MajorityNets: BNNs Utilising Approximate Popcount for Improved Efficiency

Seyedramin Rasoulinezhad*, Sean Fox*, Hao Zhou†, Lingli Wang†, David Boland*, Philip H.W. Leong*
*School of Electrical and Information Engineering, The University of Sydney, Australia 2006
†State Key Lab of ASIC and System, Fudan University, Shanghai 201203, China
Email:{seyedramin.rasoulinezhad, sean.fox, philip.leong, david.boland}@sydney.edu.au, {zhouhao, llwang}@fudan.edu.cn

***Abstract*—Binarized neural networks (BNNs) have shown exciting potential for utilising neural networks in embedded implementations where area, energy and latency constraints are paramount. With BNNs, multiply-accumulate (MAC) operations can be simplified to XnorPopcount operations, leading to massive reductions in both memory and computation resources. Furthermore, multiple efficient implementations of BNNs have been reported on field-programmable gate array (FPGA) implementations. This paper proposes a smaller, faster, more energy-efficient approximate replacement for the XnorPopcount operation, called XNorMaj, inspired by state-of-the-art FPGA look-up table schemes which benefit FPGA implementations. We show that XNorMaj is up to 2× more resource-efficient than the XnorPopcount operation. While the XNorMaj operation has a minor detrimental impact on accuracy, the resource savings enable us to use larger networks to recover the loss.**

I. Introduction

Recent research on convolutional neural networks (CNNs) has yielded a significant improvement over other techniques in cognitive domains. This ability requires a massive number of parameters and complex computations, which makes them challenging to deploy in real-time.

Quantization techniques bring significant performance enhancement by reducing both memory footprint and resource requirements of compute units. Binarized neural networks (BNNs) are the most extreme case of quantization, using a single bit for each activation and weight so the majority of energy-hungry multiply accumulate (MAC) computations can be replaced by the XnorPopcount operation [1]. Previous research has suggested modifications to field-programmable gate arrays (FPGAs) lookup-table (LUT) structures to enhance the efficiency of popcount operation [2]. Wang et al. [3] exploited the capabilities of FPGA LUTs to manipulate XnorPopcount operation to save resources in an fully-unrolled manner.

To further improve the efficiency of FPGA-based BNN architepoolingctures, we propose a smaller and faster approximation for XnorPopcount. We call this XNorMaj, based on integrating Majority and popcount circuits. We report on its efficiency on FPGA platforms, and show that it offers significant reductions in area and critical path. This is the first work focused on simplifying XnorPopcount operations in a fold-able manner with consideration of FPGA architectures. In summary, the contributions of this work:

- A novel approximate replacement for XnorPopcount, called XNorMaj, leading to new BNN architectures, MajorityNets, using the proposed XNorMaj operation.
- A quantitative evaluation of impacts of the above techniques on various BNN architectures, considering different performance metrics such as accuracy, area, and delay.

Verilog models together with a training platform are available as open source software on github.com/raminrasoulinezhad/MajorityNets.

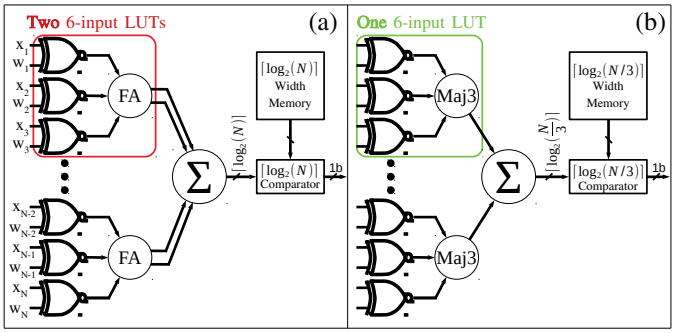

Fig. 1. a) an XnorPopcount operation and b) an XNorMaj-3 operation and their following threshold layer (according to [4])

II. XNorMaj-M Popcount (XNorMaj) operation

A. XnorPopcount operation

In CNNs, convolution (Conv) and fully-connected (FC) layers comprise the majority of computations. Each single output of both layers can be modeled by a neuron-like computation as $y = \sum_{i=0}^{N-1} x_i w_i + B$. By constraining the activations and weights to +1 and -1, and representing them by logic 1 and 0 respectively, multiplication of each pair can be done using an XNOR gate, as proposed by Courbariaux et al. [1]. Thus, high-precision MAC operations can be simplified to bit-wise XNOR operations followed by a counter, called XnorPopcount (1).

$$y = 2 \times \sum_{i=0}^{N-1} Xnor(x_i^b, w_i^b) - N + B \qquad (1)$$

By implementing XnorPopcount operation on different FPGA architectures, we observed that the implementation of Xnor gates and primary compressor circuits are fused. As depicted in Figure 1(a), every three couples of activations and weights are assigned to two LUTs to implement the three XNOR gates and their following Full-Adder (FA) counter. Then, the 2-bit answers of the mentioned blocks are summed

by a compressor tree. According to this structure, the first two LUTs offer a compression rate of 3 input pairs:2.

