compressor, and a controller as shown in Figure 1. A thermal camera feeds its captured frames to the image classifier, which classifies each frame to a number of predetermined classes. Following that, the frames are compressed by a multi-rate image encoder before being transmitted, over the wireless channel, to the destination. The classification outcome is used as an input to the controller to determine the compression rate of the classified frame. For example, a frame belonging to an unimportant class can be highly compressed (or skipped) in order to save transmission energy. The controller also uses the classification outcome to regulate the frame rate of the thermal camera. For example, it can opt to reduce the frame rate when a series of insignificant or uninteresting frames is received.

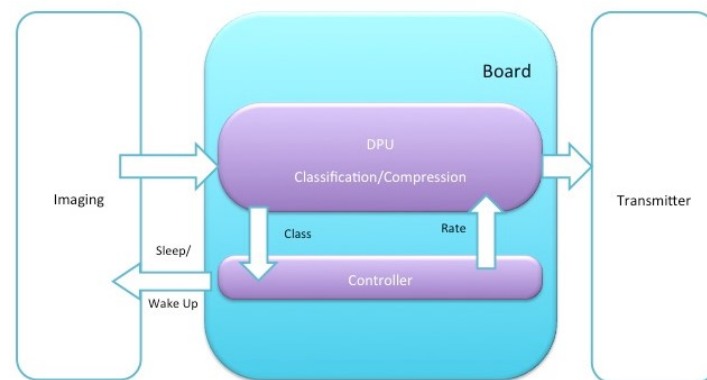


Figure 1. Block diagram of the intelligent thermal image processing unit.

The classification and compression units are designed as CNNs. We use, without loss of generality, the aerial thermal image dataset provided by SenseFly in [11] to train and test the classification and compression units. The dataset is a series of thermal images captured by a drone that monitors solar panels installation. For classification purposes, the images are grouped into two classes: images with solar panels, and images without solar panels. Sample images of the two classes are shown in Figure 2. The images are greyscale of size 512×512 . The following section provides the detailed design of the classification and compression units.

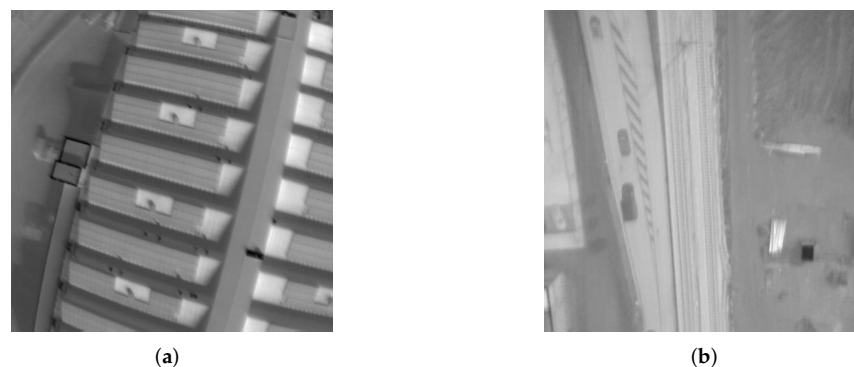


Figure 2. Sample images of the two classes from the SenseFly dataset [11]. (a) sample image from class 1 (with solar panels); (b) sample image from class 2 (without solar panels).

4. Image Classification and Compression Unit Design

4.1. Deep Learning Based Image Classification

This work focuses on deep learning-based classifiers as they automatically learn the best feature representation of the input images during a supervised training process. They only require the availability of sufficient data for training without the need for field experts to manually extract and handcraft the features for the classifier. The proposed classifier is generic and can be reused for different applications as the same CNN can be retrained to learn a different set of features from different sets of data.

The literature has plenty of CNN-based classifiers, however, the majority of them depend on key modules proposed by key classifiers such as the inception module in GoogleNet [12], the residual block in ResNet [13], the dense blocks in DenseNet [14], the separable depthwise convolution in MobileNet v1 [15], and the inverted residual block in MobileNet V2 [16]. A comprehensive survey on recent deep CNN architectures can be found in [17]. Bianco et al. [18] have compared the performance and complexity of different well-known architectures of CNN-based image classifiers. The comparison has shown that MobileNet v1, MobileNet v2, and DenseNet exhibit the best performance on resources limited applications, which suits the nature of IoT-oriented applications. However, using

such classifiers, through transfer learning, for a device with a custom thermal application is a waste of resources because they use deep CNNs with millions of parameters (around 4.24 M in MobileNet v1 and 3.47 M in MobileNet v2) as they are designed to classify over 1000 classes of RGB visible images. While in custom thermal applications the device usually needs to distinguish between few classes of thermal (greyscale) images that contain much fewer features than RGB visible images.

4.2. Proposed Image Classifier Design

The designed classifier combines the concepts of separable depthwise convolution from MobileNet v1 [15], the inverted residual from MobileNet v2 [16], and the feature map concatenation from DenseNet [14]. Figure 3 shows the architecture of different classifier models. Different models have different complexities (i.e., different numbers of parameters). For example, the models in Figure 3a–c have 5858, 2052, and 7556 parameters, respectively. The model in Figure 3d can have two forms: the thick form which has 24,802 parameters, and the thin form which has 7282 parameters. The notion thick or thin depends on the number of output features of the first 2D conv block (denoted by F in Figure 3). A thick model has $F = 32$, while a thin model has $F = 16$. As shown in the figure, all the models have the same core architecture with variation in the length (the number of stages), the thickness (the number of feature maps), and the number of dense (fully connected) layers at the output. This allows a simple hardware switch between the different models by disabling some connections/layers. The model switch should be guided by a higher-level decision based on the available battery energy, remaining operation time, and required accuracy.

Figure 4 shows the structure of the “stage” block in Figure 3. Like MobileNet v2, each “stage” block consists of three Convolutional (Conv) layers. The first Conv layer doubles the number of input feature maps, denoted by F in Figure 4. The second Conv layer is a depthwise convolution. The depthwise convolution along with the third Conv layer are the equivalent of a depthwise separable convolution. The third Conv layer halves the number of input features. The flow of feature expansion, followed by depthwise convolution and feature compression has been presented in [15,16] to reduce the design complexity. Batch normalization layers normalize the input using the batch statistics (i.e., mean and variance). This normalization keeps the input from going to very high or very low values, and consequently improves the stability of CNNs. It also has a slight regularization effect as it adds some randomness to the input features. The input of each “stage” block in Figure 3 is concatenated to the stage output to allow the propagation of coarse and fine detailed features through the network. Feature concatenation has been proposed in DenseNet [14] to reduce the need to learn redundant features, and it also allows fast access to the gradient during backpropagation. The Max Pooling layer between stages in Figure 3 reduces the size of the input to the next stage, and the global average pooling layer is used, instead of a dense layer to reduce the complexity by reducing the number of variables. The dependence on the average of each feature map in classification also reduces the chances of overfitting the training data.

