Design Methodology for Embedded Approximate Artificial Neural Networks

Adarsha Balaji†, Salim Ullah‡, Anup Das†, and Akash Kumar‡

† Drexel University, Philadelphia, USA
‡ Technische Universität Dresden, Germany

ABSTRACT

Artificial neural networks (ANNs) have demonstrated significant promise while implementing recognition and classification applications. The implementation of pre-trained ANNs on embedded systems requires representation of data and design parameters in low-precision fixed-point formats; which often requires retraining of the network. For such implementations, the multiply-accumulate operation is the main reason for resultant high resource and energy requirements. To address these challenges, we present *Rox-ANN*, a design methodology for implementing ANNs using processing elements (PEs) designed with low-precision fixed-point numbers and high performance and reduced-area approximate multipliers on FPGAs. The trained design parameters of the ANN are analyzed and clustered to optimize the total number of approximate multipliers required in the design. With our methodology, we achieve insignificant loss in application accuracy. We evaluated the design using a LeNet based implementation of the MNIST digit recognition application. The results show a 65.6%, 55.1% and 18.9% reduction in area, energy consumption and latency for a PE using 8-bit precision weights and activations and approximate arithmetic units, when compared to 16-bit full precision, accurate arithmetic PEs.

CCS CONCEPTS

• **Computing methodologies → Neural networks**; *Classification and regression trees*; • **Computer systems organization → Embedded hardware**; • **Hardware → Neural systems**.

KEYWORDS

Artificial neural networks (ANNs), approximate computing, FPGA

ACM Reference Format:
Adarsha Balaji, Salim Ullah, Anup Das, and Akash Kumar. 2019. Design Methodology for Embedded Approximate Artificial Neural Networks. In *Proc. of the Great Lakes Symposium on VLSI 2019 (GLSVLSI '19), May 9–11, 2019, Tysons Corner, VA, USA.* ACM, NY, NY, USA. 6 pages. https://doi.org/10.1145/3299874.3319490

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
GLSVLSI '19, May 9–11, 2019, Tysons Corner, VA, USA
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6252-8/19/05…$15.00
https://doi.org/10.1145/3299874.3319490

1 INTRODUCTION

Inspired by the human brain, artificial neural networks (ANNs) are computational models built using a system of interconnected computing nodes called neurons. Each neuron is associated with two input parameters, a weight vector (w) and an input vector (x). The output g_j of a neuron is defined by

$$g_i = \sigma\Big(\sum_{j=0}^{N-1} w_{i,j} x_i - \phi\Big) \quad (1)$$

where, $\sigma()$ is the activation function, N is the fan-in of the neuron j, w_{ij} is the weight of the synapse between the i-th and the j-th neurons, respectively, and ϕ is the bias added to the weighted sum. The function first computes the weighted sum of all its inputs, and then performs a non-linear activation on the computed sum to generate g_i.

Utilizing these neurons, ANN models consist of an input layer, one or more hidden layers and an output layer. Depending on the interconnect patterns of the neurons and the activation functions used, neural network models are broadly classified into feed-forward neural networks (FNN), recurrent neural networks (RNN) and convolutional neural networks (CNN). Figure. 1 shows the architecture of an FNN. During the training and inference phases, every neuron in the network computes a weighted sum of its corresponding inputs and utilizes a non-linear activation function to produce its output, as shown in Eq. 1.

Recent advances in ANNs have seen their large scale incorporation in a diverse set of applications such as synthesis, classification and recognition [4][10]. To achieve higher accuracy, present implementations of ANNs are made larger and more complex. Due to the massive number of multiply-accumulate (MAC) operations, used to compute the weighted sum at each neuron, training of ANNs is

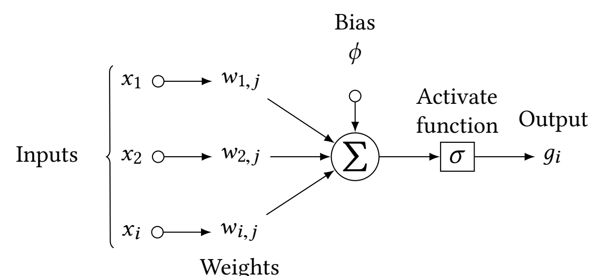

Figure 1: Design of an Artificial Neuron.

performed, preferably, using massive parallel computing platforms, such as Graphics processing units (GPUs).

