HW/SW Codesign for Approximation-Aware Binary Neural Networks

Abhilasha Dave, Fabio Frustaci *Senior Member, IEEE*, Fanny Spagnolo *Member, IEEE*, Mikail Yayla, Jian-Jia Chen *Member, IEEE*, and Hussam Amrouch *Member, IEEE*

***Abstract*—Binary Neural Networks (BNNs) are rapidly gaining remarkable attention due to their superiority in shrinking the model size, which outstandingly mitigates the fundamental "memory wall" bottleneck that is attributed to the existing von-Neumann architectures. This work investigates how principles from approximate computing can be effectively employed to further optimize BNNs. It demonstrates that HW/SW codesign, in which BNNs are either proactively trained in the presence of approximation-induced errors (i.e. design-time optimization) and/or augmented with an appropriate error-mitigation scheme (i.e., run-time optimization), is a key to realize energy-efficient yet robust BNNs.**

We unveil, for the first time, that although the underlying HW of BNNs can be implemented using simple `XNOR` gates, the complexity of the required "Popcount" circuit *super-linearly* grows with the filter kernel size. This largely impacts the area footprint, inference time, energy, and hence it severely constricts the prospective efficiency gains from BNNs. To overcome this challenge, we replace the accurate full adders constructing the Popcount with Majority gates that approximately perform the required additions. Then, our carefully-crafted error-mitigation scheme along with activations tuning considerably minimizes the induced errors. Afterward, abstracted error probabilities are derived and employed during BNN training to obtain approximation-aware BNNs, that are inherently robust against the underlying HW approximation. Differently from the typical approaches, the proposed HW/SW codesign methodology has the merit of allowing a training of the approximate BNN without the need to modify the existing software frameworks (i.e., PyTorch). This is of great importance since existing tools rely on efficient built-in functions that can be difficult and/or inefficient to be modified. An FPGA-based SoC realizing both accurate and approximation-aware BNNs is developed for validating our proposed methodology. With merely a 4.7% loss in the inference accuracy, our HW/SW codesign leads to 64% and 80.2% savings in the area and energy, respectively, at the parity of the latency. Our results are obtained using commercial EDA tool flows employing a commercial 28nm FDSOI technology node.

***Index Terms*—Approximate computing, Neural network, low-power design, FPGA.**

I. Introduction

In the past few years, Deep Neural Networks (DNNs) have emerged as a powerful methodology to enable artificial intelligence in several application fields such as computer vision, speech recognition, scientific computing, and many others [1]. Nevertheless, DNNs are still challenging for the existing von-Neumann architectures because the model size of DNNs is often very large. DNNs, in fact, demand not only a massive amount of data to be in parallel computed but, also a huge communication with memories. The former imposes a serious challenge when it comes to on-chip temperatures due to excessive power densities [2] caused by the multiply-accumulate (MAC) operations and the latter induces a profound source of efficiency loss [3], that needs to be indispensably avoided due to the inevitable "memory wall". The aforementioned challenges are further aggravated when it comes to embedded systems and edge computing not only because hardware resources are very limited in such systems, but also because cooling capability and energy budgets are tightly restricted.

After they were introduced in late 2016 [4], BNNs are continuously attracting remarkable attention due to their superiority in constructing ultra-lightweight deep learning models in which weights and activations are represented by merely a single bit. This significantly shrinks the model size leading to much less memory communication. In addition, the dominant power-hungry MAC operations in traditional DNNs can be replaced by simple `XNORs` followed by a population count (Popcount) circuit. Therefore, BNNs offer significant reductions in energy consumption and memory requirements. This considerably improves the efficiency and makes BNNs suitable for many edge-AI applications [5].

A. Our Novel Contributions and Findings within this Paper

This work originates from a preliminary analysis of the energy breakdown due to the components of a BNN convolutional layer. Our analysis demonstrates how the major bottleneck in BNN HW stems, in fact, from the Popcount circuit because its complexity super-linearly grows with the filter kernel size, contrary to the other components like `XNOR` gates and comparators. This severely constricts the efficiency gains obtained from binarizing the NN model. Towards further optimizing BNNs in which the efficiency is boosted while inference accuracy is marginally impacted:

(1) We propose a novel approximation-aware hardware-friendly activation function that drastically reduces the probability of errors induced by approximation on the outputs of the BNN layers (P_{error}). Thanks to the effectiveness of the proposed error-mitigating scheme, we are able to aggressively approximate the Popcount circuit through employing Majority gates in the deeper layers of the Popcount adder tree to

Corresponding authors: Fabio Frustaci and Hussam Amrouch

F. Frustaci and F. Spagnolo are with the DIMES Department, University of Calabria, Italy, Email: {f.frustaci, f.spagnolo}@dimes.unical.it

M. Yayla and J.-J. Chen are with the Design Automation for Embedded Systems Group, TU Dortmund University, Germany, Email: {mikail.yayla, jian-jia.chen,}@udo.edu

A. Dave and H. Amrouch are with the Chair for Semiconductor Test and Reliability (STAR), University of Stuttgart, Germany, Email: {daveaa, amrouch}@iti.uni-stuttgart.de.

enhance the energy efficiency. In such a case, BNN re-training can be avoided for a certain accuracy envelope. Compared with the simple case when the approximate BNN does not exploit any countermeasure against errors, our approximation reduces the P_{error} of the convolutional operation from 41.8% to merely 2.4% (with no energy penalty) when, for instance, a $3\times3\times64$ convolutional hardware accelerator is examined.

(2) We develop an evaluation setup using an FPGA-based SoC in which PyTorch [6] runs on top of the ARM CPU, and the latter is augmented by customized (both accurate and approximate) HW accelerators to realize BNNs. Our FPGA platform serves a dual purpose. (i) On the one hand, it enables us to *validate* the correct execution of BNNs on top of *an actual accurate and approximate HW*, instead of relying only on a pure SW execution (i.e., just PyTorch) that mimics HW approximation though injecting errors at a certain probability (P_{error}) during the BNN inference. (ii) On the other hand, our implemented FPGA platform is employed to efficiently explore the existing design space and find optimal parameters according to our approximation strategy, as well as to construct the (P_{error}) models.

(3) We demonstrate how our developed P_{error} models open doors for BNN HW/SW codesign. On the one hand, investigating, at the SW level, the sensitivity of every BNN layer against errors, enables designers to selectively employ different approximations at the HW level. On the other hand, employing the information about P_{error} allows an easy BNN re-training process that, differently from the state-of-the-art approach [7], does not need modifications of the typically used software tools (i.e. PyTorch). This is an important feature of the proposed approach since such tools rely on built-in software routines that can be difficult and/or inefficient to modify in order to mimic the intended hardware approximation. The presented approach provides designers with approximation-aware BNNs that are inherently more robust against the underlying HW approximation.

(4) We perform a deep analysis of the energy saving achievable by the proposed approximate approach under different conditions, employing a commercial 28nm FDSOI technology. We demonstrate that, when applied to implement a VGG3-based model, our approximation-aware BNN provides 80.2% less energy, while the accuracy drops by merely 4.7% with respect to the accurate implementation.

II. Related Work

Due to the ultra-low precision representations of parameters in BNNs (i.e., 1-bit representation of weights and activations), BNNs exhibit an intrinsic resiliency against bit errors, compared to quantized (e.g., 8-bits) NN models. Several works have recently investigated the bit-error tolerance of BNNs. In [8], the impact of the voltage scaling on the bit-error rate caused by write and read failures in the activations/weights memory is analyzed. It showed that BNN could tolerate the errors caused by voltage scaling without a sensible degradation in the classification accuracy. A similar analysis is carried out in [9] for BNNs where the activations/weights memories are implemented using emerging non-volatile ferroelectric memory. The work presented in [9] proposed a bit-flip based retraining framework useful to recover the BNN inference accuracy when the bit-error rate is too high.

Drawback of NN re-training: Although NN retraining is a powerful way to compensate for the reduction in accuracy due to errors, it may not always be preferable or feasible. The major drawback is that one needs to modify the built-in routines on which the software design tools rely for back-propagation step during the training process in order to take into account the effects of the approximation. This may be particularly difficult since it implies a deep technical knowledge about how such tools work. Moreover, the computational optimization of the built-in routines may be lost once they are customized for the intended approximation. Finally, retraining might not be possible because, in some cases, the training set might not even be available (e.g., proprietary models) [10]. The bit-error tolerance property of BNNs makes them suitable to be implemented while applying approximate computing concepts [11], which aim at trading-off accuracy with energy.

In approximate BNNs, errors are due to the employed approximation in the underlying HW. Several works have focused on replacing the multiplication operation in MACs with simple `XNORs`. Nevertheless, little research has been so far conducted w.r.t the Popcount operation, *which turns to be the most energy-consuming component in BNN hardware*, as we demonstrate in Section III. Since the Popcount circuit is basically an adder tree, a straightforward approximation is to substitute the Full-Adders (FAs) of the first stage of the adder tree with majority gates, as recently proposed in [7]. However, [7] does not explore the impact of further approximation in the deep layers of the adder tree, which limits its energy saving. Further, [7] does not employ any error mitigation, which makes BNN inference suffers from larger accuracy drops. Finally, the only way proposed in [7] to recover an acceptable accuracy is re-training based on customized modification of the training frameworks, such as PyTorch. As above mentioned, this maybe unfeasible because a) sometimes one has access to only the BNN pre-trained model, without the knowledge of the training set; b) customizing the typically available training frameworks, such as PyTorch, implies difficult modification of the code that can lead to a drastic increase of the execution time, since the conventional operations rely on efficient built-in functions. Moreover, defining a software model of the backward propagation based on the adopted approximation is not trivial.

Different from the state of the art [7], our approximation-aware BNN does not need retraining for a given accuracy envelope because it exploits a simple error recovery strategy. Further, when higher accuracy is needed, the proposed approximation-aware approach relies on a straightforward re-training process that is based on the developed P_{error} model. Due to its error resiliency, our approach achieves a very high accuracy (close to the nominal baseline accuracy) without retraining. At the same conditions, the strategy described in [7] leads to a BNN accuracy close to zero ($\sim 10\%$), as Section VII later illustrates, when it is performed without model retraining.

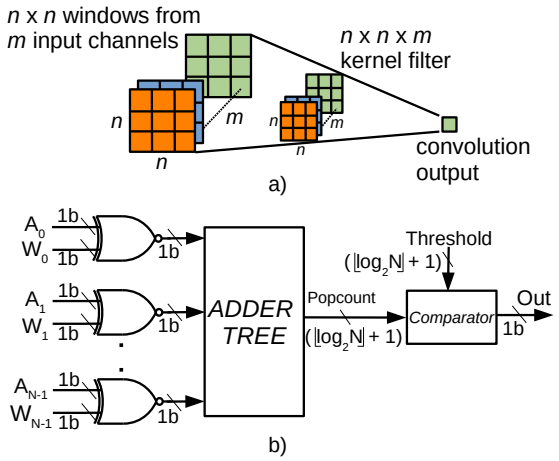


Fig. 1. (a) Convolutional operation with filter kernel size of $N = n \times n \times m$. (b) The correspondent BNN hardware architecture demonstrating the XNORs, adder tree, and comparator that does the thresholding required for activation.

