Fast Exploration of the Impact of Precision Reduction on Spiking Neural Networks

Fast Exploration of the Impact of Precision Reduction on Spiking Neural Networks

Sepide Saeedi[1], Alessio Carpegna[1], Alessandro Savino[1], and Stefano Di Carlo[1]
[1]*Control and Computer Eng. Dep., Politecnico di Torino* Torino, Italy
{sepide.saeedi, alessio.carpegna, alessandro.savino, stefano.dicarlo}@polito.it

***Abstract*—Approximate Computing (AxC) techniques trade off computational accuracy for gains in performance, energy efficiency, and area reduction. This trade-off is particularly advantageous when applications, like Spiking Neural Networks (SNNs), are naturally tolerant to some degree of accuracy loss. SNNs are especially practical when the target hardware is pushed to the edge of its computing capabilities, necessitating area minimization strategies. In this work, we utilize an Interval Arithmetic (IA)-based model that propagates approximation errors through the application's computation flow to assess these approximations' impact on the outputs. We enhance this IA-based model by introducing observation points within the computation flow to quickly detect when the level of approximation surpasses a set threshold. Experimental results demonstrate the model's effectiveness in significantly reducing exploration time, enabling more precise and fine-grained approximations that further minimize network parameters.**

***Index Terms*—approximate computing, spiking neural networks, interval arithmetic, design space exploration**

I. Introduction

Approximate Computing (AxC) techniques enable a controlled reduction in computational accuracy to improve various design metrics, including power consumption, memory utilization, and execution time [1], [2]. These benefits are particularly appealing in applications such as Artificial Neural Networks (ANNs), which are inherently tolerant to some degree of accuracy loss in computation [3], [4]. When AxC techniques are introduced into applications, the key challenge is determining which parts of the computation can be approximated and how to implement these approximations. Various methods can be used together, requiring tools and methodologies to assess how different combinations of AxC techniques affect the overall computation accuracy [5]. This decision-making process is complex because each application has its inherent tolerance for accuracy degradation. The two main strategies to address this challenge are executing the application multiple times with different configurations [6], [7] or using abstract models to predict the final application error [8], [9]. The first approach can provide precise analysis, but the time required increases with the number of available options, making exhaustive design space exploration impractical. The second approach balances the accuracy of the analysis with exploration time, enabling thorough exploration of the design space.

979-8-3503-6312-8/24/$31.00 ©2024 IEEE

Spiking Neural Networks (SNNs) are a type of ANNs that mimics human brain functionality. SNNs support information exchange based on binary spikes [10] and unsupervised learning with unlabeled data. Each neuron in an SNN performs arithmetic operations on weights and thresholds to process input spikes and generate output spikes. Due to the large size of SNN models, which involve extensive arrays of registers to store these weights and thresholds, SNNs can significantly benefit from using AxC techniques [11], especially when targeting deployment at the edge. For example, when deploying SNNs on FPGA devices, AxC techniques can help reduce the storage required for weights and thresholds, as well as the complexity of the arithmetic components [12]–[14].

Some previous works use Interval Arithmetic (IA) to modify Neural Networks (NNs) for achieving different goals. In [15], the authors propose using interval arithmetic for SNN computations and apply two specific AxC techniques. The results of the precision range analysis for a few neurons seem promising. While, in other works, for example, in [16], authors introduce an Interval Genetic Algorithm (IGA) for the neuro-evolution of nterval Neural Networkss (INNs) with interval weights and biases. The aim is to facilitate the unsupervised learning of INNs. Though the results demonstrate their capability to approximate test functions, this work does not consider the effect of approximation. While in [17], a computation flow representation is proposed to model the approximation error range for the entire SNN, using an abstract numerical model based on IA [18], which enables tracking the introduced error and offers further tuning optimization opportunities.

