electronics

MDPI

Article

Optimizing Artificial Neural Networks to Minimize Arithmetic Errors in Stochastic Computing Implementations

Christiam F. Frasser [1], Alejandro Morán [1,*], Vincent Canals [2], Joan Font [1], Eugeni Isern [1,3], Miquel Roca [1,3,4] and Josep L. Rosselló [1,3,4]

[1] Electronics Engineering Group, Industrial Engineering and Construction Department, University of the Balearic Islands, 07122 Palma, Spain; christian.franco@uib.es (C.F.F.); joan.font@uib.es (J.F.); eugeni.isern@uib.es (E.I.); miquel.roca@uib.es (M.R.); j.rossello@uib.es (J.L.R.)
[2] Energy Engineering Group, Industrial Engineering and Construction Department, University of the Balearic Islands, 07122 Palma, Spain; v.canals@uib.es
[3] Balearic Islands Health Research Institute (IdISBa), Son Espases University Hospital, 07120 Palma, Spain
[4] Artificial Intelligence Research Institute (IAIB), Universitat de les Illes Balears, 07122 Palma, Spain
* Correspondence: a.moran@uib.es

Abstract: Deploying modern neural networks on resource-constrained edge devices necessitates a series of optimizations to ready them for production. These optimizations typically involve pruning, quantization, and fixed-point conversion to compress the model size and enhance energy efficiency. While these optimizations are generally adequate for most edge devices, there exists potential for further improving the energy efficiency by leveraging special-purpose hardware and unconventional computing paradigms. In this study, we explore stochastic computing neural networks and their impact on quantization and overall performance concerning weight distributions. When arithmetic operations such as addition and multiplication are executed by stochastic computing hardware, the arithmetic error may significantly increase, leading to a diminished overall accuracy. To bridge the accuracy gap between a fixed-point model and its stochastic computing implementation, we propose a novel approximate arithmetic-aware training method. We validate the efficacy of our approach by implementing the LeNet-5 convolutional neural network on an FPGA. Our experimental results reveal a negligible accuracy degradation of merely 0.01% compared with the floating-point counterpart, while achieving a substantial 27× speedup and 33× enhancement in energy efficiency compared with other FPGA implementations. Additionally, the proposed method enhances the likelihood of selecting optimal LFSR seeds for stochastic computing systems.

Keywords: stochastic computing; edge computing; convolutional neural networks; LFSR seed; quantization (signal)

Citation: Frasser, C.F.; Morán, A.; Canals, V.; Font, J.; Isern, E.; Roca, M.; Rosselló, J.L. Optimizing Artificial Neural Networks to Minimize Arithmetic Errors in Stochastic Computing Implementations. *Electronics* **2024**, *13*, 2846. https://doi.org/10.3390/electronics13142846

Academic Editors: Francisco Javier González-Cañete, Martín González García, Antonio J. Lopez-Martin and Esteban Tlelo-Cuautle

Received: 17 June 2024
Revised: 5 July 2024
Accepted: 12 July 2024
Published: 19 July 2024

1. Introduction

The increasing use of intelligent edge devices for inference tasks utilizing deep learning (DL) models is largely due to their advantages in privacy, latency, bandwidth, energy efficiency, and cost. This contrasts with the conventional method of transmitting sensor data to the cloud for analysis [1]. In this scenario, real-time environmental monitoring minimizes the need to send data to the cloud, thus reducing the burden on data centers.

In response to the growing technological trend, numerous companies are developing specialized CPUs, DSPs, GPUs, and other processors to support general-purpose machine learning (ML) and neural network (NN) capabilities at the edge [2,3]. These processors are compatible with popular ML frameworks [4,5]. Meanwhile, some silicon companies are exploring unconventional processing methods, like approximate computations, to surpass the inherent limitations of silicon technology and achieve greater computational efficiency per unit of power [6,7]. This study examines the digital implementation of DL models using

stochastic computing (SC), a unary computing method based on random and non-weighted bitstreams [8].

The accuracy of NNs in SC models is significantly affected by the stochastic nature of bitstreams and the quantization of parameters and activations. To mitigate the accuracy decline in SC models, we analyze the error introduced by SC multiplications and integrate this analysis with the optimal selection of Linear Feedback Shift Registers (LFSRs) [9]. Additionally, we propose a novel training method that takes into account the approximate arithmetic operations in SC NNs.

This approach produces an SC inference model with minimal accuracy loss compared with its fixed-point counterpart. We demonstrate this method using the LeNet-5 model for classifying the MNIST dataset [10,11], providing results that include both accuracy and FPGA implementation metrics [12].

2. Stochastic Computing

Stochastic computing is a numerical representation methodology that deviates from conventional representation in the digital domain. Instead of using binary digits (bits) to represent numbers, SC utilizes a bitstream where the value is conveyed by the probability of encountering a logic `1`. This approach translates real numbers into probability distributions, where the number is embedded within the stochastic fluctuations of the bitstream. For instance, the value 0.5 (50% probability of observing a `1`) can be represented in SC using various bitstrings, such as `01`, `1100`, `1001100110`, or `01010101010101010101`. These examples demonstrate that the ordering of `1`'s is inconsequential for representation; rather, their proportion within the bitstream holds significance.

This unconventional numerical representation presents both advantages and limitations. One drawback is the computational overhead associated with operations, as well as the inherent loss of precision due to the stochastic nature of the bitstream. However, SC also offers compelling benefits, including reduced area and energy consumption in implementing computationally intensive tasks like multiplication and non-linear functions.

Two primary codifications exist within the SC framework: unipolarand bipolar. Unipolar encoding represents numbers within the interval [0, 1], while bipolar encoding spans the range to $[-1, 1]$. The conversion between these codifications can be accomplished using the following equation: $p^* = 2p - 1$, where p represents the unipolar representation of the number and p^* signifies the bipolar counterpart. As shown, employing the same bitstream length to represent a single number in both codifications, bipolar exhibits half the precision of unipolar coding. For instance, consider the scenario where a probability value isrepresented using a bitstream with a length of 4. The attainable values for both codifications are presented in Table 1. As observed, the unipolar representation exhibits twice the precision of the bipolar encoding. This disparity in precision can lead to elevated errors when employing bipolar coding, as elaborated in the following section.

