Energy Efficient Hardware Implementation of Fully-Connected Artificial Neural Networks Using Approximate Arithmetic Blocks

Mohammadreza Esmali Nojehdeh[1] and Mustafa Altun[1]

Abstract

In this paper, we explore efficient hardware implementation of feed-forward Artificial Neural Networks (ANNs) using approximate adders and multipliers. Due to a large area requirement in a parallel architecture, the ANNs are implemented under the time-multiplexed architecture where computing resources are re-used in the Multiply-ACcumulate (MAC) blocks. The efficient hardware implementation of ANNs is realized by replacing the exact adders and multipliers in the MAC blocks by the approximate ones taking into account the hardware accuracy. Additionally, an algorithm to determine the approximate level of multipliers and adders due to the expected accuracy is proposed. As an application, the MNIST and SVHN database are considered. To examine the efficiency of the proposed method various architectures and structures of ANNs are realized. Experimental results show that the ANNs designed using the proposed approximate multiplier have a smaller area and consume less energy than those designed using previously proposed prominent approximate multipliers. It is also observed that the use of both approximate adders and multipliers yields respectively, up to 50% and 10% reduction in energy consumption and area of the ANN design with a small deviation or better hardware accuracy when compared to the exact adders and multipliers.

Keywords: Approximate mutiplier, approximate adder, Multiply accumulate (MAC), Artificial neural network(ANN)

[1] Electronics and Communication Engineering, Istanbul Technical University, Istanbul, Turkey
Mohammadreza Esmali Nojehdeh
nojehdeh@itu.edu.tr
Mustafa Altun
altunmus@itu.edu.tr
This paper is an extension of work originally presented in ISVLSI2020 [18].

1 Introduction

In recent years, Artificial Neural Networks (ANNs) have achieved a remarkable performance in different research areas, including medical image processing [30], face detection [29], semantic segmentation [38], and control systems[7, 6, 9, 8, 10] .COVID-19 mathematical model by artificial intelligence in [43] shows the vast applicability of this method. Recent developments in Graphics Processing Units (GPUs) and Central Processing Units (CPUs) provide generous memory resources and high computation speeds for training and operation of ANNs. However, for portable devices, due to their limited memory, the number of processing units, and the battery capacity, the realization of ANNs in these devices is impractical. Here, the main concern is to reduce the ANN hardware complexity taking into account the hardware accuracy. The term hardware accuracy refers to the accuracy that is calculated by the gate level simulation results.

An investigation of ANNs complexity reduction within the literature shows that studies are categorized at the software level and hardware level commonly. Some valuable studies provide a survey of the topic's progress [16, 33]. At the software level, apart from hardware consideration, the determination of ANNs structure during the training process is intended to obtain a network with minimum parameters. On the other hand, at the hardware level, distinct from the software side, different techniques are employed for reducing the hardware cost of bulky ANNs. Consequently, training based on devoted hardware, and applicable hardware modeling through the software, provide helpful results to diminish ANNs complexity.

At the software level, [22] provides a theoretical analysis of quantization error. In this study, by focusing on the derivation of finite precession error analysis techniques, the minimum bit number for forward retrieving and back-propagation is calculated. Binary weight network and XNOR networks are proposed in [42]. These two approximations are exploited to realize the standard convolution neural networks. Logarithmic computation concept is presented in [47, 26]. This encoding method enables ANNs to eliminate bulky digital multipliers. Determining logarithmic values for weights during the training process, aides to replacing of digital multipliers by shift operations with acknowledging that the multiplicands are constant in power-two numbers. By considering that ANNs consist of multiplication of different matrices, optimizing the loops is another approach to accelerating network[28], where optimum sharing of these partial terms in the multiple constant multiplications, reduces hardware complexity[2, 3]. Beyond synthesis methods, other approaches like stochastic and approximate neural networks are common in literature. Applying stochastic computational units in neural networks results in error maintains within 10 percent of floating-point implementation[14]. The accuracy of stochastic computation may not be comparable with the conventional method but, low circuit area and power consumption make this method favorable for hardware implementation.

At the hardware level, different field-programmable gate array (FPGA) and application-specific integrated circuit (ASIC) circuits are investigated for accelerating network. To overcome the problem of memory access in large ANNs, a custom multi-chip machine-learning architecture is introduced in [15]. A specialized chip consists of a microcontroller, accelerator, and on-chip SRAM is introduced for always-on subsystems of mobile/Internet of Things(IoT)devices in [27]. Also, apart from customized chips, different hardware architectures by focusing on arithmetic operations are exploited to hinder the bulky area problem of ANNs.

ANNs realization under Multiplier ACcumulated (MAC) units reusing is an approach to reduce hardware occupied area and power consumption by considering an increase in delay. According to MAC-based implementation, ANN hardware structure can classify into two models: axonal-based[11] and dendritic-based[1, 40] models. For axonalbased model, every single input of layers is multiplied by related weights of all neurons of the layer, and all outputs calculate simultaneously, as a result, for axonal-based model obtaining all inputs at the same time is unnecessary. However, for accumulating different multiplication results, extra memory is essential. On the other hand, in dendritic-based model, the value of the next neuron is calculated by multiplying all inputs with related neuron weights and accumulating them. This method results in the sequential generation of outputs, and every step of calculation needs to obtain all inputs to start. In [41], by combining these two architecture, parallel computing is enabled in two successive layers to achieve smaller latency in the computing time of the whole network.

Since the multiplier is the core block of MAC and dominates calculation time, so designing the multipliers has become an important consideration. Conventional multipliers consist of an array of Full Adders (FA) to add partial products and final adders. Exploiting Wallace tree structure with different compressors leads to delay reduction in multipliers.

Based on the error-tolerant inherency of neural networks, approximate neural networks or ANNs with approximate blocks are a favorable approach to realize ANNs, where the tradeoff between hardware complexity and accuracy is explored through the approximate level. Approximation for both computation and memory access is investigated in [51]; also, the impact of neurons on the output quality is determined to approximate the computation and memory accesses of less critical neurons. This technique leads to achieving maximum efficiency under a given quality constraint.

The exact adders and multipliers in the MAC blocks are replaced by the approximate ones in[18]. The exploitation of approximate units yields respectively up to 64% and 43% reduction in energy and area of the ANN design for PENDIGIT dataset with a slight decrease in the hardware accuracy. An evaluation of a large pool of approximate multipliers consisting of 100 deliberately designed and 500 Cartesian Genetic Programmings (CGP) based multipliers in ANNs, is accomplished in [5]. Also, to determine the critical features of multipliers in ANNs, different error parameters' efficacy is investigated. According

to this study, the CGP-based multipliers introduced in [34] are better suited to the investigated ANN.

Beyond the hardware architecture, there are different methods which are related to ANNs based on the application, such as Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs) and Multilayer Perceptrons (MLPs) based networks. Since our priority is energy efficiency, we focus on MLPs, which provide better energy scaling [27, 49]. Each layer that comprises MLP has a computational workload, which is composed of relatively basic operations, i.e., multiplication, addition, and activation. As the network is layered, arithmetic operations run in a pipeline (feed-forward) by axonal-based model, where inputs of one layer wait for the outputs of the previous layer.

In this paper, ANNs are implemented under two different architectures using MAC blocks to explore the area and latency tradeoff. In the first one, called $SMAC\_NEURON$, a single MAC is used to realize each neuron computation in each layer, and in the second one, called $SMAC\_ANN$, a single MAC is used to implement the whole ANN. Moreover, we present efficient hardware implementation of ANNs under the time-multiplexed architectures using approximate adders and multipliers taking into account the ANN hardware accuracy. To examine the performance of the proposed method, the exact adders and multipliers in the MAC blocks and parallel units are replaced by the approximate ones. Furthermore, we introduce an algorithm to determine the approximate level of multipliers and adders where the tradeoff between the hardware complexity and accuracy can be explored by leveraging the approximation level of blocks. We note that the generation of an approximate multiplier and adder with different bit-widths of inputs under the given approximation level can be done in linear time as opposed to the methods of [13, 34] . As shown in [51] , the ANN hardware complexity can be significantly reduced by using approximate multipliers with different approximation levels for the neuron computations at different layers. Experimental results indicate that the ANNs including the proposed approximate multiplier occupy less area and consume less energy than the exact versions with a small degradation in the accuracy. It is also shown that the ANN hardware complexity can be further reduced by using approximate adders.