For the utilization of ANNs in embedded systems, the ASIC-based ANN hardware accelerators aim to provide high performance and energy-efficient *inference engines*. As ANNs are naturally designed to be error resilient [18], several approximation techniques have been proposed in literature to optimize the hardware implementations of ANN to reduce energy consumption and improve the overall efficiency. These implementations, such as [2, 7, 9, 12, 17], mainly consider the quantization of data, such as activations and weights, to reduce the resources and energy demands of the implementations.

Besides using quantized data, some recent works have also explored the utilization of approximate arithmetic components for enhancing the performance of ASIC-based ANN accelerators. Du et al. [5] propose the use of approximate multipliers to design inexact neurons in neural networks. The multipliers proposed in this design use a logic minimization algorithm to reduce the runtime latency of the multiply operation. The proposed model exploits the error-tolerance of ANNs to achieve significant energy, latency and area gains. Venkataramani et al. [15] proposed a methodology of identifying error-resilient neurons based on the back-propagation gradients. For the error-resilient neurons, an approximation using precision modification and piecewise-linear approximation of activation function was applied to create an approximate neural network. Since training is by itself an error-healing process, after creating the approximate version, the NN is retrained. They also proposed a neuromorphic processing engine platform to determine the best trade-off between the precision and energy. In a similar approach, Zhang et al. [18] propose a method to identify critical neurons in a proposed architecture. The authors present a theoretical approach to detect critical neurons by studying the effect of small errors, introduced at the input, on the neuron's computation. The neurons are then ranked according to their criticality and three separate approximation techniques are introduced, memory access reduction, data and weight quantization and approximate arithmetic circuits.

However, ASIC-based hardware accelerators are constrained by their inability to support different types of neural networks and the limited size of the inferred network. To address these limitations, efforts have been made to implement reconfigurable ANN architectures and accelerators. Eyeriss [3] is a reconfigurable accelerator designed for CNNs that minimizes the energy cost for data movement. Considering the flexibility, high-parallelism and reconfigurability of FPGAs, recent works have also considered FPGA-based ANN accelerators [6, 11, 13, 16, 17]. However, many of the state-of-the-art FPGA-based implementations, such as [17], utilize High-level Synthesis Tools (HLS), such as provided by Vivado, to generate synthesizable RTL. However, the limitation of Vivado HLS to use only Vivado-provided IPs during the generation of RTL, makes it difficult to incorporate custom approximate IPs (arithmetic units) in the generated RTL. Other FPGA-based ANN accelerators, such as [6, 16] do not consider the approximate arithmetic modules during their proposed implementations.

In this paper we propose *RoxANN*, a methodology to design area and energy efficient ANN hardware using approximate arithmetic units on FPGAs. The methodology includes:

- the design of ANN processing elements (PEs) that support the utilization of approximate multipliers, thereby achieving significant area and energy savings;
- the use of quantized activations and weights to reduces the memory demand on the FPGA;
- a weight clustering technique, used post training, to incorporate weight sharing. With a fixed number of shared weights, we propose the use of *constant multipliers* in PEs, significantly reducing the area and resource demand on the FPGA.

The remainder of this work is organized as follows. We introduce our *Rox-ANN* design methodology in Section 2. We present our evaluation methodology in Section 3. We compare our approach against a fully precise and accurate ANN hardware implementation in Section 4. Finally, we conclude the paper in Section 5.