Table 1. Comparison of unipolar and bipolar coding.

Pulses	Unipolar	Bipolar
0/4	0	−1
1/4	0.25	−0.5
2/4	0.50	0
3/4	0.75	0.5
4/4	1	1

3. Multiplication Error

Recent interest in the scientific and industrial realms has been directed towards SC circuits due to their efficient execution of computationally intensive operations, notably multiplication, while requiring minimal resources. In fact, SC multiplication requires a single logic gate, either an `AND` gate for unipolar encoding or an `XNOR` gate for bipolar encoding,

which facilitates a substantial reduction in both area and power consumption. However, this comes at the expense of increased execution time and diminished arithmetic accuracy.

In order to analytically derive an expression for the mean squared error (MSE), consider the i-th bit of two different bitstreams with activation probabilities of x and y that are i.i.d. Bernoulli random variables denoted by X_i and Y_i, respectively. These bitstreams represent bipolar variables x^* and y^*, so that the exact product is $z^* = x^*y^*$, where z^* is a bipolar variable represented by a bitstream with activation probability z. This bitstream with an activation probability z is obtained via a XNOR logic gate, i.e., the i-th bitstream bit is given by (1).

$$Z_i = X_iY_i + (1 - X_i)(1 - Y_i) \sim \text{Bernoulli}\left(\frac{x^*y^* + 1}{2}\right) \tag{1}$$

The activation probability of the resulting bitstream is approximated by counting the number of activations. In general, accumulating the Bernoulli experiment result for N iterations yields the binomial random variable Z, given by (2), so that the activation probability is approximated as Z/N.

$$Z = \sum_{i=1}^{N} Z_i \sim \text{Bin}\left(N, \frac{x^*y^* + 1}{2}\right) \tag{2}$$

Therefore, MSE associated to the bipolar variable $\hat{z}^* = (Z/N)^*$ w.r.t. the exact bipolar product $z^* = (\text{E}[Z]/N)^*$ is given by (3) and represented in Figure 1.

$$\begin{aligned} \text{MSE}[\hat{z}^*] = \text{MSE}\left[2\frac{Z}{N} - 1\right] &= \text{E}\left[\left(\left(2\frac{Z}{N} - 1\right) - \left(2\frac{\text{E}[Z]}{N} - 1\right)\right)^2\right] \\ &= \frac{4\text{Var}[Z]}{N^2} = \frac{4z(1-z)}{N} = \frac{1 - (x^*y^*)^2}{N} \end{aligned} \tag{3}$$

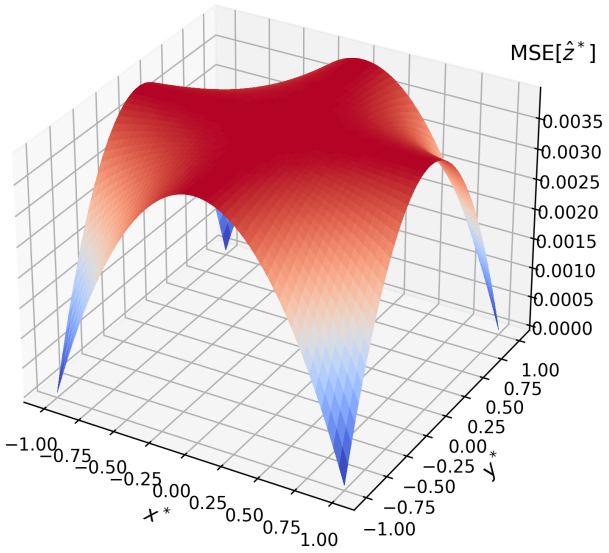

Figure 1. Mean squared error associated with the stochastic multiplication of two bipolar variables.

Moreover, during neural network (NN) inference, many multiplications are carried out. So, rather than considering the MSE associated to a single bipolar multiplication, we focus on the average MSE (AMSE), which depends on how inputs/activations and weights

are distributed. In general, the AMSE of the bipolar multiplication error is given by (4), where f and g are the probability density functions of the inputs and weights, respectively.

$$\mathrm{AMSE}[\hat{z}^*] = \frac{1}{N}\int_{-1}^{+1}\int_{-1}^{+1}\left(1-x^2y^2\right)f(x)g(y)dxdy \tag{4}$$

In addition, for simplicity, it is considered that the inputs are uniformly distributed, i.e., f is given by (5), which yields (6).

$$f(x) = \begin{cases} \frac{1}{2} & -1 \le x \le +1, \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

$$\mathrm{AMSE}_u[\hat{z}^*] = \frac{1}{2N}\int_{-1}^{+1}\int_{-1}^{+1}\left(1-x^2y^2\right)g(y)dxdy \tag{6}$$

Taking (6) as the starting point, three different scenarios are covered:

- **Normal–uniform**: Weights are normally distributed with zero mean $\mu = 0$ and standard deviation σ, i.e., g is given by (7), which results in the AMSE given by (8), where erf is the error function. Notice this is a good approximation for $\sigma \lessapprox 1/3$, and AMSE can be approximated to $\frac{1}{N}$ if σ is small.

$$g(y) = \frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{1}{2}\left(\frac{y-\mu}{\sigma}\right)^2} \tag{7}$$

$$\begin{aligned}\mathrm{AMSE}_{nu}[\hat{z}^*] &\approx \frac{1}{4N\sigma\sqrt{2\pi}}\int_{-1}^{+1}\int_{-1}^{+1}\left(1-x^2y^2\right)e^{-\frac{1}{2}\left(\frac{y-\mu}{\sigma}\right)^2}dxdy \\ &= \frac{1}{3N}\left(\left(3-\sigma^2\right)\mathrm{erf}\left(\frac{1}{\sigma\sqrt{2}}\right) - \sigma\sqrt{\frac{2}{\pi}}e^{-\frac{1}{2\sigma^2}}\right)\end{aligned} \tag{8}$$

- **Uniform–uniform**: Weights are uniformly distributed between -1 and $+1$, i.e., $f = g$, which results in the AMSE given by (9).

