Efficient Time-Multiplexed Realization of Feedforward Artificial Neural Networks

Levent Aksoy, Sajjad Parvin, Mohammadreza Esmali Nojehdeh and Mustafa Altun
Department of Electronics and Communication Engineering, Istanbul Technical University
34469, Maslak, Istanbul, Turkey
Email: {aksoyl, nojehdeh, altunmus}@itu.edu.tr, sajadparvin.stu93@gmail.com

***Abstract*—This paper presents techniques and design structures to reduce the time-multiplexed hardware complexity of a feed-forward artificial neural network (ANN). After the weights of ANN are determined in a training phase, in a post-training stage, initially, the minimum quantization value used to convert the floating-point weights to integers is found. Then, the integer weights related to each neuron are tuned to reduce the hardware complexity in the time-multiplexed design avoiding a loss on the ANN accuracy in hardware. Also, at each layer of ANN, the multiplications of integer weights by an input variable at each time are realized under the shift-adds architecture using a minimum number of adders and subtractors. It is observed that the application of the post-training stage yields a significant reduction in area, latency, and energy consumption on the time-multiplexed designs including multipliers. Moreover, the multiplierless design of ANN whose weights are found in the post-training stage leads to a further reduction in area and energy consumption, increasing the latency slightly.**

I. Introduction

Artificial neural network (ANN) is a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs [1]. ANNs have been applied to a wide range of problems, such as classification and pattern recognition, and have been realized in different design platforms, such as analog, digital, and hybrid very large scale integrated (VLSI) circuits, field programmable gate-arrays (FPGAs), and neuro-computers [2].

As illustrated in Fig. 1(a), the fundamental unit of an ANN, called neuron, sums the multiplication of input variables by weights, adds the bias value to this summation, and propagates this result to the activation function which aims to limit the amplitude of the output of the neuron [3]. In mathematical terms, the neuron is described as $y = \sum_{i=1}^{n} w_i x_i$ and $z = \phi(y + b)$. Fig. 1(b) shows an example on the architecture of an ANN design, including input, hidden, and output layers, where each circle denotes a neuron.

Observe from Fig. 1 that the hardware complexity of an ANN is dominated by the multiplication of weights by input variables. As the number of neurons increases, the area of the parallel design increases dramatically, limiting its applications on design platforms with a limited number of computing resources, such as FPGAs, and on designs having a strict area requirement [4]. In order to reduce the design area, taking into account an increase in latency, the ANN can be implemented in a time-multiplexed structure re-using the computing resources. Fig. 2 presents the time-multiplexed

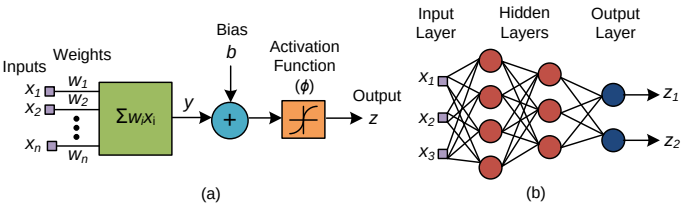

Fig. 1. (a) Artificial neuron; (b) ANN with two hidden layers.

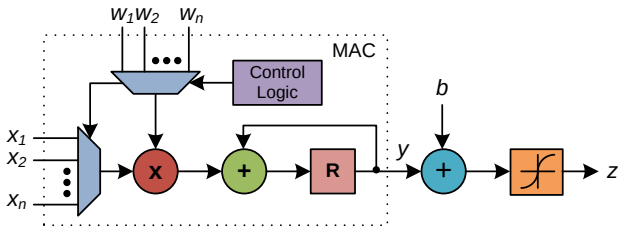

Fig. 2. Multiply-accumulate (MAC) block in the neuron computation.

realization of a neuron using a multiply-accumulate (MAC) block where the multiplication of a weight by an input variable is realized at a time synchronized by the control block, which is actually an up-counter, and is added by the accumulated value stored in the register *R*. In this figure, the clock and reset signals are omitted for the sake of clarity. Note that the number of inputs (or weights) plus 1, *i.e.*, $n + 1$, clock cycles are required to compute the neuron output. 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.*, $y = \sum_{i=1}^{n} w_i x_i$.