Using IA for error propagation is not new. In [19], authors proposed using it to analyze the error propagation through the program data flow. They perform the data flow analysis in two steps. First, during the range analysis, the tool computes the output ranges. Second, during the error analysis, the tool propagates errors and computes the worst-case round-off errors using the previously calculated ranges. The goal is to increase code readability, extensibility, and not necessarily performance. The analysis is performed once for the exact application and once for the round-off error estimation. In [17], the IA-based model provides an abstract representation of the SNN computation flow while enabling tracking of how approximation-induced errors propagate from the inputs to the outputs. The proposed method focuses on data reduction in fixed-point quantization of internal parameters to reduce the final size of the SNN model. The goal of [17] is to efficiently

and quickly estimate the approximation-induced errors for an SNN to reduce the SNN model size as much as possible for FPGA implementations while the [19] framework pursues distinct goals at the software level, such as increasing code readability and extensibility, unconcerned about reducing the exploration time.

This paper modifies the approach proposed in [17] by presenting a Design Space Exploration (DSE) technique based on the previous IA model coupled with the concept of watchers during the analysis. Watchers are specific observation points that can be placed in critical points of the computation flow to monitor violations of the acceptable accuracy of the application and help drive the DSE, thus saving time. The main benefit of this modification is that the exploration can generate the minimum bit-width required for each neuron, hence providing fine-tuning opportunities. Experiments were performed on a trained SNN designed for deploying FPGA hardware accelerators at the edge [20]. The proposed approach explored reducing the network parameters' size to fit the network in small FPGAs used for edge applications.

The rest of the paper is organized as follows: section II provides the background, while section III describes the proposed methodology. The experimental results are reported and analyzed in section section IV. And finally, section V provides the conclusions and future works.

II. Background

This section introduces an excerpt about the SNN architecture and the applied precision reduction technique from [20] and the fundamentals of the error propagation model from [17].

A. Spike Neuron Model and Network Description

Figure 1 shows the general organization of an SNN, highlighting the connections in one layer of neurons. This SNN model is based on the model employed in [20] that mimics the network structure with hardware components, and the approximation goal is to reduce the SNN size to be deployed on an FPGA. The input layer transforms the input data, the image pixels for the case study, into a sequence of spikes. Spikes are boolean single-bit information that enters the neuron through the excitatory step. Every time a spike enters the neuron, the *membrane potential* (V_0 in Figure 2) is accumulated. An output spike is generated when the *membrane potential* exceeds a threshold as indicated by the green arrows going out of the excitatory step in Figure 1. Output spikes of each layer serve as input spikes for the next layer and as inputs to the inhibitory steps inside the same layer. If other connected neurons to each neuron's inhibitory step generate any spikes, the inhibitory step decreases the neuron's *membrane potential*. Finally, the last SNN layer is followed by an output layer that can transform the received spikes into the network's final classification result.

Figure 2) depicts the computation flow of a single neuron. Suppose M different input spikes enter each neuron, and N different neurons exist in a layer. In that case, each neuron input has M weights ($w_{i,j}$) where i goes from 0 to $n-1$, indicating the neuron number in the layer and j goes from 0 to $m-1$ indicating the weight number in the neuron. Like other ANNs, the weights result from the training phase and are used in the inference phase [21].

Each time a spike is detected in a neuron's input connection, there is a chance to increase the V_i, which is $Neuron_i$ *membrane potential*. The sum of all neuron weights calculated in Equation 1 is used for increasing the V_i. Here, inp_j is the input spike and is set to 1 if a spike is detected at this input; otherwise, it is set to 0.

$$sum_i = \sum_{j=0}^{m-1} inp_j \cdot w_{i,j} \tag{1}$$

When the *membrane potential* reaches its maximum ($V_i > V_{thresh}$), the neuron fires a spike, while V_i is reset to a special *reset* value (V_{reset}), as depicted in Figure 2 following the "Yes" branch. Whereas, if no active input spike is detected, V_i should decrease, following the "No" branch in Figure 2, which indicates an *exponential decay* of the *membrane potential*. This happens by multiplying the V_i by $1 - \frac{\Delta t}{\tau}$.