III. Motivational Case-Study

Activations (A) and weights (W) in CNNs are represented in floating-point (e.g., 32 bits) or fixed-point (e.g., 8 bits) notation. On the contrary, they are quantized to the extreme level of a single bit in BNNs. Besides the large reduction in the model size, binarization allows replacing the expensive multiplication in MACs with simple XNORs, as Fig. 1 illustrates. In such a case, given a set of $N = n \times n \times m$ activations and weights (with $n \times n$ and m being the size of each kernel window and the number of input channels respectively), the convolution operation can be described as:

$$Popcount = \sum_{j=0}^{N-1} \text{XNOR}(W_j, A_j) \quad (1)$$

The addition, in this case, is a *Popcount* operation because it aims at counting the number of 1's among the outputs of all XNORs. The Popcount is then followed by the activation function defined as:

$$2 \times Popcount - N > T \quad (2)$$

where T is a threshold value obtained during the training phase of BNN. The 1-bit activation output is '1' whether the condition in Eq. (2) is satisfied ('0' if it is not).

The XNORs substitute the complex multipliers in MACs leading to high reductions in area, power, and delay. The outputs of the XNORs are composed of a single bit. Hence, an adder tree performing the Popcount operation can be employed. This is much simpler than its counterpart in a traditional CNN. Further, it is relatively easy to design such a Popcount with a high number of inputs. Importantly, the convolutional layer of a BNN usually operates on a relative high number of input channels (m) in parallel. In addition, the number N of the adder tree inputs is also high. Therefore, it can be straightforwardly predicted that the critical bottleneck in the "MAC" in a BNN HW with respect to area, latency, and energy moves from the multiplication (i.e., XNOR) towards the addition (i.e., Popcount), unlike the original situation in the "MAC" in a traditional CNN HW in which the multiplication does form the major bottleneck.

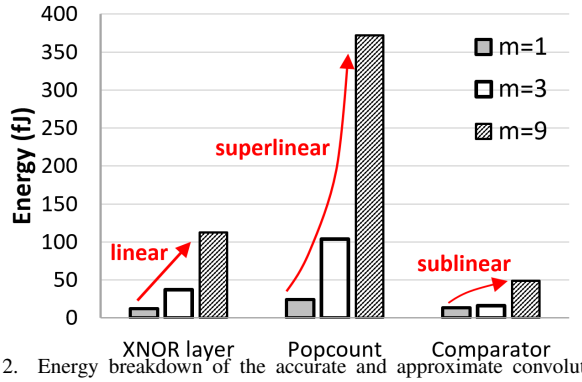


Fig. 2. Energy breakdown of the accurate and approximate convolutional layer (n=3) for different values of m.

Fig. 2 depicts the energy breakdown of the binary convolutional layer of Fig. 1 for n=3 and different values of m. The characterized hardware designs have been synthesized with Cadence Genus and mapped to a commercial 28nm FDSOI technology at their maximum operating frequency. As expected, the energy consumption of the XNOR layer has a linear relation with m because the number of XNORs is $N = n \times n \times m$. Importantly, the major contribution in the energy of convolutional layer comes from the Popcount: 48%, 66%, and 69% for m=1, m=3 and m=9, respectively. From Fig. 2, the Popcount's energy grows as m^{α}, with α that can be empirically set to 1.25 (since $\alpha > 1$, we can say that the Popcount energy is superlineraly dependent on m). The last component of the convolution operation, i.e., the comparator, is less affected by m because the number of bits of its inputs is proportional to $\lfloor \log_2 N \rfloor$. Therefore, its energy exhibits a sublinear relation with m. As an example, when m goes from 1 to 3 (9), the number of bits of the comparator inputs increases by one (three) unit(s) and the energy accordingly increases by about 1.2× (3.6×).

In summary: The Popcount is the most energy consuming operation in the convolutional layer of a BNN. To further optimize the efficiency of BNN, we aim in the next Section at applying principles from approximate computing to the Popcount circuit. The challenge is, nevertheless, how such approximations can be applied to minimize area, latency, and energy without degrading the inference accuracy.

IV. Our Popcount Approximation

The hardware architecture of the Popcount operation is essentially an adder tree summing N 1-bit inputs. This is different from an N-bit adder, of which several approximate designs have been proposed in the past [12]. The existing approximate techniques for N-bit adders focus on decreasing the mean error value of the addition result (i.e., the mean of the difference between the correct and approximate result) and the error variance. On the contrary, when approximating the Popcount in a BNN, the focus should not be on how to keep the approximate result of the sum as close as possible to the exact value but rather on how to ensure that the condition described by the activation function Eq. (2) is still satisfied

as often as possible, for various threshold T values. Hence, more approximations (i.e., more levels in the adder tree) can be applied.

A popular way to approximate the operation of counting the number of 1's in a N-bit bus is through adopting the Majority gate (MAJ) [7]. The latter has an odd number k of 1-bit inputs and one 1-bit output whose value equals the value of the majority of the inputs. Fig. 3(a) depicts the use of MAJ gates for the case k=3 [7]. It is noteworthy that the number of inputs to the accurate adder is reduced by a factor of 2 (since the MAJ gate has one output bit whereas the FA has two output bits), thus, reducing its hardware complexity, delay, and energy as a result. *In practice, the MAJ gates substitute the accurate Full-Adders (FAs) by computing only the carry-out signal.* Hence, the final result of the Popcount should be left-shifted by one-bit position. As a further benefit, the MAJ gate is more performing than a FA: in the adopted 28nm technology, its area, delay and energy consumption are 50%, 24% and 21% lower, respectively.

A. Mathematical model

In the following, we give a mathematical insight about the effect of using MAJ gates in place of FAs in terms of the error committed in evaluating $Popcount$ and the error probability in verifying the inequality (2). For a given number on input activation bits N, a particular value of Popcount=$x \in [0, N]$ can be generated by a set of input bits configurations whose number Q_x can be expressed as:

$$Q_x = \frac{N!}{x!(N-x)!} \tag{3}$$

With the term $input\ bits\ configuration$ we refer to a particular input bits sequence $(in_0, in_1, in_2, \ldots, in_{N-1})$, with in_j=0 or 1 $\forall j \in [0, N-1]$. Moreover, without lost of generality, we suppose to group the bits of a particular input configuration into $\frac{N}{3}$ sets of three consecutive bits (a_h, a_{h+1}, a_{h+2}), with h=0, 3, 6, …,(N-3), and that the generic h-th set is inputted to the FA$_h$ (3-input MAJ$_h$ gate) for the case of accurate (approximate) Popcount operation. In the following, the decimal result of the generic FA$_h$ (MAJ$_h$ gate) will be indicated as S_h ($\hat{S}_h$).

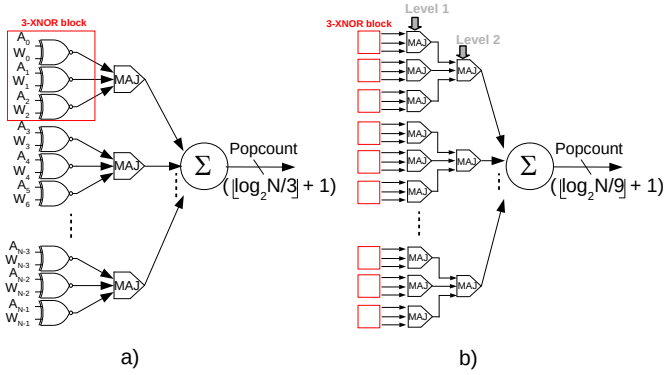


Fig. 3. Approximating the Popcount circuit: a) conventional approach in which merely *one level* is approximated through the majority gates, b) proposed approach in which *multiple levels* are approximated through the majority gates The figures shows the case of 2 levels ($L = 2$) as an example.

It is worth noting that Eq. (3) gives the number of x-element combinations of N objects, without repetition, and that $Q_x = Q_{N-x}$. When MAJ gates are used in place of FAs, the error committed by the approximate Popcount operation in calculating x (after the aforementioned one-bit left-shifting) depends on the particular input bits configuration among all the possible Q_x ones. As an example, for $N = 9$ and $x = 2$, there are 36 possible inputs configurations, such as C_1=(1,1,0,0,0,0,0,0,0) and C_2=(0,0,1,1,0,0,0,0,0). According to the adopted grouping convention, in the case of the approximate design, C_1 would give the approximate result $\hat{x}$=2 (i.e. with an error equal to 0) whereas C_2 would furnish $\hat{x} = 0$ (i.e. with an error equal to -2). In general, the error $err_h = \hat{S}_h - S_h$ committed by the generic MAJ$_h$ gate is in the range [-1, 0], Indeed, for $S_h = 0, 1, 2, 3$, $err_h = 0, -1, 0, -1$, respectively. It follows that the error Err_x=$\hat{x} - x$ in computing the approximate Popcount value $\hat{x}$ is $\in [-\frac{N}{3}, 0]$ and that it depends on how the N input bits are grouped. Towards the aim of finding Err_x, it is useful to point out that a generic input configuration C_i (with i=0,…,Q_x-1) leading to a Popcount value x can be uniquely associated to a sequence seq=(S_0,S_3,S_6,…,S_{N-3}) with $S_h \in [0, 3]$ and $\sum_{h=0}^{N-3} S_h = x$, where, as stated above, S_h is the result of the h-th FA. For a given x, the problem of finding all the possible sequences associated to that value of x is equivalent to find all the partitions of x into $\frac{N}{3}$ parts in {0, 1, 2, 3}. There is no general mathematical expression for calculating the number Seq_x of such partitions, but some closed-form expressions can be derived following a recursive process, as described in [13]. As an example, for $N = 9$, the following closed-form expression can be used:

$$Seq_x = \begin{cases} 1 + \lfloor \frac{x^2+6x}{12} \rfloor, & \text{if } x \leq 3. \\ 1 + \lfloor \frac{x^2+6x}{12} \rfloor - \sum_{i=0}^{x-4} 1 + \lfloor \frac{i}{2} \rfloor, & \text{otherwise.} \end{cases} \tag{4}$$