2 DESIGN METHODOLOGY

In this section, we present the FPGA-based Rox-ANN methodology. First, we propose the design of a generic processing element that supports the use of accurate/approximate arithmetic units, such as adders and multipliers, to compute the partial weighted sum of neurons. Next, we demonstrate the use of quantized activations and weights on the training and implementations phase of the neural network. Finally, we present a clustering based approach, used after the training phase of the neural network, to minimize the number of distinct weights on the network.

2.1 Processing Element

The proposed design of a generic PE, as shown in Figure 2, is optimized for the state-of-the-art Xilinx FPGAs (with 6-input lookup tables), however, it can also be implemented on FPGAs from other vendors, such as those provided by Intel. As described in Fig. 2(a), the neurons in the hidden layer and output layer of an ANN compute the weighted sum of all its corresponding activations. In a layered architecture, such as a Multilayer Perceptron (MLP), neurons that belong to the same hidden layer share all the inputs from the previous layer, stored in a vector x, and contains a weight for every synaptic connection, stored in a vector w. Each processing element consists of two dual-port block random access memories (BRAMs); one to store a subset of the activations $\hat{x}$ and the other to store a subset of the weights $\hat{w}$, respectively. The size of the BRAM can be configured based on the architecture of target FPGA platform, design constraints—such as energy, area and latency—and the size and type of neural network architecture being implemented on the FPGA.

Each PE contains two multipliers and two adders as shown in Fig. 2. Each multiplier and adder pair computes the partial weighted sum of the inputs $\hat{x}$ and the corresponding weights $\hat{w}$. The design of the PE is configurable, based on the demands and constraints of the neural network. Fig. 2(b) and Fig. 2(c) show two possible configurations of the proposed PE. The PE in Fig. 2(b) can compute the partial weighted sum of *two* separate neurons, with each multiplier-adder pair used to represent a single neuron. The two multiplier-adder pairs in a PE can also be configured to represent a single neuron as shown in Fig. 2(c). The configuration in Fig. 2(c) will have double the throughput of the design in Figure 2(b) for a single neuron. Similarly, depending upon target FPGA platform

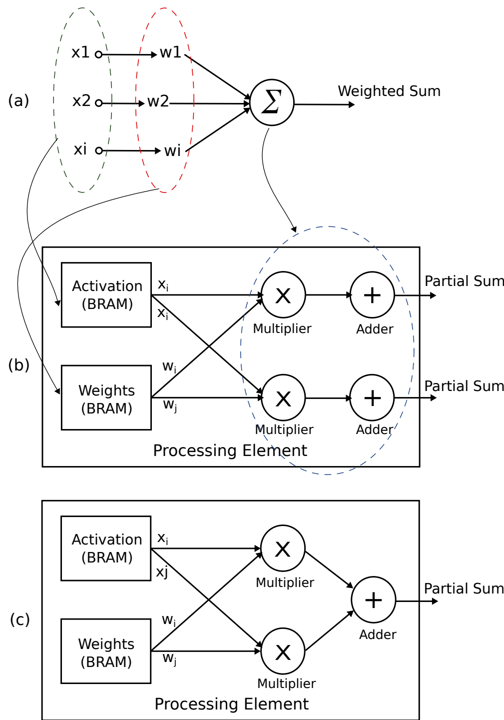

Figure 2: Implementation of a single processing element on the FPGA to compute the partial sum of single or multiple neurons.

and design requirements, a layer/neuron can be implemented by multiple concurrently executing PEs. This flexibility allows the designer to choose a PE configuration based on design constraints.

