Hindawi
International Journal of Reconfigurable Computing
Volume 2019, Article ID 7218758, 13 pages
https://doi.org/10.1155/2019/7218758

Research Article

An FPGA-Based Hardware Accelerator for CNNs Using On-Chip Memories Only: Design and Benchmarking with Intel Movidius Neural Compute Stick

Gianmarco Dinelli,[1] **Gabriele Meoni**,[1] **Emilio Rapuano**,[1] **Gionata Benelli**,[2] **and Luca Fanucci**[1]

[1]*Department of Information Engineering, University of Pisa, Pisa 56122, Italy*
[2]*IngeniArs, Pisa 56121, Italy*

Correspondence should be addressed to Gianmarco Dinelli; gianmarco.dinelli@ing.unipi.it

Received 2 May 2019; Revised 3 September 2019; Accepted 3 October 2019; Published 22 October 2019

Academic Editor: Martin Margala

During the last years, convolutional neural networks have been used for different applications, thanks to their potentiality to carry out tasks by using a reduced number of parameters when compared with other deep learning approaches. However, power consumption and memory footprint constraints, typical of on the edge and portable applications, usually collide with accuracy and latency requirements. For such reasons, commercial hardware accelerators have become popular, thanks to their architecture designed for the inference of general convolutional neural network models. Nevertheless, field-programmable gate arrays represent an interesting perspective since they offer the possibility to implement a hardware architecture tailored to a specific convolutional neural network model, with promising results in terms of latency and power consumption. In this article, we propose a full on-chip field-programmable gate array hardware accelerator for a separable convolutional neural network, which was designed for a keyword spotting application. We started from the model implemented in a previous work for the Intel Movidius Neural Compute Stick. For our goals, we appropriately quantized such a model through a bit-true simulation, and we realized a dedicated architecture exclusively using on-chip memories. A benchmark comparing the results on different field-programmable gate array families by Xilinx and Intel with the implementation on the Neural Compute Stick was realized. The analysis shows that better inference time and energy per inference results can be obtained with comparable accuracy at expenses of a higher design effort and development time through the FPGA solution.

1. Introduction

During the last years, convolutional neural networks (CNNs) found application in many different fields like object detection [1, 2], object recognition [3, 4], and KeyWord Spotting (KWS) [5, 6]. Although they proved excellent results on cloud, their applicability for portable systems is challenging because of the additional constraints in terms of memory footprint and power consumption, which generally conflict with latency and accuracy requirements. In particular, in general purpose solutions based on the use of a microcontroller, the limited available memory limits the complexity of the network, with possible impact on the accuracy of the system [7]. In the same way, microcontroller-based systems feature the worst trade-off between power consumption and timing performances [8].

For this reason, commercial hardware accelerators for CNNs such as Neural Compute Stick (NCS) [9], Neural Compute Stick 2 (NCS2) [9], and Google Coral [10] were produced. Such products feature optimized hardware architectures that allow to realize inferences of CNN models with low latency and reduced power consumption. Standard communication protocols, such as Universal Serial Bus (USB) 3.0., are generally exploited for communication purposes.

Nevertheless, since they were designed for the implementation of generic CNNs, their architectures are extremely flexible at the expense of the optimization of the single model.

For such a reason, hardware accelerators customized for a specific application might offer an interesting alternative for accelerating CNNs. In particular, field-programmable gate arrays (FPGAs) represent an interesting trade-off between cost, flexibility, and performances [11], especially for applications whose architectures have been changing too rapidly to rely on application-specific integrated circuits (ASICs) and whose production volumes might be not sufficient. FPGAs offer high flexibility at the same time, which permits the implementation of different models with a high degree of parallelism [8] and the possibility of customizing the architecture for a specific application.

The aim of this paper is to investigate the use of custom FPGA-based hardware accelerators to realize a CNN-based KWS system, analysing their performances in terms of power consumption, number of hardware resources, accuracy, and timing. A KWS system represents an example of application whose porting on the edge requires much effort, owing to the hard design trade-offs.

The study involves the use of different FPGA families by Xilinx and Intel, analysing design portability on devices with different sizes and performances. This allowed to realize a benchmark that compares the obtained results with the ones presented in our previous work for the full-SCNN (separable convolutional neural network) model [12], which implements the same architecture exploiting a NCS (version 1, mounting Myriad 2 Vision Processing Unit (VPU)).

To realize the architecture implemented on-board FPGA, a bit-true simulation was performed to appropriately quantize the model, reducing the number of resources used, saving power, and increasing throughput when compared with a floating-point approach.

The remainder of the paper is structured as follows: the *Keras* model used to describe the KWS system is presented in Section 2. Section 3 presents the approach used to quantize and compress the model to optimize its implementation on-board FPGAs. In Section 4, the results of the quantization analysis are provided and discussed. The preferred FPGA-based accelerator architecture is then described in Section 5, focusing on the analysis of design trade-offs. Results of the implementation on the different FPGA families are presented in Section 6. In Section 7, results in terms of maximum achievable clock frequency, hardware resources, and power consumption are presented and compared with the NCS solution. In Section 8, the usability of FPGA devices to accelerate the inference of CNNs is discussed with respect to the presented solution and similar applications. Finally, in Section 9, conclusions are given.

2. Architecture of the KWS System

KWS systems are a common component in speech-enabled devices: they continuously listen to the surrounding environment with the task to recognize a small set of simple commands in order to activate or deactivate specific functionalities. Commercial examples of KWS systems include "*OK Google*" and "*Hey Siri*." The proposed KWS system is designed to operate inside a domotic installation for improving the quality of life of people with disabilities. In particular, it is able to recognize 10 different commands: "yes," "no," "up," "down," "left," "right," "on," "off," "stop," and "go." Moreover, it identifies two additional classes: "silence," when no word is pronounced, and "unknown," when the pronounced word does not belong to any class.

The KWS system was pretrained in the Python framework called *Keras* [13], using *Google Speech Command dataset*.

The proposed architecture is based on the SCNN described in [12], whose architecture is shown in Figure 1.

The input of the network is a 63×13 mel frequency spectral coefficient (MFSC) matrix [14]. The bin (n, k) of the matrix contains information over the spectral content at frequency f, as shown in equation (1):

$$f = k \cdot \frac{f_{\text{sample}}}{N+1}, \quad \text{for } k = 1, \ldots, K, \tag{1}$$

where $f_{\text{sample}} = 16\,\text{kHz}$ is the sample rate and $N = 512$ (32 ms) is the number of bins used to calculate the fast fourier transform (FFT), measured at the instant n/f_{sample}, with $n \in [0, N-1]$. Every N-sample window is weighted through a Hann window and overlapped with the previous $N/2$ samples for the calculation of the FFT.

The input layer provides the 63×13 MFSC input matrix. Then, three separable convolutional (SC) layers follow, and their generic structure is shown in Figure 2.

SC layers improve standard convolutional layers by reducing the number of parameters used to process the inputs [12]. For this reason, SCNNs are particularly interesting for the realization of FPGA-based hardware accelerators because they reduce memory and computation requirements in comparison with the classic CNN approach.

A standard convolutional layer contains $c_{\text{out}}(w_f x h_f)$ filters that are convolved over.

$c_{\text{in}}(w_{\text{cin}} x h_{\text{cin}})$ input channels, producing $c_{\text{out}}(w_{\text{cin}} - w_f + 1) x (h_{\text{cin}} - h_f + 1)$ output channels. On the contrary, a separable convolution is realized through two distinct convolutions performed by means of filters, whose dimensions are, respectively, $(f_w x 1)$ and $(1 x f_h)$. Figure 3 better illustrates the difference between these two approaches.