The classifier has been trained and tested on images from the senseFly dataset [11]. The input image is classified to one of classes that indicate whether the image shows solar panels or not. Data augmentation techniques, such as rotation, horizontal and vertical flipping, and cropping, have been applied to the training images to improve the learning process of the classifier and to avoid overfitting. Table 1 shows the classification accuracy and loss of the different classifier models in Figure 3. The number of images in the training, validation, and test sets are 3732, 334, and 90 images respectively. We use the notion “Thick/Thin-1/2-Dense/” to distinguish the different classifier models presented in Figure 3. The first term indicates whether a model is thick or thin, the second term indicates the number of stages in the model, and the third term indicates if the model has two dense (fully connected) layers at the output. We refer to the models in Figure 3a–c as “Thick-1”, “Thin-1-Dense”, “Thin-2-Dense”, respectively. We refer to the thick model in Figure 3d as “Thick-2” and the thin model in Figure 3d as “Thin-2”.

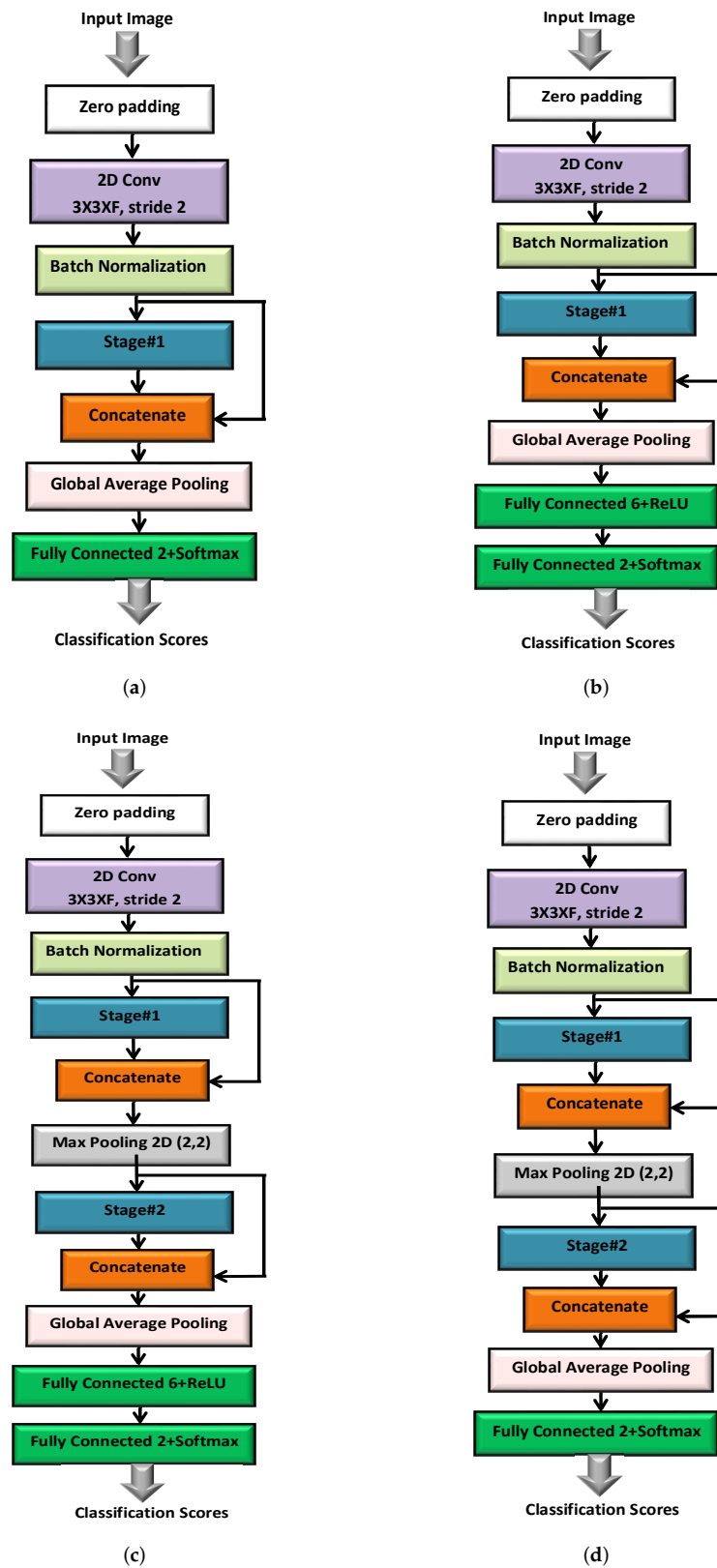


Figure 3. Different classifier models. (a) Thick ($F = 32$) one-stage model; (b) Thin ($F = 16$) one-stage model with a dense layer; (c) Thin ($F = 16$) two stages model with a dense layer; (d) Thick (thin) two stages model, $F = 32$ (16).

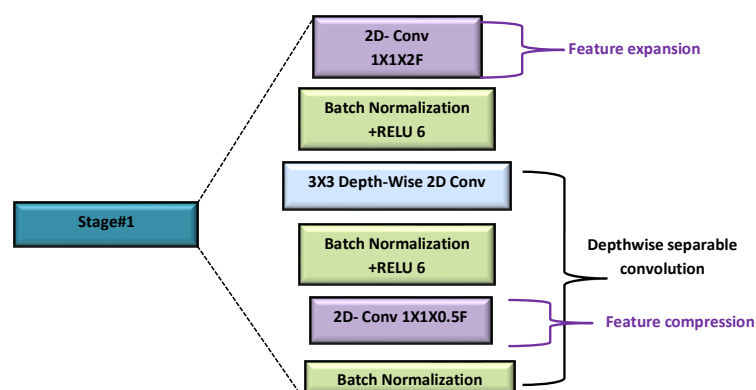


Figure 4. Architecture of the “Stage” block.

Table 1. Classifier Training, validation and test accuracy and loss on a core i7-4790 @ 3.6 GHz CPU.

Classifier	Design		Training		Validation		Test	
	Thickness	Parameters	Accuracy	Loss	Accuracy	Loss	Accuracy	Loss
Thick-1	32	5858	81.32%	0.42	79.64%	0.44	76.67%	0.46
Thin-1-Dense	16	2052	79.88%	0.46	74.85%	0.50	80.00%	0.49
Thin-2	16	7282	94.37%	0.18	92.22%	0.23	96.67%	0.14
Thick-2	32	24,802	95.66%	0.15	93.11%	0.20	97.78%	0.10
Thin-2-Dense	16	7556	98.98%	0.04	97.31%	0.08	97.78%	0.04

4.3. Deep Learning Based Image Compression

Auto-encoders are neural networks that consist of an encoding part followed by a decoding part [19]. The encoder’s job is to extract the features from the input image, while the decoder’s job is to combine the features back to restore the original image. Auto-encoders learn what features to extract through training, where the auto-encoder aims to minimize the Mean Squared Error (MSE) between the input (original) image and the output (restored) image. The encoding part and the decoding part are trained together, but after that, they operate separately as the encoder is usually located at the transmitter and the decoder at the receiver. Using auto-encoders for image compression fits the operation on edge devices with custom thermal applications because they are less complex and more resource-efficient than other universal compression algorithms (e.g., JPEG) which are designed to compress a wide range of RGB visible images that have higher visual information content (i.e., higher entropy) than thermal images.