It is worth noting that once a partition for x has been found, let's say S_i^x=($S_{0,i}^x$,$S_{3,i}^x$,$S_{6,i}^x$,…,$S_{N-3,i}^x$) (with i = 0,…,Seq_x − 1), each other sequence with the same terms $S_{h,i}^x$ but placed in a different order can be also associated to x, and all these equivalent sequences lead to the same error err_i^x when MAJ gates are employed instead of FAs. With n_i^x being the number of such equivalent sequences, the mean error committed by approximating the FAs with MAJ gates for each of the S_i^x sequences can be expressed as:

$$Err_i^x = n_i^x \times Prob_i^x \times err_i^x \tag{5}$$

where $Prob_i^x = \prod_{h=0}^{N-3} Prob(S_{h,i}^x)$ and $Prob(S_{h,i}^x)$ being the probability of each term $S_{h,i}^x$. From the FA truth table and assuming an uniform probability distribution for the input configurations, it can be easily found that $Prob(S_{h,i}^x) = \{\frac{1}{8}, \frac{3}{8}, \frac{3}{8}, \frac{1}{8}\}$ for $S_{h,i}^x = \{0, 1, 2, 3\}$, respectively. Finally, it should be noted that each sequence S_i^{N-x}=($S_{0,i}^{N-x}$,$S_{3,i}^{N-x}$,$S_{6,i}^{N-x}$,…,$S_{N-3,i}^{N-x}$) associated with the Popcount value $(N - x)$ can be obtained from each sequence S_i^x associated to the Popcount value x by making the following

substitutions (with h=0, 3, 6, ...,$(N-3)$):

$$\begin{cases} S_{i,h}^{N-x} = 3 & \text{if } S_{i,h}^{x} = 0. \\ S_{i,h}^{N-x} = 0 & \text{if } S_{i,h}^{x} = 3. \\ S_{i,h}^{N-x} = 1 & \text{if } S_{i,h}^{x} = 2. \\ S_{i,h}^{N-x} = 2 & \text{if } S_{i,h}^{x} = 1. \end{cases} \quad (6)$$

and that $err_i^x + err_i^{N-x} = -1 \times \frac{N}{3}$, where -1 is equal to the maximum error committed by a MAJ gate. From (5) and (6) it follows that the mean error $\hat{Err}$ committed when the MAJ gates are employed is:

$$\hat{Err} = \sum_{j=0}^{\lceil \frac{N}{2} \rceil} \sum_{i=0}^{Seq_j - 1} Err_i^j = -\frac{1}{2} \times \frac{N}{3} \quad (7)$$

Just as an example, let's take the case $N = 9$ and $x = 7$. There are two possible partitions $S_0 = (3, 3, 1)$ (with a probability of $\frac{1}{8} \times \frac{1}{8} \times \frac{3}{8} = \frac{3}{512}$ and an error equal to -3) and $S_1 = (3, 2, 2)$ (with a probability of $\frac{1}{8} \times \frac{3}{8} \times \frac{3}{8} = \frac{9}{512}$ and an error equal to -1). The partition S_0 (S_1) can be ordered into $n_0^2 = 3$ ($n_1^2 = 3$) ways, so that the mean error committed in the Popcount function by using MAJ gates is, according to (5): $3 \times \frac{3}{512} \times (-3) + 3 \times \frac{9}{512} \times (-1)$ = -$\frac{54}{512}$. As described at the beginning of Section IV, approximating the Popcount function may cause errors when evaluating the inequality (2). Taking into account that $Err_x = \hat{x} - x \leq 0$, the approximate Popcount circuit will produce a failure in evaluating (2) when $T - (2 \times \hat{x} - N) \geq 0$. As an example, with N=9 and T=4, the number of failure occurrences would be 36, 9 and 1 for x=7, 8 and 9, respectively.

We analyzed the effect of the MAJ-based approximation on the output of the activation function by exhaustively evaluating the condition Eq. (2) for the simple case N=9 and k=3 that is the smallest block composed of 3 XORs followed by 1 MAJ (3XORs-1MAJ block). All the possible 2^9 inputs have been considered with the threshold T varying in $[-9, 9]$. As shown in Fig. 4, the error probability in evaluating Eq. (2) is not negligible, and it can be as high as 41%, whereas its mean value is 15%.

B. The proposed error mitigation technique

To reduce the P_{error}, we propose to add a simple correction factor M equal to the mean error of the left term of Eq. (2) (i.e. $M = -2 \times \hat{Err}$):

$$2 \times Popcount - N + M > T \quad (8)$$

Note that Eq. (8) does not introduce any hardware modification of the Popcount circuit because the correction factor M can be easily integrated through modifying the threshold value itself. It is worth pointing out that the approximate Popcount function causes an erroneous thresholding result when $(2 \times \hat{x}) - N \leq T < (2 \times x - N)$. In such a case, the proposed error mitigation scheme is able to correct the errors introduced by the approximation if the condition (9) occurs:

$$\begin{cases} 2 \times (x - N) - T > 0 \\ 2 \times (Err_x) + M + (2 \times x - N) - T > 0 \end{cases} \quad (9)$$

On the other hand, when $(2 \times \hat{x}) - N \leq (2 \times x - N) \leq T$, the proposed error mitigation scheme increases the number of error occurrences if:

$$\begin{cases} 2 \times (x - N) - T \leq 0 \\ 2 \times (Err_x) + M + (2 \times x - N) - T > 0 \end{cases} \quad (10)$$

It follows that the proposed activation function (8) is actually able to reduce the number of error occurrences if condition (9) occurs more often than condition (10). This depends on the particular value of N and T. As an example, for N=9 and T=4, the number of errors using the activation function (2) is 46, as described in the example at the end of Section IV.a. When using the proposed activation function (8) (with M=3, as for (7)), the condition (9) occurs 37 times (27, 9 and 1 for x=7, 8 and 9, respectively), whereas condition (10) occurs 27 times (only for x=6). As a consequence, the proposed activation function (8) is able to correct 10 errors over 46 (i.e. about 27%). Fig. 4 shows the error probability of the activation function (8) for the case N=9 as the value of T varies. The figure demonstrates that Eq. (8) reduces P_{error} for almost every value of T: its maximum and mean values are 25% and 8%, respectively. The error mitigation capability of Eq. (8) is able to reduce the mean error probability of the approximate function by almost 50%.

In order to further validate the proposed approach, Fig 5 shows the error probability of the activation functions (2) and (8) for a larger value of N (N=24). As expected, the proposed error mitigation scheme is able to reduce the number of errors for almost each value of the threshold $T \in [-24, 24]$: the mean error probability is reduced from 16% to only 4%.

The above analysis showed that the proposed error correction methodology drastically reduce the error probability due to the MAJ-based approximation. That suggests the possibility of using a more aggressive approximation strategy.

Fig. 3(b) describes our proposed approximation in which we use MAJs to approximate the FAs belonging to the first L levels of the adder tree, contrary to *just* $L = 1$, as done in the state of the art [7]. The value of L can be leveraged to further

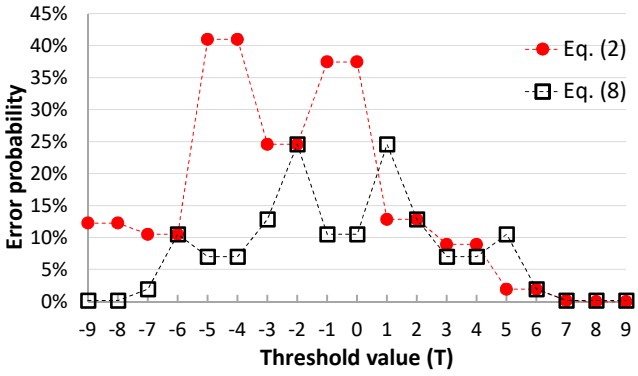


Fig. 4. Convolutional layer probability of error comparison for the case of $n = 3$ and $m = 1$. Approximating the first layer (i.e., $L = 1$) of the adder-tree Popcount through using majority gates results in a considerable error probability (see the red curve) that can be, however, mitigated (see the black curve) after using our proposed correction factor. The correction factor is employed at zero-cost through modifying the threshold value itself.

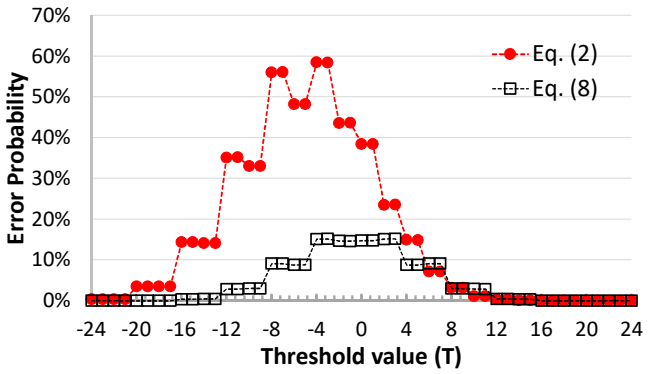


Fig. 5. Convolutional layer probability of error comparison for the case of $N = 24$. The correction factor M is 8, as calculated from Eq. (7)

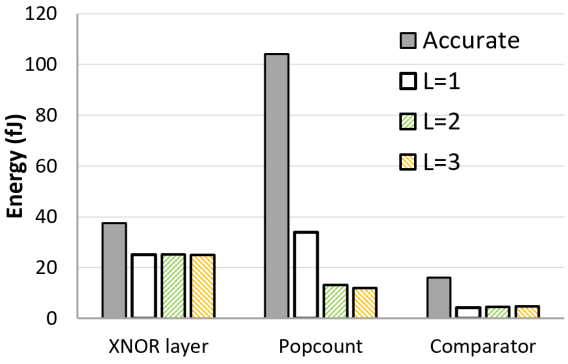


Fig. 6. Energy breakdown of the accurate and approximate convolutional layer ($n = 3$, $m = 3$) at different approximation levels of $L = 1, 2, 3$. Total energy saving: $L = 1$ →60%; $L = 2$ →72%; $L = 3$ →73%.

tradeoff the energy saving with P_{error}. For instance, Fig. 3(b) depicts the case for $L = 2$, where three 3XORs-1MAJ blocks are grouped in a macro-block whose outputs results eventually in one single bit. When creating such macro-blocks unless the approximation is applied to all the levels of the adder tree, a final accurate adder is needed, as shown in Fig. 3.b. Since the MAJ approximates the FA by computing just the carry-out bit, the inputs to the final accurate adder should be left-shifted by L bit positions and the Popcount result has at least $\lfloor \log_2 N/3^L \rfloor + 1$ non-zero bits.

It is noteworthy that the proposed approximation strategy tends to create groups with 3^L input signals (if L levels are approximated) but the size $N = n \times n \times m$ may not be a multiple of 3^L. In such a case the remaining signals can be grouped into smaller groups of size 3^P, with $P < L$. Eventually, the output of a group with 3^P inputs should be inputted to the final accurate adder, after being left-shifted by P bit positions.