Considering the structure of the MFCS input matrix, each SC layer performs two separated convolutions, realizing a "*time*" convolution followed by a "*frequency*" convolution.

A batch normalization (BN) layer, which has the role to accelerate deep network training by reducing internal covariance shift [15], follows the frequency convolution. Finally, the rectified linear unit (ReLU) is the activation function of each SC layer. ReLU is defined in equation (2) as

$$f_{\text{ReLU}}(x) = \begin{cases} 0, & \text{when } x \le 0, \\ k \cdot x, & \text{when } x > 0 \text{ with } k \in R. \end{cases} \tag{2}$$

A classic convolutional layer follows the three SC layers.

Table 1 summarizes the dimension of time/frequency filters, number of input channels (C_{in}), output channels (C_{out}), and input/output matrix dimensions for each convolutional layer of the network. Time_0 and freq_0 are,

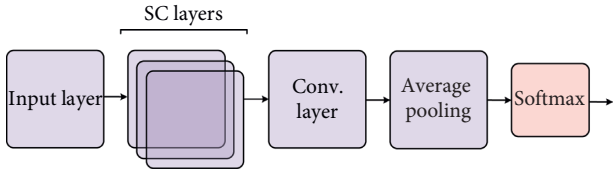

Figure 1: SCNN architecture.

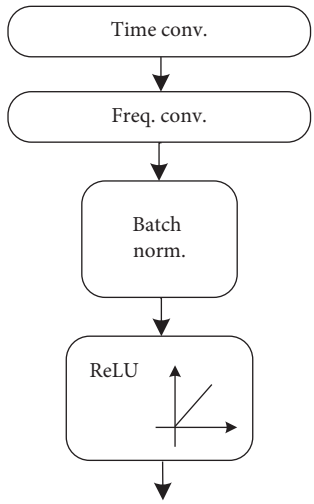

Figure 2: SC hidden layer architecture.

respectively, the temporal and frequency convolutional layer of the hidden layer 0, and similarly time_1/freq_1 for the hidden layer 1 and time_2/freq_2 for the hidden layer 2. Final_conv refers to the last convolutional layer of the network.

The average pooling layer computes the average value of each output channel of the final_conv layer, condensing them in 12 values, one for each class of the KWS system. Finally, a Softmax (or normalized exponential function) layer activation function follows. It takes a vector Z_J as input and produces an output vector in which each element $f_{\text{softmax}}(Z_J)$ is normalized in the interval [0, 1] and can be interpreted as the probability that input belongs to the class j. The standard Softmax function is described by equation (3):

$$f_{\text{softmax}}\left(Z_J\right) = \frac{e^{z_j}}{\sum_{i=1}^{K} e^{z_j}}, \quad \text{for } j = 1, \ldots, K. \tag{3}$$

In this network, the Softmax input vector is composed of 12 elements, one for each of the class of the KWS system.

The proposed SCNN model was implemented on the Intel Movidius NCS, showing an accuracy of 87.77%. The number of parameters necessary for its implementation is 15000, including bias, weights, and batch normalization parameters.

3. Keras Model Optimization toward the Hardware Implementation

In the next sections, methods to map the Keras–Python model of the KWS system on an FPGA are analysed. In fact, this model is implemented in a high-level language and its parameters are based on the floating-point representation.

The main issue about the implementation of a CNN-based model on an FPGA regards the limitation in terms of available hardware resources (combinatorial elements, sequential elements, Digital Signal Processors (DSPs), ram blocks, etc.) of such devices [11, 16, 17]. CNN algorithms are based on Multiply-and-ACcumulate (MAC) operations that require a large amount of combinatorial logic elements or DSPs. Furthermore, CNNs are characterised by a great number of parameters that shall be stored into off-chip memories if exceeding the available on-chip memory. The use of off-chip memory could be inevitable, complicating the design and increasing the inference time. For these reasons, the architecture of the hardware accelerator was carefully designed considering the trade-off between inference time and available resources.

3.1. Model Quantization. Before realizing the FPGA implementation, a quantization of the SCNN model was performed. In literature, there are many examples of quantization applied to CNNs [18–21]. The main advantage offered by a fixed-point representation is the possibility to shrink the model dimension and complexity with a negligible loss in accuracy [22]. In addition, fixed-point arithmetic requires simpler calculation than floating-point arithmetic, with advantages in terms of complexity and power consumption [23].

The quantization of the original floating-point model was performed through a bit-true simulation. The aim of the simulation is to determine the number of bits necessary to represent numbers in every internal node of the network by limiting the loss in accuracy.

The fixed-point representation of the model weights (or filter elements) w_q was calculated by using the approach described by the following equation:

$$\begin{cases} w_q = \text{round}\left(\dfrac{w}{\text{lsb}_w}\right) \cdot \text{lsb}_w, \\ \\ \text{lsb}_w = \dfrac{|w|_{\max}}{2^{b_w - 1}}, \end{cases} \tag{4}$$

where w is the floating-point representation of the weight and lsb_w is the value of the least significant bit (lsb). The latter is calculated by dividing $|w|_{\max}$, which represents the absolute value of the maximum weight over each layer, by 2^{b_w-1}, where b_w is the number of bits used to represent weights, as required by the 2's complement format. In particular, since the range of weights amplitude is roughly the same for every layer, the same value of $|w|_{\max}$ was used for each layer. Such choices are due to the necessity to reduce the conspicuous degrees of freedom in the simulation. Furthermore, in order to reduce the number of operations to implement in hardware, the effects of the BN are included in weight and bias values (BN simply consists in algebraic operations). In formulas, each weight $w(i)$ and each bias $b(i)$ belonging to a frequency convolutional layer

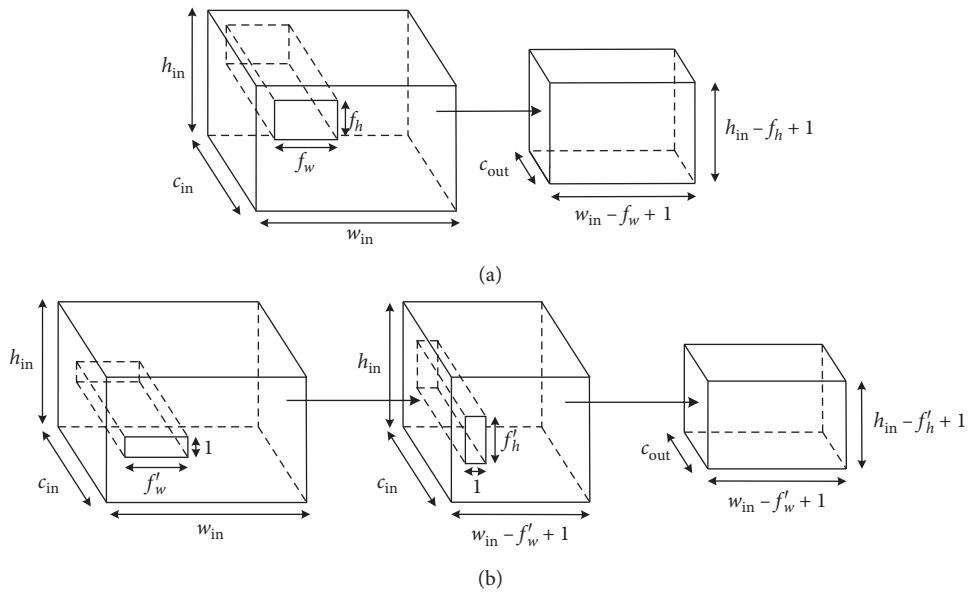

Figure 3: Convolutional layers: (a) classic CNN network and (b) SCNN network.

Table 1: Convolutional parameters for the network.