The rest of this paper is organized as follows. Background concepts and related work are given in Section II. Section III presents the MAC-based design architectures. In Section IV, the implementation of approximate adders and multipliers is described. Section V presents the experimental results, and finally, Section VI concludes the paper.
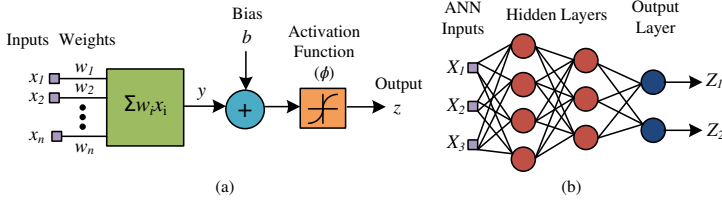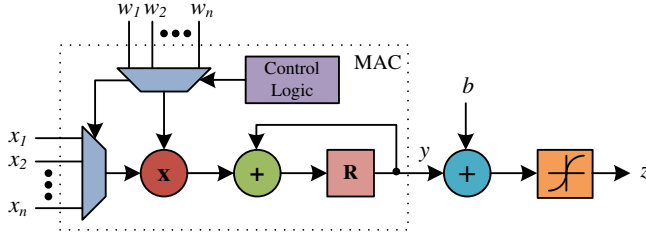
2 Background

2.1 ANN Concept

Arithmetic operations of ANN is illustrated in Fig. 1, also Fig. 2(a) represents the fundamental block of ANN, *i.e.*, neuron, which sums the multiplication of
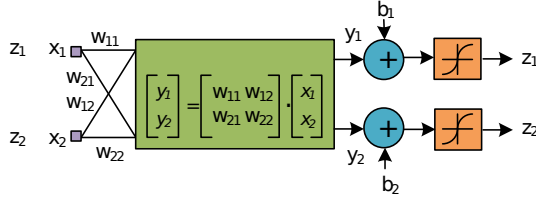
Fig. 1 ANN mathematical model for layers.

Fig. 2 (a) Artificial neuron; (b) ANN with two hidden layers.

Fig. 3 MAC block in the neuron computation.

input variables by weights, adds the bias value to this summation and propagates this result to the activation function. In mathematical terms, the neuron is described as $y = \sum_{i=1}^{n} \omega_i x_i$ and $z = \phi(y + b)$ where n denotes the number of inputs and weights. Fig. 2(b) presents an ANN design including hidden and output layers where each circle denotes a neuron.

Observe from Fig. 2 that adders and multipliers are frequently used in ANNs and dominate the hardware complexity. To reduce the ANN design area, taking into account an increase in latency, ANNs can be designed under the time-multiplexed architecture using MAC blocks. Fig. 3 shows a MAC-based realization of the neuron computation given in Fig. 2(a), re-using the multiplication and addition operations. In this figure, clock and reset signals are omitted for the sake of clarity. Observe that the multiplication of a weight by an input variable is realized at a time synchronized by the control block and is added to the accumulated value stored in the register R. We note that the control logic is actually a counter that counts between 1 and n. Under this architecture, the neuron computation is obtained after $n + 1$ clock cycles.

The design complexity of the MAC block depends on the size of the counter and multiplexers, determined by the number of weights and input variables, on the size of the multiplier, determined by the maximum bit-widths of the input

variables and weights, and on the size of adder and register, determined by the bit-width of the inner product of inputs and weights, *i.e.*, $y = \sum_{i=1}^{n} w_i x_i$.

2.2 ANN Structure

An ANN is comprised of a network of neurons which are connected to each other. The weight and bias values of ANN are determined in a training phase where the error between the desired and actual response is reduced using an iterative optimization algorithm. During training, inputs are generally normalized between -1 and 1. Such normalization may decrease the training run-time and yield an ANN with a fewer number of neurons and layers when compared to the ANN trained with un-normalized inputs, both achieving similar accuracy. Furthermore, the test data are used to provide an unbiased evaluation of the final model after the training process, and the accuracy, or misclassification rate, is computed as a performance metric [21].

2.3 ANN Implementation

To reduce the ANN hardware complexity, in [44, 17], it is shown that the weights of ANNs can be determined to include a small number of non-zero digits in training and hence, their multiplications by input variables can be realized using a small number of adders and subtractors. The floating-point weights in each layer are quantized dynamically, and the fixed-point weights are expressed in binary representation in [46].

To reduce the high latency of the MAC block, a delay-efficient structure, which uses accumulators and carry-save adders, was introduced in [36]. Efficient implementation of ANN designs using MAC blocks on FPGAs was introduced in [36]. Recently, MAC blocks have been used in the realization of neuromorphic cores using two models, namely axonal-based and dendritic-based [12]. A post-training method and a multiplierless design technique that can reduce the design complexity of a time-multiplexed ANN are given in [3].

2.4 Approximate Adders and Multipliers

Approximate computing refers to a class of methods that relax the requirement of exact equivalence between the specification and implementation of a computing system [20]. This relaxation allows trading the accuracy of numerical outputs for reductions in area, delay, or power dissipation of the design [45, 35]. Due to a high error-tolerance in ANNs, the use of approximate multipliers in ANNs is an alternative way for the reduction of the ANN hardware complexity [5].

By increasing the number of the erroneous results in 1-bit full adder's truth table the complexity of the designs decreases, based on this, 3 different 1-bit approximate full adders are proposed. [32]. In [50, 19], at the transistor level, approximate 1-bit adders are derived from the conventional mirror adders and XOR/XNOR based adders by removing transistors and/or replacing some parts of the adders with a small circuitry then, a generic approximate adder

is implemented using approximate 1-bit adders. In [13, 34], at the gate level, design tools are generated to develop efficient approximate adders. Motivated by the drawbacks of approximation methods at the transistor and gate level, a systematic synthesis technique based on a new error calculation method is introduced in [39].

In [34], the CGP method is used to generate approximate multipliers. The deliberately designed approximate multiplier of [5] is obtained through simplifications in the truth table of the multiplication operation. A novel approximate multiplier based on the input probabilities of 1-bit adders is proposed in [39]. Different approaches based on the shifting and adding operations are investigated to reduce the complexity of the multiplier. For example, the Karatsuba method based on Indian Vedic mathematics separates the operands into two sections and then generates the output by employing 4 small bit length multipliers. This algorithm possesses two accumulators and a shifter to obtain the final result. A novel algorithm by removing a multiplier and adding an accumulator in Karatsuba multipliers is proposed in [24]. Also, the approximate version of the proposed multiplier for image processing application is achieved. The simplification in the compressor for the Dadda structure is another method to reduce the complexity of the multipliers, whereby approximating the compressor, the hardware's complexity in cost of accuracy decreases and an approximate multiplier is achieved [31].

3 MAC-based ANN Design

Following the determination of the floating-point weight and bias values in the training phase, they are converted to integers since the floating-point multiplication and addition operations occupy more area and consume more energy than their integer counterparts [23]. This conversion is simply done by multiplying each floating-point weight and bias value by 2^q, where q denotes the minimum quantization factor.

To determine minimum quantization factor q following steps are obtained.

1. Set the quantization value, q, and the related ANN accuracy in hardware, $ha(q)$, to 0.
2. Increase q value by 1.
3. Convert each floating-point weight value to an integer by multiplying it by 2^q, and find the least integer greater than or equal to this multiplication result.
4. Compute $ha(q)$ value on the validation data-set using the integer weight values.
5. If $ha(q) - ha(q-1)$ is greater than 0.1%, go to Step 2.
6. Otherwise, return q as the minimum quantization value.