B. XNorMaj technique

To achieve further compression, we first rewrite Equation (1) using a two-level hierarchical summation (assume $\hat{Z} = \text{XNOR}(\hat{\mathbf{x}}, \hat{\mathbf{w}})$),

$$y = 2 \sum_{i=0}^{\frac{N}{M}-1} \sum_{m=0}^{M-1} \hat{Z}_{iM+m} - N + B \tag{2}$$

We then approximate the inner loop in Equation (2) with a scaled M-input Majority circuit (Maj-M), which indicates whether more than half of the inputs are $True$. This new operation, called XNorMaj, involves a bit-wised XNOR applied between the input activations and their corresponding weights, with each M-grouped output passed to a Maj-M circuit to generate a single bit result which is scaled by V_i^1 and V_i^0.

$$\widetilde{y} = 2 \sum_{i=0}^{\frac{N}{M}-1} (\mathbb{M}_{m=0}^{M-1}(\hat{Z}_{iM+m}) \times (V_i^1 - V_i^0) + V_i^0) - N + B \tag{3}$$

where Maj-M = $\mathbb{M}_{m=0}^{M-1} = \begin{cases} 1, & \text{if } \sum_{i=0}^{M-1} x_i \geq M/2 \\ 0, & \text{otherwise} \end{cases}$. By using a common scaling factor this simplifies to popcounting the single bit outputs of the majority circuits and scaling appropriately (Equation (4)).

$$\widetilde{y} = 2(V^1 - V^0) \sum_{i=0}^{\frac{N}{M}-1} \mathbb{M}_{m=0}^{M-1}(\hat{Z}_{iM+m}) + \frac{N \times V^0}{M} - N + B \tag{4}$$

In Equation (4), the majority circuit scale factors and bias value implement a linear transform (LT) of the neuron output. The transform is similar for all neurons in the same channel. The computation of a batch normalization (BN) layer is also a LT modeled as $\widehat{y} = \gamma(\widetilde{y} - \mu)i + \beta$, where ($\gamma$, μ, i and β) are the BN layer parameters defined per channel [5]. In cases where Conv or FC layers are followed by a BN layer, these two LT functions can be merged into a single LT function. In practice, a BN layer without separate scaling layer is sufficient and this forces BN parameters to adapt scaling factors for majority circuits in each output channel separately. Furthermore, by following the assumption in [4], the new LT function can be merged with activation function in a threshold layer. Because of the mentioned reasons, we fixed the V^1 and V^0 values to be 2.625 and 0.375 respectively, and used a BN layer after each layer using XNorMaj operation.

Using the Majority circuit introduces new trade-offs. Since majority circuits compress M-grouped inputs into a single bit, the popcount is reduced by a factor of M, leading to a smaller popcounter circuit. The comparator circuits and threshold parameters are also simplified and hence smaller. Unfortunately, the technique results in reduced accuracy. The accuracy-performance trade-offs are therefore dependent on the choice of parameter M; for this paper, we focus on XNorMaj-3 ($M = 3$) operations for the following reasons:

- **Implementation efficiency for FPGA platforms:** As demonstrated in Figure 1.b, three XNOR gates, and the following Maj-3 circuits can be combined and mapped to a single 6-input LUT. This fused computation is fed by the same input scheme comparing to the baseline implementation of three XNOR gates and the following FA, XNorFA (Figure 1.a). However, it produces a one-bit output rather than two bits leading to smaller compression trees. By synthesizing different XNorMaj-M circuits for FPGAs in Table I, it can be seen that XNorMaj-3 unite offers the best compression rate vs. complexity trade-off.
- **Accuracy:** by increasing the parameter M, the similarity of the XNorMaj-M popcount and the baseline model is reduced, and inference accuracy drops.
- **Integration with Conv layer kernels:** The number of input pairs for a neuron in an convolution layer is multiple of kernel spatial dimensions. Since M has to be an odd number, by choosing M equal to the kernel size, which is also an odd number, and applying majority logic on the pairs placed in the same channel in a row or a column, folding is possible. Also, three is the most common kernel size for Conv layers in modern BNNs [6].