The proposed methodology also supports approximate and constant multipliers for the PE implementation. The modular and RTL based implementation of the proposed PE makes it very easy to instantiate high performance approximate multipliers, such as those provided in [14], to meet the implementation's performance constraints. The utilization of approximate multipliers, for PE implementation, also results in overall area savings, which can be used for instantiating more PEs on a smaller FPGA platform with limited resources. As discussed in Section 2.3, we perform weight clustering to significantly reduce the total number of distinct weights for a layer. Utilizing the resultant fixed clustered weights, our proposed methodology can configure the PE to use constant multipliers. With constant multipliers, there is no need to store the weight vector w and only one single BRAM is used by a PE. Besides significantly reducing the overall resource utilization of a PE, the constant multipliers-based PE offer very high performance as compared to the accurate/approximate multipliers-based PE implementations.

2.2 Data Quantization

Standard implementations of ANNs use 32-bit, or word-size floating point operators, activations and weights during the training and inference phases [1]. In order to reduce the memory required to

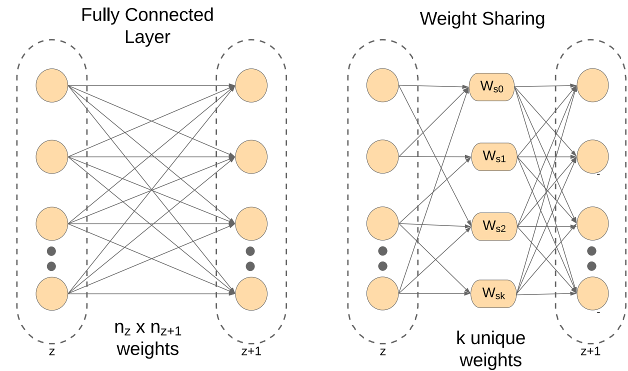

Figure 3: Proposed Weight Sharing technique compared to a Fully Connected layer.

store the weights, intermediate outputs and the memory accesses, we use 8-bit and 16-bit fixed point arithmetic operators, weights and activations during the training and inference of the neural network. Several works in literature [2] [8] have shown that quantization of operators and data, even below 8-bits, have little to no effect on the accuracy of the ANN application.

To validate this, we re-train and infer a standard MNIST handwritten digit recognition benchmark using 8-bit and 16-bit fixed precision operator. For the 8-bit and 16-bit fixed point operators, we use 1 bit and 2 bits for the integer part, respectively and 7-bits and 14-bits for the fractional part, respectively. It is important to note that the output of the arithmetic operator is truncated down to 8 and 16-bits respectively.

2.3 Weight Clustering

In a fully connected neural network, the number of weighted synapses between any two layers is represented as n_z x n_{z+1}, where z is the layer number and n is the number of neurons in the given layer. In order to reduce the memory required to store the respective weights on the PE and to minimize the latency due to memory reads, we propose a layer-wise, clustering based, *weight sharing technique*, to reduce the number of unique weights in the network.

We propose the use of Kmeans, an unsupervised learning algorithm, to divide a one-dimensional weight matrix $W_{z,z+1}$ into k clusters, where k determines the number of unique weights. The objective of the k-means algorithm is to minimize the sum of distances of every weight in a cluster to the cluster centroid C_k. The distance between the weight and the centroid C_k is computed as the *Euclidean distance*. Post clustering, weights belonging to a cluster k are replaced by the centroids C_k the respective cluster. Therefore, the centroid weight C_k is shared by all the synapses in the cluster k. Figure 3 shows the weight clustering technique used between two hidden layers of a fully connected network.

With the weights reduced to k fixed values, we propose the use of constant multipliers to design the PE. The use of constant multipliers significantly reduces the area and runtime latency of a single PE. This can be attributed to the exclusion of the redundant

Table 1: Dataset and Application Details.

Dataset	Application	Source	Samples	Input Dimensions
MNIST	Hand Writing Digit Recognition	NYU	10,000	28 * 28

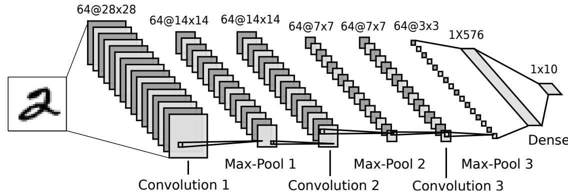

Figure 4: LeNet based MNIST digit recognition application.