The authors in [20] apply some simplifications to obtain a specific implementation of an SNN. For example, a single *membrane potential* threshold (V_{thresh}) is used for the whole layer to reduce the number of parameters of the network. Also, the *exponential decay* has been transformed into a right shift to employ a fixed-point number representation. Eventually, the classification method in [20] sees each neuron associated with an output spike counter, and decisions are based on the value of such counters. The presence of counters is not mandatory and is specific to that work.

B. Precision reduction

Authors in [20] converted the trained SNN parameters from floating-point to fixed-point format (expressed in Equation 2) to explore the precision reduction of the fractional part.

$$v = \underbrace{\sum_{i=16}^{31} b_i \cdot 2^{(i-16)}}_{integer} + \underbrace{\sum_{i=0}^{15} b_i \cdot 2^{-(16-i)}}_{fractional} \tag{2}$$

For a k-bit precision reduction of the fractional part, the approximation error ϵ follows Equation 3, where b_i is the value of the removed bit [17].

$$\epsilon = \sum_{i=0}^{k} b_i \cdot 2^{-(16-i)} \tag{3}$$

Equation 4 shows the approximation error limits [17]. The maximum error happens when each removed bit (b_i) is 1, while the minimum error occurs when each removed bit (b_i) is 0.

$$\underbrace{0}_{\forall b_i=0} \leq \epsilon \leq \underbrace{\sum_{i=0}^{k} 2^{-(16-i)}}_{\forall b_i=1} = -\frac{1-2^k}{32768} \tag{4}$$

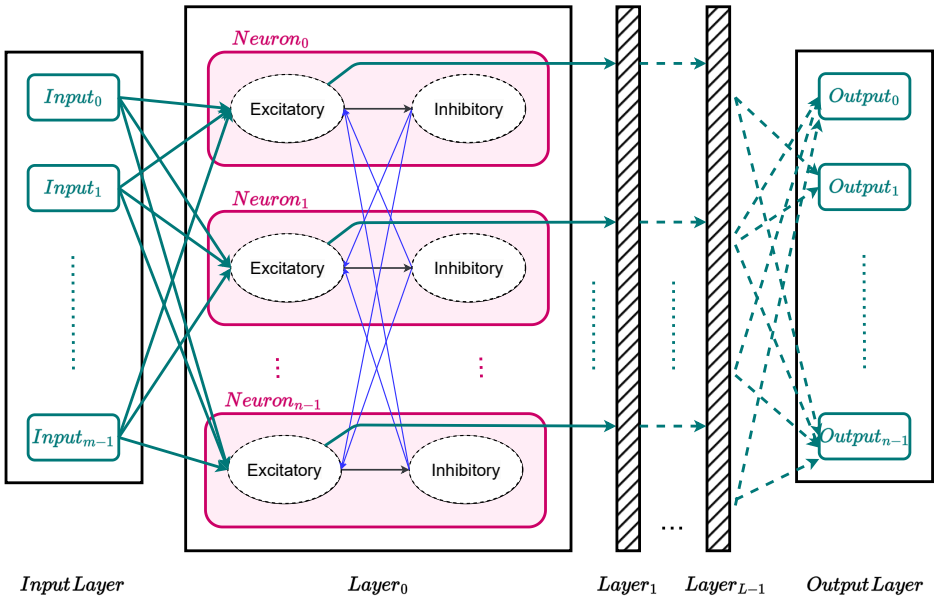

Fig. 1. SNN functional model, based on the model in [20], including a detailed description of one layer.