TABLE I
FPGA IMPLEMENTATION EFFICIENCY OF XNORMAJ-M UNITE. REPORTED DELAYS MEASURED BY REGISTERING INPUTS AND OUTPUTS. EFF: COMPRESSION RATE /(LUTS × DELAY), COMPRESSION RATE = (NUMBER OF INPUT PAIRS):(OUTPUT WIDTH), DELAY: ns

Device	Metrics	XNorFA	XNorMaj-3	XnorMaj-5	XnorMaj-7	XnorMaj-9
	Comp.	3P:2	3P:1	5P:1	7P:1	9P:1
Xilinx	LUT	2	**1**	3	5	7
	Delay	0.68	**0.64**	1.10	0.99	1.07
	Eff.	1.11	**4.67**	1.52	1.41	1.20
Intel	ALMs	2	**1**	3	5	9
	Delay	0.86	**0.70**	0.96	1.24	1.78
	Eff.	0.87	**4.26**	1.74	1.13	0.56

C. Majority Convolution and Fully-connected layers

Consider a standard convolutional layer which takes a $D_F \times D_F \times M$ feature map $\mathbf{F}$ as input, and produces a $D_G \times D_G \times N$ feature map $\mathbf{G}$ as output. The output is generated via a convolution with a $D_K \times D_K \times M \times N$ kernel $\mathbf{K}$ and addition with a N-element bias vector $\mathbf{B}$. Algorithm 1 describes the computation for standard and majority convolution (MConv) cases. Since binarization is applied on activations and weights, we modeled the majority circuit and the previously mentioned scaling factors using clip and scale functions. Scaling factor should be selected to be consistent with Equation (4). Using the same approach, a Majority fully-connected (MFC) layer can be derived.

The back-propagation algorithm for majority layers can be implemented by applying the chain rule as shown by the red arrows in Figure 2. using a straight through estimator (STE) as

Algorithm 1: Standard/Majority-D_K convolution layer

```
for n ← 0 to N do
    for k ← 0 to D_G do
        for l ← 0 to D_G do
            for m ← 0 to M do
                for i ← 0 to D_K do
                    if XNorMaj is enabled then
                        array t_in ← F_{k+i-1,:,m}
                        array t_w ← K_{i,:,m,n}
                        t_d = t_in · t_w        // Dot product
                        G_{k,l,n} += clip(t_d, (-1, 1))×Scale
                    else
                        for j ← 0 to D_K do
                            t_in ← F_{k+i-1,l+j-1,m}
                            t_w ← K_{i,j,m,n}
                            G_{k,l,n} += t_in × t_w
                G_{k,l,n} += B_n
```

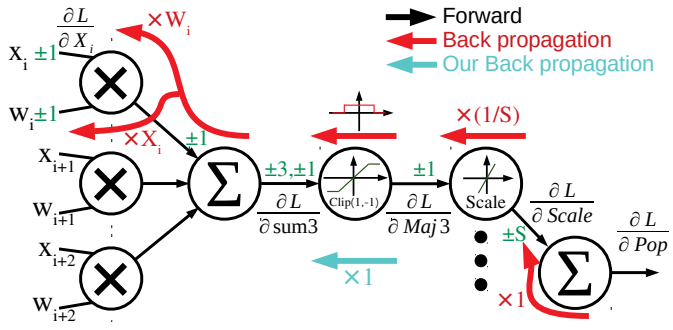

Fig. 2. Back-propagation computation for Majority layers

a proxy for the derivative of the clipping function (teal arrow). Moreover, since the scale factor is fixed for a layer, it can be applied directly on computed activation and weight gradients. With these two modifications, Majority layer back-propagation can be simplified as normal layer back-propagation.

III. Results

A. Hardware efficiency of XNorMaj vs. XnorPopcount

We synthesized, placed, and routed a Verilog model of XnorPopcount and XNorMaj circuits using several input sizes for Aria-10 (10AX016E4F29M3SG) and Zynq UltraScale+ (xczu3eg-sbva484-1-e) using Quartus-II 2017.0 and Vivado 2018.2 respectively. As Figure 3 shows, XNorMaj is 20-50% smaller, especially for the large input sizes. We also observed that critical paths are dramatically reduced in Intel architectures, with a smaller gain on Xilinx FPGAs.

