applied sciences

MDPI

Article

Flexible Convolver for Convolutional Neural Networks Deployment onto Hardware-Oriented Applications

Moisés Arredondo-Velázquez [1], Paulo Aaron Aguirre-Álvarez [2], Alfredo Padilla-Medina [2], Alejandro Espinosa-Calderon [3], Juan Prado-Olivarez [2] and Javier Diaz-Carmona [2,*]

[1] Faculty of Physical and Mathematical Sciences, Benemérita Universidad Autónoma de Puebla, Puebla 72410, Mexico
[2] Electronics Engineering Department, Tecnológico Nacional de México en Celaya, Celaya 38010, Mexico
[3] Regional Center for Optimization and Development of Equipment, Tecnológico Nacional de México, Celaya 38020, Mexico
* Correspondence: javier.diaz@itcelaya.edu.mx

Abstract: This paper introduces a flexible convolver capable of adapting to the different convolution layer configurations of state-of-the-art Convolution Neural Networks (CNNs). The use of two proposed programmable components achieves this adaptability. A Programmable Line Buffer (PLB) based on Programmable Shift Registers (PSRs) allows the generation of the required convolution masks required for each processed CNN layer. The convolution layer computing is performed through a proposed programmable systolic array configured according to the target device resources. In order to maximize the device resource usage and to achieve a shortened processing time, the filter, data, and loop parallelisms are leveraged. These characteristics allow the described architecture to be scalable and implemented on any FPGA device targeting different applications. The convolver description was written in VHDL using the Intel Cyclone V 5CSXFC6D6F31C6N device as a reference. The experimental results show that the proposed computing method allows the processing of any CNN without requiring special adaptation for a specific application since the standard convolution algorithm is used. The proposed flexible convolver achieves competitive performance compared with those reported in related works.

Keywords: convolutional neural networks (CNN); hardware accelerators; systolic array; field programmable gate arrays (FPGA); embedded systems

check for updates

Citation: Arredondo-Velázquez, M.; Aguirre-Álvarez, P.A.; Padilla-Medina, A.; Espinosa-Calderon, A.; Prado-Olivarez, J.; Diaz-Carmona, J. Flexible Convolver for Convolutional Neural Networks Deployment onto Hardware-Oriented Applications. *Appl. Sci.* **2023**, *13*, 93. https://doi.org/10.3390/app13010093

Academic Editors: Jorge Crichigno, Francesco Benedetto and Norbert Herencsar

Received: 14 November 2022
Revised: 11 December 2022
Accepted: 13 December 2022
Published: 21 December 2022

Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

This paper is an extended version of the work presented at the 2022 45th International Conference on Telecommunications and Signal Processing (TSP) [1], where a flexible general-purpose convolver is proposed and intended to be used in different state-of-the-art Convolutional Neural Networks (CNNs) applications. The proposed convolver design is intended for CNN deployment, which is one of the fundamental algorithms for deep learning applications because of their capability to learn representations at different abstraction levels from digital images. In the past decades, classic object classification techniques were limited when processing raw data since careful engineering and considerable domain expertise were required. Nowadays, CNNs, by means of their convolutional layers, are capable of extracting image features, turning raw data (such as the pixel values of an image) into a suitable numerical representation, which in turn, can be classified by an additional model, typically a Fully Connected Neural Network (FCNN) [2,3]. From reported results in 2012 on the event ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [4], CNNs have shown an outstanding breakthrough in image classification applications. The use of CNN achieves higher accuracy than conventional algorithms, as demonstrated in 2012 [5] by the reported Top-5 accuracy of 84.7%. Since then, Top-5 accuracy has been continuously improved by introducing new CNN topologies, even exceeding image classification human

capabilities. These results were also reported in the following ILSVRC events: 88.87% [6] in 2013, 93.3% [7] in 2014, 96.4% [8] in 2015, and 97.7% [9] in 2017. This achieved precision improvement is the reason why CNN is becoming a promising solution for the abstract learning required in artificial vision applications [10]. As a consequence, CNNs are becoming a handy tool in many aspects of daily human life; applications range from license plate detection [11,12], face recognition [13,14], autonomous vehicles [15,16], advances in medicine [17,18], healthcare devices [19,20], and more.

Overall, the highest accuracy was reached by introducing novel network components and irregular connectivity between layers that differ from the conventional layer types [21,22]. To deal with a wide range of CNN configurations, algorithm-level solutions [23] used in CPU- or GPU-based platforms have been employed either for their computing capabilities or the several deep learning frameworks (*Caffe*, *Theano*, *Keras* or *Tensorflow* [24,25]) that enable sharing trained network parameters for new custom models. Nevertheless, these approaches are still unsuitable for resource-constrained hardware, particularly in GPU devices, where power consumption makes implementing batteries-based systems unfeasible. As for CPU-based platforms, real-time requirements are hardly met. Furthermore, the general trend is to delve into new deeper CNN topologies, increasing the number of convolutional layers, which are the computing core of the CNN algorithm. Consequently, millions of multiplications must be computed (Convolutional layers represent over 90% of the whole CNN computation [26]), which implies the use of high computational resources for real-time processing applications.

The fast proliferation of a wide variety of state-of-the-art CNN architectures implies the need to process different CNN configurations by CNN hardware accelerators and be customizable to achieve optimum performance and high efficiency in resource utilization. Thus, the need for efficient hardware accelerators arises, particularly on resource-constrained computing platforms. Hence, the FPGA technology has become the main target for modeling and testing new convolver hardware architectures [27,28].

Several reported papers have proposed to tackle the above-described issues from a hardware design perspective. Two main design strategies are well identified in the state-of-the-art works. The first design strategy (Streaming architectures) comprises one dedicated hardware convolver to compute each one of the CNN layers, where each convolver is highly optimized for its corresponding layer. Hence, more than one layer is processed in parallel with a fixed latency, and every convolver processes information from different input images. In [29], a novel method is proposed to obtain the best parallelism degree for the convolution layers by analyzing different parallelism types that can be exploited in convolutional layers. The work [21] also employs a streaming architecture. Given a CNN, its layers are mapped as building blocks chained together for coarse pipeline processing. Furthermore, this proposal considers non-regular dataflow networks [7,8,30]. The trade-off between performance and resource utilization of each used unit is independently tuned to fulfill the requirements of each layer in the design space exploration stage. A complete FPGA reconfiguration is needed in the current design when data have to be processed by a new CNN topology. According to reported results, streaming architectures can provide an increased throughput on several CNNs, but deeper CNNs can not be mapped using one dedicated hardware per layer.

The second design strategy is based on a Single Computation Engine (SCE) [31–34]. This design strategy focuses on developing flexible components that can compute over different convolution hyperparameters (filter dimension, Stride, Image size, etc.), where techniques such as systolic arrays of Processing Elements (PEs) or multiplier trees are used to address the flexibility feature. Thus, all CNN layers are sequentially computed by the SCE since the whole CNN model having a high number of layers is difficult to be mapped within a single chip. An example of a SCE-based CNN accelerator architecture is introduced in [32]. Its main computing component is an array of PEs, where a bank of convolvers, an adder tree, and an optional pooling path are contained in each PE. This proposal uses a data quantization strategy to reduce bit-width to 8-bit with negligible accuracy loss. Another

related approach is reported in [35], where data reusing is employed through the chain of the convolution, pooling, and ReLU processes resulting in a pipeline architecture with an external memory accesses reduction. The flexibility feature is achieved by clustering the hardware operators in a collection module. The convolution operator within each collection can be configured to parallel perform one 12×12, four 6×6 , or sixteen 3×3 mask convolutions. This flexibility idea was also adopted in [36]. Data reuse of hybrid patterns for several layer sizes is described in [37], where input data, convolution kernel, and partial results reusing is exploited through a reconfigurable data path. In addition, higher energy consumption is required for data transferring to external memory, even in some cases greater than the required for data computing [38]. To address this issue, energy consumption reduction based on efficient external memory access is explored in [39] where a kernel-partition approach is used. In addition, a performance improvement is achieved by the hardware accelerator reported in [40], where intra-parallelism is favored by cascading PEs [29]. An array of multipliers and accumulators with prioritized weights is proposed in [41] to achieve energy consumption reduction and processing optimization.