The obtained value is the settle point of the accuracy based on the desired deviation. The floating-point system is employed in the training process, but the integer number system is preferable in constrained resource deployments.
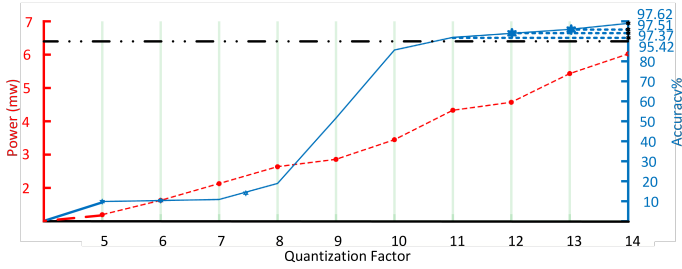
Fig. 4 Power dissipation and hardware accuracy value in terms of quantization values.

Integer-based systems have a superior performance in terms of consumed power and energy. In contrast, they have a lesser number of representations, which results in a loss of accuracy when compared to long bit-representation systems. Fig. 4 illustrates that the network's accuracy increases as the bit length increases.

Power dissipation and hardware accuracy of different quantization values for a trained ANN are shown in Fig. 4. This demonstration is a primitive graph to show the bit-width's impact on power consumption. To avoid the complexity of different structures, we chose Pendigit as an application. The exploited architecture is parallel, where 16 neurons form a hidden layer. Consider that this network's size is negligible compared to the investigated network in section VI. As expected, the power dissipation soars by incrementing the quantization value, and the accuracy value settles after some steps. The settling point is chosen based on the expected performance of the application.

In following, the SMAC_NEURON and SMAC_ANN design architectures are described in detail.

3.1 *SMAC_NEURON* Architecture

Fig. 5 presents the neuron computations at the kth layer of an ANN using m MAC blocks and a common control block where m and n denote the number of outputs (or neurons) and inputs at this layer, respectively. The control block synchronizes the multiplication of input variables by the associated weights. Assuming that an ANN includes η_i neurons at each layer, where $1 \leq i \leq \lambda$ and λ denotes the number of layers, the required number of MAC blocks is $\sum_i^\lambda \eta_i$, *i.e.*, the total number of neurons. Note that the complexity of operations and registers in the MAC blocks are determined by the number of inputs and outputs at each layer and the weight values related to each neuron of each layer.

The complexity of the control block is determined by the number of inputs at each layer. Since the neuron computations are obtained layer by layer, the neuron computations in the latter layer are started after the ones in the former layer are finished. This is accomplished by producing an output signal at each layer indicating that all neuron computations have been obtained. By doing this, we are able to reduce power dissipation and also stop the hardware from
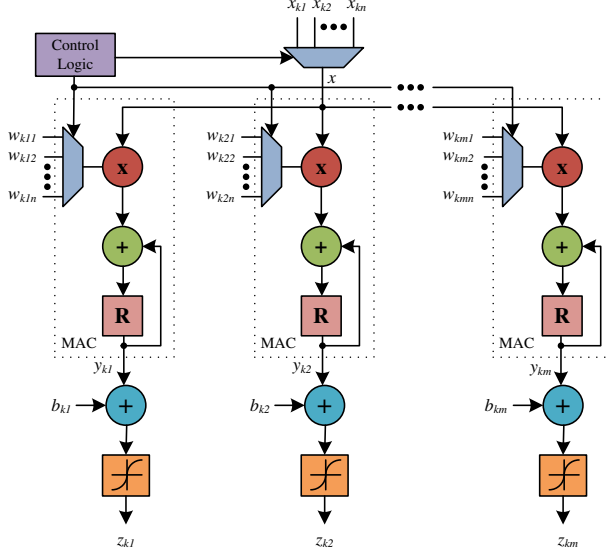
Fig. 5 Neuron computations at the kth layer of ANN using MAC blocks.

performing any further computations that are not necessary. The computation of whole ANN with λ layers and ι_i inputs at each layer, where $1 \le i \le \lambda$, is obtained after $\sum_i^{\lambda}(\iota_i + 1)$ clock cycles.

3.2 *SMAC_ANN* Architecture

Fig. 6 shows the ANN design using a single MAC block, where the clock and reset signals are omitted for the sake of clarity. In this figure, the control block includes three counters to synchronize the multiplication of a weight by an input variable, the addition of a bias value to each inner product, and applying of the activation function. These counters are associated with the number of layers, the number of inputs at each layer, and the number of outputs (or neurons) at each layer. Note that the variables $X_1, X_2, \ldots, X_n$ denote the primary inputs of ANN, and these variables are multiplied by the related weights during the computations at the first hidden layer. While the maximum number of inputs determines the size of multiplexers for the input variables at all layers, the size of multiplexers for the weight and bias values are defined by the total number of weight and bias values, respectively. The maximum bitwidth among all input variables and weights determines the MAC block's multiplier size. The maximum bitwidth of the addition of weights by input variables throughout the entire ANN determines the size of the adder and register. Moreover, the number of registers used to store the outputs at each layer is determined by the maximum number of outputs at each layer. We note that the computation of whole ANN with λ layers, ι_i inputs at each layer, and η_i neurons at each layer, where $1 \le i \le \lambda$, is obtained after $\sum_i^{\lambda}(\iota_i + 2)\eta_i$ clock cycles.
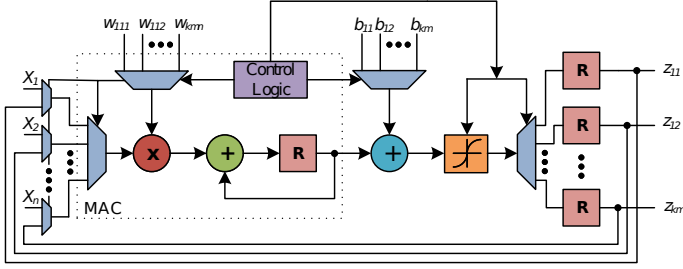
Fig. 6 ANN design using a single MAC block.

Table 1 Truth Tables of Exact and Approximate 1-bit Adders.

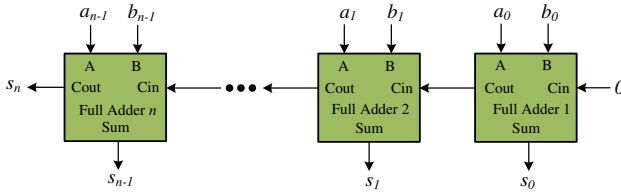| Inputs			FA			APAD1				APAD2				APAD3				APAD4			
A	B	Cin	Cout	Sum	Decimal	Cout	Sum	Error	Decimal	Cout	Sum	Error	Decimal	Cout	Sum	Error	Decimal	Cout	Sum	Error	Decimal
0	0	0	0	0	0	0✓	0✓	0	0	0✓	0✓	0	0	0✓	0✓	0	0	0✓	0✓	0	0
0	0	1	0	1	1	0✓	1✓	0	1	0✓	1✓	0	1	0✓	1✓	0	1	0✓	0✗	-1	0
0	1	0	0	1	1	1✗	0✗	+1	2	0✓	1✓	0	1	0✓	1✓	0	1	0✓	1✓	0	1
0	1	1	1	0	2	1✓	0✓	0	2	0✗	1✗	-1	1	0✗	1✗	-1	1	0✗	1✗	-1	1
1	0	0	0	1	1	0✓	1✓	0	1	1✗	0✗	+1	2	1✗	0✗	+1	2	1✗	0✗	+1	2
1	0	1	1	0	2	1✓	0✓	0	2	1✓	0✓	0	2	1✓	0✓	0	2	1✓	0✓	0	2
1	1	0	1	0	2	1✓	0✓	0	2	1✓	0✓	0	2	1✓	1✗	+1	3	1✓	1✗	+1	3
1	1	1	1	1	3	1✓	1✓	0	3	1✓	1✓	0	3	1✓	1✓	0	3	1✓	1✓	0	3

Fig. 7 Ripple carry adder.