	Layer	Input matrix	Filter	C_{in}	C_{out}	Output matrix
Hidden layer 0	Time_0	63×13	5×1	1	1	59×13
	Freq_0	59×13	1×3	1	8	59×11
Hidden layer 1	Time_1	59×11	5×1	8	8	55×11
	Freq_1	55×11	1×3	8	16	55×9
Hidden layer 2	Time_2	55×9	11×1	16	16	45×9
	Freq_2	45×9	1×3	16	192	45×7
	Final_conv	45×7	1×1	192	12	45×7

or final_conv was modified as described by equations (5) and (6):

$$w(i)' = \frac{w(i) \cdot \gamma}{\sigma}, \tag{5}$$

$$b(i)' = \beta + \frac{(b(i) - \mu) \cdot \gamma}{\sigma}, \tag{6}$$

where γ and β are the scaling factors and bias of the BN, respectively, σ the standard deviation, and μ the average of the weights of a given input channel.

At the end of each layer, the acceptable number of truncated bits b_{tr_i} and a saturation (truncation of the most significant bits) of b_{sat_i} bits were also studied through the bit-true simulation to reduce the complexity of the hardware. In terms of formulas, truncation consists in changing the value of the lsb, as described by the following equation:

$$\text{lsb}_w{}' = \text{lsb}_w \cdot 2^{b_{tr_i}}. \tag{7}$$

Instead, saturating b_{sat_i} bit means discarding the b_{sat_i} most significant bits. Such operation does not affect lsb_w. For this aim, the worst case (greatest value in absolute meaning) of each layer output was considered so as to eliminate unused bits that were previously added for avoiding overflow of arithmetic operations. To sum up, the accuracy of the model for different sets of $b_w, b_{tr1}, b_{sat1}, b_{tr2}, b_{sat2}, \ldots, b_{tr7}, b_{sat7}$ was evaluated.

A possible optimization of the model consists in quantizing separately the weights of the last convolutional layer, by using $b_{w_{last}}$ bits. Indeed, the coefficients of final_conv may be divided by the divisor of the average pooling, saving hardware operations. This optimization significantly changes the range of weights for the last layer and a different quantization should be applied to it. For this reason, a second model to evaluate the overall accuracy takes into consideration different sets $(b_w, b_{w_{last}}, b_{tr1}, b_{sat1}, b_{tr2}, b_{sat2}, \ldots, b_{tr7}, b_{sat7})$.

3.2. Pruning. Another technique to reduce the complexity of the hardware accelerator is *pruning*. It consists in dropping the least important connections of the network [24, 25] by identifying the weights or biases with a magnitude smaller than a given threshold. In this network, the biases of the temporal convolutional layers have magnitudes in the order of 10^{-9}–10^{-7}. Considering their small values with respect to the other network parameters, they were pruned to reduce the model size. Indeed, it is possible to eliminate temporal bias terms without significantly affecting accuracy and reducing the number of sums to be computed.

4. Results of the Quantization Analysis

In this section, the results obtained from the quantization process are presented and discussed.

The SCNN model of this network has many degrees of freedom. For this reason, the first simulation step is finalised to identify a starting point for a more complex analysis, and it only focuses on the quantization of input layer words and weights.

Figure 4 shows simulation results, in terms of accuracy and mean square error (MSE) in relation to the floating-point model, when only the number of bits for input words representation is quantized. A number of 4 or 5 bits optimize accuracy and minimize the MSE. This first analysis gives intuitions about a possible optimization for the input layer: in particular, the number of bits of every input can be forced to be multiple of 4 bits, so that several inputs might be contained in buses such as the Advanced eXtensible Interface 4 (AXI4) bus [26], whose size is usually a multiple of 8 bits.

Figure 5 shows simulation results, in terms of accuracy and MSE, when only the number of bits for the representation of weights has been quantized. In this case, accuracy rapidly grows between 8 and 12 bits, reaching even higher values than the original ones in correspondence of 11 and 12 bits. Finally, accuracy saturates for 16 or more bits. This parameter is crucial because it influences the number of bits necessary to represent the result of MAC operations and, consequentially, the complexity of the entire network.

This first analysis was the starting point for a more detailed design exploration, involving the number of bits for the representation of the output of each layer.

Table 2 reports the best results obtained in terms of accuracy. Only the number of bits of SC layers and final_conv outputs are presented in the table, whereas data regarding temporal convolutional sublayers are omitted. The parameters listed in the table are as follows:

(i) b_in: number of bits for the representation of input words.
(ii) b_filter: number of bits for the representation of filters.
(iii) bit_out_0: number of bits for the representation of the outputs of the first hidden layer.
(iv) bit_out_1: number of bits for the representation of the outputs of the second hidden layer.
(v) bit_out_2: number of bits for the representation of the outputs of the third hidden layer.
(vi) bit_out_fc: number of bits for the representation of the outputs of the last convolutional layer.

Collected data show that it is possible to increase model accuracy through quantization. In fact, the best accuracy obtained for the floating-point model is 87.77%, whereas for the fixed-point representation, the highest accuracy is 90.23%.

The second part of the simulation considers a different quantization for the final_conv layer due to the inclusion of the average pooling effects, as explained in Section 3.1. The results of this simulation are summarized in Table 3.

These models show smaller hardware requirements than the single-quantization versions presented in Table 2. Sets of data are the same as those in Table 2, excepting for b_last that represents the number of bits for the representation of final_conv layer weights, whereas b_filter refers only to SC layer weights. This second model allows to shrink weights representation for all convolutional layers, significantly reducing the impact of MAC operations on hardware resource requirements. Furthermore, several quantized models have an accuracy score higher than the original one (87.77%).

The model chosen for the FPGA implementation considers both accuracy and the possibility to shrink parameter representations. Model number (7) from Table 3 was selected: it has a higher accuracy than the Keras–Python model (88.09 versus 87.77), and it minimizes the number of bits necessary for the representation of layer outputs and weights. Input layer results compatible with AXI4 because Input words are represented on 4 bits. The number of bits for the representation of temporal convolutional outputs of model (7) is 10 for time_0, 8 for time_1, and 10 for time_2.

5. FPGA Hardware Architecture

This section describes the architecture of the hardware accelerator that was implemented on different FPGA families. Thanks to the reduced number of parameters of the SCNN investigated in our previous work [12], it was possible to realize a full on-chip design with high advantages in terms of latency and energy per inference, avoiding accesses to off-chip memories [11, 21].

Figure 6 shows the block diagram of the accelerator. The number of bits of the words read from and written into the Input memory and RAMs is related to our preferred model, described in the previous section.

An input memory is used as an interface between the hardware accelerator and the system that records and elaborates the audio samples. The input memory stores 4-bit input data. The time/frequency layers and final_conv layer perform convolutional operations and store the results into a RAM memory, used as a buffer. Once the previous layer completes an entire convolution, the next one starts reading out its input matrix from the memory.

Each of the seven convolutional layers has its own MAC module to perform multiply-accumulate operations. Figure 7 shows the structure of the MAC module, designed to compute one element of the output matrix per clock cycle.