4.4. Proposed Autoencoder Design

The proposed autoencoder consists of Conv layers and downsampling/upsampling layers as shown in Figure 5. The first half of the autoencoder is the encoder (at the transmitter) and the second half is the decoder (at the receiver). The Conv layers in the encoder part extract the features of the input image and the downsampling layers reduce the size of the feature maps (for compression). The size of the encoder output is a 128×128 feature map compared to a 512×512 input image, which allows a 16:1 compression ratio. The compression ratio can be further increased by setting the lowest valued features on the output feature map to 0. The benefit of such variable-rate compression is that the transmitter has the freedom to change its compression rate based on many factors including the image significance and the battery energy level. This adjustment does not require any change in the architecture or extra communication between the transmitter and the receiver to inform the receiver of the new compression rate as the receiver still receives the same expected number of features but with some of them zeroed. The Conv layers in the decoder part combine the received features to recover the original image,

upsampling layers are used in the decoder to reverse the operation of the downsampling layers in the encoder. The autoencoder has a total of 3298 parameters. Table 2 shows the auto-encoder loss. Figure 6 shows a sample input image and the decoded images when different percentage of the encoder features are set to 0.

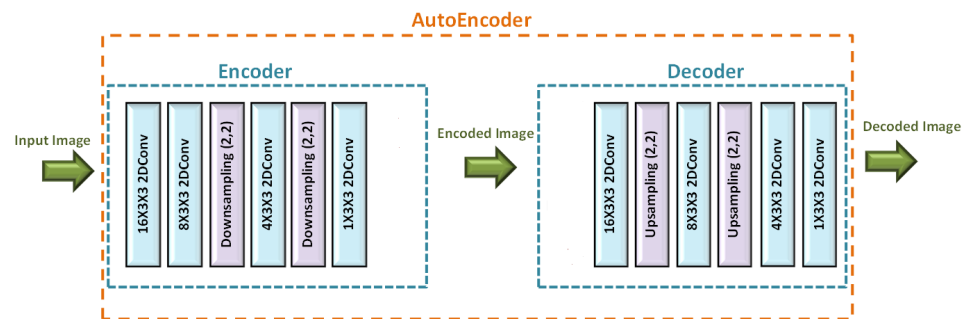


Figure 5. Autoencoder architecture.

Original Image	Decoded Image		
	No features set to 0	10% lowest valued features set to 0	20% lowest valued features set to 0
MSE	0.0003	0.002	0.003

Figure 6. Original and decoded images when different percentages of encoder output is set to zeros.

Table 2. AutoEncoder Losses on a core i7-4790 @ 3.6 GHz CPU.

Training Loss	Validation Loss	Test Loss
3.3328×10^{-4}	3.0254×10^{-4}	3.2446×10^{-4}

5. DPU Design Flow

5.1. DNN Acceleration

In terms of processing infrastructure, general-purpose processors, such as CPUs and GPUs, have been dominant for years now being deployed for DNN inference, especially the uncompressed DNN models in which the arithmetic operations are represented in floating-point matrix multiplications. However, thanks to the DNN approximation, this has opened the horizons to exploit custom hardware platforms such as Field Programmable Gate Arrays (FPGAs) and Application Specific Integrated Circuits (ASICs) to leverage the usage of fixed-point quantization, which results in faster inference with accuracy reduction less than 1% [20,21].

Since the need for faster and real-time inference is becoming more demanding, many hardware companies lean towards the custom hardware approach [22]. Concerning ASIC, Google’s Tensor Processing Unit (TPU) [23] represents a good example of a DNN accelerator implemented on ASIC as well as Intel Nervana [24]. On the other hand, the FPGA-based accelerators are found in Microsoft Brainwave [25] and Xilinx DPU [26].

Generally, the acceleration development flow takes one of two approaches to design and implement the accelerator; the first is to design the accelerator in Register-Transfer Level (RTL) using one of the famous Hardware Description Languages (HDL) such as VHDL or Verilog, or to describe the accelerator top-level design in C/C++ then high-level synthesize the code into synthesizable HDL. Table 3 summarizes the differences between

these two approaches in addition to highlighting the privileges of using the DPU as an alternative approach to reduce the accelerator design and development cycle.

Table 3. Comparison between Different Accelerator Development Flows [26–28].

	RTL	HLS	DPU
Time to Market	Slow	Faster due to using high-level language	Fastest due to using automated frameworks
Languages or Objects Supported	VHDL, Verilog	C/C++	DNN GraphDef Models, Python
Associated Xilinx Tool	Xilinx Vivado	Xilinx Vivado HLS	Xilinx Vitis AI Framework
Eventual Output	Bit file used to program FPGA	Unreadable RTL that follows the conventional RTL flow afterwards	Executable file that contains the network to run on the DPU

The DPU is an IP provided by Xilinx to accelerate DNN models, which provides a high level of parallelism, and energy efficiency that makes it an efficient alternative for IoT devices. Practically, the DPU is a programmable acceleration engine that is not dedicated to a specific CNN model or architecture. This has been achieved by a specialized instruction set that drives the DPU operations, which makes the DPU more generic to work efficiently for many CNN models. The developer is then required to convert the targeted CNN model from the famous DNN frameworks format, such as Tensorflow and Cafe, into the compatible one supported by the DPU engine. This conversion is carried out using a unified software stack called Vitis AI Framework provided by Xilinx. This AI framework is responsible for quantizing and pruning the model, compiling the model into the equivalent instructions that are supported by the DPU, and making performance profiling.

Figure 7 depicts the internal architecture of the DPU, which fundamentally consists of a scheduler module, Processing Engines (PEs), instruction unit block, and global memory pool module. The Application Processing Unit (APU) is the ARM processor, on which the application is running, serves interrupts and data transfer from and to the DPU. The instruction unit handles reading and executing the instructions associated with the different operations of the accelerated CNN. The Fetcher's main role is to fetch the DPU instructions associated with the model from the memory. Following that, the decoder is responsible for decoding the instructions to drive the PEs. The dispatcher is responsible for managing the data/instructions transfer among the PEs and the memory. The Global Memory Pool acts as a buffer for the input and output data as well as intermediate output from the DPU, which results in high throughput.

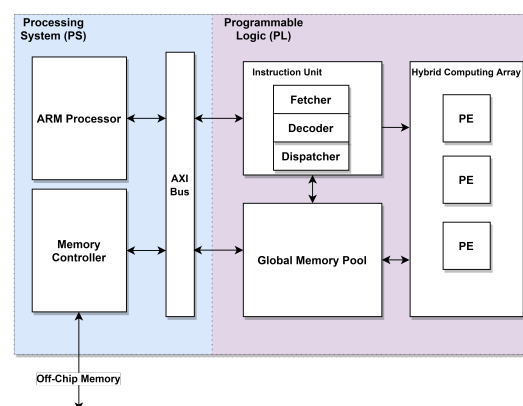


Figure 7. Xilinx Deep Learning Processing Unit (DPU) top-level architecture [10,26].

In this work, the unified DPU hardware accelerator is used to accelerate all the models proposed without making any modification to the original system setup. The accelerator is configured based on the battery energy level which makes the entire system energy adaptive, as shown in Figure 8.