$$\mathrm{AMSE}_{uu}[\hat{z}^*] = \frac{1}{4N}\int_{-1}^{+1}\int_{-1}^{+1}\left(1-x^2y^2\right)dxdy = \frac{8}{9N} \tag{9}$$

- **Custom–uniform** : Weights follow a custom probability density function given by (10), which results in the AMSE given by (11). Notice this is a good approximation for $\sigma \lessapprox 1/3$, and the AMSE can be approximated to $\frac{8}{9N} - \alpha\frac{2}{9N}$ if σ is small.

$$g(y) \approx \begin{cases} \frac{1-\alpha}{2} + \frac{\alpha}{\sigma\sqrt{2\pi}}\left(e^{-\frac{1}{2}\left(\frac{y+1}{\sigma}\right)^2} + e^{-\frac{1}{2}\left(\frac{y-1}{\sigma}\right)^2}\right) & -1 \le y \le +1, \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

$$\mathrm{AMSE}_{cu}[\hat{z}^*] \approx (1-\alpha)\frac{8}{9N} + \alpha\frac{1}{3N}\left(\left(2+\sigma^2\right)\mathrm{erf}\left(\frac{\sqrt{2}}{\sigma}\right) + \frac{4}{\sqrt{2\pi}}\sigma\right) \tag{11}$$

These AMSE expressions and the corresponding weight distributions are represented in Figure 2 using $\alpha = 0.5$ and $\sigma = 1/12$ in (11). Using these numbers and $N = 255$ (8-bit signal) results in an AMSE reduction of 19.7% for the custom–uniform case w.r.t. the normal–uniform case with $\sigma = 1/12$. Moreover, under the small σ approximation, the AMSE reduction is up to $22.\overline{2}\%$. This multiplication error is further reduced using random maximum length sequences (m-sequences) to minimize the representation error of the bitstream w.r.t. the complement number representation of the two. So, using m-sequences leads to exact multiplications by ± 1. In addition, in the experiments, random sequences

are generated by LFSRs, which are m-sequences, and also allow for fine tuning the final neural network accuracy by trying different LFSR seed pairs, as described in Section 4.

As illustrated in the histogram of Figure 3 for the LeNet-5 case, weights typically follow a normal or similar distribution. Therefore, in order to reduce the arithmetic error associated with stochastic multiplications, it is proposed to modify the probability density function of the weights during the training stage of a neural network. In particular, it is proposed to modify it so that it is similar to (11), represented in Figure 2 (green dotted line), with many more extreme values.

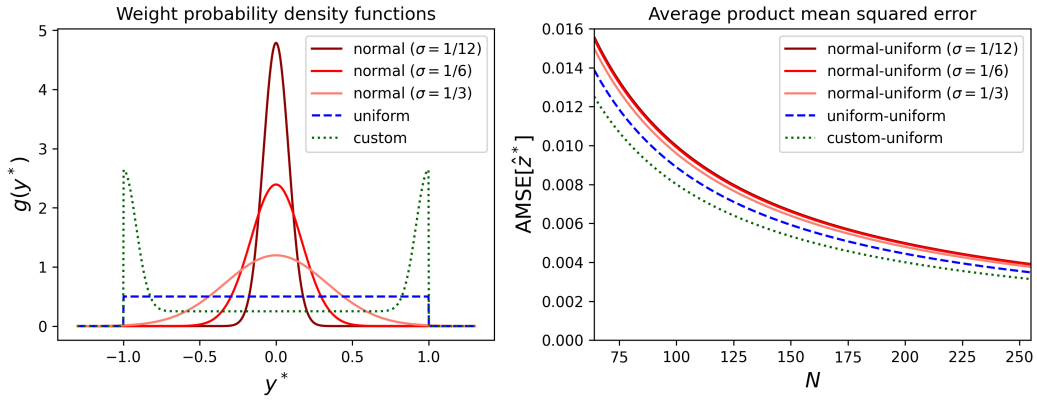

Figure 2. Average mean squared error associated with the bipolar multiplication considering the several input and weight probability density functions.

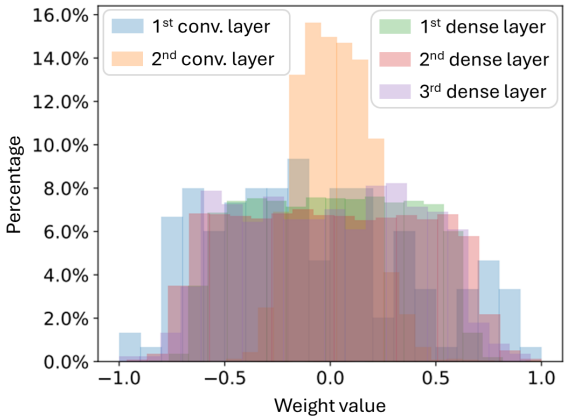

Figure 3. Weight distribution for the CNN implemented in [12].

4. LFSR Seed Selection Error

The most cost-effective approach for generating stochastic bitstreams in a real SC implementation involves utilizing a Linear-Feedback-Shift-Register (LFSR) [9,13–15]. Although there are alternatives, such as random number generators (RNGs), LFSRs have some crucial advantages: area saving (which is relevant in SC hardware, given the large area overhead due to RNGs) and the fact of generating a uniformly distributed signal $R(t)$ in the interval of all possible values of X, the digital number which, after being compared with $R(t)$, will be converted into a stochastic sequence $x(t)$ with an associated probability x. An LFSR is a circuit based on a shift register and a linear function of its previous state connected to its input. This linear function is implemented by connecting exclusive OR gates to different taps in the state registers. The pseudorandom sequence is repeated periodically, starting from the seed value. As outlined in the existing literature, it is crucial to identify an appropriate LFSR seed pair capable of producing stochastic bitstreams with minimal correlation. This is necessary to mitigate the correlation phenomenon in the multiplication operation [9,16]. Despite the significance of this requirement, devising an effective

methodology for identifying optimal LFSR seeds, especially in the case of complex systems like neural networks (NNs), remains a formidable challenge. The current state of the art has yet to offer a reliable solution, and the predominant approach typically involves trial and error. This can be a time-consuming process, given the various scenarios encountered. For instance, consider Figure 4, illustrating diverse accuracies achieved when employing all possible LFSR seeds for the 8-bit convolutional neural network (CNN) presented by C. Frasser et al. [12]. The results reveal that more than 80% of the seeds yielded inaccurate results (Acc < 0.95) due to accumulation errors in the zero region of multiplication (see Figure 1) and the correlation effects of the seeds. This phenomenon poses a significant challenge that necessitates community attention. As systems grow in complexity, the difficulty identifying LFSR seeds results in minimal AD increases.