It reads n_{elem} elements from the RAM memory, where the value of n_{elem} is shown in the following equation:

$$n_{\text{elem}} = C_{\text{in}} \cdot f_{\text{elem}}, \tag{8}$$

where C_{in} is the number of input channels and f_{elem} is the number of elements composing a channel filter. The adder-tree structure is used for accumulation, and it was chosen to reduce the overall latency of the circuit. Considering this configuration of the MAC module, the total number of clock cycles needed to complete an entire convolution for each convolutional layer is N_{clk}, as shown in the following equation:

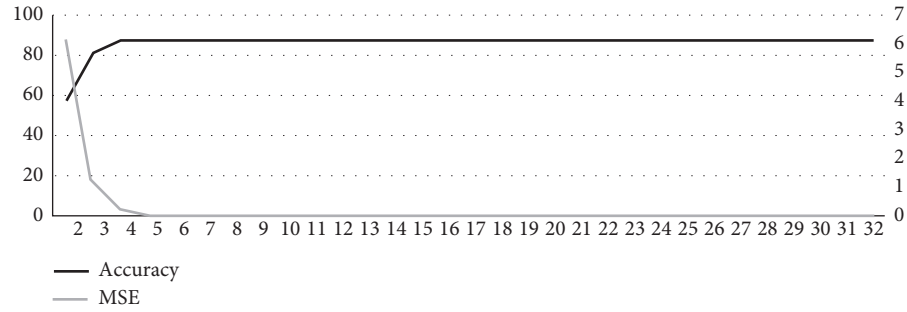

Figure 4: Accuracy and MSE to the change of the number of bits for input layer words.

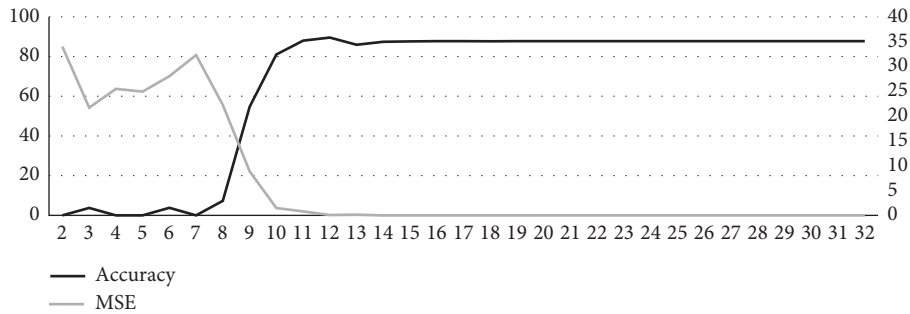

Figure 5: Accuracy and MSE to the change of the number of bits of filter elements.

Table 2: Results of the first quantization analysis.

b_in	b_filter	bit_out_0	bit_out_1	bit_out_2	bit_out_fc	Accuracy (%)
5	12	10	8	10	10	90.23
5	12	8	8	10	10	90.14
5	12	8	8	10	8	89.74
5	11	8	8	10	10	88.91
4	12	8	8	10	12	88.87
4	12	8	8	10	10	88.78
5	11	8	8	10	8	88.60
4	12	8	8	10	8	88.46
5	11	8	8	10	8	88.40
4	11	8	8	10	12	87.84
4	11	8	8	10	10	87.61

Table 3: Results of the second quantization analysis.

No.	b_in	b_filter	b_last	bit_out_0	bit_out_1	bit_out_2	bit_out_fc	Accuracy (%)
1	5	8	6	8	8	10	12	89.88
2	5	8	6	8	8	10	10	89.74
3	4	12	6	8	8	10	12	88.87
4	4	12	6	8	8	10	10	88.75
5	4	12	6	8	8	10	12	88.21
6	4	12	6	8	8	10	10	88.09
7	4	8	6	8	8	10	10	88.09
8	4	8	6	8	8	10	10	87.61
9	4	11	6	8	8	10	12	87.55
10	4	8	6	8	8	10	10	87.43
11	5	8	6	8	8	8	10	87.39
12	4	12	6	8	8	10	8	87.36
13	4	11	6	8	8	10	10	87.29
14	5	8	6	8	8	8	8	86.93

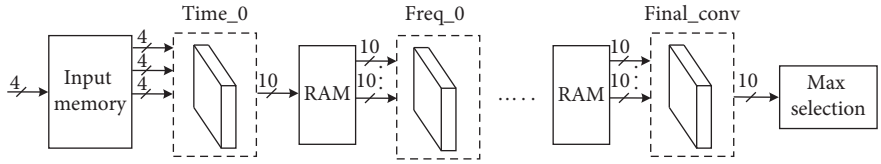

Figure 6: SCNN architecture for FPGA implementation.

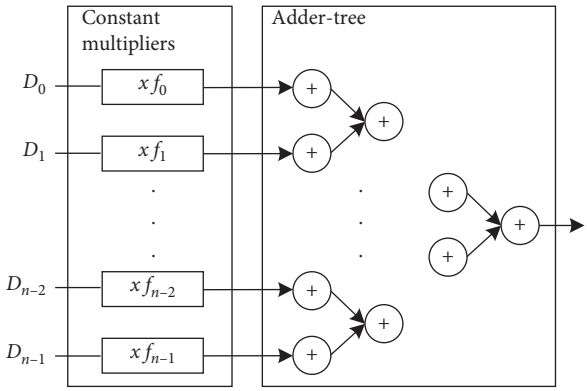

Figure 7: MAC module architecture.

$$N_{\text{clk}} = C_{\text{out}} \cdot W_{\text{out}} \cdot H_{\text{out}}, \tag{9}$$

where C_{out} is the number of output channels and W_{out} and H_{out} are the dimensions of the output matrix. Table 4 shows N_{clk} of each convolutional layer of the network considering the values of C_{out}, W_{out}, and H_{out} listed in Table 1. Finally, 819 clock cycles shall be added in order to store the 63×13 input matrix in the Input memory. A total of 90278 clock cycles are required to complete an inference.

A major parallelization of MAC operations would offer the opportunity to speed-up accelerator performances, reducing the inference time. On the other hand, it is not generally possible to perform an arbitrary number of operations per clock cycle because of the limited number of FPGA resources (combinatorial logic, DSPs, etc.). Furthermore, if the level of parallelism is too high, routing can become the bottleneck of the implementation.

It is possible to boost MAC module operations, increasing the number of output elements n computed per clock cycle. In particular, for $n > 1$, N_{clk} is reduced of a factor $1/n$, as described by the following equation:

$$N_{\text{clk}} = \frac{C_{\text{out}} \cdot W_{\text{out}} \cdot H_{\text{out}}}{n}. \tag{10}$$

Whilst this strategy leads to better timing optimization, it increases the design effort necessary to find the best combination that can fit on a specific FPGA device. Indeed, the appropriate value of n for each layer should be tuned depending on the size of the target FPGA, in order to guarantee design implementability. Furthermore, parallelizing each layer guarantees negligible advantages in terms of inference time when the number of operations necessary to carry out an entire convolution is strongly different for every layer. Considering the limitation of FPGA hardware resources, it results appropriate to parallelize MAC operations only for the layers with the highest values of N_{clk}. In this specific case, freq_2 layer contributes to 60460 over 90278 total number of clock cycles due to the very high number of output channels (192). For this reason, the MAC module of the freq_2 layer was customized so that it calculates 4 values of the output matrix per clock cycle. According to equation (10), this allows to drastically reduce freq_2 N_{clk} from 60480 to 15120 and consequently the total inference time from 90278 to 44918 clock cycles, halving the inference time. If a similar parallelization was realized for the other convolutional layers of the network, it would increase hardware resources without a significant improvement of timing performances because of their limited effect on the overall inference time.

Table 4: N_{clk} values for the various layers.

Layer	N_{clk}
Input memory	819
Time_0	767
Freq_0	5192
Time_1	4840
Freq_1	7920
Time_2	6480
Freq_2	60480
Final_conv	3780
Total	90278

As previously specified, batch normalization operations were absorbed in the frequency convolutional layer of each SC layer. The average pooling layer was included in final_conv that provides 12 outputs, corresponding to the sum of all the elements belonging the output matrix of each output channel. Finally, Softmax layer can be omitted. Indeed, to provide a direct decision on the pronounced word, it is sufficient to select the maximum value among the twelve outputs of final_conv.

This architecture was chosen because its simplicity heightens the possibility to fit the hardware accelerator in a target FPGA, reducing design time and increasing design portability among devices with different sizes.