4 Implementation of ANNs Using Approximate Arithmetic Units

In this section, we present the approximate adder of [39] and multipliers of [39, 34] used in the ANN designs, introduce an approximate multiplier and describe the implementation of ANNs using approximate adders and multipliers.
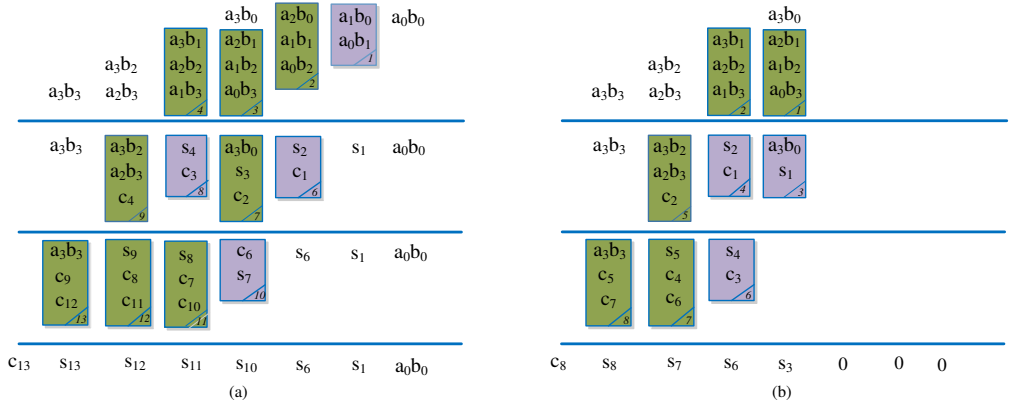
Fig. 8 (a) Exact 4-bit unsigned multiplier; (b) Approximate 4-bit unsigned multiplier with the least significant 3 bits are set to logic value 0.

4.1 Approximate Adder

Fig. 7 illustrates an n-bit ripple carry adder which consists of n 1-bit full adders (FAs). In this figure, A, B, and carry-in (Cin) represent the input bits of FA and Sum, and carry-out ($Cout$) denote its output bits. The truth table of 1-bit FA is given in Table 1. In the related studies on approximate ripple carry adders [13, 50], it is assumed that simultaneous errors on both Sum and $Cout$ outputs of FA may generate a larger erroneous result on the adder output than an error on a single output. However, this assumption neglects the fact that while an error on one output of an FA block increases the error at the adder output, another error on the other output may decrease the error. For example, as given in Table 1, on the entry of $ABCin = 010$ for the approximate 1-bit adder APAD1, both errors on the outputs of FA generate only an error with a magnitude of 1. Thus, alternating errors on both the Sum and $Cout$ outputs can provide an opportunity to simplify the hardware complexity of an approximate 1-bit adder. Based on this fact, 4 approximate 1-bit adders (APADs) with different error values and hardware complexity are introduced in [39]. The truth tables of these APADs are given in Table 1. To obtain an n-bit approximate ripple carry adder, a synthesis method that replaces the exact FAs by APADs is presented in [39].

4.2 Approximate Multipliers

The implementation of an exact multiplier consists of two stages, *i.e*, partial product generation using AND gates and accumulation of these partial products using half adders (HAs)[1] and FAs. An exact 4-bit unsigned multiplier structure is shown in Fig. 8(a), where rectangular blocks with 2 and 3 entries denote an HA and FA, respectively.

---

[1] Half adder is obtained when one of the inputs of FA is set to 0.

In the design of an approximate multiplier, based on the probability of occurrences of logic 0 and 1 at the outputs of each HA and FA, the synthesis tool of [39] replaces exact HA and FA blocks in the by their approximate versions that are called probability based approximate multipliers (PBAM). Also, the CGP method of [34] generates approximate multipliers derived from the exact multipliers.

In addition to these approximate multipliers, we propose another one, called LEBZAM, which is implemented by setting r least significant outputs of an exact multiplier to zero, where r denotes its approximation level. The synthesis method is described as follows: i) set r least significant outputs of the exact multiplier to 0; ii) eliminate all the FA and HA blocks required to realize r least significant outputs of the exact multiplier. Fig. 8(b) illustrates the realization of 4-bit approximate multiplier when r is 3.

We note that given the approximation level and the bitwidths of the inputs, an approximate multiplier LEBZAM can be easily obtained as opposed to the approximate multipliers of [34, 13]. Thus, by using approximate multipliers with different sizes and approximation levels in the MAC blocks of ANN designs under the architectures presented in Section 3, a significant reduction in the ANN hardware complexity can be achieved by taking into account the hardware accuracy. Similarly, by using approximate adders of [39] , the ANN hardware complexity can be further reduced.

4.3 Approximate Level

4.3.1 *SMAC_NEURON*

The number of least significant bits assessed under approximation blocks in layer n is referred to as the approximate level (ALn) in this study. We called it adders' approximate level (AAL) for adders and multipliers' approximate level (MAL) for multipliers.

To determine the approximate level of the multipliers and adders based on the misclassification rate (MR) for $SMAC\_NEURON$ architecture following steps are obtained.

1. Set the hidden layer number n to 1.
2. Set the approximate level AL_n to 0.
3. Increase AL_n value by 1.
4. Calculate Approximation Misclassification rate AMR.
5. If $AMR - MR < tolerable\_error$ go to Step 3, otherwise increase n value by 1.
6. If $n < n_{max} + 1$ save $AL_n - 1$ as the approximate level of nth layer and and return to step2.
7. save $AL_n - 1$ as the approximate level of output.

These steps are taken separately for adders and multipliers. We must acknowledge that starting with multipliers or adders will result in the same approximate level values. In this study, initially, we apply the proposed method

for approximate multipliers to shrink the search space of (AAL), the minimum level value of AAL is set to the determined (MAL) value incremented by one. Additionally, the error distance values of LEBZAM are negative or zero for all cases; based on this error pattern, the approximate level of multipliers and adders for all neurons in each layer are chosen identically. Contrarily selecting a higher approximate level for any neuron compared to other neurons at the same layer leads to a negative bias of that neuron i.e. the neuron with the higher approximate level posses a scanty output regarding other neurons, where this biasing results a disturb in accuracy. An arithmetic unit with m-bitwidth output, posses a number between 0 and m as a approximate level($(m + 1)$options). By considering n-bit $\times$ n-bit multiplier for MAC unit, the adder output bitwidth value will be $(2n + 1)$, and the total possible combination number of approximate level for adder and multiplier for each neuron will be $(2n + 1) \times (2n + 2)$. Also by considering that there are η neurons in λ layers, the total possible combination for a ANN is formulated as $(4n^2 + 6n + 2) \sum_i^\lambda \eta_i$.

By exploiting the proposed method, the approximate level values of the multipliers are identical for all of the neurons. According to the method, we increment the MAL value by 1 until the error deviation becomes greater than the given error limit value. MAL value of n-bit multiplier is a number between 0 and $2n$; hence by exploiting the proposed method, the total investigated case is MAL for all neurons in each layer. In line with the proposed method, the minimum number of AAL values is equivalent to the determined MAL value increment by one. Similarly, the same steps are obtained to find the AAL value; consequently, the total investigated cases is $MAL + (AAL - MAL)$ for all neurons of each layer. The total number of examined cases for the whole network is $\sum_i^\lambda (MAL_i + 1) + (AAL_i + MAL_i)$.

As an example, consider the Pendigit handwritten digit recognition problem [4], where the trained network architecture consists of 16 inputs, 50 neurons in the hidden layer, and 10 outputs. Assume the input and weights bit-width is 8, consequently the all possible combination of MAL and AAL is $(4(8^2) + (6 \times 8) + 2) \times (50 + 10) = 18360$ whereas, by exploiting the proposed method, the significant reduction in the explored cases occurs.