Table I summarizes the results obtained for a convolutional layer of a BNN for $n = 3$ along with different values of m and L, synthesized with Cadence tool flows and mapped to a commercial 28nm FDSOI 1V process technology. The energy consumption has been analysed making use of the Value Change Dump (VCD) file extracted for 1000 sets of $N = n \times n \times m$ binary input activations and weights, at an operating frequency dictated by the corresponding accurate counterparts (case L=0). The P_{error} evaluation has been obtained considering a uniform distribution for the activations/weights in $\{-1, +1\}$, and for the thresholds in $[-N, +N]$. We have chosen $n = 3$ since this is the typical kernel size in BNNs [14]. Probabilities have been computed using gate-level simulations for 1M random inputs. Note that the proposed approach based on Eq. (8) always shows a considerably lower P_{error} in comparison with the usual activation function Eq. (2). From Table I it is worth notable that the proposed simple error mitigation technique can reduce P_{error} by up to 91% for the same energy consumption, allowing the possibility to exploit a very aggressive approximation configuration. As an example, for the case $n = 3$, $m = 64$, the proposed error mitigation methodology allows to approximate the popcount adder tree up to $L = 4$, causing an error probability of just 2.4%. On the contrary, for the same approximation configuration without any error-aware countermeasure (i.e., without error-mitigation scheme), the P_{error} is unacceptable high (about 41%). As a consequence, the Popcount circuit can be more aggressively approximated, leading to an energy saving of up to 40% and, at the same time, a P_{error} that is up to 85% lower (comparing the cases $L = 1$ and $L = 4$, with $n = 3$ and $m = 64$). It is worth noting that the proposed error mitigation scheme can be simply implemented by reducing the threshold value T by the value of M, so that it does not introduce any delay, area and energy penalty with respect to approximate design without error correction

In order to provide a thorough analysis of the proposed approximation strategy, Table II reports dynamic and static power consumptions for the accurate and proposed circuits at different approximation levels and running frequencies. At a glance, it can be noted that static power impacts for a small percentage of the total. More interesting, both the static and dynamic contributions benefit from the proposed strategy, being reduced by up to 62% and 81.7%, respectively, over the accurate design configured for $m = 9$ at 1.5GHz. Table II also gives an information on how the power consumption scales when the proposed circuits are used within BNN implementations for IoT applications. Indeed, in such a case, the working frequency required by the hardware is very low (i.e. in the order of a few MHz [15]), which allows drastically reducing the overall power cost. Results obtained from synthesis and VCD files generated at the 1MHz frequency demonstrate an appreciable reduction of the power consumption through the proposed strategy also at these operating conditions.

Fig. 6 shows the energy breakdown of the convolutional layer for the case $n = 3$, $m = 3$ and $L = 1, 2, 3$ (as from Table I, the circuits have been synthesized for a clock frequency of 2GHz). As expected, the Popcount operation accounts for most of the total energy (66% for the accurate implementation). The approximate versions with $L = 1$ and $L = 2$ reduce the Popcount energy by 66% and 87%, respectively, whereas, for $L = 3$, the energy saving does not change significantly because there is only one more FA replaced with a MAJ. The energy consumption of the XOR layer is the same regardless of the approximation level because the number of XORs does not change. Nevertheless, this energy is 32% lower compared with the accurate implementation. This is mainly due to the fact that, in the accurate implementation,

the output of each XOR is inputted to a FA, whose input capacitance is higher in comparison with a MAJ. Finally, it is worth noting that L does not significantly affect the energy consumption of the comparator. Indeed, such a component is $(\lfloor \log_2 N \rfloor + 1)$-bit sized, regardless of the approximation level. On the contrary, since the proposed approximation introduces a determistic number of zero bits on the Popcount result, an energy reduction of 71% is appreciated with respect to the comparator used within the accurate design. In conclusion, the energy analysis reported in Fig. 6 and Table I reveals the following interesting property: as the number of the approximate levels L within the Popcount circuit increases, the total energy consumption decreases. Anyway, the rate at which energy decreases is not constant but it is inversely proportional with L and, eventually, it becomes roughly zero for a certain value of L. As an example, this can be appreciated in Table I, for the case m=3 and L increasing from 2 to 3, or for the case m=9 and L increasing from 3 to 4. As a result, depending on the number m of the input channels, there is a particular value of approximate levels L beyond which there is no energy advantages, but only an increase of the error probability. It is also worth noting that the energy/error analysis of Table I has a general validity regardless the particular stride value adopted in the BNN. Indeed, the energy of the convolutional engine depends only on n, m and L, whereas the error probability depends on L.

The presented analysis also demonstrates that including a simple correction factor M on top of the existing activation function (i.e., modifying the threshold value) leads to a substantial reduction of the number of errors caused by the adopted majority-based approximation strategy. Consequently, based on its size and desired approximation level, a binary convolutional layer can be designed connecting the appropriate approximate XOR-MAJ blocks.

The challenge in obtaining the correction factors: Despite the application easiness of the proposed strategy, the optimal choice of the correction factor M might not be straightforward because it depends on the activations and weights distributions. Indeed, for the sake of the proof-of-concept study, the previously-described analysis has been based on the assumption that both the activations and weights are uniformly distributed. In actual/realistic BNNs, such a hypothesis is generally not true. One may still try to model the actual weights distribution (obtained after the BNN training) to be used for the convolutional engine offline simulation and for the estimation of M. However, such an approach may be unfeasible and/or timing consuming. Moreover, modeling the activations distribution may be even more difficult since, differently from the weights, activations are often unknown in prior. Hence, distribution of activations are hard to be assumed.

Our FPGA-based solution to search and obtain the optimal correction factors: For all these above-mentioned reasons, we propose to find the optimal correction factor for each layer of the BNN through design-space exploration. It consists to make use of the $inference$ BNN model whose layers can be easily modified including the intended approximation level and a parameter M in the activation function. The inference BNN model is then run on a subset of the training data set varying M within a reasonable range: the optimum value of M is then selected as the one assuring the maximum BNN accuracy. Such a procedure is performed by applying the approximation to one layer at a time. In this way, it can be possible to perform a layer-wise analysis of the impact of the approximation on the net accuracy. It will be demonstrated that the optimal correction factor found for each layer is substantially the same as the one obtained by performing the proposed search simultaneously considering the approximation on all the layer of the BNN. In order to speed-up the described searching procedure, the inference model of the BNN has been deployed onto an heterogeneous System on Chip (SoC), where the most timing consuming tasks, i.e. convolutions, have been mapped in hardware on the FPGA section of the SoC. After finding the optimal M values, it can also be possible to extract a P_{error} model for each layer and for the adopted approximation, which will be later employed in our HW/SW codesign approach (details in Section VI) towards constructing efficient, yet robust approximation-aware BNN.

TABLE I
HARDWARE/ACCURACY ANALYSIS OF OUR APPROXIMATION. DELAY, ENERGY AND AREA RESULTS ARE OBTAINED USING CADENCE TOOL FLOWS FOR A COMMERCIAL 28NM FDSOI TECHNOLOGY.

n	m	L	Delay (ps)	Energy per op.[+](fJ)	Norm. EDP	Area[+] (μm^2)	Err. Prob. Eq.(8)(%)	Err. Prob. Eq.(2)(%)
3	1	0	350	51	1	99.8	0%	0%
3	1	1	260	21.1	0.41	83	8%	15.4%
3	1	2	240	15.2	0.3	80.7	12.3%	25.7%
3	3	0	500	157	1	109	0%	0%
3	3	1	400	63	0.4	65.8	5.1%	18.7%
3	3	2	350	43	0.27	51.9	6.2%	26%
3	3	3	300	41	0.26	49.6	9.1%	34%
3	9	0	650	533	1	342	0%	0%
3	9	1	555	228	0.42	200	2.3%	15.4%
3	9	2	460	158	0.29	168	3.6%	28.7%
3	9	3	330	99	0.18	135	5.1%	33.9%
3	9	4	260	97	0.18	132	7.2%	40.6%
3	64	0	950	5396	1	3230	0%	0%
3	64	1	920	1900	0.35	1492	1.1%	15.8%
3	64	2	830	1273	0.23	1159	1.2%	26%
3	64	3	724	1045	0.19	1027	2.2%	36%
3	64	4	500	921	0.17	957	2.4%	41.8%

[+] Obtained when synthesized for the minimum delay of the accurate implementation.

TABLE II
POWER ANALYSIS @1V SUPPLY VOLTAGE

n	m	L	Static[+](μW)	Dynamic[+](μW)	Static (μW) @1MHz	Dynamic (μW) @1MHz
3	1	0	0.0405	145	0.035	1.37
3	1	1	0.021	60.3	0.019	0.62
3	1	2	0.018	45.3	0.012	0.42
3	3	0	0.12	315	0.073	5.15
3	3	1	0.07	126	0.044	2.2
3	3	2	0.053	86	0.034	1.32
3	3	3	0.05	83	0.032	1.14
3	9	0	0.37	820	0.225	19.11
3	9	1	0.2	352	0.134	8.3
3	9	2	0.17	244	0.108	4.9
3	9	3	0.14	152	0.104	3.6
3	9	4	0.14	149.6	0.103	3.4

[+] Obtained when synthesized for the maximum running frequency of the accurate implementation (2.85GHz for m=1, 2GHz for m=3, 1.5GHz for m=9).

V. FPGA ARCHITECTURE AND DESIGN FLOW

A. FPGA-based SW/HW Codesign System Architecture

In order to analyze the proposed correction strategy, applied to the described approximation, we designed a complete embedded system that is able to run BNN models. For our experiments, we use a VGG-based BNN along with the

TABLE III
BNN ARCHITECTURE. "IN", "C", AND "FC" REFER TO INPUT LAYER, CONVOLUTION LAYER, AND FULLY-CONNECTED LAYER, RESPECTIVELY.

BNN Model	BNN Architecture
FashionMNIST BNN	In $\rightarrow$ C64 $\rightarrow$ MP2 $\rightarrow$ C64 $\rightarrow$ MP2 $\rightarrow$ FC2048 $\rightarrow$ FC10

TABLE IV
DATASET USED FOR EXPERIMENTS.

Dataset	# Train	# Test	# Dim	# Classes
FashionMNIST	60000	10000	(1,28,28)	10

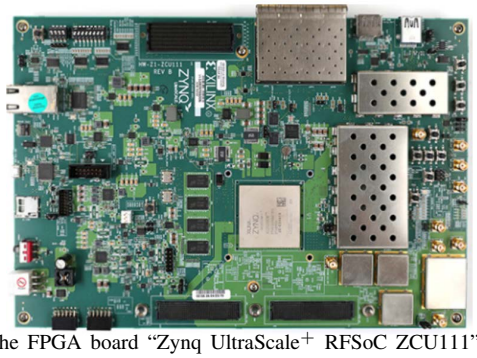


Fig. 7. The FPGA board "Zynq UltraScale$^+$ RFSoC ZCU111" Evaluation KitsBoard, which we have employed in our implementation. Details on the implemented HW/SW codesign is presented in Figs. (7 and 8).