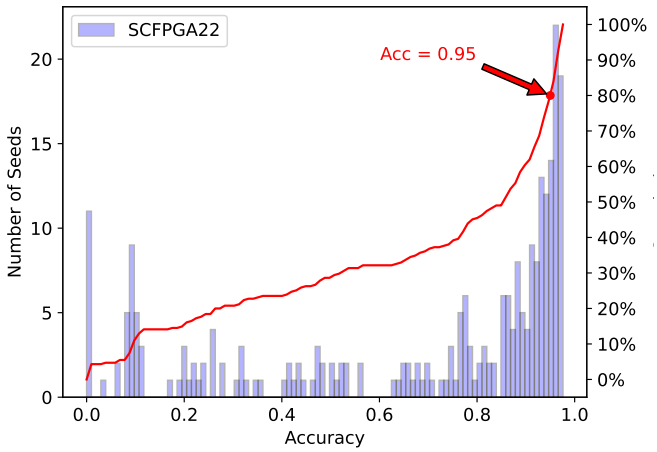

Figure 4. Number of seeds generating different accuracy levels in the SC-CNN study presented in [12]; 80% of seeds produce an accuracy smaller than 0.95.

5. Stochastic Computing Aware Training

To enhance the precision of SC hardware, several strategies are adopted. These considerations are integral to refining the accuracy of SC-NN hardware:

- **Weight clamping**: Analogous to fixed-point representation, the technique of clamping weights within a specific range $[-n_{lim}, +n_{lim}]$ is employed. This method prevents the emergence of extreme weight values in the distribution, which can otherwise escalate linear quantization errors. By restricting the variance of the weight distribution, smaller quantization errors are achieved.
- **Weight distribution uniformization**: Adjusting the weight distribution to be more uniform can mitigate the relative errors associated with smaller quantized values. Additionally, dispersing weights that are proximate to zero can diminish the incidence of large relative errors during bipolar SC multiplications.
- **Weight distribution binarization**: To enhance the accuracy of multiplication outcomes in stochastic circuits, increasing the proportion of weights represented by bipolar quantities set to -1 and $+1$ is beneficial. In the bipolar encoding framework, these values are correlated with activation rates of 0 and 1, respectively, thereby not contributing to the arithmetic error.

Grounded in the aforementioned strategies, we introduce a methodology as delineated in Algorithm 1, which incorporates an additional quantization phase. Typically, neural network (NN) training employs iterative gradient descent methodologies, as outlined in [17], consisting of three principal phases. The initial phase involves forward propagation to ascertain intermediate and predicted outputs of the network. This is followed by the backward propagation phase, wherein weight discrepancies are calculated based on a predetermined cost function, with errors being propagated in reverse through the network.

Subsequently, the weights are updated employing either basic gradient descent or more advanced algorithms such as Adam [18]. The proposed framework introduces a novel, independent fourth step that manipulates weights to enforce a pseudo-uniform distribution of quantized values upon the culmination of the training regime, as depicted in Figure 5.

Algorithm 1 Stochastic computing aware weight distribution modification and quantization steps

Require: Weights $\boldsymbol{W}$, width factor n_σ, rounding rate ε, number of integer values n_{values}

 # calculate the limit for weight values based on the width factor and standard deviation
$n_{lim} \leftarrow n_\sigma \, \mathrm{std}(\boldsymbol{W})$
 # restrict the range of weight values
$\boldsymbol{W} \leftarrow \mathrm{clamp}(\boldsymbol{W}, -n_{lim}, +n_{lim})$
 # quantized weights
$\boldsymbol{W}_q \leftarrow \left\lfloor \frac{\boldsymbol{W} n_{values}}{n_{lim}} \right\rceil \frac{n_{lim}}{n_{values}}$
 # step towards quantization
if this is the last quantization step **then**
    # training is done, keep the quantized weights
    $\boldsymbol{W} \leftarrow \boldsymbol{W}_q$
else
    # put weights closer to its quantized version based on the rounding rate
    $\boldsymbol{W} \leftarrow \boldsymbol{W} + \varepsilon(\boldsymbol{W}_q - \boldsymbol{W})$
end if
return $\boldsymbol{W}$

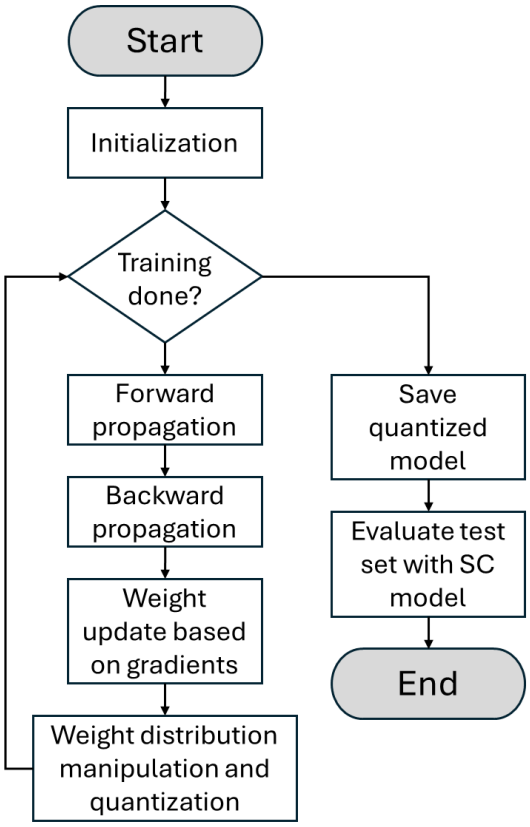

Figure 5. Stochastic computing aware training flow chart.