All of these works can achieve outstanding performance with low resource utilization. Nonetheless, the size of the convolution mask varies for every layer, which decreases the CNN computing efficiency when a single convolution component is used to perform the convolution with different mask sizes. In this sense, streaming architectures are an impractical option due to the need to design a hardware architecture for each CNN configuration. This paper describes the architecture of a systolic array-based general-purpose hardware convolver, whose main feature is the capability of adapting to any convolution hyperparameter. Unlike the state-of-the-art works, where the standard algorithm for convolution is modified to improve the performance, the proposed convolver can receive raw feature maps and weights to deliver an output whose precision is only limited by the number of bits in data representation. The convolution operation with systolic arrays is usually conducted as a matrices multiplication in related works [42], which requires a previous processing stage, and irregular data memory access [10]. Tiles-based data processing is often used too, but a data processing stage after the convolution computing is implied to compensate for the quantization error due to not being able to apply the filters on the edges of the tiles [37]. The proposed architecture design belongs to the second design strategy (SCE), which allows it to be used in classification, object recognition, or segmentation applications in digital images processing without any additional adaptation. Mobile on-chip data processing, not requiring cloud computing, avoiding internet potential security risks, is one of the potential applications [43]. The deployment of CNN with any configuration targeting limited-resources hardware is possible thanks to this new design strategy. The work described in this paper has the following differences from other reported ones:

- State-of-the-art works typically modify the convolution algorithm, which involves tuning the hardware to a single CNN configuration. The proposed method allows processing any convolution layer configuration without customizing the standard algorithm. Hence, the convolver architecture can be adapted to different CNN structures.
- The present work proposes an adaptive line buffer, where the convolution masks are dynamically generated; that is, the proposed buffer is programmable where the mask size can be defined through the external control. The need to reconfigure the design for each new CNN is avoided.
- The proposed architecture flexibility is demonstrated by its implementation in different CNN layers. From the results, a potential competitive total processing time is achieved compared with recently reported architectures.

The paper organization is as follows. Section 2 is divided into three parts: First, a brief description of CNN definitions is presented. Second, a general explanation of the proposed architecture is presented. Lastly, the detailed description of the proposed systolic architecture is depicted. The experimental results and discussion are presented in Sections 3 and 4, respectively.

2. Materials and Methods

Each convolution layer input in the CNN is the features maps set $\mathcal{F} \in \mathbb{R}^{M_{in}^{(l)} \times N_{in}^{(l)} \times CH^{(l)}}$ that is processed through the filters $\mathcal{K} \in \mathbb{R}^{w^{(l)} \times w^{(l)} \times CH^{(l)} \times K^{(l)}}$ for generating a new feature maps set $\mathcal{G} \in \mathbb{R}^{M_{out}^{(l)} \times N_{out}^{(l)} \times K^{(l)}}$ (Figure 1). The convolutional layer is represented by the index l. $\mathcal{K}(x, y, ch, k)$ represents the filter set that processes the features maps set $\mathcal{F}(x, y, ch)$, and each convolution layer must be computed by K filters. The CNN elements' dimensions for each layer are depicted in Figure 1, where $w \times w \times CH$ represent the k filters dimension, and w is typically odd. The Algorithm 1, with an optional interchange in the two external loops (CH and K), represents a convolution computing starting point, which has been considered in many state-of-the-art works [10,44,45]. The loop interchange, unrolling, or modification has effects on the hardware implementation technique. This standard algorithm is not modified in this work and is taken as a reference for the proposed convolver validation. Thus, a higher priority to the operations acceleration in the inner loops is given, and less priority to the outermost loop parallelism.

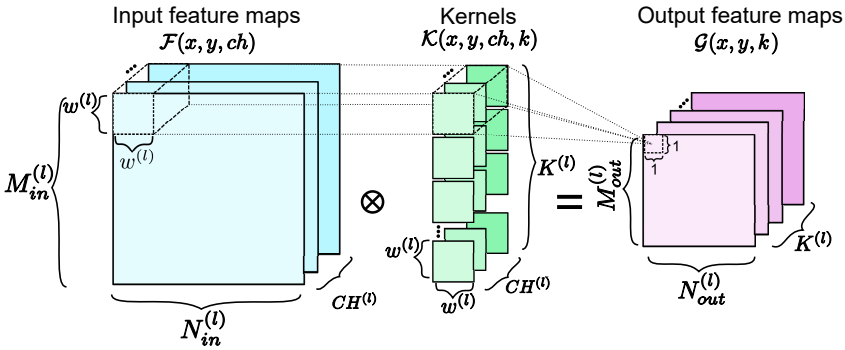


Figure 1. Elements' dimensions for the convolution process.

Algorithm 1: Standard pseudo code of a convolutional layer

```
for Ch from 1 to CH^(l) do
    for k from 1 to K^(l) do
        for y from 1 − P to N^(l+1) + P do
            for x from 1 − P to M^(l+1) + P do
                perfom G[x, y, k] += Σ_{i=0}^{w^(l)−1} Σ_{i=0}^{w^(l)−1}
                  F[x × S + i, y × S + j, ch] ⊗ K[i, j, ch, k]
```

The *Stride* (S) and *Padding* (P) parameters in the convolution layer are essential for data processing [46]. The former is helpful to avoid the whole input image being processed by the filter kernels, resulting in a dimension reduction of the resulting features maps set. The latter parameter helps to maintain the same image size regarding the obtained output feature maps, whose size is given by:

$$N_{out}^{(l)} = M_{out}^{(l)} = \frac{N_{in}^{(l)} + (2 \times P) - w^{(l)}}{S} + 1. \quad (1)$$

In general, a convolution layer is followed by a subsampling operation, where the size of the resulting convolution layer image is reduced through the computation of the maximum or average value of a specific image region. The most commonly used pooling methods are: *Maxpooling* and *Meanpooling* defined respectively by:

$$\mathcal{G}(x,y,ch) = \sum_{s=0}^{p-1}\sum_{t=0}^{p-1}\frac{1}{p^2}\mathcal{F}((x'S)+s,(y'S)+t,ch), \tag{2}$$

$$\mathcal{G}(x,y,ch) = max(\mathbf{f}(ch)), \tag{3}$$

where $\mathbf{f} \in \mathbb{R}^{w^{(l)}\times w^{(l)}\times CH^{(l)}}$ is a convolution mask of $\mathcal{F}$ whose position in $\mathcal{F}$ is (x', y') , having as origin the upper left pixel, and subsampling window dimension p. x and y are in the range $[0, M_{out}]$ whereas x' and y' have a range $[0, M_{in}]$.

In most cases, a Fully Connected Network (FCN) is the CNN last data processing stage, where a class (represented as C) is assigned to the resulting feature maps set ($\mathbf{x}$) corresponding to the previous layer. The set $\mathbf{x}$ represents the classification stage input.

An FCN has n_l layers, where L_1 and L_{nl} are the input and output layers, respectively. The l_{th} layer has Sl neurons, whereas the layer L_{nl} has C neurons equal to the number of FCN classes. The i_{th} neuron activation on l_{th} layer is denoted as $a_i^{(l)}$. This activation is given by the function $f(z_i)$, where z_i is the weighted sum of the input data values: $x_1, x_2, \ldots, x_n = \mathbf{x}$ plus the bias term $x_0 = 1$; therefore, $z_i = x_0 w_{i0} + x_1 w_{i1} + \ldots + x_n w_{in}$. The addition of the term $x_0 w_{i0}$ defines the $f(z_i)$ as a nonlinear function. The following layer activation a_i^{l+1} is defined as:

$$a_i^{l+1} = f(z_i^{l+1}) = f(\sum_{j=1}^{n} w_{ij}^{(1)} x_j + b_i^l). \tag{4}$$

A matrix of PEs is the core of a Systolic array (SA), each composed of a multiplier and accumulator module with internal registers. An SA for image processing is illustrated in Figure 2.