6. Hardware Implementation Results

This section describes the performances of the hardware accelerator on different FPGA families. The presented architecture was implemented on several Xilinx and Intel devices to analyse its design portability on FPGAs with different sizes and performances. Results are presented in terms of hardware resource occupation, maximum achievable clock frequency, inference time, and power consumption. Finally, an analysis of how MAC module

parallelization influences design portability on smaller FPGAs is provided.

The devices included in the analysis are as follows:

(i) Zynq UltraScale+ (US+), xczu9eg-ffvb1156-2-e [27]

(ii) Virtex UltraScale+, xcvu3p-ffvc1517-2-e [28]

(iii) Virtex UltraScale (US), xcvu065-ffvc1517-2-e [29]

(iv) Zynq-7000, xc7z045ffg900-1 [30]

(v) Virtex-7, xc7vx330tffg1157-2 [31]

(vi) Kintex-7 low voltage (lv), xc7k160tfbg484-2L [31]

(vii) Artix-7 low voltage (lv), xc7a200tfbg484-2 [31]

(viii) Arria 10 GX, 10AX027H3F35E2SG [32]

(ix) Stratix V GS, 5SGSMD4E1H29C1 [33]

(x) Stratix V GX, 5SEE9F45C2 [33]

(xi) Stratix V E, 5SEE9H40C2 [33]

(xii) Cyclone V, 5CEFA9U19C8 [34]

All the implementations were realized by using *Vivado design suite* for Xilinx devices and *Quartus Prime Software* for Intel devices.

Table 5 shows the hardware resources needed for the implementation of the accelerator on Xilinx FPGAs. Results are presented in terms of combinatorial elements, sequential elements, BRAMs, and LUTRAMs (LRAMs). The percentage of used resources out of the total is also indicated. Table 6 shows hardware resources needed for the implementation of the accelerator on Intel FPGAs. In this case, results are presented in terms of combinatorial elements, sequential elements, BRAMs, and DSPs.

All the implementations refer to the version of the accelerator in which the MAC module of the freq_2 layer was parallelized to compute 4 elements of its output matrix per clock cycle. The structure and the number of combinatorial/sequential elements and memory dimensions and typologies are specific for each device. Please refer to FPGA datasheets for more information about the architecture of Xilinx devices [27–31] and Intel devices [32–34].

Figures 8 and 9 show the maximum achievable clock frequency and the inference time for Xilinx and Intel FPGAs, respectively. The minimum inference time for each layer can be calculated taking into consideration MAC module optimizations for the freq_2 layer and N_{clk} values listed in Table 4. The best result is obtained for the Zynq UltraScale+ with a maximum clock frequency of 116.2 MHz and a corresponding inference time of less than 0.4 ms.

A power analysis was performed for both Xilinx and Intel FPGAs. To obtain a more accurate estimation of the power consumption for Xilinx devices, a post-implementation timing simulation was carried out by using *Questa® Advanced Simulator* to extract information about the switching activity of the internal nodes of the circuit. Since Intel devices do not support postlayout simulation, only a RTL-level estimation of the switching activity has been included in the power consumption analysis as suggested by Intel guidelines [35]. Results are shown in Table 7. In general, Xilinx devices show a lower power consumption than Intel devices for both static and dynamic power. The only exception is the Arria 10, featuring a power consumption of 1 W and resulting the second best device after the Kintex-7 lv.

6.1. Design Portability. An analysis of the hardware accelerator portability has been carried out in order to investigate how the proposed design fits in smaller FPGAs. In particular, the freq_2 layer has been customized to compute 1, 2, 4, and 8 elements (*n*_out) of a given output channel per clock cycle. Results are presented in terms of hardware resources, maximum clock frequency, and inference time.

Two FPGAs with different sizes belonging to the same family were selected:

(i) xc7z045ffg900-2 (xc7z045) and xc7z030fbg484-2 (xc7z030) for the Zynq-7000 family [30]

(ii) xczu9eg-ffvb1156-2-e (xczu9eg) and xczu3eg-sfva625-2L-e (xczu3eg) for the Zynq UltraScale+ family [27]

Tables 8 and 9 show the results in terms of hardware resource occupation for the Zynq-7000 FPGAs and for the Zynq-US+ FPGAs, respectively.

The xc7z030 and the xczu3eg have a limited number of hardware resources and only the version of the accelerator with *n*_out equal to 1, 2, and 4 can be implemented in these devices; the version with *n*_elem equals to 8 fits only in the xc7z045 and in the xczu9eg. Owing to the limited number of LUTs available on-board xc7z030 and xczu3eg DSPs are included to perform MAC operations. In addition, xczu3eg implementations exploit all the available BRAMs on-board, and LRAMs have to be included. Versions of the hardware accelerator with a lower level of MAC parallelization have worst performance in terms of inference time but can fit in smaller devices because their requirements in terms of combinatorial elements are more relaxed. Unfortunately, the number of RAMs required does not change because intralayer RAM dimensions and the number of parameters of the network do not, and it can represent a bottleneck for the implementation of the not-customized version of the accelerator on smaller FPGAs.

Figures 10 and 11 show the performance in terms of clock frequency and inference time for Zynq-7000 and Zynq-US+ FPGAs, respectively. For the xc7z045 and the xczu9eg, the maximum achievable clock frequency does not show large variation increasing the level of MAC parallelism. For the xc7z030 and the xczu3eg, maximum achievable clock frequency tends to decrease because the limited size of these devices leads to a less optimized routing and consequently to worse timing performances. For this reason, inference times for xc7z030 with *n*_mac equals to 4 and *n*_mac equals to 2 are almost the same.

Similarly, when *n*_out is equal to 4, timing performance of the xc7z030 solution features an implementation loss of the 31% with respect to the solution on-board the xc7z045, and the xczu3eg solution features an implementation loss of 47% with respect to the one on-board the xczu9eg.

Table 5: Hardware accelerator implementation on Xilinx FPGAs.

FPGA family	Comb. elem.	Comb. elem. (%)	Seq. elem.	Seq. elem. (%)	BRAM	BRAM (%)	LRAM	LRAM (%)
Zynq US+	81345	30	860	<1	228	25	2560	2
Virtex US+	81367	21	864	<1	228	32	2560	1
Virtex US	81427	23	952	<1	228	18	2560	3
Zynq-7000	76283	35	632	<1	244	45	0	0
Virtex-7	76163	37	632	<1	244	33	0	0
Kintex-7 lv	81737	81	633	<1	244	75	0	0
Artix-7 lv	87406	86	1081	<1	228	70	0	0

Table 6: Hardware accelerator implementation on Intel FPGAs.

FPGA family	Comb. elem.	Comb. elem. (%)	Seq. elem.	Seq. elem. (%)	BRAM	BRAM (%)	DSP	DSP (%)
Arria 10 GX	23722	23	296	<1	344	46	323	39
Stratix V GS	25532	18	2851	<1	344	36	323	31
Stratix V GX	23370	7	1860	<1	344	13	323	92
Stratix V E	23099	7	1843	<1	344	13	323	92
Cyclone V	24111	21	2911	<1	392	32	323	94

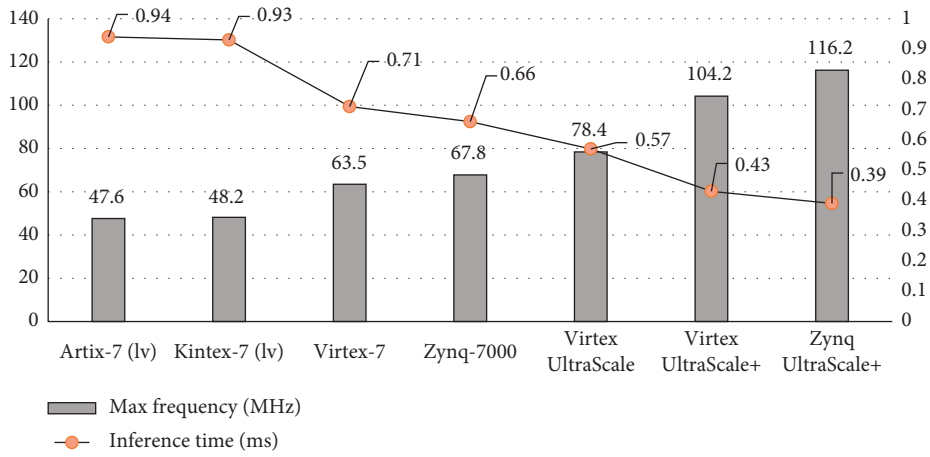

Figure 8: Maximum clock frequency and inference time for different Xilinx FPGA families.