The MR deviation percentage for different $MALs$ and $AALs$ values are shown in Fig.9. Based on the proposed method, after seven iterations, the MAL_1 value is set to 6. To find this value all the other arithmetic units are set to their exact versions, and their approximate level is 0. Note that setting MAL_1 value to 7 causes MR value to become greater than the given value, which is considered 1% of MR for this example. The same steps are employed to obtain the MAL_2 value. According to the proposed algorithm, to investigate AAL_1 and AAL_2 values, the starting points are set to their corresponding MAL value in each layer. As shown in Fig.9, the AAL value for layer1 and layer2 is 10. Note that the total examined case number for this example is $(7 + (10 - 7)) + (10 + (10 - 10)) = 20$, which is negligible in comparison to all possible 18360 cases.
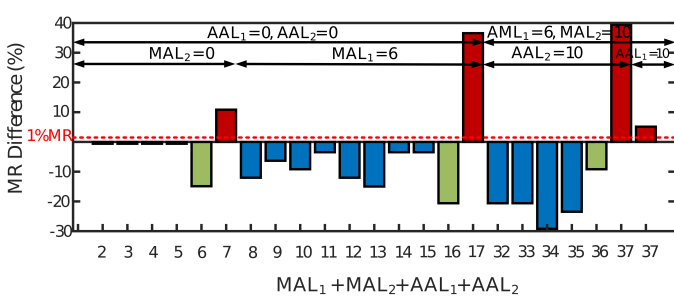
Fig. 9 Misclassification Rate for SMAC ₋ NEURON architecture by the different approximate levels of multipliers and adders.
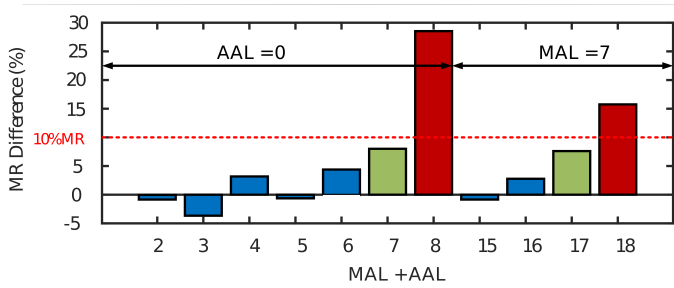
Fig. 10 Misclassification Rate for SMAC₋ ANN architecture by the different approximate levels of multipliers and adders.

4.3.2 *SMAC_ANN*

To determine the approximate level of multipliers and adders in $SMAC\_ANN$ architecture following 7 steps are obtained.

1. Set the MAL and AAL to 0.
2. Increase MAL value by 1.
3. Calculate Approximation Misclassification rate AMR.
4. If $AMR - MR < tolerable\_error$ go to Step 2, otherwise save $MAL - 1$ as the approximate level of multiplier.
5. Increase AAL value by 1.
6. Calculate Approximation Misclassification rate AMR.
7. If $AMR - MR < tolerable\_error$ go to Step 4, otherwise save $AAL - 1$ as the approximate level of adder.

The $SMAC\_ANN$ architecture comprises a singular MAC unit as the arithmetic unit; consequently, determining the approximate level of multiplier and adder is more straightforward compared to the $SMAC\_ANN$ architecture. Distinct from the ANN structure, the total possible combination of MAL and AAL values is correlated with the output bit-width of arithmetic units. By assuming that the multipliers and adders output bit-widths are j and k, respectively, the number of all possible combinations of MAL and AAL will

be $j \times k$. On the other hand, by exploiting the proposed method, the number of investigated cases shrinks to $MAL + (AAL - MAL)$ cases.

As an example, Pen-digit handwritten digit recognition problem is employed by the same parameters but under $SMAC\_ANN$ architecture. The applied method results are shown in Fig .10. Starting by the multiplier, after 8 iterations the MAL value is set to 7, and the tolerable error value is considered as 1.1 of MR. According to the proposed method, AAL initial value is set to $MAL + 1$, and after 4 iterations, the AAL value is set to 10 correspondingly. Take consider the total examined cases is 12, whereas all possible combination is (16×20) by considering that, the output bit-width is 16 and 20 for multiplier and adder respectively.

According to the Section 3.2 the proposed method obtains the results after 12 steps whereas the number of the total possible case is (3200) for $SMAC\_ANN$. On the other hand, this technique achieves the result after 20 steps as mentioned in Section 3.1 for $SMAC\_NEURON$ among 18360 possible cases.

5 Experimental Results

As an application, we considered the MNIST handwritten digit recognition problem [25]. The dataset consists of gray-scale images from NIST, which are normalized to fit into (28×28) pixel boxes. The ANN is employed to predict the digit among 10 integers(0-9) based on the input pixels. To examine the performance of the proposed method on a different structure, we implemented the feedforward ANN with two different structures; 3 hidden layers by 256 neurons for the first case and a hidden layer by 128 neurons for the second case. The ANN was trained using the deep learning toolbox of MATLAB [48], where the training and test inputs were normalized in between -1 and 1, the weights were initialized randomly, and they were adjusted to minimize the error in between the actual and desired response using a backpropagation-based learning method. The activation functions in the hidden and output layer were symmetric saturating linear and softmax, respectively. The ANN was trained using 50000 data and tested using 10000 data. To consider the overfitting problem 10000 data sets are considered as the validation data during the training process. Also, the quantization factor q was set to 12 for employed ANN by exploiting the proposed steps in section3. By converting input data to 12 bits integer, 116 of 784 input pixels remained unchanged for the train and test data, as eliminating these fixed values will not affect the performance of ANN.

By training ANN through MNIST database at the software level, the computed MRs for test data were calculated as 2.60 and 2.24 for 668-256-256-256-10 and 668-128-10 structure, respectively.

After the floating-point weight and bias values were converted to integers when the quantization value q was set to 12, the ANN design using exact adders and multipliers was described in a behavioral fashion, and the hardware

Table 2 Results of SMAC_NEURON architecture for 668-128-10 structure using approximate multipliers and adders.

| Multiplier Type | Approximation Level | | | | area | delay | latency | power | energy | HMR | area gain | energy gain |
| | Hidden | | Output | | | | | | | | | |
	Mul	Add	Mul	Add								
Behavioral	0	0	0	0	200831	9.56	7.63	21.52	164.10	2.63	0%	0%
PBAM [39]	7	0	16	0	206506	9.29	7.41	18.70	138.62	2.54	-3%	16%
LEBZAM	7	0	16	0	189796	7.75	6.18	17.88	110.52	2.65	5%	33%
EvoAPP[34]	HDG/KQ	0	HDG/KQ	0	224621	8.71	6.95	17.67	122.73	2.64	-12%	25%
EvoAPP[34]	HFZ/K5	0	HFZ/K5	0	214757	8.30	6.62	17.64	116.82	2.66	-7%	29%
EvoAPP[34]	GAT/NM	0	GAT/NM	0	157275	5.85	4.66	17.28	80.61	2.78	22%	51%
EvoAPP[34]	2KM	0	0	0	229000	8.39	6.70	18.78	125.83	2.63	-14%	23%
EvoAPP[34]	0	0	HDG/KQ	0	202615	8.94	7.13	22.38	159.54	2.62	-1%	3%
EvoAPP[34]	0	0	12N/GAT	0	203342	9.14	7.29	21.68	158.07	2.64	-1%	4%
EvoAPP[34]	0	0	K5/HFZ	0	200349	9.56	7.63	21.87	166.89	2.64	0%	-2%
PBAM [39]	7	8	14	15	191431	8.98	7.17	12.63	90.48	2.64	5%	45%
PBAM [39]	8	8	15	17	185200	9.18	7.33	13.67	100.18	2.61	8%	39%
PBAM [39]	7	7	16	16	191925	10.47	8.35	11.87	99.13	2.54	4%	40%
LEBZAM	7	7	15	17	191286	9.09	7.25	12.88	93.42	2.56	5%	43%
LEBZAM	7	8	15	15	189343	8.76	6.99	12.11	84.66	2.61	6%	48%
LEBZAM	7	8	16	17	187396	9.03	7.21	12.85	92.62	2.63	7%	44%
LEBZAM	7	7	16	16	190466	9.55	7.62	13.19	100.53	2.65	5%	39%
LEBZAM	8	8	14	14	188112	6.70	5.35	14.31	76.49	2.76	6%	53%

Table 3 Results for fully connected layer of convolutional neural network by SMAC_NEURON architecture for 468-128-10 structure using approximate multipliers and adders.