After each optimizer step, we calculate new limits for the weight values. To do this, n_σ is defined as a new hyperparameter corresponding to the number of standard deviations allowed in the previous weight distribution (width factor). Therefore, the actual absolute

limit value is given by (12) and the weight values are restricted using a simple clamping operation given by (13) for every weight value in a given weight tensor $\boldsymbol{W}$, as denoted by line 4 in Algorithm 1.

$$n_{lim} = n_\sigma \text{std}(\boldsymbol{W}) \tag{12}$$

$$\text{clamp}(\omega, -n_{lim}, n_{lim}) \equiv \begin{cases} n_{lim} & \text{if } \omega > n_{lim} \\ -n_{lim} & \text{if } \omega < -n_{lim} \\ \omega & \text{otherwise} \end{cases} \tag{13}$$

For illustration purposes, Figure 6 demonstrates the evolution of the weight distribution from a Gaussian distribution through three sequential iterations of the clamping process across various n_σ settings. Post clamping, the weight values achieve a more uniform distribution, with the exception of the extremities at -1 and $+1$ (it is noteworthy that these extreme values are devoid of associated SC multiplication errors, rendering the resultant weight distribution more advantageous than an ideally uniform distribution). This uniformity enhances the accuracy of uniform quantization, particularly in comparison with the normal distributions. Accordingly, we have implemented a straightforward uniform symmetric quantization approach, given by (14).

$$\boldsymbol{W}_q = \left\lfloor \frac{\boldsymbol{W} n_{values}}{n_{lim}} \right\rceil \frac{n_{lim}}{n_{values}} \tag{14}$$

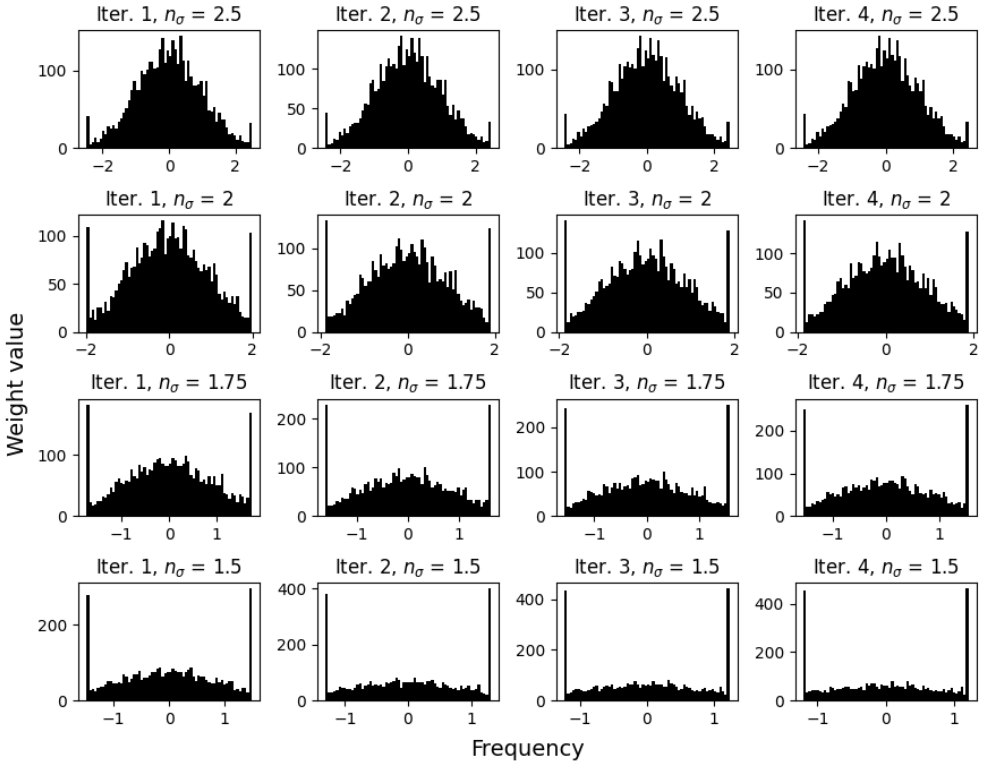

Figure 6. Weight distribution for several values of n_σ after applying weight clamping three consecutive times.

In the above expression, the operation $\lfloor \cdot \rceil$ signifies rounding to the nearest integer. For radix-2 data representation, it is pragmatic to select n_{values} as a power of two minus one; for instance, in the case of 8-bit weights, we set $n_{values} = 255$. To facilitate a controlled transition from continuous weight values to their quantized counterparts, we introduce the rounding rate ε as a hyperparameter. This hyperparameter governs the incremental adjustment of the weights $\boldsymbol{W}$ towards their quantized form $\boldsymbol{W}_q$ with each iteration, as

detailed in (15). The hyperparameter is adjusted as a function that increases monotonically with the number of training epochs, starting at $\varepsilon = 0$ and ending at $\varepsilon = 1$ towards the conclusion of the training period.

$$W_{next} = W + \varepsilon(W_q - W) \tag{15}$$

Although this adaptive training strategy has proven to increase the accuracy for SC-NN implementations, it requieres the careful management of two additional hyperparameters: the width factor n_σ and the rounding rate ε.

6. Experimental Results

Experimental outcomes detailed in this section stem from the dynamic adjustments made to the additional hyperparameters n_σ and ε, as visualized in Figure 7. Employing a batch size of 64, we adhered to default Adam optimizer parameters ($\lambda = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$), and utilized a cross-entropy cost function. The width factor, initially set to $n_\sigma = 6$ with no constraints, underwent exponential reduction until approximately $n_\sigma \approx 2$ across 60 epochs. Meanwhile, the rounding rate followed a logistic function shape, commencing near 0 with negligible quantization and progressively ascending to 1 ($W = W_q$) upon training completion.