Observe from Fig. 2 that the design complexity in the MAC block of each neuron, and consequently, the ANN hardware complexity, can be reduced if small weight values are used, taking the ANN accuracy into consideration. Hence, in this paper, we introduce a post-training stage where initially the minimum quantization value used to convert the floating-point weights to integers is found, sacrificing a little loss in the ANN accuracy in hardware. Note that when all the weights associated with a neuron are multiples of 2^k, where $k > 0$, the inner product can be realized as $y = (\sum_{i=1}^{n} c_i x_i) \ll k$, where $c_i = w_i/2^k$ and $\ll$ stands for the left shift operation. Parallel shifts can be implemented using only wires in hardware without representing any area cost. Thus, the bit-widths of constants to be multiplied by the input variables in the MAC block, and consequently, the sizes of the multiplier, adder, and

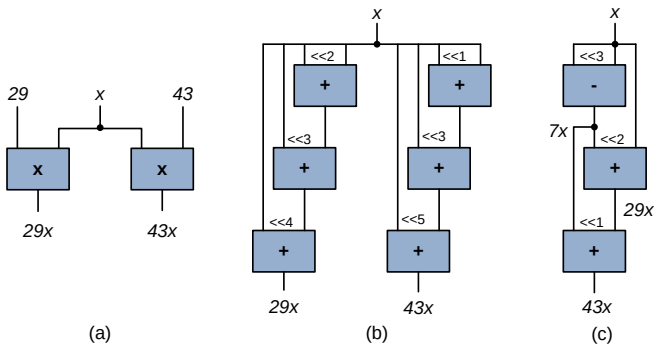

Fig. 3. (a) Constant multiplications of $29x$ and $43x$; their shift-adds realizations: (b) DBR method [5]; (c) the exact algorithm [6].

register can be reduced. Hence, in the post-training stage, we also explore all the weights associated with each neuron in such a way that they can be set to multiples of 2^k value, where k is maximum, without losing any accuracy in hardware and exceeding the bit-width of the inner product. Moreover, we introduce the multiplierless realization of the time-multiplexed ANN design. To do so, in each layer, the multiplications of weights by an input variable at each time are implemented using a multiple constant multiplication (MCM) block. Since the weights are determined beforehand, these constant multiplications can be realized using only shift, addition, and/or subtraction operations under the shift-adds architecture [5]. The multiplierless realization of the MCM block is found using the exact algorithm of [6] with a minimum number of adders/subtractors. Results show that the application of the post-training stage yields a significant reduction in hardware complexity and the multiplierless design of ANN with the weights found in the post-training stage reduces the complexity further when compared to designs using multipliers.

The rest of this paper is organized as follows: Section II presents the preliminaries. Section III introduces the time-multiplexed realization of ANNs. Results are given in Section IV and finally, Section V concludes the paper.

II. Preliminaries

A. Multiplierless Design of Multiple Constant Multiplications

Multiplication of constants by variable(s) is a ubiquitous and crucial operation in many applications, such as digital signal processing (DSP), cryptography, and compilers [7]. The MCM operation realizes the multiplication of multiple constants by an input variable x. A straight-forward shift-adds design technique, called the digit-based recoding (DBR) [5], can realize constant multiplications in two steps given as follows: i) define the constants under a particular number representation, such as binary or canonical signed digit (CSD) [8]; ii) for the non-zero digits in the representation of constants, shift the input variable according to digit positions and add/subtract the shifted variables with respect to digit values. As a simple example, consider $29x$ and $43x$ shown in Fig. 3(a). Fig. 3(b) presents the solution of the DBR method with a total of 6 adders found when the constants are defined under the binary representation as $29x = (11101)_{bin}x$ and $43x = (101011)_{bin}x$. The number of adders/subtractors in the MCM block can be further reduced by maximizing the sharing of common partial products among the constant multiplications [6], [9], [10]. Returning to our example, the exact algorithm of [6] finds a solution with a minimum number of 3 operations as shown in Fig. 3(c).