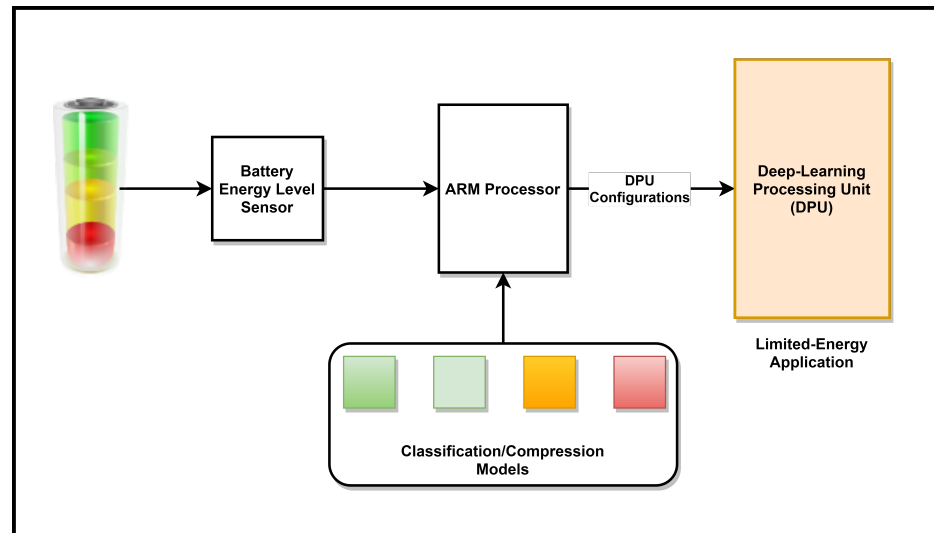


Figure 8. Energy adaptive DPU configurations.

5.2. Model Quantization and Compilation

Quantization is the process of representing the data in a smaller number of bits in a scheme that minimizes the error between the original and quantized data. This, accordingly, results in reducing the energy and storage cost area. Therefore, the memory access gets optimized since it dominates the energy consumption [5]. In this work, the quantization and channel pruning techniques are leveraged to address these issues while achieving high performance and high energy efficiency with little degradation in accuracy.

Within the Vitis AI stack, a specialized quantizer is used to convert the numerical representation of the model weights from 32-bit floating point to 8-bit fixed point precision. Furthermore, the Vitis AI quantizer prunes the network from the ineffective connections, which also enhances the overall accelerator performance.

Besides, the framework also entails a core component, which is the pruner. The pruner is responsible for lowering the number of model weights without resulting in significant precision loss. This happens through an iterative process that takes advantage of the redundant and near-zero parameters, which aims at reducing the number of computations in the model. However, this can cause accuracy loss, which can be overcome by the next stage which is the fine-tuning [29].

Vitis AI comes with a domain-specific compiler that maps the quantized model into the associated supported sequence of instructions that drive the DPU. This happens by recognizing each layer and convert it into equivalent instructions. By the end of such process, the main goal is to generate the kernels that shall be deployed on the FPGA and then used by some provided API functions to drive the accelerators.

In this work, the Vitis AI compiler is used to compile the classifiers and the compression encoder after quantizing them into 8-bit fixed-point precision. By nature, the compiler supports a range of different CNN layers that can be accelerated by the DPU. However, there are some layers that force the developer to implement them in software so that they will eventually get executed on the CPU which is denoted by Hardware-Software Co-Design. For example, the softmax activation function and the Global Average Pooling Layer are not supported by the DPU, and correspondingly, they have been implemented on the software side.

At the end of the process, the DPU kernels that contain the parameters of the separated regions are generated. In run-time, these kernels shall be loaded into the DPU ahead of

propagating the input image over the model. In order to perform this task, Vitis AI provides a uniform coded API that handles the execution of the model on the heterogeneous platform of the CPU and the DPU.

5.3. Vitis AI API

The main objective of the Vitis API functions is to configure the DPU for the CNN model desired to be accelerated. This includes reading the DPU instruction sequence of the model and loading weights in the DPU. Besides, the API functions are used to handle the data exchange between the CPU and the DPU, which allows the data to be pre-processed before being fed to the DPU or post-processed after carrying out the inference, which is accelerated by the DPU.

The Vitis API functions are programmed to load the different classifiers proposed in this work as well as the compression encoders based on several parameters, such as battery energy level, and ata transmission rate. The API propagates the images through the classifier on the DPU for accelerated inference, then takes the output data, which is then used to configure the DPU again with the compression encoder weights and instructions for accelerated compression. Finally, the API takes the compressed image out from the DPU, store it in the off-chip memory, then send it to the specified destination.

5.4. FPGA Implementation of DPU

The DPU hardware accelerator has been implemented on the targeted Xilinx Zynq UltraScale+ MPSoC ZCU104 FPGA Evaluation Kit. Resource utilization and power consumption are considered as the main hardware performance metrics. The DPU design parameters and configurations are elaborated in Table 4.

Table 4. Configuration parameters of the implemented DPU.