BRAM used to store weights, the reduced number memory reads and the size of a constant multiplier.

3 EXPERIMENTAL SETUP

To evaluate the performance and efficiency of the proposed Rox-ANN design methodology, we have used VHDL, Xilinx Vivado 17.4 and Xilinx Artix-7 AC701 evaluation platform (xc7a200tfbg676-2 device) for the implementation of proposed PE design. We have used the 2-neurons configuration of the PE, shown in Fig. 2(b), for describing the effects of utilizing accurate, approximate and constant multipliers on area utilization, critical path delay and energy consumption. For EDP calculations, Vivado Simulator and Power Analyzer tools have been used.

We evaluate the following metrics.

Application Accuracy. The classification accuracy of the benchmark application when retrained using quantized activations and weights, and approximate arithmetic units.

Energy Consumption. The static and dynamic energy consumed by the proposed PE is mainly attributed to the size of the BRAMs and the arithmetic units used to design the PE.

Area Overhead. The area (number of utilized LUTs) of a PE's implementation is also mainly defined by the size of BRAM and choice of arithmetic units.

Latency. The latency of the designed PE is measured as the critical path delay for a single multiply-accumulate (MAC) operation.

We perform experiments on the LeNet based implementation of the MNIST digit recognition application. Figure 4 shows the architecture of the neural network used to train and infer the MNIST digit recognition application. Table 1 shows the details of the MNIST dataset.

4 RESULTS

In this section, we present results that demonstrate the accuracy obtained and also the energy efficiency, reduced latency and area reduction achieved using the Rox-ANN design.

4.1 Accuracy

The MNIST handwritten digit recognition application, shown in Figure 4, is used as a benchmark to observe the effects of quantized data and approximate arithmetic units on the accuracy. Table 2 shows the accuracy of the benchmark application, re-trained using fixed bit-width (8-bit and 16-bit) operators, activations and weights and approximate neurons. The accuracy is compared to a standard implementation of the benchmark application using 32-bit float point data and accurate multipliers. From the table, we observe that the accuracy of the MNIST benchmark application trained and implemented using 8-bit and 16-bit fixed precision data and approximate hardware neurons suffer a 1.77% and 0.37% loss in accuracy when compared to the standard implementation.

Table 3 shows the accuracy of the MNIST benchmark application, implemented using 8-bit and 16-bit data, and a *constant multiplier* based hardware neuron. In order to design the constant multiplier, we use a weight clustering technique, described in Section 2.3, to reduce the number of distinct weights between any two fully connected layers of the ANN. In the benchmark application, we introduce constant multiplier based PEs to implement the dense layers of the LeNet architecture. In Table 3, we observe a marginal drop in application accuracy with a reduction in the number of constant multipliers used to implement the dense layers.

4.2 Performance Metrics of Processing Element

The proposed PE design, shown in Fig. 2(b), has been implemented using two different quantization schemes and utilizing accurate, approximate and constant multipliers. For accurate and approximate multipliers-based PEs, both neurons (in a single PE) reads 576 different activations (vector x) and corresponding weights (vector w) from the associated BRAMs. For constant multipliers-based PE design, the clustered weights define the constant multipliers, and hence, only the activations (vector x) is read from a single associate BRAM. Table 4 shows the area, energy consumption and latency of the *four* proposed configurations of the PE. The results demonstrate that the use of approximate PEs designed using *Rox-ANN* have a significant impact on the area, energy and latency of an ANN implementation, when compared to accurate multipliers-based PE implementation. For accurate multipliers-based PE, our methodology utilizes Vivado speed-optimized multiplier IPs. For implementing approximate signed multipliers, we have used 8×8 and 16×16 precision-reduced multipliers. For these multipliers, the two LSBs of each input operand have been truncated to use 6×6 and 14×14 multiplier IPs (provided by Vivado) to implement precision-reduced 8×8 and 16×16 multipliers respectively. An accurate 8×8 multiplier occupies 75 LUTs, whereas 42 LUTs are occupied by a precision-reduced 8×8 approximate mutiplier. From Fig. 5 we see that the use of 8-bit approximate arithmetic units achieves a 65.6%, 55.1%, and 18.9% reduction in the area, energy consumption and latency for a single PE when compared to a fully-accurate PE. The