B. Related Work

Time-multiplexed constant multiplication (TMCM) block realizes the multiplication of a constant variable selected at a time by an input variable x. Among alternative realizations, its implementation using an MCM block and multiplexer is described in [11], [12].

A delay-efficient MAC structure, which uses accumulators and carry-save adders, was introduced in [13] to reduce its high latency. Efficient implementation of ANN designs using MAC blocks on FPGAs was presented in [4].

For the multiplierless realization of neural networks, in [14], [15], the weights of ANNs are 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.

III. Time-Multiplexed ANN Design

The design procedure has three steps: i) given the ANN structure, including the number of inputs, outputs, hidden layers, neurons in the hidden layers, and the type of activation function of neurons in each layer, train the ANN using state-of-art techniques and find the weight and bias values; ii) in a post-training stage, using a validation data set, determine the minimum quantization value for the weights and biases taking into account the ANN accuracy in hardware, convert the floating point weight and bias values to integers, and tune them such that the hardware complexity of the time-multiplexed design is reduced while keeping and/or improving the ANN accuracy in hardware; iii) describe the time-multiplexed ANN design in hardware and verify the design. Below, these steps are described in detail.

A. Training

We developed our training tool to determine the weights and bias values. It includes the conventional and stochastic gradient descent methods and the Adam optimizer [16] as an iterative optimization algorithm. It has different weight initialization techniques, such as Xavier [17], He [18], and a fully random method. It includes several stopping criteria, *e.g.*, the number of iterations, early stopping using validation data set, and saturation of loss function. It can define a number of activation functions for neurons in each layer, namely sigmoid, hyperbolic tangent, hard sigmoid, hard hyperbolic tangent (hardtanh), linear, rectified linear unit, and softmax [19].

B. Hardware-aware Post-training

Since the floating-point multiplication and addition operations occupy larger area and are less energy efficient than its integer counterparts [20], the floating-point weight values found during training are converted to integers. We aim to reduce the bit-widths of weights and explore possible weight

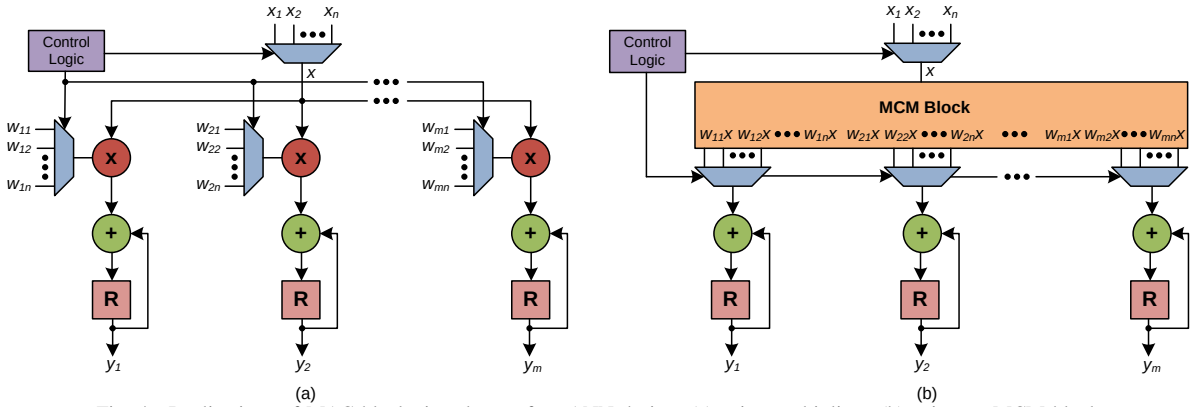

Fig. 4. Realizations of MAC blocks in a layer of an ANN design: (a) using multipliers; (b) using an MCM block.

values which may lead to a decrease in the design area and/or yield an increase in the ANN accuracy in hardware. To do so, initially, we generate a validation data set, which is used to compute the hardware accuracy, by moving 30% of the training data set to the validation data set randomly. The minimum quantization value, which is used to convert the floating-point weight values to integers, is found as follows:

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 finding 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 $ha(q) > 0$ and $ha(q) - ha(q-1)$ is greater than 0.1%, go to Step 2.
6) Otherwise, return q as the minimum quantization value.