FashionMNIST dataset. In practice, the constructed BNN is a modified version of the VGG [16], adapted for the image sizes in FashionMNIST.

The details of the dataset and BNN architecture are presented in Table III and Table IV, respectively. The BNN uses convolutional (C) layers with size 3×3, fully connected (FC) layers, maxpool (MP) with size 2×2, and batch normalization (BN) layers followed by binary activation functions. We use a batch size of 256 and an initial learning rate of 10^{-3} for all cases. We halve the learning rate every 25^{th} epoch. For the optimizer, we run Adam, similar to [4]. We train the BNNs for 200 epochs in each case.

After training using our PyTroch (modified to support BNN training), the obtained BNN model is then executed on an SoC implemented on the Xilinx Zynq UltraScale$^+$ RFSoC ZCU111 board depicted in Fig. 7. It is furnished with an ARM Cortex A53 Processing System(PS) and UltraScale$^+$ Programmable Logic (PL). Fig. 8 describes the HW/SW architecture. The SoC is augmented with our customized HW that realizes the BNN convolution layer and BNN fully-connected layer for both accurate and approximate (at different levels of L) scenarios. The PyTorch framework is executed on the PS (i.e. the ARM core), which runs a Linux OS kernel. Correspondingly, we have developed the custom HW IPs for Convolutional and Fully-Connected layers on PL. The communications between the ARM CPU and the custom HW IPs are performed through an Advanced eXtensible Interface (AXI). Our FPGA setup allows us to seamlessly execute any trained BNN and evaluate the impact of HW approximation on inference accuracy.

B. Validating our FPGA-based BNN implementation

First, we validate the functionality of our system by implementing exact hardware accelerators on the PL and by running the inference phase on the trained model. Further, we compare the inference accuracy and all tensor outputs obtained from our setup against the corresponding accuracy and tensor outputs obtained from the golden PyTorch (i.e., a pure SW-based BNN execution). This analysis showed an identical matching, which proves the correct functinality of our implemented FPGA system. Then, we modify the hardware accelerator on the PL to implement our approximation and to analyze its impact on the accuracy. Finally, by comparing the tensor outputs of the approximate BNN with the exact ones, we derived the P_{error} model for each layer and for the adopted approximation: it will be exploited later to further increase the BNN accuracy. Fig. 9 depicts the FPGA-based HW/SW co-design system flow. The input and output layers are purely executed on the FPGA's ARM CPU, and all the intermediate layers are performed between the ARM CPU and FPGA fabric. Based on our design flow in Fig. 9, batch normalization and quantized activation happen on the FPGA's ARM CPU, and the convolutional and Fully Connected logic of the proposed approximated circuit is laid out on the FPGA fabric.

C. Searching and finding the optimal correction factors using our FPGA-based implementation

As described earlier, the proposed error-mitigation scheme is based on adding a correction factor M to the left-side of Eq. (2), whose value is dependent on the weights and activations distributions. Hence, we need a value of M for each convolutional and fully-connected layers of the BNN. In this work, we propose to perform a layer-wise research by running the inference of the BNN model on a subset

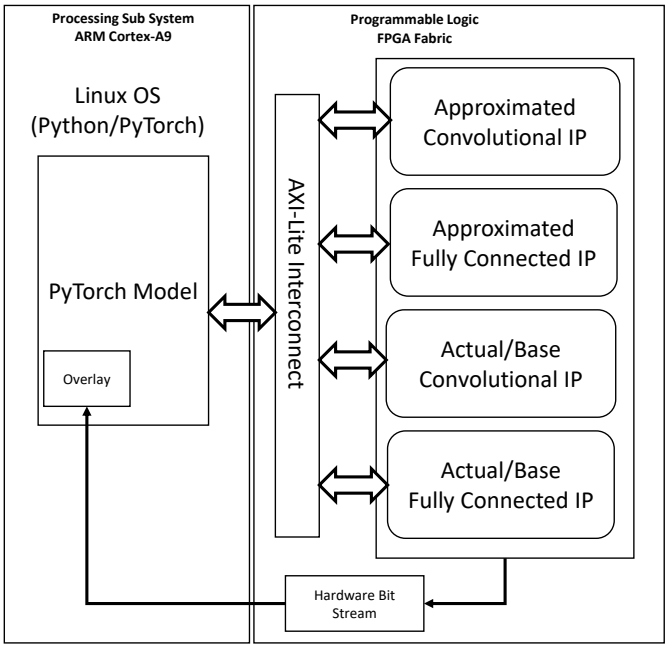


Fig. 8. Overview block diagram demonstrating our FPGA-based architecture.

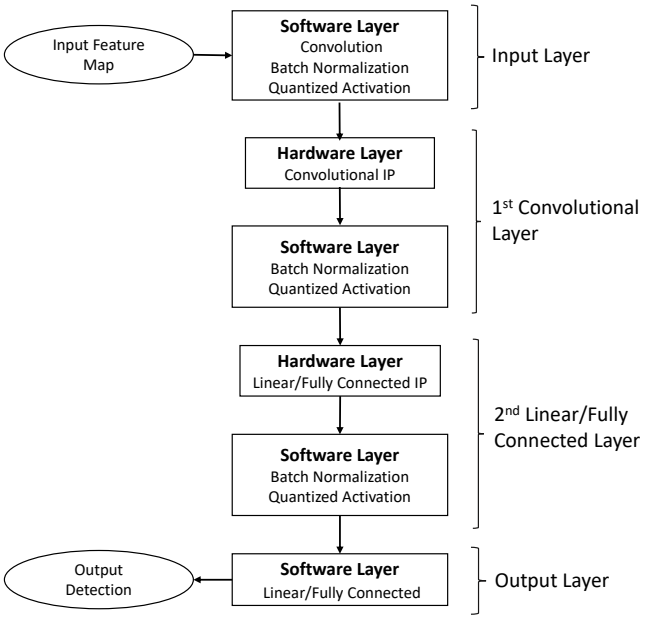


Fig. 9. Our implemented FPGA-based system to realized the proposed HW/WS codesign for BNN. The FPGA platform is employed to evaluate the BNN accuracy drop under different approximation levels, with/without correction factor and with/without error-aware training. The FPGA board used in this implementation is shown in Fig. 7.

of the training set inputs and to then analyze the obtained accuracy drop induced by the approximation just applied to one layer at the time. Such a procedure is repeated by varying the value of M related to each convolutional and fully-connected layers within a reasonable range. Eventually, the optimum values of the correction factors are those assuring the maximum accuracy result. Towards this aim, we made use of the designed embedded system to speed-up the intended procedure. Each BNN layer has been described in VHDL for different approximation levels, so that an hardware IP has been generated for each value of L. After that, the IPs related to the approximation to be investigated have been deployed onto the FPGA-based PL. Since the thresholding operation runs on the PS, the operation of varying the value of M has been straightforwardly performed by subtracting in software the value of M from the threshold, as it can be easily inferred from Eq. (8). Fig. 10 depicts the obtained results. The x-axis of each subplot represents the values of M used in the searching process, with a layer-wise adoption of the proposed approximation strategy, whereas the y-axes indicates the obtained inference accuracy. The value of M has been increased with a variable step in order to speed-up the searching procedure: we used a coarse step (20 units) when the accuracy was below 70%, a finer one (2 units) otherwise. The accuracy has been evaluated over one hundred inputs, randomly picked from the training set. It is worth noting that the accuracy always shows a maximum value for a particular value of M, that is selected as the optimum correction factor to be used in the proposed error mitigation scheme. It is noteworthy that when approximating a layer without error correction (i.e. M=0), the BNN accuracy is essentially null. This is in accordance with the results of Table I, where it has been shown that (2) has a much higher error probability than (8). As a final remark, the described procedure enables us also find the P_{error} model for each BNN layer by comparing the layer outputs of the approximate BBN with those related to the BNN without any approximation obtained from the "golden" run. Such obtained P_{error} model can be then employed in the proposed HW/SW codesign to further optimize the BNN, as will be described in the next Section.

VI. HW/SW Codesign for BNN Optimization

The optimum correction factors M have been integrated in the embedded system which has been used to run the inference phase with the 10,000 input images of the FashionMNIST dataset. In conjunction with the error mitigation scheme, we also investigated the possibility to increase the inference accuracy by means of a simple retraining strategy. To that end, we employed a framework based on PyTorch [6]. To model the applied HW approximation, we modify PyTorch to allow error injection into the activation values of each layer, according to the P_{error} model found as previously described. In such a way, there is no need to implement the actual approximation function within PyTorch, thus maintaining the excellent optimizations that PyTorch offers. In addition, this also allows to train the BNNs in the presence of the errors expected from the underlying HW when it is being approximated, without modifying the PyTorch code of the backward propagation. Using our built setup, we investigate the following scenarios.
(i) The SW is *unaware* of HW approximation: Here, the BNN is trained in the *absence* of any error and hence the SW level is *unaware* of the existing approximation in the HW level. After training, the trained parameters are fed to the embedded system to run the inference phase and eveluate the impact of the HW approximation on the accuracy.
(ii) The SW is *aware* of HW approximation: Here, the BNN is trained in the *presence* of induced errors. Hence, the SW level is now *aware* of the existing approximation in the HW level. During the BNN training, the corresponding P_{error} capturing the applied HW approximation is employed. For error-aware training, we use the modified hinge loss to train the BNNs, with the hyperparameter $b = 128$ [17]. After the error-aware retraining, the BNN is expected to be more robust against the errors stemming from the approximate HW.
(iii) The HW is *aware* of SW Sensitivity: Here, the approximation in the underlying HW is *selectively applied for every BNN layer* towards maximizing the energy saving for a certain accuracy drop budget. To achieve this goal, we perform a layer-wise analysis to determine the sensitivity of every BNN layer against errors. This, in turn, enables us to determine the appropriate level of approximation that can be tolerated for each particular layer. If a layer shows high resilience to errors, then a more aggressive approximation can be applied. Otherwise, a light approximation can be applied.