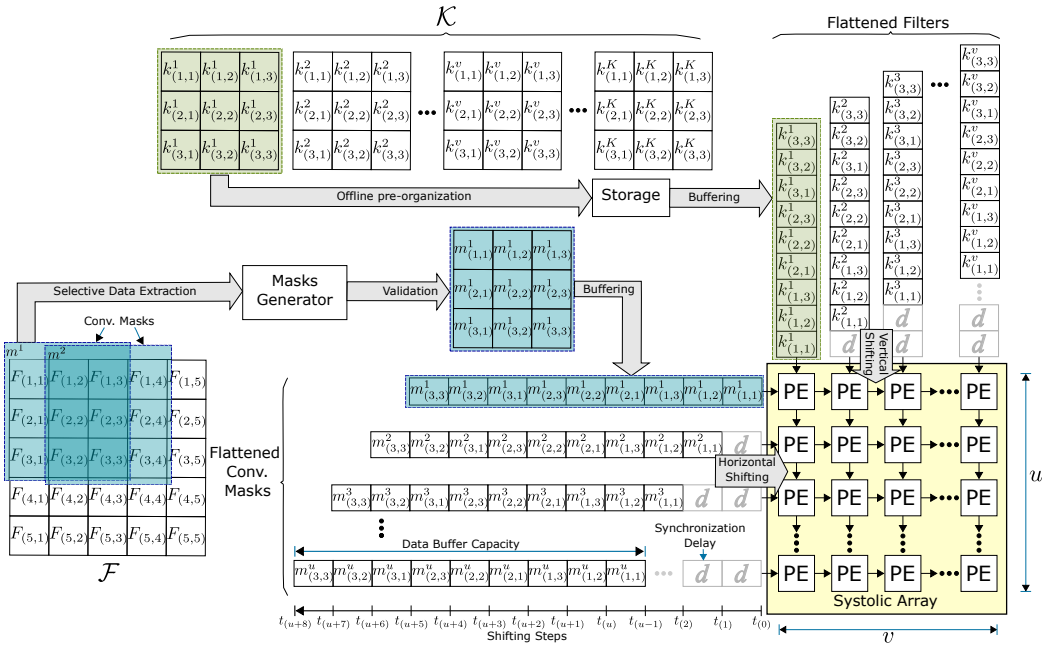


Figure 2. SA convolution processing procedure. As an example, a convolution $\mathcal{F} \otimes \mathcal{K} = \mathcal{G}$ where $w = 3$, $S = 1$, $P = 0$, $M_{in} = N_{in} = 5$ and $K = 4$ was taken as a reference.

The central computing feature of these strategies is the parallel processing of each CNN operation. In order to achieve an improvement in the CNN processing time for given hardware resources of a particular device, different types of parallelism and computation patterns are taken into account in the SA architecture design. The parallelism types implemented in the proposed computing are described in [29,47]. The state-of-the-art parallelism types are equivalent but have different terms; therefore, we adopt the following:

Layer parallelism: Data from different layers are pipeline processed through parallel computing of two or even more convolution layers. This type of parallelism is characteristic of streaming architectures and can be used when the target device contains enough hardware resources to host a convolver for each convolution layer. For this work, this type of parallelism is not considered.

Loop parallelism: This type of parallelism is based on the fact that all convolutions between feature maps and kernels within a convolutional layer can be parallel processed due to their independence. Hence, this parallelism can be exploited as: *intra-output* (denoted as n_{fi}) and *inter-output* (denoted as n_{fo}) parallelisms [48]. The number of input/output feature maps that are parallel computed is related to the first/second parallelism type.

Operation parallelism: The Basic CNN computing operator is the convolution (between a $w \times w$ convolution mask and a $w \times w$ kernel). It can be understood that the input image is scanned from left to right and top to bottom, creating $N_{out}^{(l)} \times N_{out}^{(l)}$ convolution masks with a $w \times w$ size. An element-wise multiplication is obtained for each mask position by the SA. In this case, operation parallelism is represented as n_m and corresponds to the number of used multipliers in such element-wise multiplication.

Data parallelism: Different input feature maps set areas are processed by performing a convolution with each convolution mask; these operations are simultaneously computed by replicating the SA rows.

Task parallelism: This parallelism type consists of processing more than one input image in parallel and implementing more than one CNN within the same chip. This type of parallelism is the least explored since state-of-the-art works focus on accelerating the convolution operation in one layer, taking most of the chip's resources.

Besides parallelism types, computing patterns are considered. These refer to the number of data movements for weight, input, and output data [37] and are defined as:

Weight Reuse (WR): Power consumption reduction is achieved by reusing the filters readings from external memory.

Input Reuse (IR): The reuse of the processed feature maps within the convolution layer. Such reuse type is the basics of the described hardware architecture. The number of external memory accesses is reduced because each pixel of the input feature maps set is read only one time.

Output Reuse (OR): The computed convolution layer data are stored in an output buffer until the final result is obtained.

The SA operands' inputs are defined by the SA edges and propagated by the PE internal registers when each multiplication–accumulation operation is finished. The number of rows in the SA determines the number of convolution masks that can be processed in parallel (nv). At the same time, the number of columns indicates the number of filters also to be parallel processed (nf_o). Convolution filters can be organized offline before processing. Depending on the memory capacities of the target device, the weights can be on- or off-chip stored typically on RAM memory. Regardless of where the weights are stored, each filter must be flattened and allocated in its respective column within the SA.

The data flow, which is called Time Multiplexed stationary Outpus (TMSO), establishes that the filter weights and masks values are shifted up to down and left to right, respectively, so that each PE stores the partial results until the computation is completed and is read from the accumulator to start new computing. Similarly, the convolution masks are ordered in different SA rows as long as they belong to a valid mask since, due to stride and padding, not all masks must be processed. It should be noted that synchronization delays must be added in the SA input registers in order to operate correctly on the data. Specifically, the number of synchronization delays for each row is equivalent to its position. Because

the convolution mask is generated sequentially, the storage in the SA input registers is also sequential. Thus, the top row does not require delay components since it is the first position where a convolution mask is stored. The following rows will then require 1, 2, 3, ... synchronization delays. In each SA column, the adjacent output image pixels corresponding to the same channel are obtained through the PEs of each SA row. Image pixels of different output feature maps are obtained in each SA column [49].

In SCE-based architectures, the image processing is given by a sequential computation of the convolutional layers. Therefore, the hardware components represented in the block diagram of Figure 3 are used for experimentation. The described strategy is not restricted to one specific device; the hardware architecture can be implemented on any FPGA fulfilling the required hardware resources for a given limited-resources application. The hardware design was developed in VHDL with Quartus Prime Lite 20.1, taking the FPGA Cyclone V SE5CSXFC6D6F31C6N as a reference, which is the one available in the employed ALTERA DE10-Standard board.

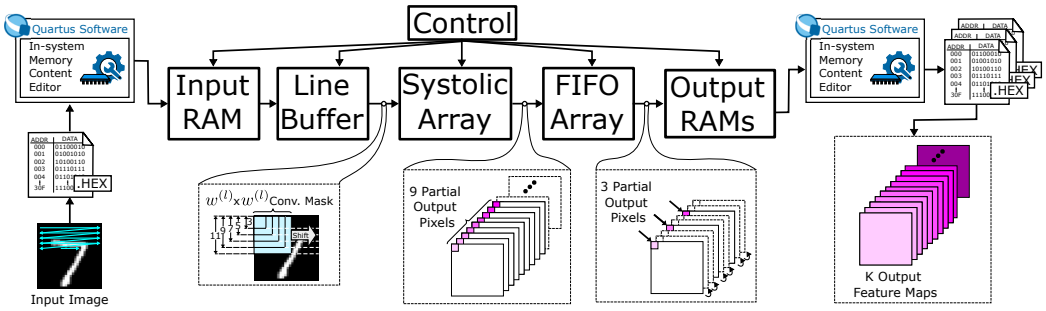


Figure 3. General block diagram of the proposed system.