B. Hardware Efficiency of MConv and MFC layers

Figure 4 demonstrates the Conv/FC layer implementation used to measure the efficiency of XNorMaj-3. In this implementation, the input feature map is streamed over channels in parallel. Each stream is saved in a channel buffer to provide a window of $D_K \times D_K \times M$ activations to all processing units (PU). Each PU is responsible for computing pixels of a channel of the output feature map. For FC layers, we assume inputs are available in parallel and there is no need for sliding. If Conv/FC layers are followed by a pooling layer, with the same buffering scheme, the pooling layer is implemented on top of that layer. Also, a layer following pooling layers can be folded to keep the same throughput rate, achieving full utilization. For instance, a Conv layer after a Maxpool layer with a 2×2 kernel size should be folded $4\times$ more than the previous layer. In our design, we fold the number of PEs rather than their input size by folding factor (FF). This approach prevents high-precision MAC operations on partial results.

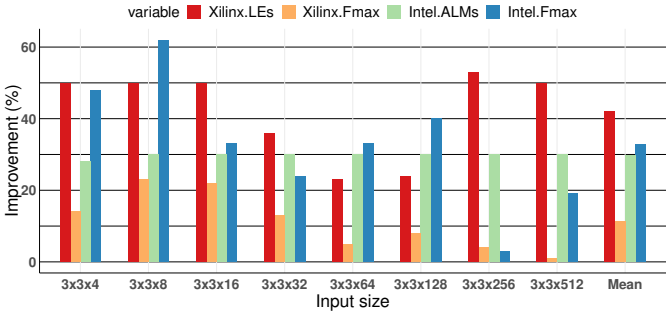

Fig. 3. XNorMaj-3 improvements using Xilinx/Intel FPGA architectures (Zynq Ultrascale+ / Aria 10). F_{max}: Maximum working frequency

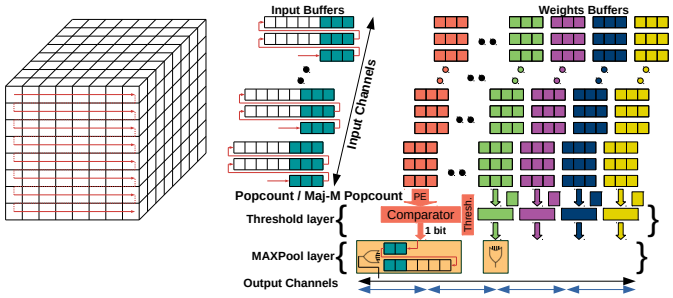

Fig. 4. Illustration of accelerator operation

Table II summarises the implementation results for different layers of the padded version of CNV [4] (CNV-P) network, where all Conv layers are using padding. Using XNorMaj-3 reduces the required LUTs by 20-43% per layer and 30% in total. By resynthesizing the layers regardless of throughput balancing, the logic element (LE) reduction grows to 43% which shows the affect of folding on reduction rate (last column of the Table II). To measure the efficiency for highly-folded implementations we increased the FF by $8\times$ for all layers which limits the resource reduction to 13%.

C. Accuracy

To explore the effect of using XNorMaj on the BNNs, we trained different models on different datasets. The training platform is available on the GitHub repository, which is based on the open-source project in reference [7].

TABLE II
LE USAGE COMPARISON OF (M)CONV AND (M)FC LAYERS USING CNV-P. (LAYER #1 IS NOT INCLUDED), ⋆: PADDED TO BE MULTIPLE OF 3

Layer configurations				Conv	MConv	LE Improvement	
#	Cin	Cout	FF	/FC	/MFC	folded	non-folded
Conv2	64	64	1	55k	40k	27%	27%
Conv3	64	128	4	38k	30k	20%	26%
Conv4	128	128	4	86k	63k	27%	36%
Conv5	128	256	16	43k	33k	27%	36%
Conv6	256	256	16	87k	50k	43%	56%
FC1	4096⋆	512	64	96k	67k	30%	36%
FC2	512⋆	512	64	11k	8k	22%	36%
FC3	512⋆	10	10	1K	0.7k	29%	30%
Total				417k	291k	30%	43%