Multiplier Type	Approximation Level				area	delay	latency	power	energy	HMR	area gain	energy gain
	Hidden		Output									
	Mul	Add	Mul	Add								
Exact	0	0	0	0	134651	5.77	0.0034	14.53	0.050	8.81	0%	0%
LEBZAM	3	0	0	0	125350	6.22	0.0037	11.70	0.043	10	7%	13%
LEBZAM	4	0	0	0	122349	5.93	0.0035	11.88	0.042	12.21	9%	18%
LEBZAM	3	5	3	5	121482	5.80	0.0035	9.25	0.032	9.10	10%	36%
LEBZAM	4	6	4	6	117278	6.06	0.0036	8.07	0.029	10.24	13%	42%

Table 4 Results of SMAC_NEURON architecture for 668-256-256-256-10 structure using approximate multipliers and adders.

Multiplier Type	Approximation Level								area	delay	latency	power	energy	HMR	area gain	energy gain
	Hidden1		Hidden2		Hidden3		Output									
	Mul	Add	Mul	Add	Mul	Add	Mul	Add								
Behavioral	0	0	0	0	0	0	0	0	939076	8.05	11.60	141.37	1639.62	3.96	0%	0%
PBAM [39]	7	0	7	0	7	0	14	0	971948	7.31	10.53	131.53	1384.58	4.22	-4%	16%
PBAM [39]	7	7	7	7	7	7	14	14	892979	9.03	13.00	77.10	1002.29	4.22	5%	39%
LEBZAM	6	0	6	0	6	0	14	0	929330	9.00	12.96	107.94	1398.73	4.25	1%	6%
LEBZAM	6	6	6	6	6	6	14	14	919664	9.17	13.20	84.85	1120.19	4.25	2%	32%
EvoAPP[34]	DG/KQ	0	DG/KQ	0	DG/KQ	0	DG/KQ	0	1051004	9.22	13.28	111.91	1486.26	3.99	-12%	9%
EvoAPP[34]	FZ/K5	0	FZ/K5	0	FZ/K5	0	FZ/K5	0	1000478	9.57	13.79	105.15	1449.53	4.05	-7%	12%
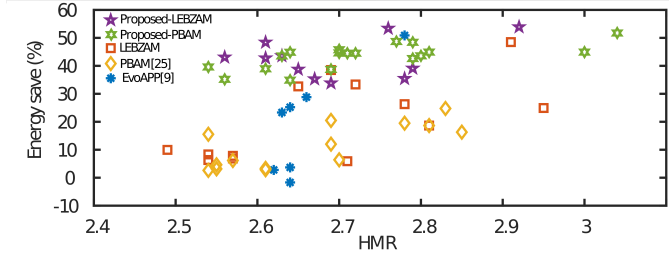
Fig. 11 Energy save percentage of ANN for different approximate methods in terms of Hardware Misclassification Rate.
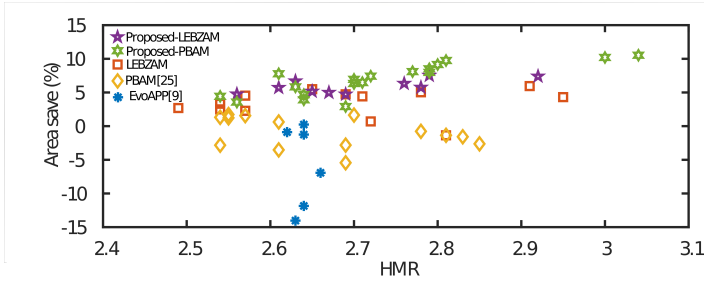
Fig. 12 Area save percentage of ANN for different approximate methods in terms of Hardware Misclassification Rate.

misclassification rate (HMR) was found as 2.63% for 668-128-10 structure, and 3.69% for 668-256-256-256-10 structure. In this study, the ANN designs were implemented using approximate adders and multipliers without exceeding the HMR limit, which was set to 10% deviation in HMR.

The proposed multipliers are compared to CGP-based multipliers that are introduced in [34] in order to assess the effectiveness of the multipliers and the algorithm in comparison to other literature. According to [5] study, CGP-based multipliers holds better result among all the deliberately approximate multipliers.

The ANNs were implemented under the SMAC_NEURON and SMAC_ANN architectures using the approximate adders of [39], the approximate multipliers of [34], and our proposed multiplier LEBZAM. The signed approximate multipliers of [34], have constant 12-bit×12-bit and 16-bit×16-bit inputs. Also, according to $SMAC$ synthesis method, inputs bit-widths of multipliers are non-identical for this architecture. To adopt the [34] multipliers with the employed structure, the multipliers from the library of [34] by maximum bit-width were selected, and then we removed the gates and in-outs of extra bits. Note that for [39] and LEBZAM multipliers, we systematically determined the approximation levels of adders and multipliers on the hidden and output layers taking into account the HMR value. The ANN designs were described in Verilog and synthesized using the Cadence Genus tool with the TSMC 40nm design library.

Also, the SVHN[37] database is considered to examine the performance of the proposed approximate blocks in the fully connected layer of convolutional neural networks. SVHN is a real-world image dataset for developing machine learning and object recognition algorithms with minimal data preprocessing and formatting requirements. SVHN is obtained from house numbers in Google Street View images.

TensorFlow is exploited in the convolutional neural network training for the SVHN dataset. The trained network architecture is 5 layers deep, with 3 pooling layers included. The convolution layer by 52 filters extracts the input features. Flatten layers transform the 3D inputs to feed a 1D fully connected layer after three sequential convolutions and pooling layers. The fully connected layers consist of 128 neurons in the hidden layer and 10 neurons in the output layer. Also, the TensorFlow post-training strategy is exploited to convert the 32-bit floating-point numbers (such as weights and activation outputs) to the nearest 8-bit fixed-point numbers.

The energy and area save of an ANN by 668-128-10 structure and under $SMAC\_NEURON$ architecture for 63 cases by 5 different approximation methods are shown in Fig. 11 and Fig. 12 respectively. Consider that in Figs. 11 and 12, the approximate adders and multipliers for LEBZAM and PBAM are applied separately, but in the proposed technique, the approximate multipliers and adders are used concurrently. As shown in these figures, exploiting simultaneously approximate multipliers and adders according to the proposed method always hold more savings in energy and area with the same HMR values compared to the other methods.

Tables 2-5 present the gate-level results, where *area*, *delay*, and *power* stand respectively for total area in μm^2, the delay in the critical path which is determined to be the clock period in ns, and total power dissipation in mW. Also, *latency* denotes the time in ms required for the ANN output to be obtained after an input is applied, determined as the multiplication of clock period by the number of clock cycles to obtain the ANN output. The number of clock cycles required to obtain the ANN output under the SMAC_NEURON and SMAC_ANN is respectively computed as 798 and 87060 for 668-128-10 structure and, 1440 and 306196 for 668-256-256-256-10 structure. Moreover, *energy* presents the energy consumption in μJ computed as the multiplication of latency by power dissipation. We note that the clock period was improved using the retiming technique in the synthesis tool iteratively. The switching activity data required for the computation of power dissipation was generated using the test data in simulation. The test dataset was also used to verify the ANN design.

Table 2 and 4 present the gate-level results of ANN designs under the SMAC_NEURON architecture where the exact multipliers and adders in the MAC blocks are replaced by the approximate ones. Observe that since the approximate multipliers of [34] are optimized for energy consumption, the ANN designs including these multipliers may have worse area values than those of

Table 5 Results of SMAC_ANN architecture for 668-128-10 structure using approximate multipliers and adders.

Multiplier Type	Approximation Level		latency	power	energy	HMR	energy gain
	Mul	Add					
Behavioral	0	0	688.04	9.33	6418.81	2.63	0.00%
LEBZAM	6	7	666.18	8.01	5333.17	2.63	17%
PBAM [39]	7	7	720.94	7.73	5570.29	2.7	13%
EvoAPP[34]	HDG	0	695.78	8.30	5773.67	2.04	10%
EvoAPP[34]	GAT	0	659.57	8.59	5667.82	2.78	12%

ANN using exact multipliers. This is also due to the fact that the logic synthesis tool uses optimized exact multipliers and adders. On the other hand, the use of approximate multipliers of [39] can reduce the ANN hardware complexity by finding the appropriate approximation levels of multipliers at the hidden and output layers. Furthermore, the proposed approximate multiplier LEBZAM leads to the largest reduction in area, latency, and energy consumption. Observe that the tradeoff between hardware complexity and accuracy can be explored by simply changing the approximation level of multipliers and adders.