VII. Evaluations and Comparisons

A. Impact of Approximation on BNN Accuracy

Towards comprehensively investigating the impact of our proposed approximation-aware technique, we first analyzed the inference accuracy obtained when the approximation is

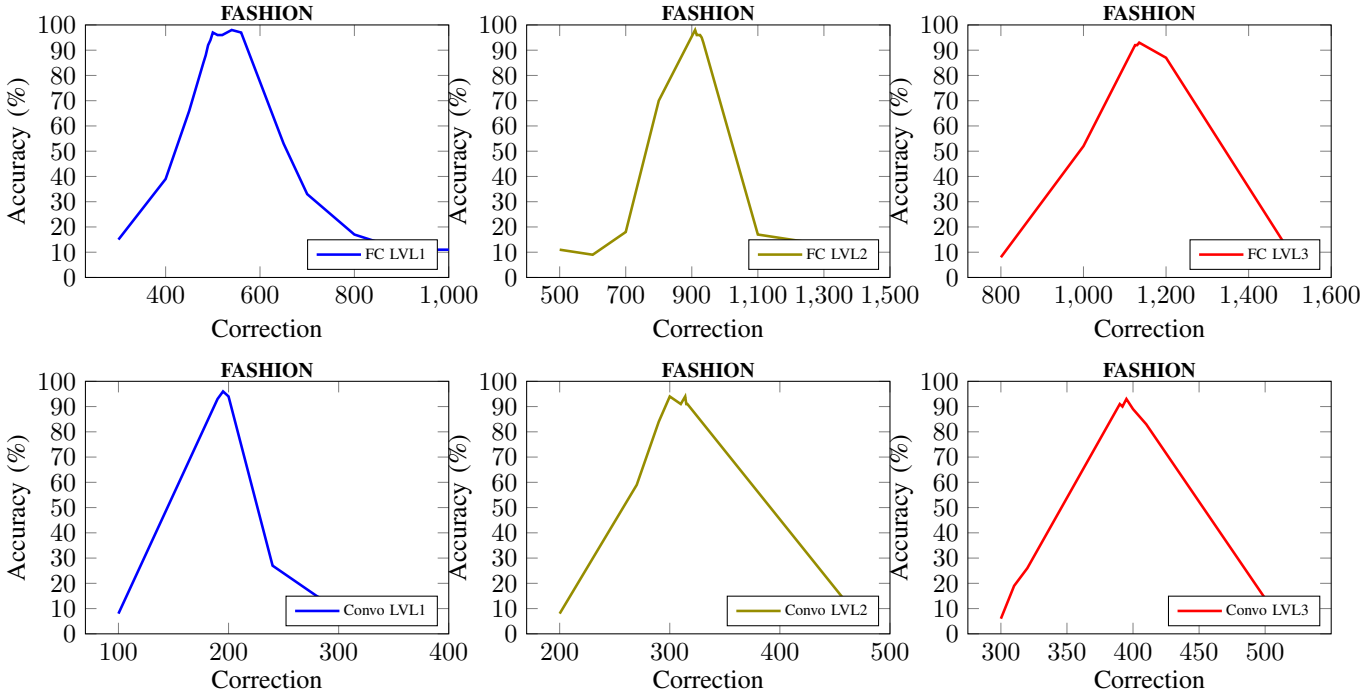


Fig. 10. The optimum correction factors searching procedure on a subset of input images within the training data set, for different approximation levels in the CV and FC layers. The optimum values of M are those leading to the maximum BNN accuracy.

TABLE V
HW/SW CO-DESIGN ACCURACY FOR 10000 TEST IMAGES.

Cases	HW Convolutions IP	HW Fully Connected IP	Convolution CF	Fully Connected CF	Test Accuracy	Test Model
1	Base Line Hardware	Base Line Hardware	NA	NA	89.49%	Without Error Train
2	Approx Level1 Hardware	Base Line Hardware	195	NA	86.40%	Without Error Train
3	Approx Level2 Hardware	Base Line Hardware	315	NA	85.27%	Without Error Train
4	Approx Level3 Hardware	Base Line Hardware	395	NA	82.88%	Without Error Train
5	Base Line Hardware	Approx Level1 Hardware	NA	495	87.00%	Without Error Train
6	Base Line Hardware	Approx Level2 Hardware	NA	920	87.31%	Without Error Train
7	Base Line Hardware	Approx Level3 Hardware	NA	1130	87.12%	Without Error Train
8	Approx Level2 Hardware	Approx Level2 Hardware	315	930	80.00%	Without Error Train
9	Approx Level2 Hardware	Approx Level2 Hardware	315	930	85.18%	With Error Train
10	Approx Level2 Hardware	Approx Level2 Hardware	318	930	84.90%	With Error Train
11	Approx Level2 Hardware	Approx Level3 Hardware	315	1140	81.41%	Without Error Train
12	Approx Level2 Hardware	Approx Level3 Hardware	315	1140	84.71%	With Error Train

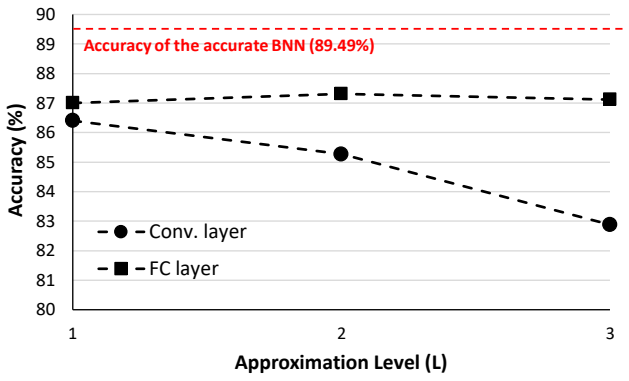


Fig. 11. BNN accuracy sensitivity vs. the approximation level applied to the convolutional and fully-connected layers for our proposed error-mitigation scheme.

layer-wise applied in conjunction with the optimum correction factors M, using the 10,000 images of the test dataset. The obtained results are collected in Table V, from row 2 to row 7. An approximation spanning from the 1^{st} to the 3^{rd} level of the popcount tree has been analyzed for each layer. For each approximation configuration, the relative correction factor M has been integrated in the correspondent layer. It is evident that the simple adoption of M allows a drastic reduction of the inference accuracy drop. This is more underlined in Fig. 11, where the accuracy sensitivity vs. the adopted approximation is plotted for each layer. It is worth noting that the maximum accuracy drop is of about 6.5%. Moreover, the inference accuracy sensitivity to the approximation level is higher in the convolutional layer than in the fully-connected layer. Indeed, in the latter layer, the accuracy is almost constant regardless the approximation level of the popcount tree. This result is in accordance with the preliminary analysis summarized in Table I, where it is evident that, for the proposed technique, the accuracy drop is inversely proportional to the layer size at the parity of the adopted approximation level. Finally, the analysis of Fig. 11 suggests that it is preferable to adopt a more aggressive approximation to the deepest layers of the

BBN, whereas a moderate approximation can be applied to the first layers. In order to verify that the searching procedure actually produces the optimum values of M for each layer, the same procedure has been carried out on the images belonging to the test dataset. Hence, we replicated the analysis shown in Fig 10 for a subset of the inputs within the training set, and the obtained results are plotted in Fig. 12. It is interesting to note that the actual optimum values of M found using the test images are identical to those obtained using a subset of the training images. This proves that the proposed searching procedure is effective in obtaining the optimum values of the correction factors. As a further evidence of the effectiveness of the proposed error mitigation scheme, the inference accuracy drops to only 10% when M is set to 0 (i.e. when the approximation is applied without any countermeasure as for (2)). Table V also collects the inference accuracy when the approximation is applied to more than one layer simultaneously. We applied approximation until levels 1-2 and levels 1-3 to the convolutional and fully-connected layer, respectively. Interestingly, the optimum values of M are practically identical to those obtained when the approximation is applied to one layer at the time. As an example, the obtained values of M for the convolutional (fully-connected) layer is 315 (930) when an approximation until the level 2 is applied and the searching procedure is performed by approximating just one layer at the time. Such a scenario is reported in Table V as *case 9*. A two-dimensional searching procedure has also been carried out by applying the approximation simultaneously to both the layers and let the two correction factors (one for each layer) to vary at the same time. Results are collected in Table V as *case 10*. It is worth noting that the obtained values of M are practically the same than those related to *case 9*, same thing for the inference accuracy. Moreover, results in Table V clearly shows that the accuracy improves when the error mitigation technique is applied in conjunction with the proposed retraining methodology, thus confirming the effectiveness of the proposed HW/SW co-design optimization.

For the sake of comparison, we investigate different scenarios and compare them to the state-of-the-art technique in BNN approximation and to the golden reference:

Reference: The golden baseline accuracy to compare against. Both BNN training and inference are performed without any errors (i.e., no approximation is applied).

No error corr.: An HW approximation is applied only to the first level (i.e., $L = 1$) in the adder tree of the Popcount circuit as proposed in [7] to the convolutional and fully-connected BNN. The BNN is trained without errors injection.

Case1 [our]: An HW approximation is applied until level 2 (level 3) in the adder tree of the Popcount for the convolutional (fully-connected) layer, along with our error-mitigation scheme as illustrated in Eq. (8).

Case2 [our]: This is the same as in Case1 scenario, but the BNN is retrained in the presence of errors using the corresponding P_{error} calculated from the induced approximation.

Fig. 13 demonstrates the inference accuracy for the two cases above, the case with no error correction and just one level of approximation in the popcount ([7]), and the reference. As shown, without any error mitigation technique, the BNN accuracy largely (over 70%) drops. When our approximations in case1 is applied (i.e., no error-aware training is yet used), a much higher BNN accuracy is achieved (81.41%). Finally, after employing error-aware training, our approximation-aware BNN provides, in case2, an inference accuracy of 84.71%. In short, unlike the technique described in [7], that compulsorily requires a complex retraining procedure based on the PyTorch framework modification, the accuracy drop in our approximation-aware BNN is merely 4.8%, for a more aggressive approximation (L=2 and L=3 for the convolutional and fully-connected layers, respectively).

TABLE VI
ENERGY CONSUMPTION (NJ) FOR THE LAYERS OF THE ANALYZED BNN AT DIFFERENT APPROXIMATION LEVELS (L1-L3).

VGG-based BNN				
Approx. Level:	**Accurate**	**L=1**	**L=2**	**L=3**
Layer 2 (C64)	58.3	20.5	13.7	-
Layer 3 (FC2048)	31	9.4	6.51	4.06

TABLE VII
VGG3-BASED BNN ENERGY CONSUMPTION FOR THE SIX ANALYZED SCENARIOS DESCRIBED IN FIG. 13.

Scenario	**Energy(nJ)**	**Accuracy**
Reference	89.3	89.49%
State of the art [7]	29.9	10%
Case 1 [our]	17.7	81.41%
Case 2 [our]	17.7	84.71%

B. Impact of Approximation on BNN Energy

To evaluate energy, the layers of the VGG-based analyzed BNN have been synthesized with Cadence Genus and mapped to a commercial 28nm FDSOI technology. For the synthesis of the approximate designs, the timing constraint has been set to the minimum delay achievable by the accurate design.