Parameter	Value
DPU Architecture	B4096
DPU Operational Frequency	300 MHz

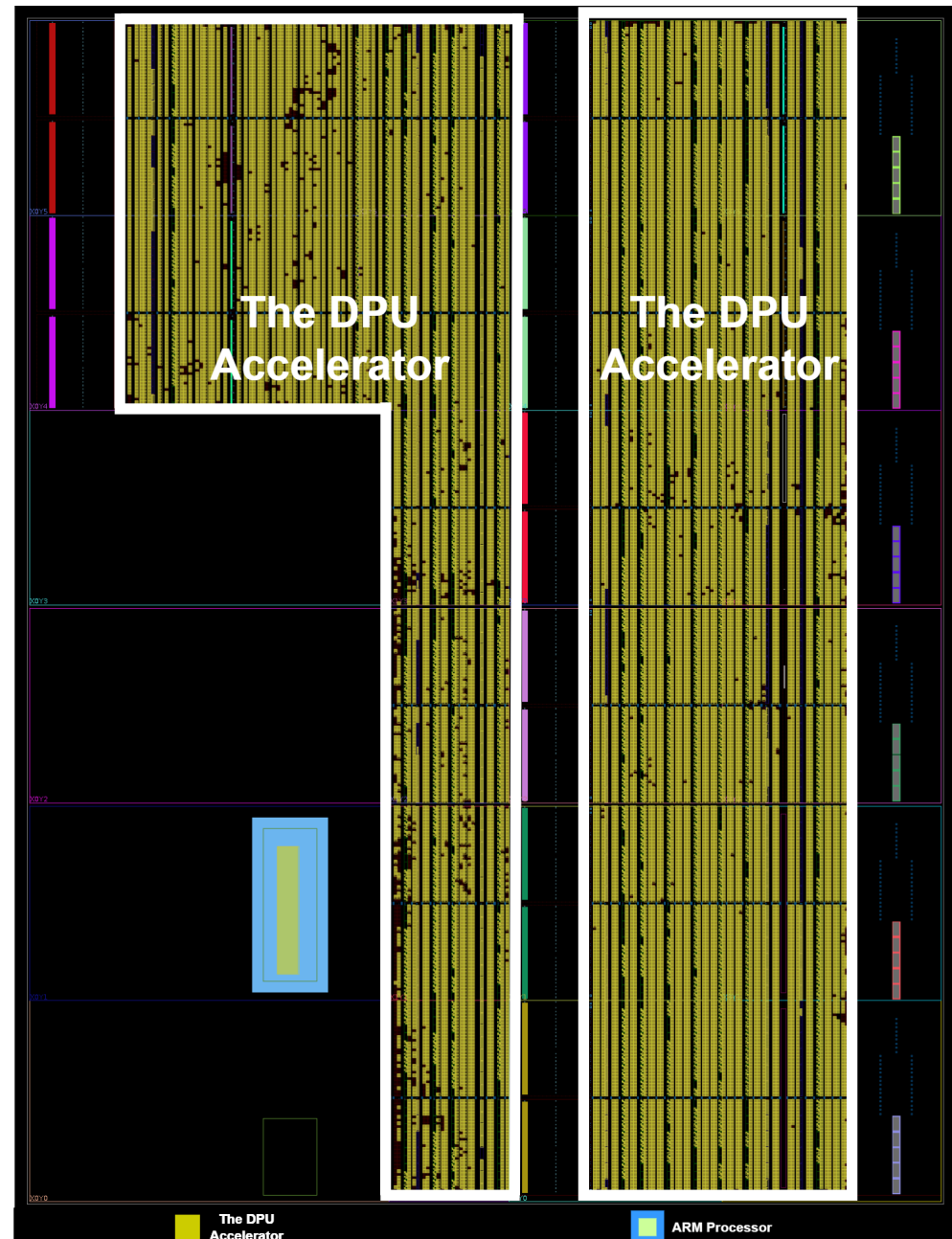
As demonstrated in Figure 7, the DPU is connected to the ARM processor on the FPGA chip to manage task scheduling and offloading weights and data to the DPU. The DPU is connected to the ARM core through an AXI Bus that carries data and weights to the DPU. In addition, Mixed-Mode Clock Manager (MMCM) block is used to generate required frequencies. Table 5 demonstrates the resources utilization of the DPU on the ZCU 104. The MMCM block consumes less than 1% of the total DPU resources. Table 6 shows the power consumption of the DPU at 300 MHz. The DPU power consumption and resource utilization are optimized by leveraging special UltraRAM slices [30]. The UltraRAM is a novel memory solution by Xilinx, which introduces higher memory speed with low energy consumption and resource utilization. The DPU operates with 4 MB memory attached to it coming from both URAM blocks as well as BRAM blocks. This amount of memory is enough for this application and for any future scaling of the system. The floorplan of the DPU is illustrated in Figure 9.

Table 5. DPU Resources Utilization on ZCU 104.

	CLB LUTS	CLB Registers	BRAM	DSP Slices
Total Resources	230.4 K	460.8 K	312	1.72 K
The DPU	103.7 K	198.9 K	290	1.38 K
Resource Utilization	45%	43%	92.9%	80.2%

Table 6. The DPU dynamic power consumption on ZCU 104 in Watts.

Utilization	Clocks	Signals	Data	BRAM	DSP
11.65	2.2	4	3.87	0.635	2.118

**Figure 9.** FPGA floorplan of the DPU accelerator.

6. System Setup and Experimental Results

6.1. System Setup

In these experimental results, Xilinx Zynq UltraScale+ MPSoC ZCU104 Evaluation Kit is used. Figure 10 shows the DPU hardware accelerator coupled with the ARM processor. On the top of the system, the five classifier models and encoder model reside and are managed by a custom API that uses Vitis AI library functions to manage the communications between the APU (The ARM Processor Unit) and the DPU accelerator. The Vitis AI API library communicates with the DPU driver to handle the data movement between both

sides of the system, which are all controlled by the Petalinux OS that runs on top of the ARM processor.

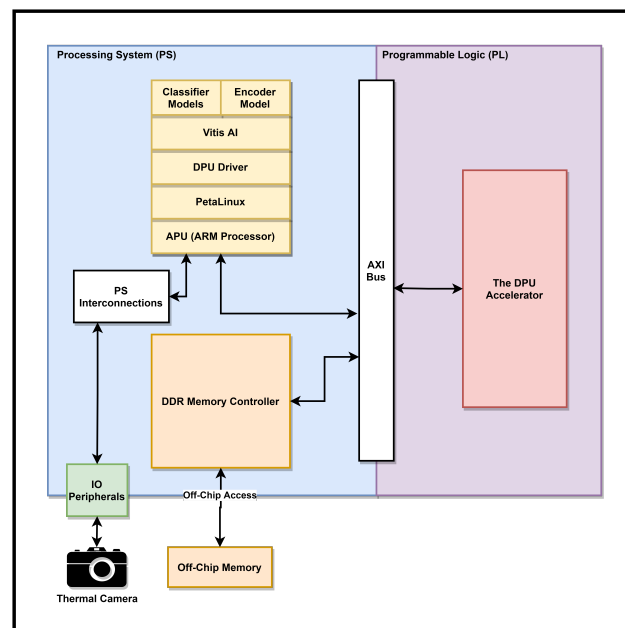


Figure 10. Proposed system block diagram.

The thermal camera sends a number of frames to the APU, which controls the entire system. The frames are then propagated one by one through the classifier in order to determine its class. The output of the inference process is then sent to the APU in order to configure the DPU to accelerate the encoding with the required compression parameters, based on the battery energy level and the transfer rate. Following that, the compressed frames are sent to the specified destination.

At the system level, the data are initially stored in the off-chip DDR-RAM, which is controlled by the DDR-Controller through the AXI-Bus. The system data is initially loaded in the off-chip memory before being moved to the on-chip DPU buffers as a part of configuring the DPU for acceleration. Once the DPU finishes a process, the data is taken from the output on-chip buffers back to memory for any desired post-processing. The data include the model input image, with a size of is 512×512 , the model weights and biases, and model associated instructions.

On start-up, the DPU fetches the model associated instructions from the off-chip memory, which are then used to control the PEs inside the Hybrid Computing Array shown in Figure 7. The data is then loaded in the DPU to accelerate the inference (in case of classification) or the compression (in case of accelerating the encoder). To reduce the overall memory bandwidth, the data remain used as much as possible, which enhances the overall performance of the system in terms of latency as well as energy consumption.

Furthermore, the API takes an adaptive approach, based on a set of decision parameters, that picks which classifiers and which encoder compression rate should be used to configure the DPU.

6.2. Experimental Results and Discussions

As the main objective of this flow is to accelerate the inference process while maintaining the lowest accuracy loss, the performance of the proposed system is characterized with respect to certain trade-offs. The performance results are shown in Tables 7 and 8 for the classifiers and the compression engine, respectively, on both CPU and DPU. The performance (i.e., accuracy, speed, and energy) of each model has been evaluated on Intel Core i7-4790 that operates at 3.6 GHz with 16 GB RAM coupled. In this context, the Intel processor is referred to as the CPU platform.