The In-system memory content editor tool was used to read and write over the input and output memories. Thus, input images and weights in a personal computer are transferred to the FPGA before processing starts. Likewise, the results were sent to the PC for further analysis or processing. At the beginning of the processing, the convolution masks are generated by the Programmable Line Buffer (PLB) and directly conducted toward the SA that performs the convolution. The convolution results are stored in an array of FIFO memories and later transferred to a RAM memory containing the final output results. A control module synchronizes the operation of the main components.

An example of generating a maximum 5×5 window size and capable of obtaining a 3×3 one is shown in PLB output in Figure 4. Although the depicted PLB case is for 5×5 or 3×3 generation, the design has the processing capacity for any combination of filter sizes (scalable architecture) by adding the required derivations. Components in blue indicate the configurable data paths used to control the mask sizes. Firstly, the pixels of the input image are stored in FFD $R_{1,1}$, and then the data are clock-synchronously moved to the right-located register (depicted arrow flow direction). The data path in multiplexers for generating a 3×3 convolution mask in $R_{1,1}$ to $R_{3,3}$ are defined through the control module when $w = 3$. In the processing of a new convolution layer, both w parameter and the size of the input feature map are programmed. The subsequent value is smaller in lower layers than in higher ones. The PSR comprises several shift registers in cascade where data are transferred or skipped depending on the number of feature map columns being processed. The length of each shift register is laid out in powers of two; thus, for instance, when $N_{in} = 32$ and $w = 5$, shift registers SR16, SR8, SR2 and SR1 must be used. The control unit is notified by a synchronizer when a valid to-be-processed convolution mask was generated within the PLB. Such operation is completed depending on the P and S values; for instance, when $P > 0$, the PLB is filled up with zero values, hence the image data are not read from the input RAM memory.

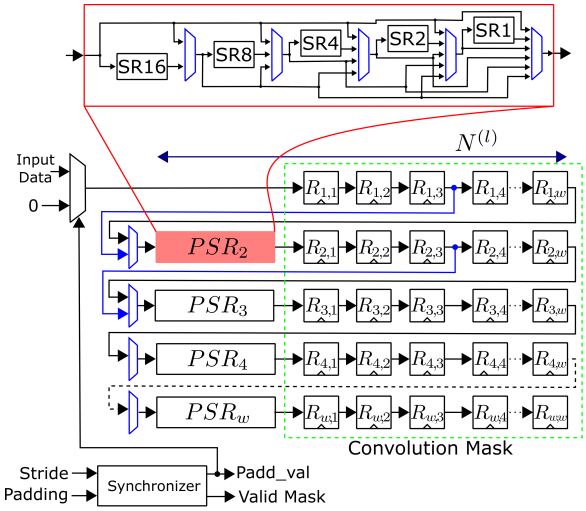


Figure 4. Generation of 3×3 or 5×5 convolution masks through the programmable line buffer.

The block diagram of the proposed flexible SA is depicted in Figure 5. When a valid convolution mask is generated by PLB, Mask buffers (MB), which serve as row inputs for the SA, concurrently receive the reshaped $w \times w$ masks. The number of SA rows determines the number of masks to be parallel processed (nv); thus, each mask is transferred to each MB as the PLB generates them. Once the MBs are full, the general control should stop entering data into the PLB until the processing in the SA is finished. Previously, the filter coefficients must be loaded as shift registers sets, called Passive Weight Buffers (PWBs), in order to start the processing when the data content of MB is complete. When the PWB registers are filled, they are transferred to new registers called Active Weight Buffers (AWB). The purpose of performing this transfer is to be able to operate on the values stored in the AWB while the PWB is being loaded with new coefficients. This ping-pong strategy reduces processing latency and dead time.

Given a w value, the size of the AWB register is established through the multiplexers to carry out a weight loading to the PEs' nearest position. The data flow of TMSO is accomplished through the SA, and the control unit does the PE result parallel reading for each available column. The SA row number is defined by the square of the current minimum value of w because of the use of a feedback path (Fb) for the weights transfer to the PE input to be reused on the following convolution masks. Once all weights are moved to the PEs in one column, and the w value is greater, it must be returned to the position number obtained by the squared w minus the number of SA rows. For instance: if $w = 5$, the weight value is returned to the 16th position of the PWB nearest register to the first PEs' row. The vertical shift function supplies the filter coefficients for each loaded mask sequentially. Hence, parallel data processing is established, defined by the number of mask buffers. Filter parallelism is allowed by the number of SA columns. More parallel processing filters imply a greater number of SA columns.

At the end of the processing, the partial results are distributed in the SA PEs. Each stored value is obtained from the convolution between a mask and the corresponding filter. The control module sequentially selects each PE to read its stored value, adds the bias in the bias module, and then stores the values in a set of FIFO memories. Output Feature Maps (OFM) are formed and stored by reading from the FIFO memories in the respective order. These values remain stored in a double port RAM that allows the values to be accumulated when writing in the same address. With this, it is possible to store the partial and final results of $\mathcal{G}$ in the same memory.

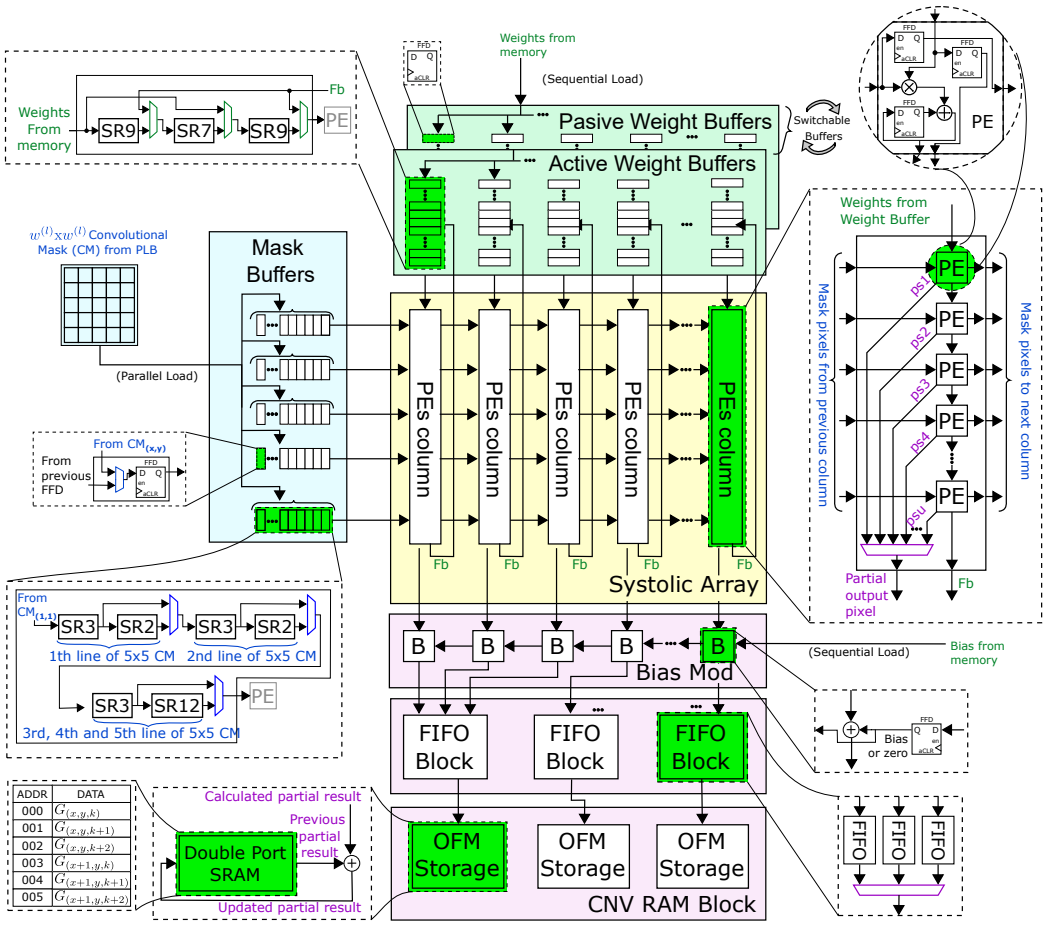


Figure 5. Flexible SA block diagram for convolution with filter sizes of 3×3 and 5×5.