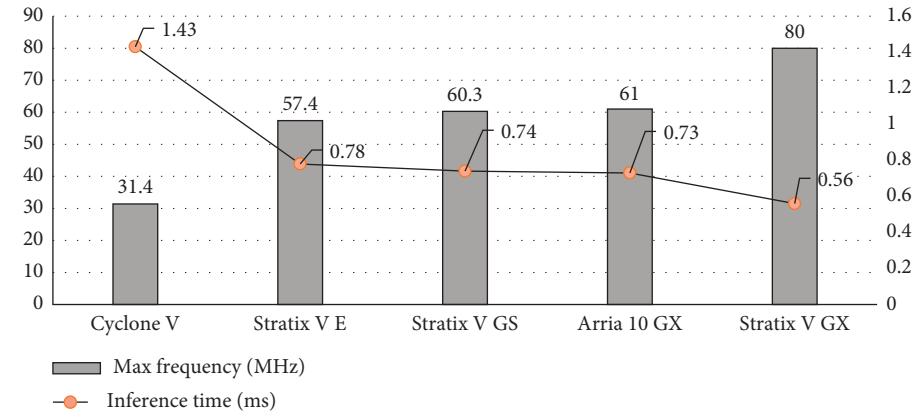

Figure 9: Maximum clock frequency and inference time for different Intel FPGA families.

7. Comparison with Intel Movidius Neural Compute Stick

In this section, the FPGA-based accelerator is compared with a commercial hardware accelerator for machine learning on the edge: the Intel Movidius Neural Compute Stick.

The same model of SCNN keyword spotting was implemented on the NCS in our previous work [12], and a direct comparison between the performances of the two

Table 7: Power consumption for Xilinx and Intel FPGAs.

Device	Static power (W)	Dynamic power (W)	Total power (W)
Artix 7	0.151	0.892	1.043
Kintex-7 lv	0.110	0.859	0.969
Zynq-7000	0.215	1.172	1.387
Virtex 7	0.204	1.147	1.351
Virtex-US	0.626	1.235	1.861
Virtex-US+	0.839	1.302	2.141
Zynq-US+	0.627	1.532	2.259
Cyclone V	0.570	1.731	2.301
Stratix V E	1.607	2.150	3.757
Stratix V GS	0.857	3.153	4.010
Arria 10	0.272	0.730	1.002
Stratix V GX	1.244	2.141	3.385

Table 8: Design portability analysis for Zynq-7000 family.

Zynq-7000 family	Num mac.	Comb. elem. (%)	Seq. elem. (%)	BRAM (%)	DSP (%)
xc7z030	1	85	<1	92	0
	2	88	<1	92	0
	4	55	<1	92	100
	8	—	—	—	—
xc7z045	1	33	<1	45	0
	2	35	<1	45	0
	4	38	<1	45	0
	8	44	<1	45	0

Table 9: Design portability analysis for Zynq-US+ family.

Zynq-US+ family	Num mac.	Comb. elem. (%)	Seq. elem. (%)	BRAM (%)	LRAM (%)	DSP (%)
xczu3eg	1	54	<1	100	16	92
	2	64	<1	100	16	100
	4	70	<1	100	16	100
	8	—	—	—	—	—
xczu9eg	1	25	<1	27	0	0
	2	26	<1	27	0	0
	4	30	<1	27	0	0
	8	36	<1	27	0	0

solutions, in terms of inference time, power consumption, and energy per inference, is now presented.

The NCS is a commercial deep learning hardware accelerator hosting the Myriad 2 VPU by Intel Movidius [9]. The VPU includes the following:

(i) 4Gb of LPDDR3 DRAM

(ii) 12 very long instruction word (VLIW) streaming hybrid architecture vector engine (SHAVE) processors optimized for machine vision used to run parts of a neural network in parallel

(iii) 2MB on-chip memory shared between SHAVE processors and fixed-function accelerators

(iv) 2 Leon microprocessors that coordinate the reception of the network graph file and of inputs via USB connection

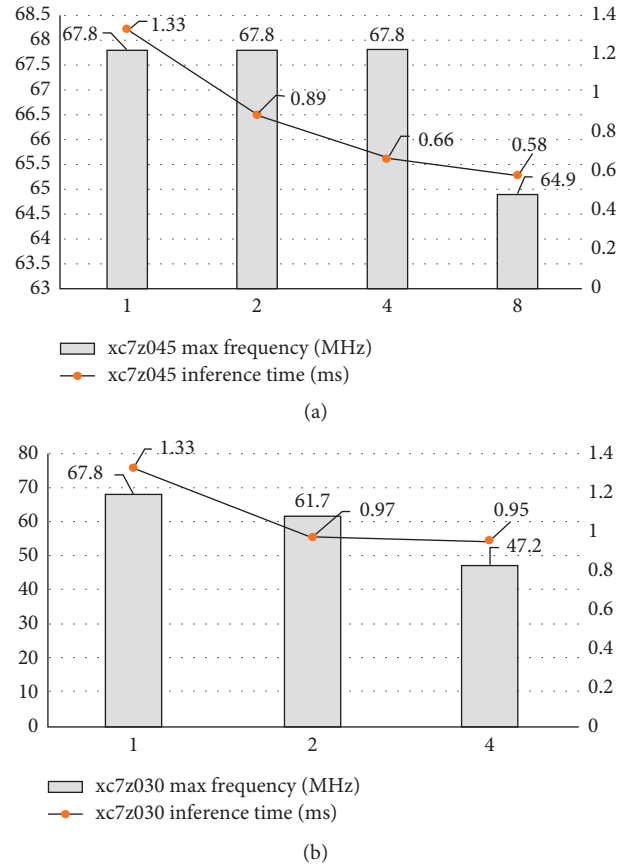

Figure 10: Max frequency and inference time for the Zynq-7000 family.

The Myriad 2 VPU supports fully connected, convolutional (with arbitrary sized kernel), and depthwise convolutional layers.

The NCS implements the floating-point version of the SCNN model with a maximum accuracy of 87.77. Quantization allows to increase this value to 90.23%, even if our preferred implementation has an accuracy of 88.09%.

The inference time for the SCNN implemented on the NCS is approximately 10 ms. The FPGA-based accelerator has a lower inference time for all the FPGA implementations presented, swinging from 1.45 ms for the Cyclone V to 0.39 ms for the Zynq-US+. Finally, the NCS power consumption is 0.81 W. Such a result is provided by considering the hardware setup of our previous work [12], featuring a Rasperry PI 3B [36] connected to the NCS. Power consumption can be estimated by subtracting the Raspeberry PI 3B power consumption in the absence of the NCS (1.3 W) to the total power consumption of the system during an inference (2.11 W).