By exploiting the proposed algorithm, the deviation limit is set to 2.5% of HMR, and regarding that, the HMR value is 2.63, and the maximum tolerable HMR is 2.69 for 668-128-10 structure. The MAL value of LEBZAM multipliers are obtained as 7 and 16 for the hidden and output layers. Also, AAL values are calculated as 8 and 17, respectively, for the hidden layer and output layer. Furthermore, to obtain the performance of the proposed approximate level algorithm, different cases beyond this algorithm are given in Table 2. It must be noted that, due to the large search space of MAL and AAL values, the proposed algorithm only finds near-optimal values by acceptable variance. By investigation of Table 2 results, observe that the simultaneous use of approximate multipliers with the introduced approximate adders in [39], reduces the ANN hardware complexity significantly. The maximum gain on area and energy consumption reaches up to 6% and 48% using the approximate multipliers and adders with improving in the accuracy.

Table 3 indicate the efficiency of the LEBZAM in Multiplier ad adder for the network which is trained for SVHN database. According to the gate-level results, the LEBZAM multipliers and adders save up to 36% in energy consumption and 10% in occupied are, by only 0.29 deviation in HMR percentage.

Table 4 presents the gate-level results of ANN designs under the SMAC_NEURON architecture for 668-256-256-256-10 architecture. According to the table result, exploiting approximate multipliers and adders yields up to 39%, and 5% save in energy and area, respectively, with a small degradation in the accuracy.

Table 5 presents the gate-level results of ANN designs under the SMAC_ANN architecture where the single exact multiplier and adder in the MAC block is replaced by the approximate one. Although there exists only one multiplier

and adder to be replaced, the proposed approximate units lead to the largest gains in energy consumption.

It is worth noting that the complexity of control units overwhelms the complexity of the whole design in $SMAC\_ANN$, whereas the ratio of computation parts to control parts is negligible. As a result, the area is an unreliable metric for comparing the effectiveness of various approximate approaches. As a result, the area results are not included in Table 5 to avoid confusion. Conversely, the energy parameter shows the efficiency of computation units in $SMAC\_ANN$ architecture. To obtain a better performance, the investigated blocks must possess a better function in terms of delay and consumed power simultaneously.

According to Table 2 and Table 5 results, $SMAC\_ANN$ takes a much longer latency compared to the $SMAC\_NEURON$ as we expected. Also, the energy consumption increase due to the increased control complexity; additionally, the single multiplier and adder bit lengths are the maximum possible bit length of the network for $SMAC\_ANN$. Contrarily, for $SMAC\_NEURON$, multipliers and adders bit lengths are based on the corresponding neuron weight values.

6 Conclusion

In this paper, we presented hardware efficient implementation of ANN designs under the time-multiplexed architecture using approximate adders and multipliers. We also introduced an approximate multiplier, which leads to a significant reduction in area and energy consumption in the ANN design when compared to the previously proposed approximate multipliers. Also, we showed that exploiting proper approximate adders based on the employed multipliers can reduce the complexity of the structure without changing the accuracy. To exploit the proposed multipliers and adders in ANNs structure based on the desired accuracy, we offered the approximate level as a novel error metric. The generation of the approximate arithmetic units based on this error metric can be done in linear times for different bit-width inputs as opposed to the other methods. According to the experimental results, the introduced metric has a linear relationship with ANN accuracy. Furthermore, we proposed an algorithm to determine the approximate level of multipliers and adders by considering the desired accuracy. Experimental results clearly show that the use of approximate adders and multipliers in the ANN designs reduces the design complexity significantly with the same hardware accuracy compared to the ANN designs using exact adders and multipliers.

References

[1] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G. Nam, B. Taba, M. Beakes, B. Brezzo, J. B. Kuang, R. Manohar, W. P. Risk, B. Jackson, and D. S. Modha. Truenorth: Design and tool flow of a 65 mw

1 million neuron programmable neurosynaptic chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(10): 1537–1557, 2015.

[2] L. Aksoy, E. da Costa, P. Flores, and J. Monteiro. Exact and approximate algorithms for the optimization of area and delay in multiple constant multiplications. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(6):1013–1026, June 2008. ISSN 0278-0070. doi: 10.1109/TCAD.2008.923242.

[3] Levent Aksoy, Sajjad Parvin, Mohammadreza Esmali Nojehdeh, and Mustafa Altun. Efficient time-multiplexed realization of feedforward artificial neural networks. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, 2020. doi: 10.1109/ISCAS45731.2020. 9181002.

[4] F. Alimoglu and E. Alpaydin. Combining multiple representations and classifiers for pen-based handwritten digit recognition. In *International Conference on Document Analysis and Recognition*, pages 637–640, 1997.

[5] M. S. Ansari, V. Mrazek, B. F. Cockburn, L. Sekanina, Z. Vasicek, and J. Han. Improving the accuracy and hardware efficiency of neural networks using approximate multipliers. *IEEE Transactions on Very Large Scale Integration Systems*, 28(2):317–328, 2020.

[6] Adnène Arbi. Novel traveling waves solutions for nonlinear delayed dynamical neural networks with leakage term. *Chaos, Solitons & Fractals*, 152:111436, 2021.

[7] Adnène Arbi and Najeh Tahri. Almost anti-periodic solution of inertial neural networks model on time scales. In *MATEC Web of Conferences*, volume 355. EDP Sciences, 2022.

[8] Adnène Arbi and Najeh Tahri. Almost anti-periodic solution of inertial neural networks model on time scales. In *MATEC Web of Conferences*, volume 355. EDP Sciences, 2022.

[9] Adnene Arbi, Chaouki Aouiti, and Abderrahmane Touati. Uniform asymptotic stability and global asymptotic stability for time-delay hopfield neural networks. In *IFIP International Conference on Artificial Intelligence Applications and Innovations*, pages 483–492. Springer, 2012.

[10] Adnene Arbi, Jinde Cao, and Ahmed Alsaedi. Improved synchronization analysis of competitive neural networks with time-varying delays. *Nonlinear Analysis: Modelling and Control*, 23(1):82–107, 2018.

[11] J. V. Arthur, P. A. Merolla, F. Akopyan, R. Alvarez, A. Cassidy, S. Chandra, S. K. Esser, N. Imam, W. Risk, D. B. D. Rubin, R. Manohar, and D. S. Modha. Building block of a programmable neuromorphic substrate: A digital neurosynaptic core. In *The 2012 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2012.

[12] J. V. Arthur, P. A. Merolla, F. Akopyan, R. Alvarez, A. Cassidy, S. Chandra, S. K. Esser, N. Imam, W. Risk, D. B. D. Rubin, R. Manohar, and D. S. Modha. Building block of a programmable neuromorphic substrate: A digital neurosynaptic core. In *International Joint Conference on Neural*

Networks (IJCNN), pages 1–8, 2012. doi: 10.1109/IJCNN.2012.6252637.

[13] A. Bernasconi and V. Ciriani. 2-spp approximate synthesis for error tolerant applications. In *Euromicro Conference on Digital System Design*, pages 411–418, 2014.

[14] B. D. Brown and H. C. Card. Stochastic neural computation. i. computational elements. *IEEE Transactions on Computers*, 50(9):891–905, Sep. 2001. ISSN 0018-9340. doi: 10.1109/12.954505.

[15] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam. Dadiannao: A machine-learning supercomputer. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 609–622, 2014.

[16] Jian Cheng, Pei-song Wang, Gang Li, Qing-hao Hu, and Han-qing Lu. Recent advances in efficient computation of deep convolutional neural networks. *Frontiers of Information Technology & Electronic Engineering*, 19(1):64–77, Jan 2018.