Tab. VI provides examples for the obtained mean energy values for different layers and different approximation levels. As stated in Fig. 11, the convolutional layer is more sensitive to the approximation, so we set its maximum approximation L=2. Conversely, the fully-connected layer is more robust to the errors induced by the proposed approximation and, hence, we analyzed the impact on the energy for a deeper approximation level (i.e. L=3). It can be noted from Table VI that the energy consumption per layer decreases as the approximation level increases. For instance, the energy saving for the convolutional layers can be up to 77%, and even higher energy reduction is observed in the fully-connected layers.

Tab. VII summarizes the energy consumption of the VGG-based BNN for the four different scenarios described in Fig. 13 (see Sec. VII-A). Our approximations (Case 1-2) reduce the energy by 80% compared to the reference (i.e., accurate BNN) and outperform the state of the art [7]. For instance, Case 1 achieves a considerably higher accuracy (81.41% versus a 10%) along with a 41% lower energy. When error-aware BNN training is applied (i.e., case 2), the accuracy largely increases to 84.71%. *In short, our approximation-aware BNN provides* 80% *less energy with a marginal drop (*4.7%*) in accuracy.*

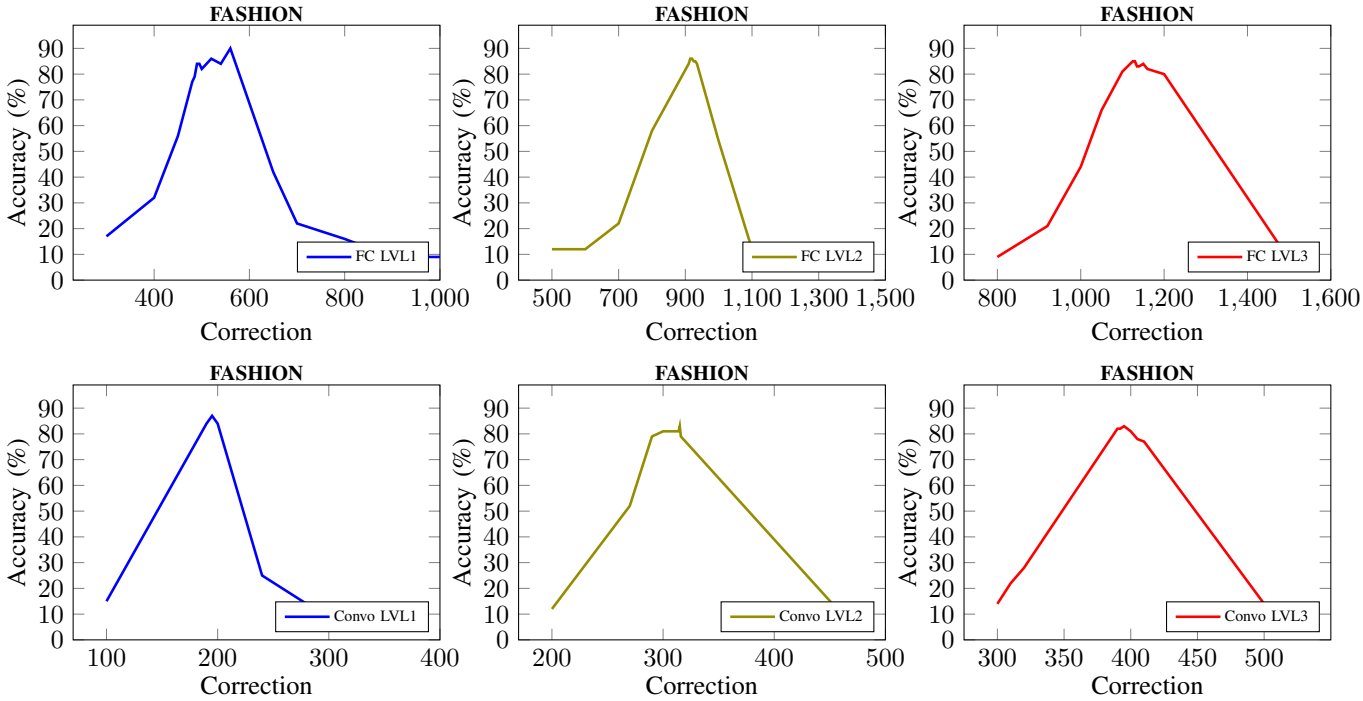


Fig. 12. BNN inference accuracy over the 10,000 input images of the testing data set for different values of the correction factors M and approximation levels in the CV and FC layers.

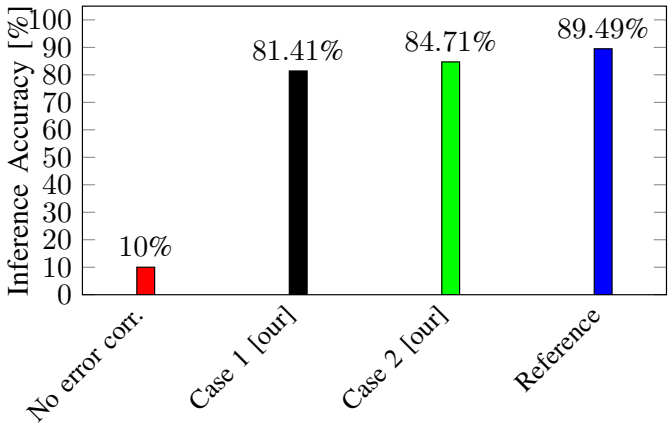


Fig. 13. BNN Inference accuracy of the FashionMNIST for our four different approximation cases (see Sec. VII-A), compared to the case where no error correction scheme is employed (e.g., [7]) and the reference accuracy.

C. Analysis of the proposed methodology on a larger BNN

The applicability of the proposed methodology has been tested also on a larger BNN. In particular, we used a VGG7-based BNN with the CIFAR10 dataset [16]. It is a modified versions of the VGG-architectures, adapted for the image sizes in the above dataset, which is a suitably sized example for resource and power constrained inference [16]. Table VIII summarizes the BNN topology. Fig. 14 depicts the application of the above described optimum correction factor searching procedure on the VGG7-based BNN, with L=1 for each layer, and the optimum values of M are tabled in Table IX. When the approximation is applied to all the layers (each with its own correction factor), the BNN reaches an accuracy of 71% (without retraining), that is just 4% lower than the accuracy the original net (75%). As a remarkable advantage, the proposed approximation can reduce the energy dissipation of about 64.8% when running the VGG7-based BNN with with L=1.

As a final remark, from Fig. 14 it can be noted that the deeper is the layer of the BNN, the lower is the accuracy sensitivity w.r.t. the value of M. This is inline with the results obtained in Section VIII.a for the smaller BNN, and also with recent works where it has been shown that, in conventional CNN, the deeper is the accumulation tree of the MAC operation, the higher is the possibility to apply more aggressive approximation schemes while maintaining an acceptable accuracy [18].

TABLE VIII
VGG7-BASED BNN ARCHITECTURE.

BNN Model	BNN Architecture
VGG7-based BNN	In → C128 → MP2 → C256 → C256 → MP2 → C512 → C512 → MP2 → FC1024 → FC10

TABLE IX
OPTIMUM VALUES OF M FOR THE VGG7-BASED BNN

VGG-based BNN	
BNN layer:	M
Layer 1 (C128)	390
Layer 2 (C256)	365
Layer 3 (C256)	785
Layer 4 (C512)	760
Layer 5 (C512)	1530
Layer 6 (FC1024)	2790

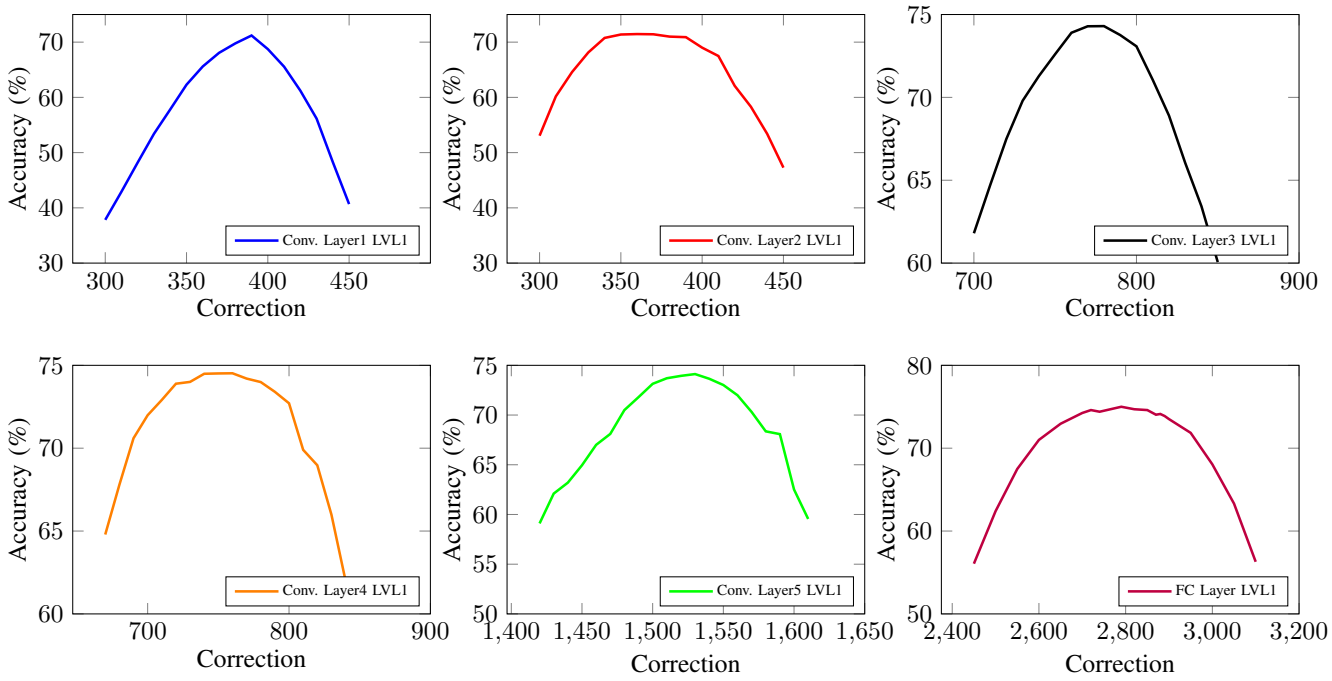


Fig. 14. The optimum correction factor tuning and obtaining procedure for the VGG7-based BNN.