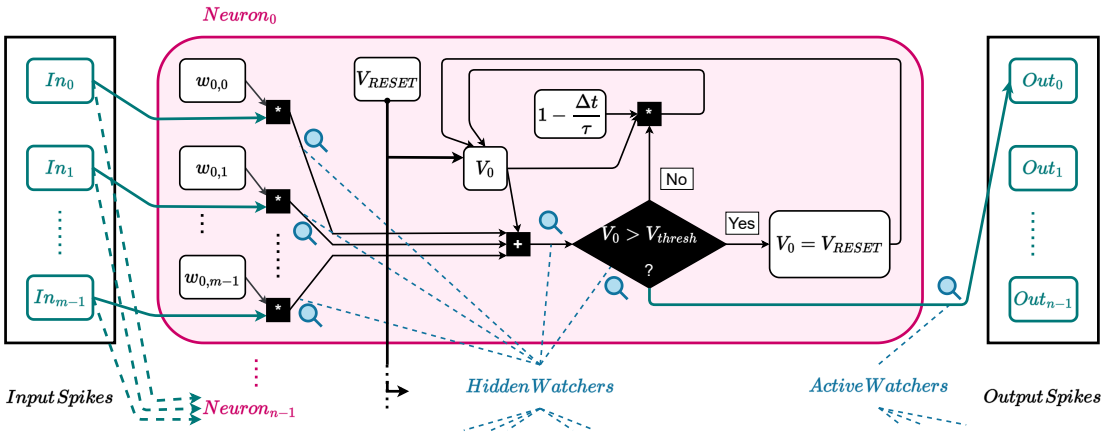

Fig. 2. Computation Flow of an SNN layer modified by the watchers for the exploration algorithm.

Using Equation 2, any approximated value v can be expressed as $v - \epsilon$, where ϵ represents a value subtracted from the fractional part of each number [17].

C. The Interval Arithmetic Error Propagation Model

Based on the error limits defined in Equation 4, [17] introduced an error propagation model resorting to the IA theory.

IA [18] defines mathematical operations over intervals instead of single values. An interval is defined as $[v] \equiv [v_{min}, v_{max}]$, where v_{min} and v_{max} are respectively the minimum and maximum numbers included in the interval. The exact value v can be replaced with an interval ($[v]$), where the two boundaries are the original v value itself ($[v] \equiv [v, v]$). Equation 4 already defines the error due to a precision reduction as a range $[\epsilon]$. Hence, [17] defines a generic approximated value as a IA number (denoted as $|ian|$) as a pair, including two intervals $\{[v], [\epsilon]\}$.

Since the applied approximation technique in [17] is precision reduction, the $[\epsilon]$ is subtracted from $[v]$. So, here, $|ian| \simeq [v] - [\epsilon]$. Also, the precision reduction errors formulated in Equation 4 are monotonic, hence, $[\epsilon] \equiv [\epsilon_{min}, \epsilon_{max}]$. These formulations still support the IA operators introduced in [18].

$$|ian_1| + |ian_2| = \{\underbrace{[v_{1_{min}} + v_{2_{min}}, v_{1_{max}} + v_{2_{max}}]}_{[v]}, \underbrace{[\epsilon_{1_{min}} + \epsilon_{2_{min}}, \epsilon_{1_{max}} + \epsilon_{2_{max}}]}_{[\epsilon]}\} \quad (5)$$

$$|ian_1| - |ian_2| = \{\underbrace{[v_{1_{min}} - v_{2_{max}}, v_{1_{max}} - v_{2_{min}}]}_{[v]}, \underbrace{[\epsilon_{1_{min}} - \epsilon_{2_{max}}, \epsilon_{1_{max}} - \epsilon_{2_{min}}]}_{[\epsilon]}\} \quad (6)$$

Using IA theory, [17] modeled addition, subtraction, multiplication, right shift, and comparison operators required by subsection II-A. For example, addition and subtraction between two values ($|ian_1|$ and $|ian_2|$) can be defined using the linearity of the two mathematical operations and the monotonic shape of the ranges as in Equation 5 and Equation 6. For more details on how the other mathematical operators are modeled, the reader may refer to [17], from which the IA-based modeling for this work was obtained.