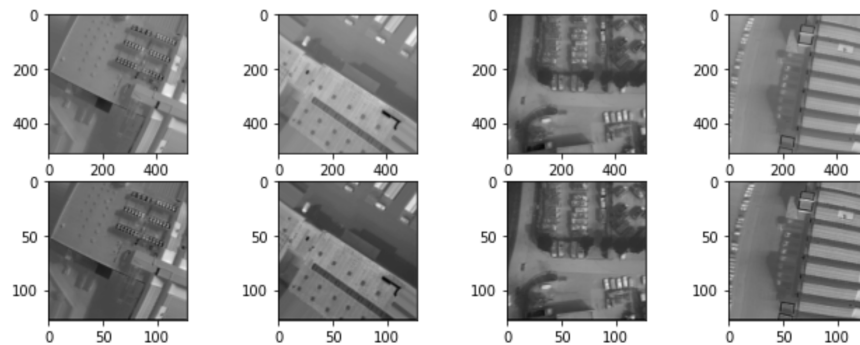
Table 7. Classifiers Implementation parameters.

Classifier	Quantization			Throughput (Image/Sec)			Energy (J/Image)		
	Pre-Accuracy	Post-Accuracy	Loss	CPU	DPU	Speedup	CPU	DPU	Reduction
Thick-1	76.67%	75.56%	1.45	5.2	13.26	2.55×	16.154	0.884	94.53%
Thin-1-Dense	80%	67.78%	15.28	12	24.04	2×	7	0.488	93.03%
Thin-2	96.67%	84.44%	12.65	8	38.46	4.81×	10.5	0.304	97.10%
Thick-2	97.78%	86.67%	11.36	3.5	22.32	6.38×	24	0.525	97.81%
Thin-2-Dense	97.78%	93.33%	4.55	8	38.61	4.83×	10.5	0.304	97.10%

Table 8. Compression Engine Performance on CPU vs. DPU.

Throughput (Image/Sec)			Energy (J/Image)		
CPU	DPU	Speedup	CPU	DPU	Reduction
22.2	123.46	5.56×	3.784	0.095	97.49%

First, the post-quantization accuracy for each model was evaluated using the evaluation model generated by the Vitis AI quantizer using a test set that has 90 test cases. On the other hand, the encoder accuracy is evaluated by estimating the Mean-Squared Error (MSE) between the compressed images before and after quantization based on the evaluation model. In such scenario, the MSE between the original and post-quantization encoder models is 7.194×10^{-5} . Additionally, Figure 11 highlights the differences between the compressed images generated by the original and quantized encoders.

**Figure 11.** Encoder compressed output image before quantization on the first row and post-quantization encoder output on the second.

Furthermore, the system throughput is evaluated by the overall inference time on both CPU (Intel Processor) and the FPGA to show the hardware acceleration benefits. The Intel CPU power consumption is estimated to be 84 W [31]. As demonstrated in Table 6, the total power consumption of the DPU is measured and equals 11.65 W.

6.2.1. Classifiers

Several trade-offs should be considered in the hardware accelerator design in the proposed system. Firstly, the accuracy loss that results from quantization should be evaluated for each model. In this context, the Thick-1 model has achieved the least accuracy loss of all models, which is nearly 1.45%. On the other hand, the Thin-1-Dense model has achieved the least post-quantization accuracy and accuracy loss which are about 68% and 15.3%, respectively. Moreover, the Thick-2 model has achieved the highest accuracy among all models before and after quantization. The speed increase of the classifiers is ranged from 2× to about 6.4× compared to the CPU (Intel Processor). The former belongs

to Thin-1-Dense, while the latter is associated with the Thick-2 model. Figure 12 reflects the gap between the throughput on the DPU compared to the CPU. In Figure 13, the number of parameters and the speed increase achieved by the accelerator is plotted. It can be observed that a direct proportion can describe the effect of the number of model parameters on the DPU to accelerate the model inference.

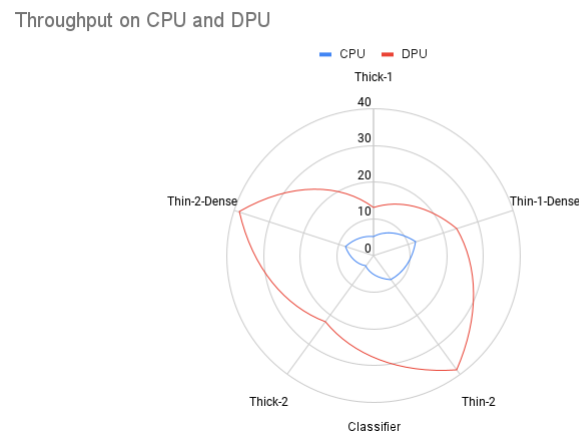


Figure 12. Gap between throughput on CPU and DPU.

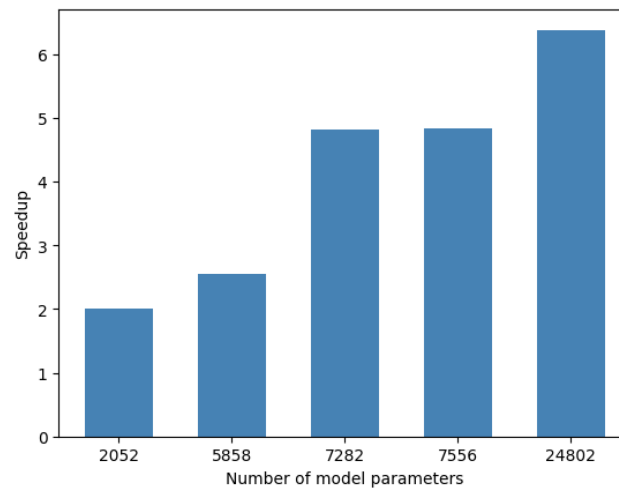


Figure 13. Relationship between the number of model parameters and speed increase factor after deployment on the DPU.

In terms of energy consumption, the DPU achieves a significant reduction in the energy consumed during inference time, which saves at least 93% of its counterpart on the CPU, whereas the maximum energy saving achieved has been 97.8%. As a result, this makes all the models suitable for IoT applications. Based on the battery energy level, one model is configured to the DPU trading-off accuracy to lengthen the battery life [32]. The gap between the energy consumption on the CPU and the DPU can be realized from Figure 14.

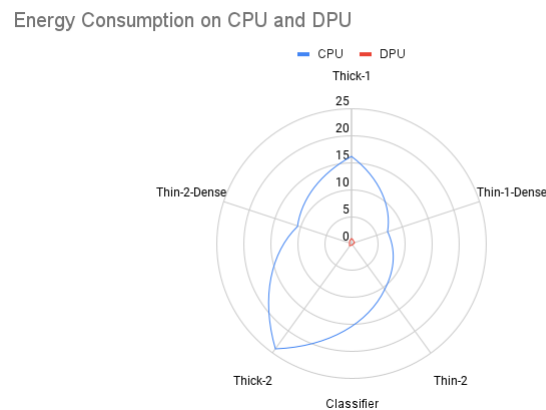


Figure 14. Gap between energy consumption on CPU and DPU.

6.2.2. Compression Encoder

The CNN-based encoder achieves the highest throughput and lowest energy consumption per image compression process, compared to the classifiers. The MSE between the original and the quantized models is 7.194×10^{-5} , which is considered insignificant. Hereby, the compressed images should be capable of inheriting the main features of the original images while being significantly compressed. This can be observed by the encoder output images in Figure 11.