Observe that we sacrifice maximum 0.1% loss in the ANN accuracy in hardware computed on the validation data set in order to use small size weights. After the weight values are converted to integers using the minimum quantization value q, for each neuron, we aim to maximize the smallest left shift value among all its weights, denoted as sls, while avoiding a decrease in the ANN accuracy in hardware. As a simple example, the sls value for the integer values $12 = 3 \ll 2$, $14 = 7 \ll 1$, and $16 = 1 \ll 4$, is computed as 1. The tuning procedure is described as follows:

1) Set $ha(q)$ computed in finding the minimum quantization value to the best ANN accuracy in hardware, bha.
2) For each neuron in the ANN design, n_i, compute the sls value for its weights.
   a) For each weight associated with n_i, w_{n_ij}, find its largest left shift value, lls.
   b) If lls is equal to sls, determine the first possible weight as $pw_1 = w_{n_ij} - (w_{n_ij} \mod 2^{lls+1})$. If the bit-width of pw_1 is not greater than the maximum bit-width of weights associated with n_i, compute the ANN accuracy in hardware, ha_1, when w_{n_ij} is replaced by pw_1. Determine the second possible weight as $pw_2 = pw_1 + 2^{lls+1}$ and compute the ANN accuracy in hardware, ha_2, similarly.
   c) If the maximum of ha_1 and ha_2 is equal to or greater than bha, then replace w_{n_ij} by the possible weight that leads to the maximum ANN accuracy in hardware and update bha.
   d) Otherwise, assuming that w_{n_ij} is replaced by the possible weight that leads to the maximum ANN accuracy in hardware, change the bias value of the neuron, b_{n_i}, in between $[b_{n_i} - 4, b_{n_i} + 4]$ and compute the ANN accuracy in hardware. If the ANN accuracy in hardware in one of these cases is equal to or better than bha, update the values of w_{n_ij}, b_{n_i}, and bha accordingly.
3) If sls value of any neuron is improved, go to Step 2.
4) Otherwise, return the weight and bias values.

C. Hardware Design and Multiplierless Design Optimization

After the integer weight and bias values are determined, the ANN design is built taking into account the ANN structure given to the training tool. In addition, the bit-widths of inputs and outputs of the ANN design are specified. In the time-multiplexed design, the MAC block can be realized with multipliers and also, with an MCM block. The realization of MAC blocks in a layer of an ANN design with n inputs and m outputs is illustrated in Fig. 4(a). In the multiplierless design, all the weight multiplications are computed in the MCM block and they are diverted to the corresponding adders using multiplexers as shown in Fig. 4(b). Rather than using an MCM block for each neuron, a single MCM block, which realizes the multiplication of all the weights in a layer by an input variable, is used to increase the sharing of partial products, and thus, to reduce the required number of adders/subtractors. The exact algorithm of [6] is used to find the shift-adds realization of the MCM block using a minimum number of adders/subtractors.

IV. Experimental Results

The pen-based handwritten digit recognition problem [21] was used as an application. In the convolutional neural network design of this application, we implemented 5 feedforward ANN structures with different number of hidden layers and number of neurons in the hidden layers, denoted as p_i–p_{h_1}–…–p_{h_n}–p_o, where p_i and p_o stand for the number of inputs and outputs, respectively and p_{h_j}, where $1 \leq j \leq n$,

TABLE I
SUMMARY OF RESULTS OF ANN DESIGNS WITHOUT POST-TRAINING.

Structure	Training Details		PARALLEL				TM-MUL				TM-MCM				
	sta	hta	A	L	P	E	A	L	P	E	add	A	L	P	E
16-10	84.6	83.3	19779	3.3	8.6	28.4	6098	55.0	0.6	35.4	110	9222	63.5	0.9	57.7
16-10-10	94.1	92.9	30765	6.2	15.8	98.4	11682	96.7	1.2	113.9	169	15854	110.7	1.2	128.4
16-16-10	96.0	94.7	42464	6.5	24.0	155.7	14301	127.3	1.5	186.3	212	18941	148.4	1.4	213.4
16-10-10-10	94.7	91.9	39952	9.0	26.1	235.2	16697	145.3	1.6	226.8	216	19909	153.4	1.1	172.1
16-16-10-10	96.6	94.6	50737	7.3	28.7	209.3	19100	168.3	1.7	293.1	232	23001	183.3	1.2	228.4
Average	93.2	91.5	36739.4	6.5	20.6	145.4	13575.6	118.5	1.3	171.1	187.8	17385.4	131.9	1.2	160.0

TABLE II
SUMMARY OF RESULTS OF ANN DESIGNS WITH POST-TRAINING.