Table 2: Experimental Statistics for Approximate Arithmetic Neurons

Dataset	Fixed Bit-Width Implementation		Float-Point Implementation	
	Configuration	Accuracy	Configuration	Accuracy
MNIST	S_8	97.33 %	S_32	99. 10 %
	S_16	98.73%		

Table 3: Experimental Statistics for Constant Multiplier Approach.

Dataset	Fixed Bit-Width Implementation			Float-Point Implementation		
	Configuration	Clusters	Accuracy	Configuration	Clusters	Accuracy
MNIST	S_8	4	96.41%	S_32	1	99.10 %
		8	96.93%			
		16	97.19%			
	S_16	4	97.28%			
		8	97.94%			
		16	98.21%			

Table 4: Area, Energy and Latency of the proposed PEs.

PE Design	Area (LUTs)	Multiplier Area (LUTs)	Energy (pJ)	Latency (ns)
8-bit Accurate	1233	75	3.18	9.65
16-bit Accurate	2488	287	4.83	11.34
8-bit Approximate	856	42	2.17	9.20
16-bit Approximate	2181	220	4.84	10.69
8-bit Constant	150	19	2.30	7.59
16-bit Constant	256	52	3.04	8.70

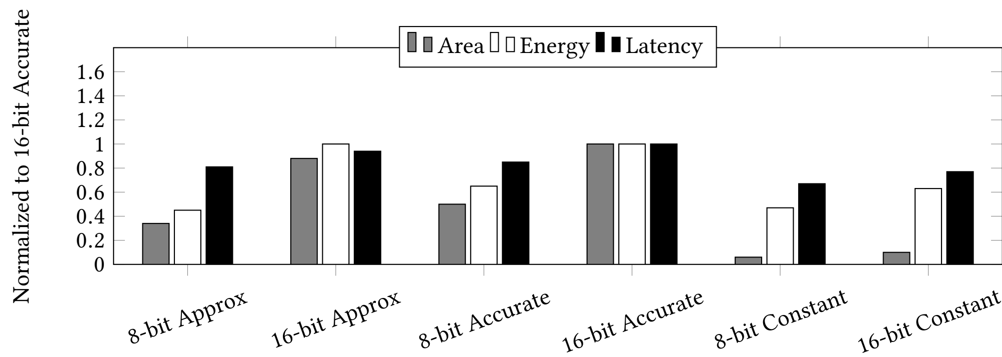

Figure 5: Area, Energy and Latency of the proposed PEs, normalized to the 16-bit Accurate PE.

performance of the ANN can be further improved by incorporating constant multipliers based PEs in the design of the fully-connected layers of the ANN. PEs designed using 8×8 and 16×16 constant multipliers demonstrate a 95.08% and 89.98% reduction in area respectively, and a 33.05 % and 23.3% reduction in critical path delay, respectively, when compared to a 16-bit accurate PE. The LUTs occupied by 8×8 and 16×16 constant multipliers are 19 and 52 respectively.

5 CONCLUSION

Approximate computing is a design paradigm used to reduce the resources required to implement artificial neural networks (ANNs) on hardware. In this paper, we present Rox-ANN, an FPGA-based methodology to design processing elements (PEs) that supports the use of approximation techniques, such as approximate arithmetic units and quantized data, to implement artificial neural networks (ANNs). With the proposed PEs, we demonstrate a 65.6%, 55.1% and 18.9% reduction in area, energy consumption and latency for a 8-bit approximate PE when compared to a PE designed using 16-bit

accurate arithmetic units. Furthermore, we propose a technique to implement a fully-connected layer of an ANN using constant-multipliers based PEs.