III. Methodology

This section describes how the DSE methodology in [17] was modified by adding the watchers to the DSE framework, which is the main contribution of this paper.

The proposed modifications to accelerate the DSE methodology of [17] is depicted in Figure 2. The idea is to observe the approximation effects at specific points of the computation flow to monitor the output of the chosen arithmetic operation and modify the precision reduction procedure accordingly. The watchers are placed after every mathematical operation and are of two types: (i) *hidden watchers* are placed within the computation flow and evaluate intermediate computations, and (ii) *active watchers* observe the output of a neuron. Watchers allow a comparison between the interval at the monitored point and the acceptable error range.

Algorithm 1 outlines the evaluation strategy. The exploration process begins by setting the maximum precision reduction for all values in the model, including weights and thresholds (lines 5-9). The minimum error corresponds to a 1-bit precision reduction, while the maximum error occurs when a 15-bit precision reduction is applied (line 7). The IA model, introduced in subsection II-C, enables a rapid evaluation of the impact across the full range of precision reductions from 1-bit to 16-bit in a single step.

Implementing Watchers involves creating two key methods. The first method, `compareData(...)`, checks whether the error range at the observed point, expressed as an $|ian|$, falls within an acceptable error range using the redefined comparison operator provided by IA. An internal flag is set to True if the error range exceeds the acceptable limits. This flag can be checked using a second method, `watchFired()`.

The acceptable error range for all watchers is determined as the mathematical average between the minimum and maximum possible approximation errors, as calculated in Equation 4. The approximation-induced errors are also uniformly distributed because the network parameters, such as weights and thresholds, follow a uniform distribution. Thus, the mathematical average between the minimum and maximum approximation errors serves as the acceptable error threshold.

The definitions provided in subsection II-B, based on IA, enable any operation on the $|ian|$ and facilitate the implementation of the `compareData()` method. This versatile approach can be applied to the computation flow of any other application modeled using IA.

```
1  N := Number of Neurons;
2  A := Number of Active Watchers;
3  H := Number of Hidden Watchers;
4  explorationDone ← False;
5  for i = 0; i < N; i++ do
6  |   for j = 0; j < H; j++ do
7  |   |   k_best[i][j] ← 15;
8  |   end
9  end
10 while explorationDone <> True AND ∀k_best > 0
   do
11 |   explorationDone ← True;
12 |   runExploration();
13 |   for i = 0; i < N; i++ do
14 |   |   if actWatcher[i].watchFired() == True
   |   |   then
15 |   |   |   explorationDone ← False;
16 |   |   |   for j = 0; j < H; j++ do
17 |   |   |   |   if hidWatcher[i][j].watchFired() ==
   |   |   |   |   True then
18 |   |   |   |   |   k_best[i][j] −= 1;
19 |   |   |   |   end
20 |   |   |   end
21 |   |   end
22 |   end
23 end
```

Algorithm 1: DSE algorithm

In algorithm 1, lines 10 to 23 handle the exploration procedure. The iterative process continues until either all model values have no further approximation (when the condition $\forall k > 0$ at line 10 is no longer true) or when no active watcher is triggered after a full iteration, indicated by the $explorationDone$ flag remaining False. The first condition addresses the scenario where no further approximation is possible because all precision reductions have already been set to 0-bit reduction.

Line 12 is where the model processes all inputs within the while loop. After the model run, the watchers are checked as outlined in lines 13 to 22. If none of the active watchers, as depicted in Figure 2, detect an error range violation, the $explorationDone$ flag stays False (set in line 15), signaling the end of the exploration process. However, a backward analysis is performed if some active watchers detect a violation. This involves checking the hidden watchers from the end to the beginning of the computation flow to identify which watchers were triggered. The corresponding weights are then adjusted by reducing the number of cut bits by 1. This adjustment is possible because the IA model differentiates between the error and the value.