VIII. Summary and Conclusion

BNNs are rapidly emerging as an attractive methodology for ultra-lightweight neural networks. In this work, we demonstrated that the Popcount circuit forms a bottleneck that seriously hampers the efficiency gains promised by BNNs. State of the art merely approximates the first layer in the adder tree that constructs the Popcount, yet results in a severe drop in the inference accuracy unless model retraining is performed. On the contrary, we apply more aggressive approximations covering more layers in the adder tree, leading to a higher energy saving (80.2%), while still achieving significantly larger accuracy (84.71%) akin to our carefully-designed error-mitigation scheme.

Our methodology was evaluated using a commercial 28nm technology and validated using an FPGA-based SoC. We have shown how our FPGA-based system enables designers to search the space towards obtaining the optimal correction factors required for error-mitigation scheme. Further, the FPGA-based system also enables use to compute abstracted probability model for the errors induced by approximation. Such error probabilities are later employed to perform an error-aware BNN training that further optimize the BNN and further minimize the accuracy loss. Most notably, the proposed HW/SW codesign approach allows for a easy retraining of the BNN that does not require any modification of the existing software design tools.

All in all, HW/SW codesign is a key to construct approximation-aware BNNs that are considerable more efficient, yet robust against errors. In future work, we plan to extend our analyses for other, larger BNN models that feature different layer types (such as depthwise separable convolution) and hyperparameters (such as stride and convolution window size). Such additional experiments require to evaluate anew the trade-offs regarding level of approximation and accuracy.

Acknowledgment

This work was supported by Deutsche Forschungsgemeinschaft (DFG) project OneMemory (405422836), by the DFG project ACCROSS (428566201), by the Collaborative Research Center SFB 876 “Providing Information by Resource-Constrained Analysis” (project number 124020371), subproject A1 (http://sfb876.tu-dortmund.de) and in part by PON Ricerca Innovazione - MUR (grant 062_R24_INNOVAZIONE), Ministero dellUniversit e della Ricerca, Italian Government.

References

[1] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, *et al.*, “In-datacenter performance analysis of a tensor processing unit,” in *Int. Symp. Computer Architecture*, pp. 1–12, 2017.

[2] H. Amrouch, G. Zervakis, S. Salamin, H. Kattan, I. Anagnostopoulos, and J. Henkel, “Npu thermal management,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 11, pp. 3842–3855, 2020.

[3] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, “How to evaluate deep neural network processors: Tops/w (alone) considered harmful,” *IEEE Solid-State Circuits Magazine*, vol. 12, no. 3, pp. 28–41, 2020.

[4] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, “Binarized neural networks,” in *Advances in Neural Information Processing Systems* (D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, eds.), vol. 29, Curran Associates, Inc., 2016.

[5] F. Conti, P. D. Schiavone, and L. Benini, “Xnor neural engine: A hardware accelerator ip for 21.6-fj/op binary neural network inference,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2940–2951, 2018.

[6] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, “Pytorch: An imperative style, high-performance deep learning library,” in *Advances in Neural Information Processing Systems 32*, pp. 8026–8037, 2019.

[7] S. Rasoulinezhad, S. Fox, H. Zhou, L. Wang, D. Boland, and P. H. Leong, “Majoritynets: Bnns utilising approximate popcount for improved efficiency,” in *2019 International Conference on Field-Programmable Technology (ICFPT)*, pp. 339–342, 2019.

[8] L. Yang, D. Bankman, B. Moons, M. Verhelst, and B. Murmann, “Bit error tolerance of a cifar-10 binarized convolutional neural network processor,” in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, 2018.

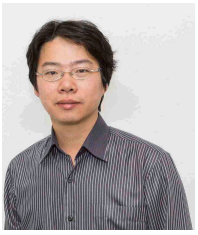
[9] M. Yayla, S. Buschjager, A. Gupta, J.-J. Chen, J. Henkel, K. Morik, K.-H. Chen, and H. Amrouch, "Fefet-based binarized neural networks under temperature-dependent bit errors," *IEEE Transactions on Computers*, pp. 1–14, 2021.
[10] V. Mrazek, Z. Vasícek, L. Sekanina, M. A. Hanif, and M. Shafique, "ALWANN: automatic layer-wise approximation of deep neural network accelerators without retraining," in *Proceedings of the International Conference on Computer-Aided Design, ICCAD 2019, Westminster, CO, USA, November 4-7, 2019* (D. Z. Pan, ed.), pp. 1–8, ACM, 2019.
[11] M. Alioto, V. De, and A. Marongiu, "Energy-quality scalable integrated circuits and systems: Continuing energy scaling in the twilight of moores law," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 8, no. 4, pp. 653–678, 2018.
[12] M. Pashaeifar, M. Kamal, A. Afzali-Kusha, and M. Pedram, "A theoretical framework for quality estimation and optimization of dsp applications using low-power approximate adders," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 66, no. 1, pp. 327–340, 2019.
[13] G. E. Andrews, "The theory of partitions," in *Encyclopedia of Mathematics and its Applications, Vol. 2, Addison-Wesley Publishing Co., Reading, MA-London-Amsterdam*, 1976.
[14] S. Liang, S. Yin, L. Liu, W. Luk, and S. Wei, "Fp-bnn: Binarized neural network on fpga," *Neurocomputing*, vol. 275, pp. 1072–1086, 2018.
[15] B. Liu, H. Cai, Z. Wang, Y. Sun, Z. Shen, W. Zhu, Y. Li, Y. Gong, W. Ge, J. Yang, and L. Shi, "A 22nm, 10.8 μw/15.1 μw dual computing modes high power-performance-area efficiency domained background noise aware keyword- spotting processor," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 12, pp. 4733–4746, 2020.
[16] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations, ICLR*, 2015.
[17] S. Buschjäger, J.-J. Chen, K.-H. Chen, M. Günzel, C. Hakert, K. Morik, R. Novkin, L. Pfahler, and M. Yayla, "Margin-maximization in binarized neural networks for optimizing bit error tolerance," *DATE '21*.
[18] B. Liu, Z. Wang, X. Wang, R. Zhang, A. Xue, Q. Shen, N. Xie, Y. Gong, Z. Wang, J. Yang, and H. Cai, "An efficient bcnn deployment method using quality-aware approximate computing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 11, pp. 4217–4228, 2022.

Abhilasha Dave is currently pursuing a Ph.D. degree at the chair "Semiconductor Test and Reliability (STAR)" in the University of Stuttgart, Germany under the supervision of Prof. Hussam Amrouch. Her Ph.D. research focuses on Approximate Computing for Deep Neural Network Acceleration. She received her Master's degree in Computer Engineering at California State University, Fresno, USA, with a thesis focused on the application of machine learning in digital circuit verification and testing. During her Masters's degree, she was nominated for the Dean's medalist award at the Electrical and Computer Engineering Department. After her Master's degree, for almost 2 years, she worked as a R&D FPGA Engineer in Silicon Valley.

Fabio Frustaci (M'14-SM'22) is an Associate Professor with the Computer Science, Electronics, Modeling and Systems Department at the University of Calabria, Rende, Italy. He received the M.S. and the Ph.D. degree in electronic engineering from the University Mediterranea of Reggio Calabria, Italy, in 2003 and 2007, respectively. In 2006, he was a Visiting Scholar at the ECE Department of the University of Rochester, Rochester, NY. In 2011-2013 he was a Visiting Researcher at the EECS Department of the University of Michigan, Ann Arbor, MI. He has authored more than 70 papers in the field of VLSI design. Currently, he is a member of the editorial board of Microelectronics Journal. His research interests include low power and high performance VLSI circuits, design techniques for emerging technologies, reconfigurable architectures, embedded systems.

Fanny Spagnolo (M'20) is an Assistant Professor with the Computer Science, Electronics, Modeling and Systems Department at the University of Calabria. She received the masters degree in Electronics Engineering and the Ph.D. degree in Information and Communication Technologies from the University of Calabria, Italy, in 2016 and 2019, respectively. In June 2016, she won a Research Grant funded by the Department of Informatics, Modeling, Electronics and System Engineering, University of Calabria. She has co-authored of more than 30 papers in the field of VLSI design. Her research interests include embedded systems design, VLSI architectures for image processing, high-performance reconfigurable circuits and approximate computing techniques for low-power Deep Neural Networks accelerators.

Mikail Yayla is currently pursuing the Ph.D. degree at the informatics chair "Design Automation for Embedded Systems" in the Technical University of Dortmund, under the supervision of Prof. Jian-Jia Chen. His research focuses on robust and efficient machine learning for emerging resource-constrained systems. He has published in major EDA conferences and journals, including DAC, ICCAD, DATE, TCAS-I, and TC. He has one best paper nomination at DATE'21.



Jian-Jia Chen is Professor at Department of Informatics in TU Dortmund University in Germany. He was Juniorprofessor at Department of Informatics in Karlsruhe Institute of Technology (KIT) in Germany from May 2010 to March 2014. He received his Ph.D. degree from Department of Computer Science and Information Engineering, National Taiwan University, Taiwan in 2006. He received his B.S. degree from the Department of Chemistry at National Taiwan University 2001. Between Jan. 2008 and April 2010, he was a postdoc researcher at ETH Zurich, Switzerland. His research interests include real-time systems, embedded systems, energy-efficient scheduling, power-aware designs, temperature-aware scheduling, and distributed computing. He received the European Research Council (ERC) Consolidator Award in 2019. He has received more than 10 Best Paper Awards and Outstanding Paper Awards and has involved in Technical Committees in many international conferences.

Hussam Amrouch (S'11-M'15) is a Jun.-Professor heading the Chair of Semiconductor Test and Reliability (STAR) within the Computer Science, Electrical Engineering Faculty at the University of Stuttgart as well as a Research Group Leader at the Karlsruhe Institute of Technology (KIT), Germany. He currently serves as Editor at the Nature Scientific Reports Journal. He received his Ph.D. degree with the highest distinction (Summa cum laude) from KIT in 2015. His main research interests are design for reliability and testing from device physics to systems, machine learning for CAD, HW security, approximate computing, and emerging technologies with a special focus on ferroelectric devices. He holds eight HiPEAC Paper Awards and three best paper nominations at top EDA conferences: DAC'16, DAC'17 and DATE'17 for his work on reliability. He has served in the technical program committees of many major EDA conferences such as DAC, ASP-DAC, ICCAD, etc. and as a reviewer in many top journals like Nature Electronics, T-ED, TCAS-I, TVLSI, TCAD, TC, etc. He has around 200 publications in multidisciplinary research areas across the entire computing stack, starting from semiconductor physics to circuit design all the way up to computer-aided design and computer architecture. His research in HW security and reliability have been funded by the German Research Foundation (DFG), Advantest Corporation, and the U.S. Office of Naval Research (ONR). ORCID 0000-0002-5649-3102.