3. Results

The performance results for the flexible convolver are presented in this section. The proposed method was validated by implementing different representative convolution layer configurations of state-of-the-art CNNs. For its hardware implementation, the convolution filters' coefficients were generated in Matlab using random numbers. Subsequently, these coefficients were converted to their 8-bit fixed-point representation in the decimal part since this represents a trustworthy standard for its implementation in FPGAs [50]. Arbitrary images with pixel values normalized to the range [0,1] were used as convolver input data. The convolver output data were compared with those obtained through software processing (Matlab) to validate correctness. It should be noted that the employed validation method is a convolver performance evaluation taking into account processing time, number of operations per second, and power consumption. Nonetheless, the proposed method does not lose generality to be applied to different convolution layer configurations; the system, in fact, is intended to be applied to configurations (config) such as those shown in Table 1.

The achievable processing time by the proposed architecture is defined by Equation (5), where T is the period of the clock signal and C_{conv} is the required clock cycles to perform either a shift in the PLB or a multiply–accumulate operation in the SA. Due to the nature of the processing, $nm = 1$, thus operation parallelism is not exploited. Instead, hardware resources (multipliers) are intended for filter and data parallelism. In addition, the intra-output parallelism (n_{fi}) was set to the minimum value.

Table 1. Convolutional layers configurations for popular CNN models.

	Alexnet						VGG16						Darknet19					
# Conf	$M_{in} \times N_{in}$	CH	K	w	S	P	$M_{in} \times N_{in}$	CH	K	w	S	P	$M_{in} \times N_{in}$	CH	K	w	S	P
1	227×227	3	96	11	4	0	224×224	3	64	3	1	1	224×224	3	32	3	1	1
2	27×27	96	256	5	1	2	224×224	64	64	3	1	1	112×112	32	64	3	1	1
3	13×13	256	384	3	1	1	112×112	64	128	3	1	1	56×56	64	128	3	1	1
4	13×13	384	256	3	1	1	112×112	128	128	3	1	1	56×56	128	64	3	1	0
5	-	-	-	-	-	-	56×56	128	256	3	1	1	28×28	128	256	3	1	1
6	-	-	-	-	-	-	56×56	256	256	3	1	1	28×28	256	128	1	1	0
7	-	-	-	-	-	-	28×28	256	512	3	1	1	14×14	256	512	3	1	1
8	-	-	-	-	-	-	28×28	512	512	3	1	1	14×14	512	256	1	1	0
9	-	-	-	-	-	-	14×14	512	512	3	1	1	7×7	512	1024	3	1	1
10	-	-	-	-	-	-	-	-	-	-	-	-	7×7	1024	512	1	1	0
11	-	-	-	-	-	-	-	-	-	-	-	-	7×7	1024	1000	1	1	0

$$Pt = T \times C_{conv} \times [((w^2 + n_{fo} + nv) \times \frac{(N_{out})^2}{nv} + nv) \times \frac{K}{n_{fo}} \times \frac{CH}{n_{fi}}] \quad (5)$$

It should be noted that Equation (5) only considers the processing time for convolutions on valid masks. Additionally, the time needed to shift data in the PLB (dead time) in order to generate valid masks must be considered. The number of cycles required by design to generate a not valid mask is defined as:

$$Lat_{buffer} = \underbrace{N_{in}^{(l)} \times (w-1) + w}_{\text{initial latency}} + \underbrace{((N_{in}^{(l)})^2 - (N_{out}^{(l)})^2)}_{\text{invalid mask latency}} \quad (6)$$

The initial latency represents the necessary clock cycles to generate a valid mask from the first value read from the input memory. The invalid mask latency is produced by the parameter S; thus, the greater the value of S, the greater the dead time. Table 2 shows the processing time calculated with Equations (5) and (6) for the convolutional layers configurations shown in Table 1. Additionally, the processing time for the convolutional layers of CNNs taken as test cases in previous studies [22] is shown. These are one-, two-, and three-convolution layers of naive CNNs trained for classification tasks under the MNIST [51] dataset. Information regarding the processing time of a modified CNN Alexnet has also been included. It has the characteristic of operating with input image dimensions of $67 \times 67 \times 3$ and being trained with the CALTECH101 dataset [52]. For each CNN, the processing time required to process all the convolution layers is shown. Note that overall time processing corresponds, in fact, to all the layers of the convnets, not only to those shown in the table. This provides an estimation of the attainable processing time for each CNN. The fourth column reports the processing time obtained in hardware. The difference in time is due to overhead processing that is related to the control logic.

Table 2. Processing time calculated from Equations (5) and (6) and obtained in hardware processing for different state-of-the-art CNN convolution layers.

CNN Model	Config	Processing Time (Calculated)	Processing Time (Hardware)
CNN1	1/Overall	245.6 us	245.6 us
CNN2	1	82.14 us	100.26 us
	2	59.32 us	61.57 us
	Overall	141.46 us	161.83 us
CNN3	1	109.98 us	134.49 us
	2	88.30 us	91.69 us
	3	35.45 us	35.57 us
	Overall	233.73 us	261.75 us
Modified Alexnet	1	1.73 ms	1.73 ms
	2	10.54 ms	10.54 ms
	3	8.72 ms	8.73 ms
	4	13.09 ms	13.09 ms
	5	8.72 ms	8.72 ms
	Overall	42.80 ms	42.81 ms
Alexnet	1	22.96 ms	23.05 ms
	2	155.04 ms	155.07 ms
	3	99.16 ms	99.17 ms
	4	99.16 ms	99.17 ms
	5	66.11 ms	66.12 ms
	Overall	442.44 ms	442.58 ms
VGG16	1	55.06 ms	56.57 ms
	2	1174.45 ms	1175.97 ms
	3	587.27 ms	587.66 ms
	4	1174.55 ms	1174.93 ms
	5	587.49 ms	587.58 ms
	6	1174.98 ms	1175.08 ms
	7	588.35 ms	588.37 ms
	8	1176.70 ms	1176.73 ms
	9	295.89 ms	295.90 ms
	Overall	9.75 s	9.76 s
Darknet19	1	27.53 ms	29.05 ms
	2	146.82 ms	147.20 ms
	3	146.88 ms	146.97 ms
	4	110.17 ms	110.26 ms
	5	147.09 ms	147.11 ms
	6	110.39 ms	110.41 ms
	7	147.95 ms	147.95 ms
	8	111.25 ms	111.25 ms
	9	151.39 ms	151.39 ms
	10	114.69 ms	114.69 ms
	11	225.79 ms	225.79 ms
	Overall	2.62 s	2.89 s