Structure	Training Details		PARALLEL				TM-MUL				TM-MCM				
	q	hta	A	L	P	E	A	L	P	E	add	A	L	P	E
16-10	5	86.6	8943	2.8	2.7	7.4	4278	47.2	0.4	19.0	11	3633	47.8	0.4	17.8
16-10-10	7	93.5	16838	5.4	8.6	46.0	8676	85.8	0.9	76.1	34	7648	91.5	0.6	57.5
16-16-10	7	95.9	20063	5.5	10.7	58.6	10014	114.0	1.0	112.3	33	8358	122.4	0.8	101.2
16-10-10-10	7	93.5	22994	7.9	13.7	108.4	13020	121.2	1.2	150.1	43	10743	134.6	1.0	132.1
16-16-10-10	7	95.6	25181	5.5	13.2	73.2	13368	147.8	1.3	199.3	29	11033	152.4	1.0	152.1
Average	6.6	93.0	18803.8	5.4	9.8	58.7	9871.2	103.2	1.0	111.4	30.0	8283.0	109.7	0.8	92.1

indicates the number of neurons in the j^{th} hidden layer. The ANNs were trained using 7494 data while the activation function of each neuron in the hidden and output layers was *hardtanh* and *sigmoid*, respectively. During the training, the Adam optimization algorithm was used and the weights were initialized using the Xavier technique. The training was stopped whenever the accuracy on the validation data set is decreased. Due to the randomness in our tool, the ANN was trained 30 times and the weights, that lead to the best software test accuracy, denoted as *sta*, were determined.

In this work, we present the results of ANN designs implemented in three different architectures: i) parallel; ii) time-multiplexed using multipliers, TM-MUL; iii) time-multiplexed using MCM block(s), TM-MCM. In the parallel design, to make a fair comparison, flip-flops were added to the outputs of the ANN design. In parallel and time-multiplexed designs using multipliers, the constant multiplications were described in a behavioral fashion. In all the ANN designs, the bit-widths of inputs and outputs were set to 8. The ANN designs were described in Verilog and were synthesized using the Cadence RTL Compiler with the TSMC 40nm design library.

Table I presents the gate-level results of ANN designs implemented without applying the post-training stage. The quantization value used to convert the floating-point weight values to integer was 8. In this table, *hta* denotes the hardware test accuracy, *A*, *L*, *P*, and *E* stand respectively for total area in μm^2, latency in ns, power dissipation in mW, and energy consumption in pJ, and *add* indicates the total number of adders/subtractors in the MCM block(s). The latency is computed as the multiplication of the minimum clock period by the number of clock cycles to obtain the ANN output. The minimum clock period was found using the retiming technique in the synthesis tool iteratively. The switching activity data required for the computation of power dissipation was generated using 3498 test data in simulation. This test data set was used to verify the ANN design. Energy consumption is computed as the multiplication of latency and power dissipation.

Observe from Table I that the hardware test accuracy is close to the software test accuracy. Note that the difference is mainly due to the bit-widths of inputs and outputs, the quantization value used to convert floating-point weights to integers, and the weights themselves. Observe from Table I that the time-multiplexed realization of ANNs using multipliers leads to a $2.7\times$ area reduction on average when compared to the parallel realization. However, the latency and energy consumption are increased by 18.2 and 1.2 times on average, respectively. On the other hand, the multiplierless time-multiplexed designs occupy larger area and have greater latency when compared to the time-multiplexed designs using multipliers. This is because the weights have large and diverse values, requiring a large number of adders/subtractors to replace a small number of multipliers at each layer. For example, 110 adders/subtractors are required to replace 10 multipliers in the 16-10 structure.