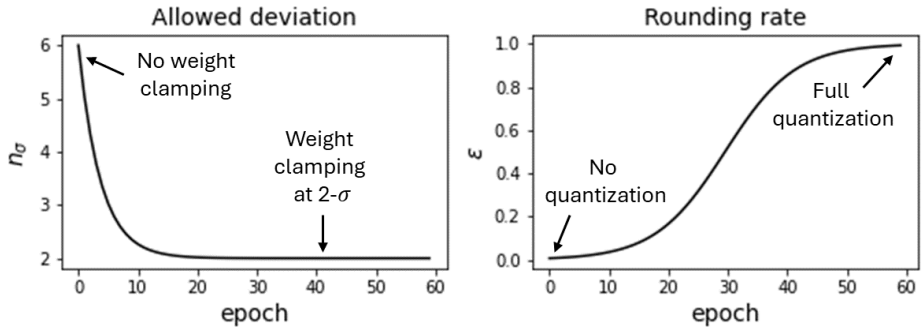

Figure 7. Width factor and rounding rate hyperparameter setup for our experiment.

To validate the efficacy of our training methodology, we executed a hardware (FPGA) implementation [12] of the renowned CNN architecture, LeNet-5 [10], utilizing stochastic computing. The network underwent training utilizing the MNIST handwriting dataset, comprising 60 k training images and 10 k testing images [11]. Our implementation adhered to the CNN architecture originally proposed by J. Lecun et al. [10], which includes two convolutional layers, two max-pooling layers, and three fully connected layers. A comprehensive outline of the hardware design employed in this endeavor is delineated in Figure 8, with further details provided in reference [12]. The implementation was executed on a GIDEL PROC10A board [19], equipped with an Intel 10AX115H3F34I2SG FPGA operating at 150 MHz.

The software model achieved an accuracy of 98.98% after training using the SC-aware methodology. Meanwhile, the hardware accuracy with 8-bit precision stood at 98.97%, and at 97.64% for 6-bit precision. The associated weight distribution following the proposed training method is depicted in Figure 9.

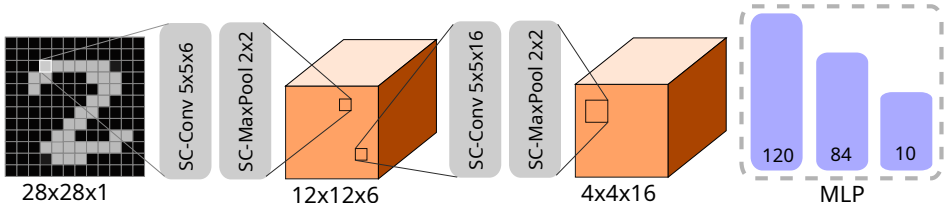

Figure 8. Fully parallel stochastic CNN architecture [12].

Table 2 illustrates the advancements achieved through the adoption of the training methodology advocated in this research compared with utilizing the same hardware architecture without such training. It is apparent that the implementation of this approach facilitated a transition from 8 bits to 6 bits, resulting in a 4× increase in throughput and performance, accompanied by a 17.4× improvement in energy efficiency and a 0.06% enhancement in accuracy. It is noteworthy that despite only a 1.08× reduction in area, there is a 4.3× decrease in power consumption. This phenomenon is attributed to the proposed training method, which induces a significant reduction in signal switching, given that the weights tend to approximate to 1 or −1 in SC.

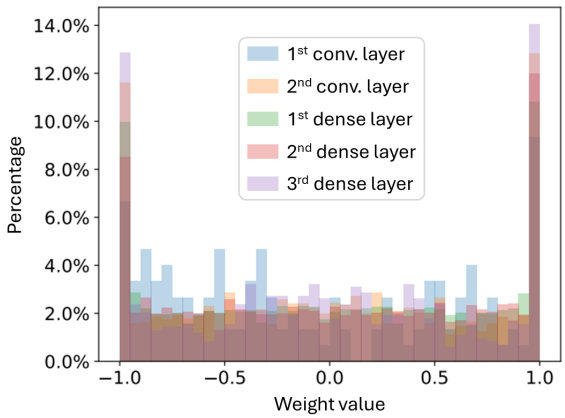

Figure 9. Weight distribution after the arithmetic aware training.

Table 2. Enhancements in FPGA implementation compared to prior work (SCFPGA22 [12]).

Metric	SCFPGA22 [12]	This Work
Year	2022	2024
Architecture parallelism	Parallel	Parallel
Computing paradigm	SC	SC
Activation/Weight bits	8/8	6/6
FPGA platform	Arria10 GX1150	Arria10 GX1150
Frequency (MHz)	150	150
Software Acc (%)	98.60	98.98
Hardware Acc (%)	97.58	**97.64**
Acc Degradation (%)	**1.02**	1.34
Throughput (Images/s)	294,118	1,190,476
Performance (Images/s/MHz)	1961	**7937**
Power (W)	21.0	4.9
Energy efficiency (Images/J)	14,006	**243,171**
Logic used K (LUT or ALM)	343	318
DSP (blocks)	0	0
Memory (Mbits)	0.00	0.00

Table 3 presents a performance comparison with other FPGA implementations, including SC, BNN, and TC approaches. The table emphasizes the two most favorable outcomes for the pertinent metrics. In comparison with the most accurate study (BNFPGA18 [20]), our approach yields a four-fold increase in throughput, a four-fold improvement in performance, and a 21.6× enhancement in energy efficiency, albeit with a marginal 0.88% reduction in accuracy degradation (AD). Relative to a more recent investigation (SCFPGA24 [21]), our results exhibit a slightly inferior performance, characterized by a mere 0.61% reduction in accuracy degradation, a 1.44× slower processing speed, and a 1.97×

decrease in performance. However, our approach showcases a mere 1.05× reduction in energy efficiency.

When exclusively considering SC implementations of the LeNet-5 architecture documented in the literature, we juxtapose the accuracy degradation among them when using the 8-bit approach (see Table 4). The row designated as "Test Platform" delineates the methodology employed for accuracy assessment, with "Sim" indicating simulation and "FPGA" indicating FPGA implementation. It is evident that accuracy degradation poses a significant challenge for SC-CNN designers, with the most favorable scenario achieving an AD of 0.14%. Conversely, in our study, we attained an AD of merely 0.01%, representing a 14-fold enhancement over the best-performing scenario (SCCNN24 [21]), while concurrently achieving the highest accuracy (98.97%). These findings underscore the efficacy of the proposed training approach, which systematically addresses SC-associated limitations while preserving hardware integrity [22].

Table 3. FPGA LeNet-5 implementations comparison. Notice the results for SCFPGA24 are for the 64 bitstream length case.