The amount of memory used by varying n_{fo} and n_{fi} are the register number hosted within the n_{fi} line buffers if the device is large enough to leverage intra-output parallelism, where it is established that it is necessary to use a line buffer for each parallel processed feature map, where the amount of memory used in a convolver (quantified in FFDs) can be estimated by:

$$Mem_{PLB} = nf_i \times (N_{in}^{(d)} \times (w^{(j)} - 1) + w^{(j)}), \tag{7}$$

where j represents the convolution layer that contains the largest filter size and a d layer that receives the feature maps with the largest number of rows/columns, which is usually the first layer. The memory usage in the SA depends on its dimensions. The number of columns is related to n_{fo}, which is selected based on the common divisors (powers of 2) of the typical numbers of filters. Since the number of multipliers in the device used is 112, the number of columns in the SA was set to 16, so nv must be equal to 7, adjusting the architecture to the device capabilities. Each PE contains three FFDs, resulting in 336 FFDs within the SA (Mem_{SA}). The memory used in the AWB and PWB is equivalent to $Mem_{WB} = Mem_{AWB} + Mem_{PWB} = (w^{(j)})^2 \times n_{fo} \times 2$; in the case that $w^{(j)} = 5$, 800 FFDs would be used. Finally, since the number of weight buffers equals 7 ($nv = 7$), the number of FFDs can be calculated as $Mem_{MB} = ((w^{(j)})^2 \times nv) + nv$. It should be noted that additional memory is used for the convolver input and output memories. Nevertheless, only the memory directly consumed by the convolver architecture is taken into account for its quantification. Hence, the total memory (Mem) is given by:

$$Mem = Mem_{PLB} + Mem_{MB} + Mem_{WB} + Mem_{SA}, \tag{8}$$

Table 3 summarizes the proposed convolver's performance results, taking into account seven different CNNs. Likewise, information about the obtained results by other authors in related works is provided. This table shows the design strategy (method) for each work, the complexity of the CNN model, the throughput (quantified in millions of operations per second), and the time required to process an image. In the case of complexity, for the works [22,53] and ours, the total number of multiplications required by the network is considered. For the remaining works, it is assumed that complexity is expressed in multiplications–accumulations (GMACs or GOps) since no information is provided on how this metric was quantified. The processing time shown is obtained through hardware processing. The processing time per image for our architecture is 245.6, 161.83, and 261.75 us for the one-, two-, and three-convolution layer CNNs, respectively. In the case of the custom CNN with three convolution layers, this time is less than that obtained by our previous works [22,53] and that reported in [54] that uses a CNN of similar complexity. The throughput for our architecture is in the range of [0.95, 2.71] GOp/s, which is superior to the works that are based on a similar CNN model.

Table 3. Performance comparison with recently reported works.

Work	Method	CNN Model	Complexity	Throughput (GOp/s)	Pt/Image
This Work	PLB for flexible mask generation and flexible SA for standard convolution	One Layer Custom CNN	668,000	2.71	245.6 us
		Two Layers Custom CNN [a]	240,266	1.48	161.83 us
		Three Layers Custom CNN [a]	250,250	0.95	261.75 us
		Modified Alexnet [a]	65.81 MMACs	1.53	42.81 ms
		Alexnet [a]	1.076 GMACs	2.43	442.58 ms
		VGG16 [a]	15.34 GMACs	1.57	9.76 s
		Darknet19 [a]	2.79 GMACs	0.96	2.89 s
[22] 2020	Layer-operation-chaining-based processing on a streaming architecture for a single-image processing	Three layers custom CNN	250,250	0.354	706.36 us
[53] 2019	convolvers reusability on a streaming architecture employing layer-operation-chaining approach	Three layers custom CNN	250,250	0.6271	399 us
[39] 2021	Kernel partition SA	Alexnet VGG16	1.45 GOPs 30.72 GOPs	220 230.1	6.6050 ms 137.3626 ms
[40] 2020	Channel oriented data pattern SA	Alexnet [a] VGG16 [a]	1.076 GMACs 15.3466 GMACs	10.5 12.5	- -
[41] 2019	Parallel MAC modules and DMA input data control	Modified AlexNet	1.44 GOPs	198.1	12.2 ms
[54] 2020	RAM based LB for fixed 5×5 mask generation and parallel DSP block multiplier	Custom CNN for MNIST	209,280	0.598 GMAC/s	526 us
[55] 2021	Cascade of a parallel multichannel point-wise convolution stage + multichannel depth-wise convolution stage	RAd-MobileNet	43.54 MMACs	-	-
[56] 2020	Controlled loading of inputs to each PE from separate memories and CORDIC radix-4 computing instead of multiplications	VGG-16 [a]	15.3446 GMACs	177.3	5.76 ms

[a] Conv. Layers processing only.

Throughput depends on the convolutional layer hyperparameters and hardware resources intended for operations. As shown in Table 4, our design uses only 112 multipliers, which restricts maximum throughput; only Ref. [40] reports a usage of 64 multipliers (DSPs) which is less than ours; nonetheless, details of the rest of the hardware resources utilization is not provided. Regarding logic elements usage (Slices in the case of Xilinx-FPGAs-based architectures and Adaptive Logic Modules (ALMs) when Intel FPGAs are used), our architecture is the one with the least hardware resources usage, excluding work [56] where a multiplierless processing is proposed. To estimate power consumption, the Intel *Early Power Estimator* (EPE) tool was used. The result shows that the proposed convolver consumes only 0.289 W, being the best-reported value. The memory usage for our case only considers the flexible convolver. Hence, the amount of memory required to store input images is excluded. It is important to highlight that the results shown in Table 4 for our

work correspond to a convolver configured to operate over the convolution layer settings shown in Table 2.

Table 4. Resource utilization in related convolver designs.

Work	Device	Working Frequency (MHz)	Slices	Registers	DSPs	Memory	Power Consumption (W)
This Work	Cyclone V SE5CSXFC6D	100	11,597 [a]	24,105	112	698.11 kb	0.289
[39] 2021	Virtex 7 VC709	200	121,472	159,872	664	16.812 Mb	9.61
[40] 2020	Zynq XC7Z045	100	-	-	64	-	-
[41] 2019	Zynq XC7Z045	200	71,741	79,347	576	1.476 MB	-
[54] 2020	Artix-7 XC7A200T-SBG484	50	88,756	42,038	571	LUTRAM = 596, BRAM = 218	1.225
[55] 2021	Virtex-7 xc7vx980	225	57,438	79,327	937	5.11	3.25
[56] 2020	Virtex 5 XC5VLX5OT	496	1465	519	0	45 kB on-chip and 134 MB off chip	54

[a] ALMs.

4. Discussion

The validation process was carried out considering a low-end device. The attained processing time (Table 2) indicates that the proposed architecture can be deployed in applications based on CNN models with low computational complexity. The implementation on a device with a greater amount of hardware resources (mainly DSPs) would reduce the processing time as well as increase performance. With the exception of work [40], the other architectures use more than 5 times the number of multipliers, which limits their migration to devices with less hardware resources. The performance achieved by our architecture was obtained based on the number of multiplications (complexity) of the CNN model; if the number of multiplications and shifts are also considered, the performance would be at least twice as high since the number of additions in hardware corresponds to the number of multiplications. Note that the results in this work are obtained by the standard algorithm for convolution, which gives generality to the architecture.

The power consumption and processing time achieved by the described hardware architecture for the considered low-complexity CNN models show a potential use in portable real time image processing applications with a sampling frame rate of 30 and 60 images per second. Although the achieved throughput was not the best, the data processing is achieved through small hardware resources, which is appropriate for restricted-resources applications such as embedded systems. In addition, the reduced power consumption makes the proposed convolver suitable for battery-based systems.

The proposed adaptation to different CNN parameters variabilities such as stride, padding, size of input images, and filters is the base of the flexibility feature for the presented convolver architecture. This flexibility feature allows its use in different scenarios where convolution computation is required. According to the convolutional layers to be computed, PLB must be configured, creating the corresponding derivations both on shift registers and the FFD matrix. Likewise, the proposal of a programmable systolic array allows operation on different convolution layer configurations, exploiting data and filter parallelism based on the number of rows and columns of the SA, respectively. This logic is easily scalable to different images and kernel sizes as required, allowing for taking the present design as a reference for selecting

the device to use. Furthermore, the described strategy is not restricted to one specific device; the hardware architecture can be implemented on any FPGA fulfilling the required hardware resources for a given limited-resources application.

It should be noted that the processing time achievable by any architecture based on the standard algorithm is defined by the following:

$$Pt = T \times C_{conv} \times [w^2 \times \frac{(N_{out})^2}{nv} \times \frac{K}{n_{fo}} \times \frac{CH}{n_{fi}}] \quad (9)$$

Future work includes adding a data flow strategy that removes the additional terms from Equation (5). This would allow for getting a system based on the standard algorithm capable of obtaining the shortest possible processing time, having as a new area of research the reduction of both hardware resources and energy consumed.