Table II presents the gate-level results of ANN designs implemented after the post-training stage is applied. In this table, q denotes the determined minimum quantization value. Note that the post-training leads to high hardware test accuracy, where it even exceeds the software test accuracy in the 16-10 structure, and yields ANN designs with small hardware complexity. Observe from Table II that the time-multiplexed realization of ANNs using multipliers leads to a $1.9\times$ area reduction on average when compared to the parallel realization. The multiplierless time-multiplexed designs occupy less area and consume less energy than the time-multiplexed designs using multipliers. This is because the weights determined during the post-training stage have small and very similar values when they are shifted left. Note that the latency in multiplierless designs is increased due to a large number of adders/subtractors in series in MCM block(s).

V. CONCLUSIONS

This paper presented efficient techniques to reduce the hardware complexity of a time-multiplexed feedforward ANN design. It introduced a post-training stage where the size and values of constant weights are explored to reduce the hardware complexity. It also presented the multiplierless realization of the time-multiplexed ANN design where the number of addition/subtraction operations is optimized. Experimental results indicated that the proposed techniques yield a significant reduction in design area when compared to the parallel and time-multiplexed designs using multipliers.

ACKNOWLEDGMENT

This work is funded by TUBITAK-1001 project #117E078.

REFERENCES

[1] R. Hecht-Nielsen, *Neurocomputing*. Addison-Wesley, 1990.

[2] J. Misra and I. Saha, "Artificial neural networks in hardware: A survey of two decades of progress," *Neurocomputing*, vol. 74, no. 1-3, pp. 239–255, 2010.

[3] S. Haykin, *Neural Networks: A Comprehensive Foundation*. Prentice-Hall, 1999.

[4] 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*, vol. 72, no. 10, pp. 2171 – 2179, 2009.

[5] M. Ercegovac and T. Lang, *Digital Arithmetic*. Morgan Kaufmann, 2003.

[6] L. Aksoy, E. O. Gunes, and P. Flores, "Search algorithms for the multiple constant multiplications problem: Exact and approximate," *Microprocessors and Microsystems, Embedded Hardware Design (MICPRO)*, vol. 34, no. 5, pp. 151–162, 2010.

[7] O. Gustafsson, "Lower bounds for constant multiplication problems," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 54, no. 11, pp. 974–978, 2007.

[8] R. Hartley, "Subexpression sharing in filters using canonic signed digit multipliers," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 43, no. 10, pp. 677–688, 1996.

[9] O. Gustafsson, "A difference based adder graph heuristic for multiple constant multiplication problems," in *International Symposium on Circuits and Systems (ISCAS)*, 2007, pp. 1097–1100.

[10] M. P. Y. Voronenko, "Multiplierless multiple constant multiplication," *ACM Transactions on Algorithms*, vol. 3, no. 2, 2007.

[11] S. Demirsoy, I. Kale, and A. Dempster, "Reconfigurable multiplier constant blocks: Structures, algorithm and applications," *Springer Circuits, Systems and Signal Processing*, vol. 26, no. 6, pp. 793–827, 2007.

[12] L. Aksoy, P. Flores, and J. Monteiro, "Multiplierless design of folded DSP blocks," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 20, no. 1, pp. 14:1–14:24, 2014.

[13] Y. Seo and D. Kim, "A new vlsi architecture of parallel multiplier–accumulator based on radix-2 modified booth algorithm," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 18, no. 2, pp. 201–208, 2010.

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

[15] 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 (ASP-DAC)*, 2018, pp. 1–8.

[16] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv e-prints*, 2014, arXiv:1412.6980.

[17] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *International Conference on Artificial Intelligence and Statistics*, 2010, pp. 249–256.

[18] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *arXiv e-prints*, 2015, arXiv:1502.01852.

[19] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation functions: Comparison of trends in practice and research for deep learning," *arXiv e-prints*, 2018, arXiv:1811.03378.

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

[21] F. Alimoglu and E. Alpaydin, "Combining multiple representations and classifiers for pen-based handwritten digit recognition," in *International Conference on Document Analysis and Recognition (ICDAR)*, 1997, pp. 637–640.