Metric	**TCFPGA17 [23]**	BNFPGA18 [20]	SCFPGA20 [24]	SCFPGA24 [21]	**This Work**
Year	2017	2018	2020	2024	2024
Architecture parallelism	Sequential	Sequential	Semi-Parallel	Parallel	Parallel
Computing paradigm	TC	BNN	SC	SC	SC
Activation/Weight bits	16/8	8/1	9/9	6/6	6/6
FPGA family	Virtex7	Stratix V	Zynq	Kintex7	Arria 10
FPGA name	VX690T	5SFSD8	XC7Z020	XC7K325T	GX1150
Frequency (MHz)	100	150	60	110	150
Software Acc (%)	99.17	98.70	98.67	98.36	98.98
Hardware Acc (%)	98.16	98.24	98.13	97.63	97.64
Acc Degradation (%)	1.01	**0.46**	**0.54**	0.73	1.34
Throughput (Images/s)	10,617	294,118	170	**1,718,800**	**1,190,476**
Performance (Images/s/MHz)	106	1961	3	**15,626**	**7937**
Power (W)	25.2	26.2	3.7	6.8	4.9
Energy efficiency (Images/J)	421	11,226	46	**254,373**	**243,171**
Logic used K (LUT or ALM)	233	0.182	28	153	318
DSP (blocks)	2907	20	0	0	0
Memory (Mbits)	17.2	44.2	1.7	0.0	0.0

Table 4. Comparison with other SC LeNet-5 implementations.

Metric	**Year**	**Software Acc (%)**	**Hardware Acc (%)**	**Acc Degradation (%)**	**Test Platform**
SCCNN19 [25]	2019	98.47	97.94	−0.53	Sim
SCFPGA20 [24]	2020	98.67	98.13	−0.54	FPGA
SCCNN21 [26]	2021	98.75	97.50	−1.25	Sim
SCFPGA22 [12]	2022	98.60	97.58	−1.02	FPGA
SCFPGA24 [21] (8-bits)	2024	98.36	98.22	−0.14	FPGA
This work (8-bits)	2024	98.98	98.97	**−0.01**	FPGA

Figure 10 depicts a comparison of the number of LFSR seeds producing various accuracies across different bit precisions for two academic endeavors: the SCFPGA22 study [12] and the present work under discussion. Remarkably, the training methodology introduced in this study facilitates the identification of optimal seeds for LFSR implementation. As the bit precision decreases, the extent of improvement becomes more pronounced. For the 8-bit precision scenario, it is noted that 60% of the seeds in the present study yield accuracies surpassing 98.5% (with an AD of merely 0.5%), while no seeds have been identified to achieve a comparable absolute difference value in the SCFPGA22 study [12]. Furthermore,

at 6-bit precision, the maximum accuracy achieved in the [12] study is 17.7%, whereas this study achieves accuracies exceeding 95% with five different seeds.

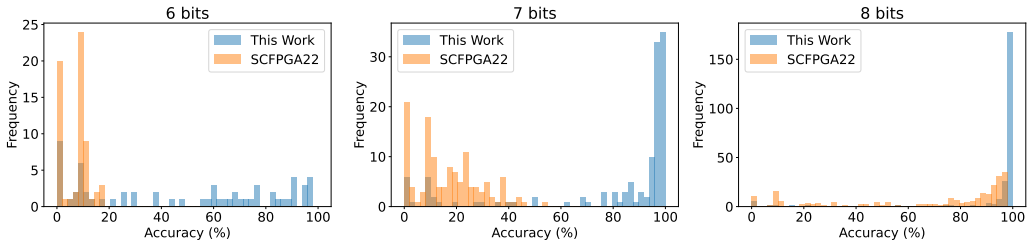

Figure 10. Number of LFSR seeds producing various accuracies for different bit precisions.

7. Conclusions

Stochastic computing (SC) emerges as a viable approach for overcoming the difficulties of deploying neural networks (NNs) on hardware platforms. This research has concentrated on resolving the accuracy degradation (AD) challenges inherent in SC when applied to actual hardware systems by employing an arithmetic-aware training strategy. The outcomes have shown considerable advancements, achieving a reduction in AD by a factor of 100 without necessitating any modifications to the hardware setup. In comparison with similar studies, our method attains an exceptionally low AD rate of only 0.01%, which represents a 54-fold improvement over the best comparison case, while also delivering an enhanced performance in terms of speed (27 times faster) and energy efficiency (33 times more efficient). Furthermore, our training technique has aided in selecting LFSR seeds suitable for SC hardware applications, where 60% of the seeds show an AD below 0.5% and 83% maintain an AD under 5%. These results indicate that our method is more effective in securing precise outcomes with minimal variations from the desired accuracy, all while preserving the integrity of the hardware configuration. Finally, it is worth highlighting that the proposed method has been tested with the hardware proposed in [27], with a baseline MNIST test accuracy degradation of 1.02%, which is improved to 0.01%. Nevertheless, the proposed technique is not limited to this particular accelerator. In fact, implementations resulting in more precise SC multipliers like SCFPGA24 [21] are expected to improve in terms of accuracy degradation for 8-bit and lower weight and activation precision, just by modifying the way the NN is trained.

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

Funding: This research was funded by the Spanish Ministry of Science and Innovation (MCIN/AEI/10.13039/501100011033) and the European Regional European Development Founds (ERDF) under Grants PID2020-120075RB-I00 and PDC2021-121847-I00.