6.3. System Limitations

Despite the fact that the proposed system handles a variety of CNN layers in an energy-adaptive manner, the system still tackles some limitations. Firstly, the current system does not fully leverage multi-core DPU implementation to accelerate one stage of the system on one core and the other stage on the other core. Furthermore, the CNN compiler associated with the DPU (i.e., Vitis AI compiler) does not support some operations like Global Average Pooling and Softmax, hence, these operations have to be handled by the CPU. In such case, the data is moved from the DPU memory to the CPU memory so that the CPU executes these operations then send it back again to the DPU memory to resume the acceleration flow. Hereby, this results in an additional overhead on the system.

7. Conclusions

In this paper, the implementation of an intelligent thermal imaging classification and compression is discussed. The system is adaptive to a set of factors, such as energy budget. Image classification is carried out using different CNN architectures with an average accuracy drop of 4.22% and an average speed increase of $4.11\times$ after deployment on the DPU. A compression model is proposed which compresses the image from 512×512 into 128×128 pixels using an auto-encoder architecture. After training the auto-encoder, the encoder part, which is used to compress the image, is deployed to the DPU with a speed increase of $5.56\times$. The compression model can be integrated with another classification model for more optimization in computations and memory usage. The system performs faster than commercial CPUs, which led to a more energy-efficient behavior per frame. Furthermore, in terms of speed increase, it was found that the number of the parameters of the proposed models depicted had a significant impact on the acceleration speed increase rate. In the future, the system can be extended to support manual process assignment to each of the DPU cores so that a higher level of parallelism can be leveraged. Additionally, in order to achieve better optimization, it is important to investigate the software stack of the Vitis-AI to enhance the quantizer and the compiler for more efficient task scheduling.

Author Contributions: Conceptualization, Y.F., H.M. and K.N.S.; methodology, A.S.H., A.A., Y.F., H.M., K.N.S. and M.K. software and validation, A.S.H., A.A. and M.K.; writing—original draft preparation, A.S.H. and M.K.; writing—review and editing, Y.F., H.M. and K.N.S.; supervision, Y.F., H.M. and K.N.S.; project administration, Y.F. and H.M. All authors have read and agreed to the published version of the manuscript.

Funding: This work is funded by the Information Technology Industry Development Agency (ITIDA), Information Technology Academia Collaboration (ITAC) Program, Egypt—Grant Number PRP2019.R26.2.

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

Appendix A. Parameters of the Proposed Image Classifier and Autoencoder

The parameters of the proposed image classifier models in Figure 3 are presented in Tables A1 and A2. Moreover, the parameters of the proposed auto-encoder in Figure 5 are presented in Table A3.

Table A1. Parameters of the Image Classification Unit: models Thick-1 and Thin-1-Dense.

Model	Thick-1		Thin-1-Dense	
Layer	Description	#Parameters	Description	#Parameters
Zero Pad.	1 pixel	—	1 pixel	—
2D Conv, Stride = 2	32 Filter, $3 \times 3 \times 1$ Kernel	288	16 Filter, $3 \times 3 \times 1$ Kernel	144
Batch Norm.	—	128	—	64
2D Conv, Stride = 1	64 Filter, $1 \times 1 \times 32$ Kernel	2048	32 Filter, $1 \times 1 \times 16$ Kernel	512
Batch Norm.+ ReLU6	—	256	—	128
Depthwise 2D Conv Stride = 1	64 Filter, $3 \times 3 \times 1$ Kernel	576	32 Filter, $3 \times 3 \times 1$ Kernel	288
Batch Norm.+ ReLU6	—	256	—	128
2D Conv, Stride = 1	32 Filter, $1 \times 1 \times 64$ Kernel	2048	16 Filter, $1 \times 1 \times 32$ Kernel	512
Batch Norm.	—	128	—	64
Concat.	—	—	—	—
Global Avg. Pool	—	—	—	—
Dense+ReLU	N/A	N/A	6 Neurons	198
Dense+SoftMax	2 Neurons	130	2 Neurons	14

Table A2. Parameters of the Image Classification Unit: models Thin-2-Dense and Thick(Thin)-2.

Model	Thin-2-Dense		Thick(Thin)-2	
Layer	Description	#Parameters	Description	#Parameters
Zero Pad.	1 pixel	—	1 pixel	—
2D Conv, Stride = 2	16 Filter, $3 \times 3 \times 1$ Kernel	144	32 (16) Filter, $3 \times 3 \times 1$ Kernel	288 (144)
Batch Norm.	—	64	—	128 (64)
2D Conv, Stride = 1	32 Filter, $1 \times 1 \times 16$ Kernel	512	64 (32) Filter, $1 \times 1 \times 32$ (16) Kernel	2048 (512)
Batch Norm.+ ReLU6	—	128	—	256 (128)
Depthwise 2D Conv Stride = 1	32 Filter, $3 \times 3 \times 1$ Kernel	288	64 (32) Filter, $3 \times 3 \times 1$ Kernel	576 (288)
Batch Norm.+ ReLU6	—	128	—	256 (128)

Table A2. Cont.

Model	Thin-2-Dense		Thick(Thin)-2	
Layer	Description	#Parameters	Description	#Parameters
2D Conv, Stride = 1	16 Filter, $1 \times 1 \times 32$ Kernel	512	32 (16) Filter, $1 \times 1 \times 64$ (32) Kernel	2048 (512)
Batch Norm.	—	64	—	128 (64)
Concat.	—	—	—	—
2D Max Pool.	2×2	—	2×2	—
2D Conv, Stride = 1	64 Filter, $1 \times 1 \times 32$ Kernel	2048	128 (64) Filter, $3 \times 3 \times 64$ (32) Kernel	8192 (2048)
Batch Norm.+ RelU6	—	256	—	512 (256)
Depthwise 2D Conv Stride = 1	64 Filter, $3 \times 3 \times 1$ Kernel	576	128 (64) Filter, $3 \times 3 \times 1$ Kernel	1152 (576)
Batch Norm.+ RelU6	—	256	—	512(256)
2D Conv, Stride = 1	32 Filter, $1 \times 1 \times 64$ Kernel	2048	64 (32) Filter, $1 \times 1 \times 128$ (64) Kernel	8192 (2048)
Batch Norm.	—	128	—	256 (128)
Concat.	—	—	—	—
Global Avg. Pool	—	—	—	—
Dense+RelU	2 Neurons	390	N/A	N/A
Dense+SoftMax	2 Neurons	14	2 Neurons	258 (130)

Table A3. Parameters of the Image Compression Unit:Auto-encoder.