The adaptability to different CNN configurations opens the possibility of exploring emerging areas, such as the early detection of COVID-19 [57] or Attention-deficit/hyperactivity disorder (ADHD) [58] through image analysis, where the investigation of different CNN configurations is common in order to achieve the highest possible efficiency in terms of recognition efficiency and reliability degree.

5. Conclusions

A flexible convolver with an adaptation capacity to meet the different configurations required in the convolution layers of a CNN is described in this paper. Unlike the convolution accelerators reported in the state-of-the-art works, no modifications of the standard convolution algorithm are required for the proposed convolver, which can be directly applied without any additional configuration. The flexibility feature is achieved using two proposed Programmable components; The first is a Programmable Line Buffer based on a novel Programmable Shift Register that allows mask convolution generation through configuration bits. The second one is the proposed flexible systolic array based on the standard algorithm for convolution operation. This way, there is no need to change the convolver design for every new CNN configuration. Likewise, the systolic array is designed from the available multipliers on the target device, which makes the proposed design independent of the used device. The performance results show that the present architecture is suitable for deploying low computational complexity CNN models onto low-end devices and battery-based systems.

Author Contributions: Conceptualization, M.A.-V. and J.D.-C.; methodology, J.D.-C. and A.P.-M.; software, M.A.-V. and J.P.-O.; validation, M.A.-V., P.A.A.-Á., and J.P.-O.; formal analysis, J.D.-C. and A.E.-C.; investigation, M.A.-V. and P.A.A.-Á.; resources, J.D.-C. and A.E.-C.; data curation, M.A.-V. and A.P.-M.; writing—original draft preparation, M.A.-V.; writing—review and editing, J.D.-C.; visualization, P.A.A.-Á.; supervision, M.A.-V.; project administration, A.P.-M.; funding acquisition, J.D.-C. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Tecnológico Nacional de México (TECNM) and Consejo de Ciencia y Tecnología (CONACyT) with Grant Nos. 14099.22-P and 1109193, respectively.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

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

References

1. Aguirre-Álvarez, P.A.; Diaz-Carmona, J.; Arredondo-Velázquez, M. Hardware Flexible Systolic Architecture for Convolution Accelerator in Convolutional Neural Networks. In Proceedings of the 2022 45th International Conference on Telecommunications and Signal Processing (TSP), Prague, Czech Republic, 13–15 July 2022; pp. 305–309.
2. Lecun, Y.; Bengio, Y.; Hinton, G. Deep learning. *Nature* **2015**, *521*, 436–444. [CrossRef] [PubMed]

3. Lindsay, G.W. Convolutional Neural Networks as a Model of the Visual System: Past, Present, and Future. *J. Cogn. Neurosci.* **2020**, 1–15. ._a_01544. [CrossRef] [PubMed]
4. Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M.; et al. ImageNet Large Scale Visual Recognition Challenge. *Int. J. Comput. Vis.* **2015**, *115*, 211–252. [CrossRef]
5. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. Imagenet classification with deep convolutional neural networks. *Commun. ACM* **2017**, *60*, 84–90. [CrossRef]
6. Zeiler, M.D.; Fergus, R. Visualizing and understanding convolutional networks. In Proceedings of the European Conference on Computer Vision, Zurich, Switzerland, 6–12 September 2014; Springer: Berlin/Heidelberg, Germany, 2014; pp. 818–833.
7. 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.
8. 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.
9. Hu, J.; Shen, L.; Sun, G. Squeeze-and-excitation networks. In Proceedings of the IEEE Conference on COMPUTER Vision and Pattern recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 7132–7141.
10. 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]
11. Xie, L.; Ahmad, T.; Jin, L.; Liu, Y.; Zhang, S. A new CNN-based method for multi-directional car license plate detection. *IEEE Trans. Intell. Transp. Syst.* **2018**, *19*, 507–517. [CrossRef]
12. Pham, T.A. Effective deep neural networks for license plate detection and recognition. *Vis. Comput.* **2022**, 1–15.. [CrossRef]
13. Kim, J.; Kang, J.K.; Kim, Y. A Resource Efficient Integer-Arithmetic-Only FPGA-Based CNN Accelerator for Real-Time Facial Emotion Recognition. *IEEE Access* **2021**, *9*, 104367–104381. [CrossRef]
14. Vu, H.N.; Nguyen, M.H.; Pham, C. Masked face recognition with convolutional neural networks and local binary patterns. *Appl. Intell.* **2022**, *52*, 5497–5512. [CrossRef]
15. Aladem, M.; Rawashdeh, S.A. A single-stream segmentation and depth prediction CNN for autonomous driving. *IEEE Intell. Syst.* **2020**, *36*, 79–85. [CrossRef]
16. Arefnezhad, S.; Eichberger, A.; Frühwirth, M.; Kaufmann, C.; Moser, M.; Koglbauer, I.V. Driver monitoring of automated vehicles by classification of driver drowsiness using a deep convolutional neural network trained by scalograms of ECG signals. *Energies* **2022**, *15*, 480. [CrossRef]
17. Le, E.; Wang, Y.; Huang, Y.; Hickman, S.; Gilbert, F. Artificial intelligence in breast imaging. *Clin. Radiol.* **2019** , *74*, 357–366. [CrossRef]
18. Ankel, V.; Shribak, D.; Chen, W.Y.; Heifetz, A. Classification of computed thermal tomography images with deep learning convolutional neural network. *J. Appl. Phys.* **2022**, *131*, 244901. [CrossRef]
19. Jameil, A.K.; Al-Raweshidy, H. Efficient CNN Architecture on FPGA Using High Level Module for Healthcare Devices. *IEEE Access* **2022** , *10*, 60486–60495. [CrossRef]
20. Mohana, J.; Yakkala, B.; Vimalnath, S.; Benson Mansingh, P.; Yuvaraj, N.; Srihari, K.; Sasikala, G.; Mahalakshmi, V.; Yasir Abdullah, R.; Sundramurthy, V.P. Application of internet of things on the healthcare field using convolutional neural network processing. *J. Healthc. Eng.* **2022**, *2022* , 1892123. [CrossRef]
21. Venieris, S.I.; Bouganis, C.S. fpgaConvNet: Mapping Regular and Irregular Convolutional Neural Networks on FPGAs. *IEEE Trans. Neural Netw. Learn. Syst.* **2018**, *30*, 326–342. [CrossRef]
22. Arredondo-Velazquez, M.; Diaz-Carmona, J.; Torres-Huitzil, C.; Padilla-Medina, A.; Prado-Olivarez, J. A streaming architecture for Convolutional Neural Networks based on layer operations chaining. *J. Real-Time Image Process.* **2020**, *17*, 1715–1733. [CrossRef]
23. Arredondo-Velazquez, M.; Diaz-Carmona, J.; Barranco-Gutierrez, A.I.; Torres-Huitzil, C. Review of prominent strategies for mapping CNNs onto embedded systems. *IEEE Lat. Am. Trans.* **2020**, *18*, 971–982. [CrossRef]
24. NVIDIA. Deep Learning Frameworks. 2019. Available online: https://developer.nvidia.com/deep-learning-frameworks (accessed on 16 July 2019).
25. Erickson, B.J.; Korfiatis, P.; Akkus, Z.; Kline, T.; Philbrick, K. Toolkits and libraries for deep learning. *J. Digit. Imaging* **2017**, *30*, 400–405. [CrossRef]
26. Cong, J.; Xiao, B. Minimizing Computation in Convolutional Neural Networks. In *Proceedings of the Artificial Neural Networks and Machine Learning—ICANN 2014*; Wermter, S., Weber, C., Duch, W., Honkela, T., Koprinkova-Hristova, P., Magg, S., Palm, G., Villa, A.E.P., Eds.; Springer International Publishing: Cham, Switzerland, 2014; pp. 281–290.
27. Cong, J.; Fang, Z.; Huang, M.; Wei, P.; Wu, D.; Yu, C.H. Customizable Computing—From Single Chip to Datacenters. *Proc. IEEE* **2018**, *107*, 185–203. [CrossRef]
28. Hailesellasie, M.T.; Hasan, S.R. MulNet: A Flexible CNN Processor With Higher Resource Utilization Efficiency for Constrained Devices. *IEEE Access* **2019**, *7*, 47509–47524. [CrossRef]
29. Liu, Z.; Dou, Y.; Jiang, J.; Xu, J.; Li, S.; Zhou, Y.; Xu, Y. Throughput-Optimized FPGA Accelerator for Deep Convolutional Neural Networks. *ACM Trans. Reconfigurable Technol. Syst. (TRETS)* **2017**, *10*, 17. [CrossRef]
30. 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.