As shown in Table 7, power consumption for the design implemented on-board all FPGAs is higher than that for the NCS one. Nevertheless, for all the implementations, the energy dissipated during an inference (E_{inf}) is lower than the one of NCS. In fact, it is possible to calculate E_{inf} as shown in the following equation:

$$E_{\text{inf}} = P \cdot t_{\text{inf}}, \tag{11}$$

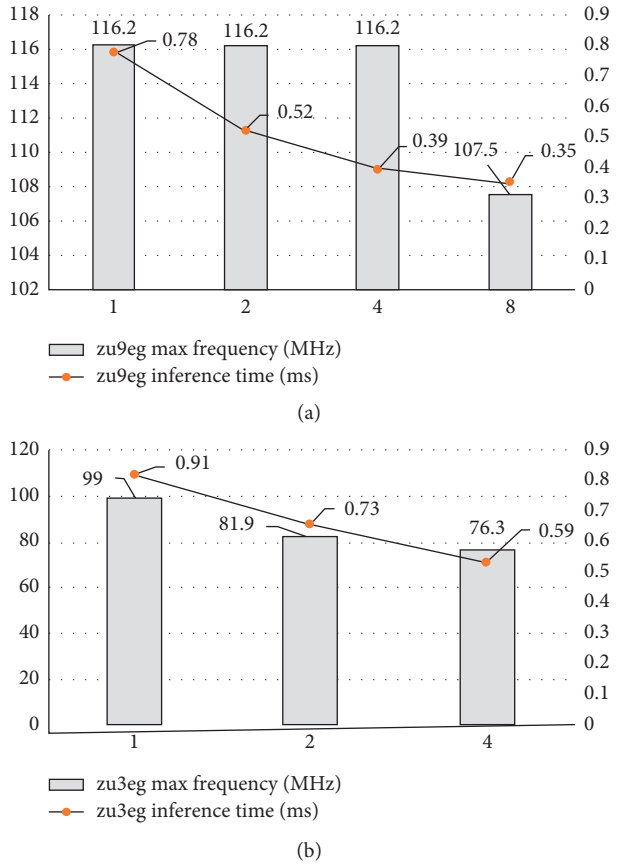

Figure 11: Max frequency and inference time for the Zynq-US+ family.

where P is the average power consumption during an inference and t_{inf} is the inference time.

Indeed, even if Xilinx and Intel devices show a higher power consumption, the significantly lower t_{inf} leads to a reduced E_{inf}.

Table 10 shows a comparison among FPGAs and NCS in terms of inference time, power, and energy, by using model (7) of Table 3. The power analysis was performed by considering the maximum achievable clock frequency (f_{clk}) of each FPGA in order to minimize the inference time.

Results show that FPGAs offer great design flexibility, allowing to tune inference time and power consumption through the choice of the different platforms. FPGAs are promising devices for the implementation of CNN-based hardware accelerators for portable applications and in particular for those requiring low latency and high accuracy. Indeed, inference time results to be diminished approximately of a factor between 7 and 25 and energy per inference is reduced, respectively, of a factor between 2.5 and 9 in the investigated cases.

Finally, Figure 12 provides a graphical representation of the power consumption/inference time results shown in Table 10. It is evident from results that all FPGA solutions feature a reduced inference time with respect to the NCS implementation at expense of a higher power consumption, even if comparable for some devices.

Table 10: Performance comparison between Xilinx FPGAs, Intel FPGAs, and NCS.

Device	f_{clk} (MHz)	Inference time (ms)	Total power (W)	Energy (mJ)
Xilinx FPGA families				
Artix 7	47.6	0.94	1.043	0.98
Kintex-7 lv	48.2	0.93	0.969	0.90
Zynq-7000	67.8	0.65	1.387	0.90
Virtex 7	63.5	0.71	1.351	0.96
Virtex-US	78.4	0.57	1.861	1.01
Virtex-US+	104.2	0.43	2.141	0.92
Zynq-US+	116.4	0.39	2.259	0.88
Intel FPGA families				
Cyclone V	31.4	1.43	2.301	3.29
Stratix V E	57.4	0.78	3.757	2.9
Stratix V GS	60.3	0.74	4.010	2.96
Arria 10	61	0.73	1.002	0.73
Stratix V GX	80	0.56	3.385	1.9
Intel movidius neural compute stick				
NCS	600	10	0.810	8.1

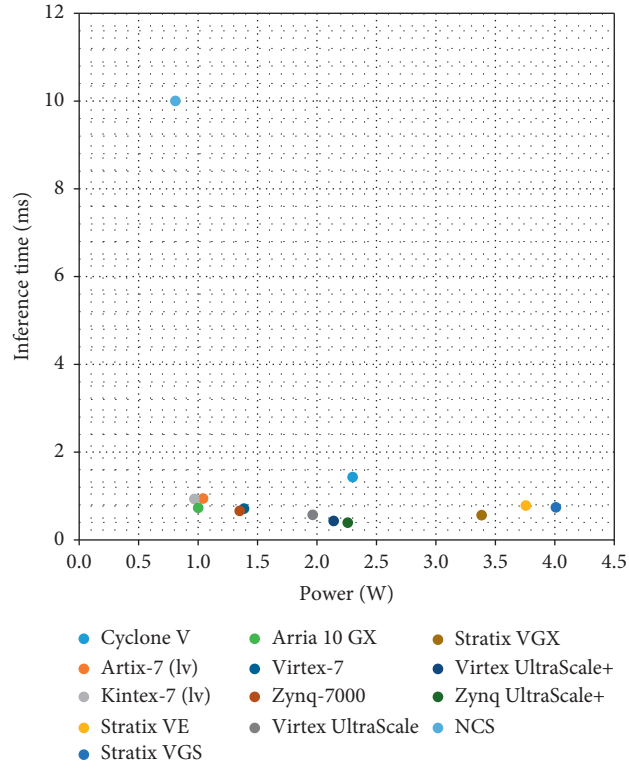

Figure 12: Inference time/power consumption trade-off analysis.

8. Discussion

The results presented in this work highlight the value of the FPGA solutions to accelerate inference of CNNs. They offer a remarkable trade-off between power consumption and

inference time, resulting in interesting solutions for on the edge computing.

It is necessary to underline that these results were possible, thanks to the use of a CNN model optimized for resource-constrained devices [12], featuring a reduced number of parameters and layers. In view of that, a full on-chip design was achievable, with strong advantages in terms of latency and power consumption. Consequently, results are pertinent for applications requiring relatively small models, such as digit and letter recognitions systems [37, 38], audio [39], and mobile vision applications [40].

Finally, the proposed full on-chip design guarantees a straightforward processing architecture (i.e., no data scheduling from external memories and no management of shared inference processing elements), further reducing the overall system design time. However, when compared with NCS and other *plug and play* solutions, the use of FPGA still requires much more design effort and competences, in view of the higher and heterogeneous design steps (i.e., model quantization and architecture definition) and of the broader design space.

9. Conclusions

This article presents a full on-chip FPGA-based hardware accelerator for on the edge keyword spotting. The KWS system is described focusing on its realization through a machine-learning algorithm and on traducing AI on the edge paradigm.

Starting from a *Keras–Python* model of a KWS based on a SCNN, the parameters of the network were quantized in order to shrink the hardware resources needed for its realization. CNNs have a large number of parameters and are characterized by multiplying and accumulating operations that make their implementation on an FPGA device challenging. Quantization analysis shows that fixed-point representation does not significantly affect model accuracy. On the contrary, it is possible to increase it for particular combinations of input words, weight, and layer output representations. Then, the accelerator architecture is described, focusing on design effort to exploit the intrinsic parallelism of these devices. The SCNN accelerator was implemented on several Xilinx and Intel FPGAs to analyse design portability on different families. The obtained results are presented in terms of maximum achievable clock frequency, hardware resources needed for the network implementation, energy per inference, and power consumption. Finally, the proposed accelerator was compared with a commercial solution for on the edge AI applications: the Intel Movidius NCS. This analysis shows that with a FPGA-based solution, it is possible to overcome NCS performances in terms of inference time and energy per inference.