Encoder-Part		
Layer	Description	#Parameters
2D Conv, Stride = 1	16 Filters, $3 \times 3 \times 1$ Kernel, Bias = True, Activation = Tanh	160
2D Conv, Stride = 1	8 Filters, $3 \times 3 \times 16$ Kernel, Bias = True, Activation = Tanh	1160
2D Max Pool	2×2	—
2D Conv, Stride = 1	4 Filters, $3 \times 3 \times 8$ Kernel, Bias = True, Activation = Tanh	292
2D Max Pool	2×2	—
2D Conv, Stride = 1	1 Filters, $3 \times 3 \times 4$ Kernel, Bias = True, Activation = Tanh	37
Dncoder-Part		
Layer	Description	#Parameters
2D Conv, Stride = 1	16 Filters, $3 \times 3 \times 1$ Kernel, Bias = True, Activation = Tanh	160
2D upsampling	2×2	—
2D Conv, Stride = 1	8 Filters, $3 \times 3 \times 16$ Kernel, Bias = True, Activation = Tanh	1160
2D upsampling	2×2	—
2D Conv, Stride = 1	4 Filters, $3 \times 3 \times 8$ Kernel, Bias = True, Activation = Tanh	292
2D upsampling	2×2	—
2D Conv, Stride = 1	1 Filters, $3 \times 3 \times 4$ Kernel, Bias = True, Activation = Tanh	37

References

1. Krišto, M.; Ivašić-Kos, M. Thermal imaging dataset for person detection. In Proceedings of the 2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, Croatia, 20–24 May 2019; pp. 1126–1131.
2. Santangeli, A.; Chen, Y.; Klueen, E.; Chirumamilla, R.; Tiainen, J.; Loehr, J. Integrating drone-borne thermal imaging with artificial intelligence to locate bird nests on agricultural land. *Sci. Rep.* **2020**, *10*, 10993. [CrossRef] [PubMed]
3. Vadivelu, S.; Ganesan, S.; Murthy, O.R.; Dhall, A. Thermal imaging based elderly fall detection. In Proceedings of the Asian Conference on Computer Vision, Taipei, Taiwan, 20–24 November 2016; Springer: Cham, Switzerland, 2016; pp. 541–553.
4. Kafafy, M.; Fahmy, Y. Joint coding bit-rate and activity rate optimisation in wireless visual sensor networks. *IET Commun.* **2020**, *14*, 3428–3439. [CrossRef]
5. Sze, V.; Chen, Y.H.; Yang, T.J.; Emer, J.S. Efficient processing of deep neural networks: A tutorial and survey. *Proc. IEEE* **2017**, *105*, 2295–2329. [CrossRef]
6. Qasaimeh, M.; Denolf, K.; Khodamoradi, A.; Blott, M.; Lo, J.; Halder, L.; Vissers, K.; Zambreno, J.; Jones, P.H. Benchmarking vision kernels and neural network inference accelerators on embedded platforms. *J. Syst. Archit.* **2021**, *113*, 101896. [CrossRef]
7. 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.
8. Redmon, J.; Farhadi, A. Yolov3: An incremental improvement. *arXiv* **2018**, arXiv:1804.02767.
9. Chan, C.W.H.; Leong, P.H.; So, H.K.H. Vision Guided Crop Detection in Field Robots using FPGA-Based Reconfigurable Computers. In Proceedings of the 2020 IEEE International Symposium on Circuits and Systems (ISCAS), Sevilla, Spain, 10–21 October 2020; pp. 1–5.
10. 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]
11. SenseFly Dataset. Available online: <https://www.sensefly.com/education/datasets/?sensors%5B%5D=26> (accessed on 30 October 2021).
12. Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; Rabinovich, A. Going deeper with convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Boston, MA, USA, 7–12 June 2015; pp. 1–9.
13. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.
14. Huang, G.; Liu, Z.; Van Der Maaten, L.; Weinberger, K.Q. Densely connected convolutional networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 4700–4708.
15. Howard, A.G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv* **2017**, arXiv:1704.04861.
16. Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; Chen, L.C. Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 4510–4520.
17. Khan, A.; Sohail, A.; Zahoor, U.; Qureshi, A.S. A survey of the recent architectures of deep convolutional neural networks. *Artif. Intell. Rev.* **2020**, *53*, 5455–5516. [CrossRef]
18. Bianco, S.; Cadene, R.; Celona, L.; Napolitano, P. Benchmark analysis of representative deep neural network architectures. *IEEE Access* **2018**, *6*, 64270–64277. [CrossRef]
19. Bank, D.; Koenigstein, N.; Giryes, R. Autoencoders. *arXiv* **2020**, arXiv:cs.LG/2003.05991.
20. 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 44th Annual International Symposium on Computer Architecture, Toronto, ON, Canada, 24–28 June 2017; pp. 1–12.
21. Shen, J.; Huang, Y.; Wang, Z.; Qiao, Y.; Wen, M.; Zhang, C. Towards a uniform template-based architecture for accelerating 2D and 3D CNNs on FPGA. In Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 25–27 February 2018; pp. 97–106.
22. Wang, E.; Davis, J.J.; Zhao, R.; Ng, H.C.; Niu, X.; Luk, W.; Cheung, P.Y.; Constantinides, G.A. Deep Neural Network Approximation for Custom Hardware: Where We’ve Been, Where We’re Going. *ACM Comput. Surv. (CSUR)* **2019**, *52*, 1–39. [CrossRef]
23. Jouppi, N.P.; Young, C.; Patil, N.; Patterson, D. A domain-specific architecture for deep neural networks. *Commun. ACM* **2018**, *61*, 50–59. [CrossRef]
24. Intel AI. *Intel Nervana Neural Network Processors (NNP) Redefine AI Silicon*; Intel: Santa Clara, CA, USA, 2017.
25. Chung, E.; Fowers, J.; Ovtcharov, K.; Papamichael, M.; Caulfield, A.; Massengil, T.; Liu, M.; Lo, D.; Alkalay, S.; Haselman, M.; et al. Accelerating persistent neural networks at datacenter scale. *Hot Chips* **2017**, *29*. Available online: https://old.hotchips.org/wp-content/uploads/hc_archives/hc29/Hc29.22-Tuesday-Pub/Hc29.22.60-NeuralNet1-Pub/Hc29.22.622-Brainwave-Datacenter-Chung-Microsoft-2017_08_11_2017.pdf (accessed on 8 January 2021).
26. Xilinx Zynq DPU v3.1 Product Guide. Available online: https://www.xilinx.com/support/documentation/ip_documentation/dpu/v3_1/pg338-dpu.pdf (accessed on 8 January 2021).

27. Xilinx Vivado Design Suite User Guide—High-Level Synthesis. Available online: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_2/ug902-vivado-high-level-synthesis.pdf (accessed on 8 January 2021).
28. Xilinx Vitis AI User Guide. Available online: https://www.xilinx.com/support/documentation/sw_manuals/vitis_ai/1_2/ug1414-vitis-ai.pdf (accessed on 8 January 2021).
29. Xilinx. *Vitis AI Optimizer User Guide*; XILINX: San Jose, CA, USA, 7 July 2020.
30. Xilinx. *White Paper: UltraRAM: Breakthrough Embedded Memory Integration on UltraScale + Devices*; Technical Report WP477; XILINX: San Jose, CA, USA, 14 June 2016.
31. Intel® Core™ i7-4790 Processor. Available online: <https://ark.intel.com/content/www/us/en/ark/products/80806/intel-core-i7-4790-processor-8m-cache-up-to-4-00-ghz.html> (accessed on 15 December 2021).
32. Hassan, S.; Attia, S.; Salama, K.N.; Mostafa, H. EANN: Energy Adaptive Neural Networks. *Electronics* **2020**, *9*, 746. [CrossRef]