31. Zhang, C.; Li, P.; Sun, G.; Guan, Y.; Xiao, B.; Cong, J. Optimizing fpga-based accelerator design for deep convolutional neural networks. In Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 22–24 February 2015; ACM: New York, NY, USA, 2015; pp. 161–170.
32. Guo, K.; Sui, L.; Qiu, J.; Yu, J.; Wang, J.; Yao, S.; Han, S.; Wang, Y.; Yang, H. Angel-Eye: A complete design flow for mapping CNN onto embedded FPGA. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2017**, *37*, 35–47. [CrossRef]
33. Chen, Y.H.; Krishna, T.; Emer, J.S.; Sze, V. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE J. Solid-State Circuits* **2017**, *52*, 127–138. [CrossRef]
34. Abdelouahab, K.; Pelcat, M.; Sérot, J.; Bourrasset, C.; Berry, F. Tactics to Directly Map CNN graphs on Embedded FPGAs. *IEEE Embed. Syst. Lett.* **2017**, *9*, 113–116. [CrossRef]
35. Dundar, A.; Jin, J.; Martini, B.; Culurciello, E. Embedded streaming deep neural networks accelerator with applications. *IEEE Trans. Neural Netw. Learn. Syst.* **2017**, *28*, 1572–1583. [CrossRef]
36. Du, L.; Du, Y.; Li, Y.; Su, J.; Kuan, Y.C.; Liu, C.C.; Chang, M.C.F. A Reconfigurable Streaming Deep Convolutional Neural Network Accelerator for Internet of Things. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2018**, *65*, 198–208. [CrossRef]
37. Tu, F.; Yin, S.; Ouyang, P.; Tang, S.; Liu, L.; Wei, S. Deep Convolutional Neural Network Architecture With Reconfigurable Computation Patterns. *IEEE TRansactions Very Large Scale Integr. (VLSI) Syst.* **2017**, *25*, 2220–2223. [CrossRef]
38. Ma, Y.; Cao, Y.; Vrudhula, S.; Seo, J.s. Optimizing the convolution operation to accelerate deep neural networks on FPGA. *IEEE TRansactions Very Large Scale Integr. (VLSI) Syst.* **2018**, *26*, 1354–1367. [CrossRef]
39. Li, J.; Un, K.F.; Yu, W.H.; Mak, P.I.; Martins, R.P. An FPGA-based energy-efficient reconfigurable convolutional neural network accelerator for object recognition applications. *IEEE TRansactions Circuits Syst. II Express Briefs* **2021**, *68*, 3143–3147. [CrossRef]
40. Chen, Y.X.; Ruan, S.J. A throughput-optimized channel-oriented processing element array for convolutional neural networks. *IEEE Trans. Circuits Syst. II Express Briefs* **2020**, *68*, 752–756. [CrossRef]
41. Gilan, A.A.; Emad, M.; Alizadeh, B. FPGA-based implementation of a real-time object recognition system using convolutional neural network. *IEEE Trans. Circuits Syst. II Express Briefs* **2019**, *67*, 755–759.
42. Xu, R.; Ma, S.; Wang, Y.; Chen, X.; Guo, Y. Configurable multi-directional systolic array architecture for convolutional neural networks. *ACM Trans. Archit. Code Optim. (TACO)* **2021**, *18*, 1–24. [CrossRef]
43. Jafari, A.; Page, A.; Sagedy, C.; Smith, E.; Mohsenin, T. A low power seizure detection processor based on direct use of compressively-sensed data and employing a deterministic random matrix. In Proceedings of the Biomedical Circuits and Systems Conference (BioCAS), Atlanta, GA, USA, 22–24 October 2015; pp. 1–4.
44. Sze, V.; Chen, Y.H.; Yang, T.J.; Emer, J.S. Efficient processing of deep neural networks. *Synth. Lect. Comput. Archit.* **2020**, *15*, 1–341.
45. Xiyuan, P.; Jinxiang, Y.; Bowen, Y.; Liansheng, L.; Yu, P. A Review of FPGA-Based Custom Computing Architecture for Convolutional Neural Network Inference. *Chin. J. Electron.* **2021**, *30*, 1–17. [CrossRef]
46. Stankovic, L.; Mandic, D. Convolutional neural networks demystified: A matched filtering perspective based tutorial. *arXiv* **2021**. arXiv:2108.11663
47. Lacey, G.; Taylor, G.W.; Areibi, S. Deep Learning on FPGAs: Past, Present, and Future. *arXiv* **2016**, arXiv:1602.04283.
48. Chakradhar, S.; Sankaradas, M.; Jakkula, V.; Cadambi, S. A dynamically configurable coprocessor for convolutional neural networks. In Proceedings of the ACM SIGARCH Computer Architecture News, New York, NY, USA, 3 June 2010; ACM: New York, NY, USA, 2010; Volume 38, pp. 247–257.
49. Samajdar, A.; Zhu, Y.; Whatmough, P.; Mattina, M.; Krishna, T. Scale-sim: Systolic cnn accelerator simulator. *arXiv* **2018**. arXiv:1811.02883.
50. Fu, Y.; Wu, E.; Sirasao, A.; Attia, S.; Khan, K.; Wittig, R. Deep Learning with INT8 Optimization on Xilinx Devices. White Paper. 2016 . Available online: https://docs.xilinx.com/v/u/en-US/wp486-deep-learning-int8 (accessed on 13 November 2022).
51. Deng, L. The MNIST database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Process. Mag.* **2012**, *29*, 141–142. [CrossRef]
52. Li, F.-F.; Andreto, M.; Ranzato, M.A.; Perona, P. Caltech 101. 2022. . Available online: https://data.caltech.edu/records/mzrjq-6wc02 (accessed on 6 April 2022). [CrossRef]
53. Arredondo-Velázquez, M.; Diaz-Carmona, J.; Torres-Huitzil, C.; Barranco-Gutiérrez, A.I.; Padilla-Medina, A.; Prado-Olivarez, J. A streaming accelerator of convolutional neural networks for resource-limited applications. *IEICE Electron. Express* **2019**, *16*, 20190633. [CrossRef]
54. Shan, D.; Cong, G.; Lu, W. A CNN Accelerator on FPGA with a Flexible Structure. In Proceedings of the 2020 5th International Conference on Computational Intelligence and Applications (ICCIA), Beijing, China, 19–21 June 2020; pp. 211–216.
55. Bouguezzi, S.; Fredj, H.B.; Belabed, T.; Valderrama, C.; Faiedh, H.; Souani, C. An efficient FPGA-based convolutional neural network for classification: Ad-MobileNet. *Electronics* **2021**, *10*, 2272. [CrossRef]
56. Parmar, Y.; Sridharan, K. A resource-efficient multiplierless systolic array architecture for convolutions in deep networks. *IEEE Trans. Circuits Syst. II Express Briefs* **2019**, *67*, 370–374. [CrossRef]

57. Bassi, P.R.; Attux, R. A deep convolutional neural network for COVID-19 detection using chest X-rays. *Res. Biomed. Eng.* **2022**, *38*, 139–148. [CrossRef]
58. Wang, D.; Hong, D.; Wu, Q. Attention Deficit Hyperactivity Disorder Classification Based on Deep Learning. *IEEE/ACM Trans. Comput. Biol. Bioinform.* **2022** . [CrossRef]

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