[17] R. Ding, Z. Liu, R. D. Blanton, and D. Marculescu. Quantized deep neural networks for energy efficient hardware-based inference. In *Asia and South Pacific Design Automation Conference*, pages 1–8, 2018.

[18] M. Esmali Nojehdeh, L. Aksoy, and M. Altun. Efficient hardware implementation of artificial neural networks using approximate multiply-accumulate blocks. In *2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 96–101, 2020.

[19] V. Gupta, D. Mohapatra, S. P. Park, A. Raghunathan, and K. Roy. Impact: Imprecise adders for low-power approximate computing. In *IEEE/ACM International Symposium on Low Power Electronics and Design*, pages 409–414, 2011.

[20] J. Han and M. Orshansky. Approximate computing: An emerging paradigm for energy-efficient design. In *European Test Symposium*, pages 1–6, 2013.

[21] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, Upper Saddle River, NJ, 1999.

[22] J. L. Holi and J. . Hwang. Finite precision error analysis of neural network hardware implementations. *IEEE Transactions on Computers*, 42(3):281–290, March 1993. ISSN 0018-9340. doi: 10.1109/12.210171.

[23] M. Horowitz. Computing's energy problem (and what we can do about it). In *IEEE International Solid-State Circuits Conference*, 2014.

[24] Riya Jain and Neeta Pandey. Approximate karatsuba multiplier for error-resilient applications. *AEU - International Journal of Electronics and Communications*, 130:153579, 2021. ISSN 1434-8411. doi: https://doi.org/10.1016/j.aeue.2020.153579. URL https://www.sciencedirect.com/science/article/pii/S1434841120327837.

[25] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. at&t labs, 2010.

[26] E. H. Lee, D. Miyashita, E. Chai, B. Murmann, and S. S. Wong. Lognet: Energy-efficient neural networks using logarithmic compleutation. In *2017*

IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 5900–5904, March 2017. doi: 10.1109/ICASSP.2017.7953288.

[27] S. K. Lee, P. N. Whatmough, D. Brooks, and G. Wei. A 16-nm always-on dnn processor with adaptive clocking and multi-cycle banked srams. *IEEE Journal of Solid-State Circuits*, 54(7):1982–1992, 2019.

[28] G. Li, F. Li, T. Zhao, and J. Cheng. Block convolution: Towards memory-efficient inference of large-scale cnns on fpga. In *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1163–1166, March 2018. doi: 10.23919/DATE.2018.8342188.

[29] Haoxiang Li, Zhe Lin, Xiaohui Shen, Jonathan Brandt, and Gang Hua. A convolutional neural network cascade for face detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 5325–5334, 2015.

[30] Q. Li, W. Cai, X. Wang, Y. Zhou, D. D. Feng, and M. Chen. Medical image classification with convolutional neural network. In *International Conference on Control Automation Robotics Vision*, pages 844–848, 2014.

[31] Karri Manikantta Reddy, M.H. Vasantha, Y.B. Nithin Kumar, and Devesh Dwivedi. Design and analysis of multiplier using approximate 4-2 compressor. *AEU - International Journal of Electronics and Communications*, 107:89–97, 2019. ISSN 1434-8411. doi: https://doi.org/10.1016/j.aeue.2019.05.021. URL https://www.sciencedirect.com/science/article/pii/S1434841118330085.

[32] Mohammad Mirzaei and Siamak Mohammadi. Low-power and variation-aware approximate arithmetic units for image processing applications. *AEU - International Journal of Electronics and Communications*, 138:153825, 2021. ISSN 1434-8411. doi: https://doi.org/10.1016/j.aeue.2021.153825. URL https://www.sciencedirect.com/science/article/pii/S1434841121002223.

[33] Janardan Misra and Indranil Saha. Artificial neural networks in hardware: A survey of two decades of progress. *Neurocomputing*, 74(1):239 – 255, 2010.

[34] V. Mrazek, R. Hrbacek, Z. Vasicek, and L. Sekanina. Evoapproxsb: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods. In *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 258–261, 2017.

[35] S. H. Nawab, A. V. Oppenheim, A. P. Chandrakasan, J. M. Winograd, and J. T. Ludwig. Approximate Signal Processing. *Journal of VLSI signal processing systems for signal, image and video technology*, 75:177 – 200, 1997.

[36] N. Nedjah, R. M. da Silva, L. M. Mourelle, and M. V. C. da Silva. Dynamic MAC-based architecture of artificial neural networks suitable for hardware implementation on FPGAs. *Neurocomputing*, 72(10):2171 – 2179, 2009.

[37] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised

feature learning. 2011.

[38] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. In *IEEE International Conference on Computer Vision*, pages 1520–1528, December 2015.

[39] Mohammadreza Esmali Nojehdeh and Mustafa Altun. Systematic synthesis of approximate adders and multipliers with accurate error calculations. *Integration*, 70:99 – 107, 2020. ISSN 0167-9260. doi: https://doi.org/10.1016/j.vlsi.2019.10.001.

[40] Mohammadreza Esmali Nojehdeh, Sajjad Parvin, and Mustafa Altun. In *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 402–405. IEEE, 2021.

[41] H. Park and T. Kim. Structure optimizations of neuromorphic computing architectures for deep neural network. In *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 183–188, 2018.

[42] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *Computer Vision - ECCV 2016*, pages 525–542, Cham, Switzerland, 2016. Springer. doi: 10.1007/978-3-319-46448-0_32. URL https://doi.org/10.1007/978-3-319-46448-0_32.

[43] Zulqurnain Sabir, Muhammad Asif Zahoor Raja, Sharifah E Alhazmi, Manoj Gupta, Adnène Arbi, and Isa Abdullahi Baba. Applications of artificial neural network to solve the nonlinear covid-19 mathematical model based on the dynamics of siq. *Journal of Taibah University for Science*, 16(1):874–884, 2022.

[44] S. S. Sarwar, S. Venkataramani, A. Raghunathan, and K. Roy. Multiplierless artificial neurons exploiting error resiliency for energy-efficient neural computing. In *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 145–150, 2016.

[45] M. Schaffner, F. Gurkaynak, A. Smolic, H. Kaeslin, and L. Benini. An approximate computing technique for reducing the complexity of a direct-solver for sparse linear systems in real-time video processing. In *Design Automation Conference (DAC)*, pages 1–6, 2014.

[46] H. Tann, S. Hashemi, R. I. Bahar, and S. Reda. Hardware-software codesign of accurate, multiplier-free deep neural networks. In *Design Automation Conference (DAC)*, pages 28:1–28:6, 2017.

[47] H. Tann, S. Hashemi, R. I. Bahar, and S. Reda. Hardware-software codesign of accurate, multiplier-free deep neural networks. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2017. doi: 10.1145/3061639.3062259.

[48] The MathWorks Inc. *Deep Learning Toolbox*. Natick, Massachusetts, United States, 2020. URL https://www.mathworks.com/help/deeplearning/.

[49] Y. Yamada, T. Sano, Y. Tanabe, Y. Ishigaki, S. Hosoda, F. Hyuga, A. Moriya, R. Hada, A. Masuda, M. Uchiyama, M. Jobashi, T. Koizumi, T. Tamai, N. Sato, J. Tanabe, K. Kimura, Y. Ojima, R. Murakami, and

T. Yoshikawa. A 20.5 tops multicore soc with dnn accelerator and image signal processor for automotive applications. *IEEE Journal of Solid-State Circuits*, 55(1):120–132, 2020. doi: 10.1109/JSSC.2019.2951391.

[50] Z. Yang, A. Jain, J. Liang, J. Han, and F. Lombardi. Approximate xor/xnor-based adders for inexact computing. In *IEEE International Conference on Nanotechnology*, pages 690–693, 2013.

[51] Qian Zhang, Ting Wang, Ye Tian, Feng Yuan, and Qiang Xu. Approxann: An approximate computing framework for artificial neural network. In *Design, Automation and Test in Europe Conference and Exhibition*, pages 701–706, 2015.