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AD	Accuracy degradation
AMSE	Average mean squared error
BNN	Binarized neural network
CNN	Convolutional neural network
FPGA	Field Programmable Gate Array
LFSR	Linear feedback shift register
ML	Machine learning
MSE	Mean squared error
NN	Neural network
SC	Stochastic computing

References

1. Chen, J.; Ran, X. Deep learning with edge computing: A review. *Proc. IEEE* **2019**, *107*, 1655–1674. [CrossRef]
2. Suda, N.; Loh, D. *Machine Learning on Arm Cortex-M Microcontrollers*; Arm Ltd.: Cambridge, UK, 2019.
3. Google. Google Edge TPU. Available online: https://cloud.google.com/edge-tpu (accessed on 16 July 2024) .
4. Abadi, M.; Barham, P.; Chen, J.; Chen, Z.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Irving, G.; Isard, M.; et al. Tensorflow: A system for large-scale machine learning. In Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementatio, Savannah, GA, USA, 2–4 November 2016; Volume 16, pp. 265–283.
5. Paszke, A.; Gross, S.; Chintala, S.; Chanan, G.; Yang, E.; DeVito, Z.; Lin, Z.; Desmaison, A.; Antiga, L.; Lerer, A. Automatic differentiation in PyTorch. In Proceedings of the 31st Conference on Neural Information Processing Systems, Long Beach, CA, USA, 4–9 December 2017. Available online: https://openreview.net/forum?id=BJJsrmfCZ (accessed on 16 July 2024).
6. Akopyan, F.; Sawada, J.; Cassidy, A.; Alvarez-Icaza, R.; Arthur, J.; Merolla, P.; Imam, N.; Nakamura, Y.; Datta, P.; Nam, G.J.; et al. Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2015**, *34*, 1537–1557. [CrossRef]
7. Davies, M.; Srinivasa, N.; Lin, T.H.; Chinya, G.; Cao, Y.; Choday, S.H.; Dimou, G.; Joshi, P.; Imam, N.; Jain, S.; et al. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro* **2018**, *38*, 82–99. [CrossRef]
8. Gaines, B.R. Stochastic computing systems. *Adv. Inf. Syst. Sci.* **1969**, *2*, 37–172.
9. Frasser, C.F.; Roca, M.; Rosselló, J.L. Optimal stochastic computing randomization. *Electronics* **2021**, *10*, 2985. [CrossRef]
10. LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2324. [CrossRef]
11. Lecun, Y. The MNIST Database of Handwritten Digits. Available online: http://yann.lecun.com/exdb/mnist/ (accessed on 16 July 2024).
12. Frasser, C.F.; Linares-Serrano, P.; de los Ríos, I.D.; Morán, A.; Skibinsky-Gitlin, E.S.; Font-Rosselló, J.; Canals, V.; Roca, M.; Serrano-Gotarredona, T.; Rosselló, J.L. Fully Parallel Stochastic Computing Hardware Implementation of Convolutional Neural Networks for Edge Computing Applications. *IEEE Trans. Neural Netw. Learn. Syst.* **2022**, *34*, 10408–10418. [CrossRef] [PubMed]
13. Li, Z.; Chen, Z.; Zhang, Y.; Huang, Z.; Qian, W. Simultaneous area and latency optimization for stochastic circuits by D flip-flop insertion. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2018**, *38*, 1251–1264. [CrossRef]
14. Neugebauer, F.; Polian, I.; Hayes, J.P. Building a better random number generator for stochastic computing. In Proceedings of the 2017 Euromicro Conference on Digital System Design (DSD), Vienna, Austria, 30 August–1 September 2017; pp. 1–8.
15. Morán, A.; Parrilla, L.; Roca, M.; Font-Rossello, J.; Isern, E.; Canals, V. Digital Implementation of Radial Basis Function Neural Networks Based on Stochastic Computing. *IEEE J. Emerg. Sel. Top. Circuits Syst.* **2023**, *13*, 257–269. [CrossRef]
16. Anderson, J.H.; Hara-Azumi, Y.; Yamashita, S. Effect of LFSR seeding, scrambling and feedback polynomial on stochastic computing accuracy. In Proceedings of the 2016 Design, Automation Test in Europe Conference Exhibition (DATE), Dresden, Germany, 14–18 March 2016; pp. 1550–1555.
17. LeCun, Y.; Bengio, Y.; Hinton, G. Deep learning. *Nature* **2015**, *521*, 436–444. [CrossRef] [PubMed]
18. Kingma, D.P.; Ba, J. Adam: A method for stochastic optimization. *arXiv* **2014**, arXiv:1412.6980.
19. Gidel Company. PROC10A Board Image. Available online: https://gidel.com/product/proc10a/ (accessed on 10 June 2020).
20. Liang, S.; Yin, S.; Liu, L.; Luk, W.; Wei, S. FP-BNN: Binarized neural network on FPGA. *Neurocomputing* **2018**, *275*, 1072–1086. [CrossRef]
21. Lee, Y.Y.; Halim, Z.A.; Ab Wahab, M.N.; Almohamad, T.A. Stochastic Computing Convolutional Neural Network Architecture Reinvented for Highly Efficient Artificial Intelligence Workload on Field-Programmable Gate Array. *Research* **2024**, *7*, 0307. [CrossRef] [PubMed]
22. Costoya, A.M. Compact Machine Learning Systems with Reconfigurable Computing. Ph.D. Thesis, Universitat de les Illes Balears, Palma, Spain, 2022.

23. Li, Z.; Wang, L.; Guo, S.; Deng, Y.; Dou, Q.; Zhou, H.; Lu, W. Laius: An 8-Bit Fixed-Point CNN Hardware Inference Engine. In Proceedings of the 2017 IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications (ISPA/IUCC), Guangzhou, China, 12–15 December 2017; pp. 143–150. [CrossRef]
24. Muthappa, P.K.; Neugebauer, F.; Polian, I.; Hayes, J.P. Hardware-Based Fast Real-Time Image Classification with Stochastic Computing. In Proceedings of the 2020 IEEE 38th International Conference on Computer Design (ICCD), Hartford, CT, USA, 18–21 October 2020; pp. 340–347.
25. Zhang, Y.; Zhang, X.; Song, J.; Wang, Y.; Huang, R.; Wang, R. Parallel Convolutional Neural Network (CNN) Accelerators Based on Stochastic Computing. In Proceedings of the 2019 IEEE International Workshop on Signal Processing Systems (SiPS), Nanjing, China, 20–23 October 2019; pp. 19–24. [CrossRef]
26. Sadi, M.H.; Mahani, A. Accelerating Deep Convolutional Neural Network base on stochastic computing. *Integration* **2021**, *76*, 113–121. [CrossRef]
27. Frasser, F.; Camilo, C. Hardware Implementation of Machine Learning and Deep-Learning Systems oriented to Image Processing Ph.D. Thesis, Universitat de les Illes Balears, Palma, Spain, 2022.