IV. Experimental Results

The proposed approach was employed to modify the DSE performed using the methodology in [17] to optimize the trained SNN from [20]. This use case was selected to allow a fair comparison with the original work and evaluate if the proposed modification to the methodology in [17] can provide benefits. The reference SNN was trained with the images from the MNIST dataset [22]. The MNIST dataset consists of 70,000 handwritten images of digits (from 0 to 9), of which 60,000 images are in the training set and 10,000 are in the test set. For generating the input spikes from the MNIST images, authors in [20] converted the 784 pixels composing each image of size 28x28 pixels into a sequence of spikes employing a random Poisson process [23], which finally resulted in 3,500 input spikes for each pixel.

The reference SNN has one layer with 400 neurons. The structure of the network decision model comprises 400 counters placed at the end of each neuron, which are split into groups of 40 counters, and the classification is made by selecting the group with the highest number of counts. Since the SNN performs a digit classification task on the MNIST dataset, each of the ten groups represents a digit. The SNN primary outputs are the output spike counters, and the secondary outputs are the classification decisions made by majority voting among the spike counters at the final decision layer. Since the approximation impacts the number of output spikes indicated by the spike counter, if the majority voting results are unaffected for a specific approximation compared to the exact SNN, the SNN classification accuracy remains the same. In other words, the change in counters value due to approximation can change the classification results only if the counter of the right group does not show the highest value compared to the counters of the wrong groups. As a result, applying approximation can decrease the network classification accuracy when the number of times a handwritten digit image is classified into the wrong group increases.

The proposed approach was applied considering the network behavior when processing 1,000 images of the MNIST test set. The IA-based model was built based on the computational flow in Figure 2. Using the algorithm 1 described previously, the model was then enhanced to have watchers in place. On average, the exploration stopped after eight iterations when no active watcher went off anymore. Figure 3 shows the number of neurons updating the precision reduction cut at each iteration. In the highest precision reduction case, the final reduction was down to 10 bits removed from almost all weights as in the reference work [20]. Nevertheless, the exploration pointed out some weights that were kept with an even higher reduction of 11 bits removed. This is a huge difference from the reference work, where the authors proposed a fast approach to size reduction and imposed the same precision reduction on all values in the model.

To double-check the obtained results at the end of the exploration, the classification results obtained from the counters computed by the model were compared with the classification results of the original approximate SNN in [20] when applied the least bit reduction. The classification accuracy for the approximated SNN version obtained here compared to the original approximated SNN was untouched. This approach for comparing the reduced size SNN results with the approximated SNN results from the original work [20] is the same approach taken by [17]. It is impossible to seek exact correspondence of counters values since this approach's reduced size SNN includes different bit precision reductions for different network weights. At the same time, the original work applied a precision reduction with a specific number of bits to all the network parameters each time to obtain an approximated SNN. For better comparison, the least bit reductions obtained by the proposed approach can be denoted as l and the highest bit reductions as h. Hence, the original approximated SNN with l bit reduction for all network weights could be further approximated with some weights undergoing h bit reduction. In contrast, this higher approximation does not impact the resulting network classification accuracy. It is important to mention that on average, for the 1000 images from the MNIST test set, the h equals 10 bits and l equals 8 bits.

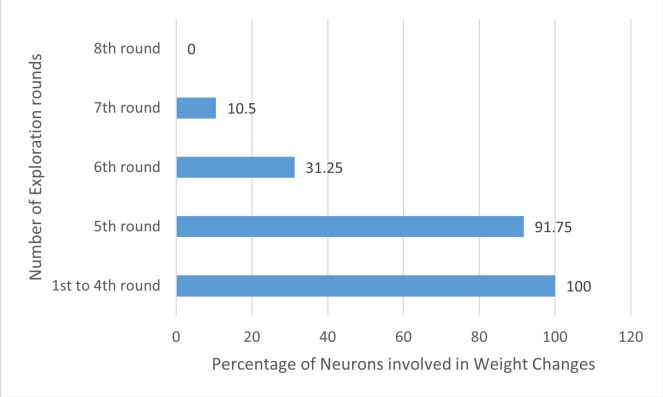

Fig. 3. Percentage of neurons that needed more precision for the weights at each iteration of exploration for one image.