Data Availability

The data used to support the findings of this study are included within the article.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

References

[1] D. M. Ramík, C. Sabourin, R. Moreno, and K. Madani, "A machine learning based intelligent vision system for autonomous object detection and recognition," *Applied Intelligence*, vol. 40, no. 2, pp. 358–375, 2014.

[2] H. Zhang, K.-F. Wang, and F.-Y. Wang, "Advances and perspective on applications of deep learning in visual object detection," *Acta Automatica Sinica*, vol. 43, no. 8, pp. 1289–1305, 2017.

[3] L. Zhang, Z. He, and Y. Liu, "Deep object recognition across domains based on adaptive extreme learning machine," *Neurocomputing*, vol. 239, pp. 194–203, 2017.

[4] R. Nian, B. He, and A. Lendasse, "3D object recognition based on a geometrical topology model and extreme learning machine," *Neural Computing and Applications*, vol. 22, no. 3-4, pp. 427–433, 2013.

[5] G. Retsinas, G. Sfikas, N. Stamatopoulos, G. Louloudis, and B. Gatos, "Exploring critical aspect of CNN-based keyword spotting. A phocnet study," in *Proceedings of the 13th IAPR International Workshop on Document Analysis Systems*, pp. 13–18, Vienna, Austria, April 2018.

[6] Y. B. Ayed, D. Fohr, J. P. Haton, and G. Chollet, "Keyword spotting using support vector machines," in *Proceedings of the 5th International Conference on Text, Speech and Dialogue*, vol. 2448, pp. 285–292, Brno, Czech Republic, September 2002.

[7] Y. Zhang, N. Suda, L. Lai, and V. Chandra, "Hello edge: keyword spotting on microcontrollers," 2017, https://arxiv.org/abs/1711.07128.

[8] Q. Zhang, M. Zhang, T. Chen et al., "Recent advances in convolutional neural network acceleration," *Neurocomputing*, vol. 323, pp. 37–51, 2019.

[9] Neural compute Stick Documentation. https://software.intel.com/en-us/movidius-ncs.

[10] Google Coral Datasheet: https://coral.withgoogle.com/tutorials/accelerator-datasheet/.

[11] S. Mittal, "A survey of FPGA-based accelerators for convolutional neural networks," *Neural Computing and Applications*, pp. 1–31, 2018.

[12] G. Benelli, G. Meoni, and L. Fanucci, "A low power keyword spotting algorithm for memory constrained embedded system," in *Proceedings of the 26th IFIP/IEEE International Conference on Very Large Scale Integration*, Verona, Italy, October 2018.

[13] Keras Documentation. https://keras.io/.

[14] A. Mohamed, "Deep neural network acoustic models for ASR," Doctoral thesis, Toronto University, Toronto, Canada, 2014.

[15] S. Ioffe and C. Szegedy, "Batch normalization: accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on Machine Learning, ICML*, vol. 1, pp. 448–456, Lille, France, July 2015.

[16] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural network," in *Proceedings of the FPGA 2015—2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 161–170, Monterey, CA, USA, February 2015.

[17] J. Qiu, J. Wang, S. Yao et al., "Going deeper with embedded FPGA platform for convolutional neural network," in *Proceedings of the International Symposium on Field-Programmable Gate Arrays*, pp. 26–35, Monterey, CA, USA, February 2016.

[18] E. Nurvitadhi, D. Sheffield, J. Sim et al., "Accelerating binarized neural networks: comparison of FPGA, CPU, GPU and ASIC," in *Proceedings of the International Conference on Field-Programmable Technology (FPT)*, pp. 77–84, Tokyo, Japan, December 2016.

[19] S. Moini, B. Alizadeh, M. Emad, and R. Ebrahimpour, "A resource-limited hardware accelerator for convolutional neural networks in embedded vision applications," *IEEE Transactions on Circuits and Systems II Express Briefs*, vol. 64, no. 10, pp. 1217–1221, 2017.

[20] L. Jiao, C. Luo, W. Cao, X. Zhou, and L. Wang, "Accelerating low bit-width convolutional neural networks with embedded FPGA," in *Proceedings of the 27th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 1–4, Leuven, Belgium, September 2017.

[21] J. Park and W. Sung, "FPGA based implementation of deep neural networks using on-chip memory only," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1011–1015, Shanghai, China, March 2016.

[22] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: a tuturial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[23] G. Feng, Z. Hu, S. Chen, and F. Wu, "Energy-efficient and high-throughput FPGA-based accelerator for convolutional neural networks," in *Proceedings of the 13th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*, pp. 624–626, Hangzhou, China, October 2016.

[24] X. Zhang, X. Liu, A. Ramachandran et al., "High-performance video content recognition with long-term recurrent convolutional network for FPGA," in *Proceedings of the 26th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 1–4, Leuven, Belgium, September 2017.

[25] J. H. Kim, B. Grady, R. Lian, J. Brothers, and J. H. Anderson, "FPGA-based CNN inference accelerator synthesized from multi-threaded C software," in *Proceedings of the 30th IEEE International System-On-Chip Conference (SOCC)*, pp. 268–273, Munich, Germany, September 2017.

[26] AMBA Advanced Extensible Interface 4 Specifications. https://www.arm.com/products/silicon-ip-system/embedded-system-design/amba-specifications.

[27] Zynq UltraScale+ Family Datasheet. https://www.xilinx.com/support/documentation/data_sheets/ds891-zynq-ultrascale-plus-overview.pdf.

[28] Virtex UltraScale+ Family Datasheet. https://www.xilinx.com/support/documentation/data_sheets/ds923-virtex-ultrascale-plus.pdf.

[29] Virtex UltraScale Family Datasheet. https://www.xilinx.com/support/documentation/data_sheets/ds890-ultrascale-overview.pdf.

[30] Zynq-7000 Family Datasheet. https://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf.

[31] Virtex-7, Kintex-7 and Artix-7 Families' Datasheet. https://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf.

[32] Arria 10 Family Overview. https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/arria-10/a10_overview.pdf.

[33] Stratix V Family Datasheet. https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/stratix-v/stx5_51001.pdf.

[34] Cyclone V Family Datasheet. https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/cyclone-v/cv_51001.pdf.

[35] Quartus Prime Standard Edition. https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/qts/archives/qts-qps-handbook-16.0.pdf.

[36] Rasberrey PI 3B Datasheet. https://www.terraelectronica.ru/pdf/show?pdf_file=%252Fds%252Fpdf%252FT%252FTechicRP3.pdf.

[37] Y. Hout and H. Zhao, "Handwritten digit recognition based on depth neural network," in *Proceedings of the International Conference on Intelligent Informatics and Biomedical Sciences (ICIIBMS)*, Shanghai, China, November 2017.

[38] D. C. Ciresan, U. Meier, and J. Schmidhuber, "Transfer learning for Latin and Chinese characters with deep neural networks," in *Proceedings of the 2012 International Joint Conference on Neural Networks (IJCNN)*, Brisbane, Australia, June 2012.

[39] T. Secu, C. Puhrsh, B. Kingsbury, and Y. LeCun, "Very deep multilingual convolutional neural networks for LVCSR," in *Proceedings of the 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Shanghai, China, March 2016.

[40] A. G. Howard, M. Zhu, B. Chen et al., "MobileNets: efficient convolutional neural networks for mobile vision applications," 2017, https://arxiv.org/abs/1704.04861.