REFERENCES

[1] Murat Alçın, İhsan Pehlivan, and İsmail Koyuncu. 2016. Hardware design and implementation of a novel ANN-based chaotic generator in FPGA. *Optik* 127, 13 (2016), 5500–5505.

[2] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. 2014. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *ACM Sigplan Notices*, Vol. 49. ACM, 269–284.

[3] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. 2017. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits* 52, 1 (2017), 127–138.

[4] George E Dahl, Dong Yu, Li Deng, and Alex Acero. 2012. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Transactions on audio, speech, and language processing* 20, 1 (2012), 30–42.

[5] Zidong Du, Krishna Palem, Avinash Lingamneni, Olivier Temam, Yunji Chen, and Chengyong Wu. 2014. Leveraging the error resilience of machine-learning applications for designing highly energy efficient accelerators. In *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 201–206.

[6] K. Guo, L. Sui, J. Qiu, J. Yu, J. Wang, S. Yao, S. Han, Y. Wang, and H. Yang. 2018. Angel-Eye: A Complete Design Flow for Mapping CNN Onto Embedded FPGA. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37, 1 (Jan 2018), 35–47. https://doi.org/10.1109/TCAD.2017.2705069

[7] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2017. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research* 18, 1 (2017), 6869–6898.

[8] Kyuyeon Hwang and Wonyong Sung. 2014. Fixed-point feedforward deep neural network design using weights+ 1, 0, and- 1. In *2014 IEEE Workshop on Signal Processing Systems (SiPS)*. IEEE, 1–6.

[9] Patrick Judd, Jorge Albericio, Tayler Hetherington, Tor Aamodt, Natalie Enright Jerger, Raquel Urtasun, and Andreas Moshovos. 2015. Reduced-precision strategies for bounded memory in deep neural nets. *arXiv preprint arXiv:1511.05236* (2015).

[10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[11] UroÅą Lotric and Patricio Bulic. 2012. Applicability of approximate multipliers in hardware neural networks. *Neurocomputing* 96 (2012), 57–65.

[12] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. *CoRR* abs/1603.05279 (2016). arXiv:1603.05279 http://arxiv.org/abs/1603.05279

[13] Mohammad Samragh, Mohammad Ghasemzadeh, and Farinaz Koushanfar. 2017. Customizing neural networks for efficient fpga implementation. In *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 85–92.

[14] Salim Ullah, Sanjeev Sripadraj Murthy, and Akash Kumar. 2018. SMApproxlib: Library of FPGA-based Approximate Multipliers. In *Proceedings of the 55th Annual Design Automation Conference (DAC '18)*. ACM, New York, NY, USA, Article 157, 6 pages. https://doi.org/10.1145/3195970.3196115

[15] Swagath Venkataramani, Ashish Ranjan, Kaushik Roy, and Anand Raghunathan. 2014. AxNN: energy-efficient neuromorphic systems using approximate computing. In *Proceedings of the 2014 international symposium on Low power electronics and design*. 27–32.

[16] Y. Wang, J. Xu, Y. Han, H. Li, and X. Li. 2016. DeepBurning: Automatic generation of FPGA-based learning accelerators for the Neural Network family. In *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. 1–6. https://doi.org/10.1145/2897937.2898002

[17] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. 2015. Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '15)*. ACM, New York, NY, USA, 161–170. https://doi.org/10.1145/2684746.2689060

[18] Qian Zhang, Ting Wang, Ye Tian, Feng Yuan, and Qiang Xu. 2015. ApproxANN: An approximate computing framework for artificial neural network. In *Proceedings of the 2015 DATE*. 701–706.