Regarding time impact, the reference system is a laptop computer with 16GB RAM and an Intel Core i7 processor. In the experiments with 1,000 images, eight iterations required a total exploration time of 539.842 seconds, with each iteration averaging 67.480 seconds. In contrast, without the IA model and watchers, the reference network takes 0.159 seconds to complete a single image inference. Although the raw numbers suggest that the original model is faster, further evaluation is necessary from an exploration perspective. Considering the model size, each neuron has 784 weights, and with 400 neurons, the SNN contains a total of 313,600 weights to optimize. Since the precision reduction involves 16 different values (ranging from 15 to 0 bits removed), this results in a potential number of combinations of different approximated values equal to 4.18×10^{74}, as defined by the combination formula in Equation 7.

$$\binom{n}{r} = \frac{n!}{r!(n-r)!} = \binom{313600}{16} = \frac{313600!}{16!(313584)!} \quad (7)$$

To fairly compare the exploration approach with the reference network, it is necessary to evaluate the time required to explore all possible combinations of approximations. Running the inference, which takes 0.159 seconds per instance, across all 4.18×10^{74} combinations would take approximately 6.65×10^{73} seconds, which is unfeasible. It is important to note that the authors in [20] did not explore the entire space, which is a critical consideration when interpreting the results. By approximating each value in the model, the final configuration is fine-tuned compared to the original, allowing the proposed approach to apply different precision reductions to different SNN parameters, resulting in a more optimized final configuration.

V. Conclusion

This paper modified a previously proposed DSE approach for precision reduction of an SNN model. The algorithm builds upon an IA-based model of the SNN computation flow that allows tracking the error propagation and comparing the approximation error value to its threshold at each point of the computation flow. The experiments on a trained SNN available in the literature with a preliminary model reduction obtained the same results as the original work using the DSE methodology reasonably quickly. Experimental results comparing this modified model to the original work confirm that the exploration time is reduced significantly while providing the opportunity to reduce the precision of weights with more fine-tuned bit precision reductions.

The model can extend to other applications with complex computations, specifically other ANNs and even more complex SNNs. Also, different AxC techniques can be employed and compared to further investigate this approach's extensibility. Another direction might include exploiting this approach to enhance the exploration with a multi-objective evaluation, including other design parameters, such as power consumption and application execution time.

Acknowledgment

This work has received funding from the APROPOS project in the European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement No 956090.

References

[1] A. Bosio, S. D. Carlo, P. Girard, E. Sanchez, A. Savino, L. Sekanina, M. Traiola, Z. Vasicek, and A. Virazel, "Design, verification, test and in-field implications of approximate computing systems," in *2020 IEEE European Test Symposium (ETS)*, 2020, pp. 1–10.

[2] P. Stanley-Marbell, A. Alaghi, M. Carbin, E. Darulova, L. Dolecek, A. Gerstlauer, G. Gillani, D. Jevdjic, T. Moreau, M. Cacciotti, A. Daglis, N. E. Jerger, B. Falsafi, S. Misailovic, A. Sampson, and D. Zufferey, "Exploiting errors for efficiency: A survey from circuits to applications," *ACM Comput. Surv.*, vol. 53, no. 3, jun 2020. [Online]. Available: https://doi.org/10.1145/3394898

[3] D. Wu and J. S. Miguel, "Special session: When dataflows converge: Reconfigurable and approximate computing for emerging neural networks," in *2021 IEEE 39th International Conference on Computer Design (ICCD)*, 2021, pp. 9–12.

[4] A. Piri, S. Saeedi, M. Barbareschi, B. Deveautour, S. D. Carlo, I. O'Connor, A. Savino, M. Traiola, and A. Bosio, "Input-aware approximate computing," in *2022 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR)*, 2022, pp. 1–6.

[5] A. Savino, M. Portolan, R. Leveugle, and S. Di Carlo, "Approximate computing design exploration through data lifetime metrics," in *2019 IEEE European Test Symposium (ETS)*, 2019, pp. 1–7.

[6] S. C. Smithson, G. Yang, W. J. Gross, and B. H. Meyer, "Neural networks designing neural networks: Multi-objective hyper-parameter optimization," in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2016, pp. 1–8.

[7] E. Dupuis, D. Novo, I. O'Connor, and A. Bosio, "On the automatic exploration of weight sharing for deep neural network compression," in *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2020, pp. 1319–1322.

[8] M. Traiola, A. Savino, and S. Di Carlo, "Probabilistic estimation of the application-level impact of precision scaling in approximate computing applications," *Microelectronics Reliability*, vol. 102, p. 113309, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0026271418309442

[9] A. Savino, M. Traiola, S. D. Carlo, and A. Bosio, "Efficient neural network approximation via bayesian reasoning," in *2021 24th International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*, 2021, pp. 45–50.

[10] W. Maass, "Networks of spiking neurons: The third generation of neural network models," *Neural networks*, vol. 10, no. 9, pp. 1659–1671, 1997.

[11] M. Capra, B. Bussolino, A. Marchisio, G. Masera, M. Martina, and M. Shafique, "Hardware and software optimizations for accelerating deep neural networks: Survey of current trends, challenges, and the road ahead," *IEEE Access*, vol. 8, pp. 225 134–225 180, 2020.

[12] T. Ayhan and M. Altun, "Approximate fully connected neural network generation," in *2018 15th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*, 2018, pp. 93–96.

[13] Z. Peng, X. Chen, C. Xu, N. Jing, X. Liang, C. Lu, and L. Jiang, "Axnet: Approximate computing using an end-to-end trainable neural network," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2018, pp. 1–8.

[14] Z. Wang, M. A. Trefzer, S. J. Bale, and A. M. Tyrrell, "Approximate multiply-accumulate array for convolutional neural networks on fpga," in *2019 14th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, 2019, pp. 35–42.

[15] J. P. Turner and T. Nowotny, "Arpra: An arbitrary precision range analysis library," *Frontiers in Neuroinformatics*, vol. 15, 2021. [Online]. Available: https://www.frontiersin.org/article/10.3389/fninf.2021.632729

[16] H. Okada, T. Matsuse, T. Wada, and A. Yamashita, "Interval ga for evolving neural networks with interval weights and biases," in *2012 Proceedings of SICE Annual Conference (SICE)*, 2012, pp. 1542–1545.

[17] S. Saeedi, A. Carpegna, A. Savino, and S. Di Carlo, "Prediction of the impact of approximate computing on spiking neural networks via interval arithmetic," in *2022 IEEE Latin-American Test Symposium (LATS)*, 2022.

[18] "Ieee standard for interval arithmetic," *IEEE Std 1788-2015*, pp. 1–97, 2015.

[19] E. Darulova, A. Izycheva, F. Nasir, F. Ritter, H. Becker, and R. Bastian, "Daisy-framework for analysis and optimization of numerical programs (tool paper)," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2018, pp. 270–287.

[20] A. Carpegna, A. Savino, and S. Di Carlo, "Spiker: an fpga-optimized hardware accelerator for spiking neural networks," in *IEEE ISVLSI 2022*, IEEE, Ed., 2022.

[21] D. Padovano, A. Carpegna, A. Savino, and S. Di Carlo, "Spikeexplorer: Hardware-oriented design space exploration for spiking neural networks on fpga," *Electronics*, vol. 13, no. 9, 2024. [Online]. Available: https://www.mdpi.com/2079-9292/13/9/1744

[22] C. C. Yann LeCunn and C. J. Burges. The mnist database. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[23] D. Heeger, "Poisson model of spike generation," *Handout, University of Standford*, 10 2000.