TABLE III
ERROR RATE (%) OF DIFFERENT BNNS

BNNs	Dataset	Error(%)		details		LE
		baseline	Ours	MConv	MFC	Improve
SFC	MNIST	3.47%	3.75%	-	All	45%
LFC	MNIST	2.66%	2.86%	-	All	35%
CNV-P	SVHN	5.67%	6.28%	except 1st Conv	-	23%
CNV-P	CIFAR10	13.35%	15.01%		-	
VGG-like	CIFAR10	10.78%	11.24%		-	30%

First, we applied the proposed idea on two multi-layer perceptrons, SFC and LFC networks [4]. By replacing their all three FC layers with MFC layers, the accuracy drop for the MNIST dataset is 0.2% and 0.3% while reducing the LEs by 45% and 35% respectively for SFC and LFC in the mentioned implementation method with no folding. In the same approach, using CNV-P [4] and VGG-like [7] networks, by replacing 2nd-6th Conv layers with MConv, the LE reduction is about 23% and 30% with the cost of 1.7% and 0.5% accuracy drop for CNV-P and VGG-like respectively (Table III).

The area reduction makes the CNV-P with majority layers implementation the same cost as a non-padded with XnorPopcount operation. However, by using padding, in CNV-P, convolution layers receive and produce larger feature maps which increase the computations and keep the weight parameters the same for those layers. It also affects the first FC layer, where the number of inputs is increased, *e.g.*, 16 times in the case of in the first MFC layer of CNV-P network leading to 16 times more computation and parameters. Since the error rate of adding padding and using majority layers is 2% less, we recovered accuracy using the same area.

In addition, we explored the effect of arbitrary picking the layers for deployment of XNorMaj-3 operation. We select CNV-P network and the most contributor layers in terms of LEs which are Conv2-6 and the first FC layers. We use a six-character notation, where each character can be B or M, representing whether a layer is using XnorPopcount or XNorMaj. The first five characters specify the configuration of Conv2-6 layers and the last character gives the FC1 layer

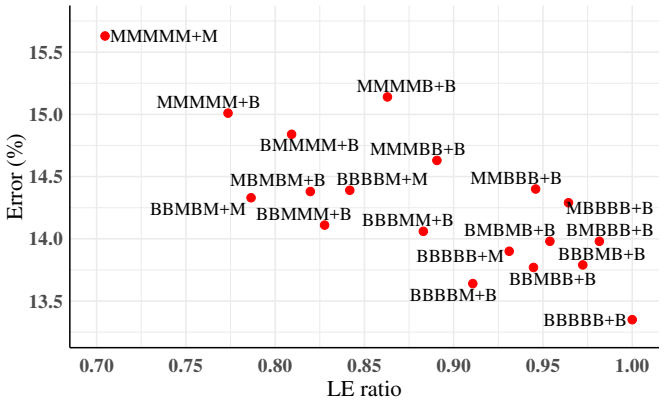

Fig. 5. Comparison of accuracy vs. LE usage (points on the lower left represent a Pareto set of efficient implementations)

with "+" being used to separate the last character. Figure 5 shows performance per LUT of different configurations using our mentioned architecture, enabling a user to trade accuracy for performance by following the Pareto-optimal curve. As an example, the BBMBM+M configuration delivers saves 22% of LEs with accuracy reduced by only 1%.

IV. CONCLUSION

We proposed XNorMaj-M popcount, a new approximate XnorPopcount operation, that reduces resource requirements. Compared to a conventional implementation using XNOR and a compression tree, XNorMaj is 20-50% more area efficient in FPGA platforms. Furthermore, the technique enjoys an average 20% critical path reduction for Xilinx and Intel FPGA architectures. Using XNorMaj, an semi-unrolled, padded version of the CNV network with the same LUT utilization enjoys 2% better accuracy. In future work we will explore the effect of using Maj-5/7/9 circuits. In addition to that, partially usage of the XNorMaj in a layer would be explored. We will also show the efficiency of XNorMaj operations on application specific integrated circuits (ASIC) implementations.

REFERENCES

[1] M. Courbariaux and Y. Bengio, "BinaryNet: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1," *CoRR*, vol. abs/1602.02830, 2016.

[2] J. H. Kim, J. Lee, and J. Anderson, "FPGA architecture enhancement for efficient BNN implementation," in *Int. Conf. on Field-Programmable Technology*, 2018, pp. 1–8.

[3] E. Wang, J. J. Davis, P. Y. K. Cheung, and G. A. Constantinides, "LUTNet: Rethinking inference in FPGA soft logic," in *IEEE Annual Int. Symp. on Field-Programmable Custom Computing Machines FCCM*, 2019.

[4] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. H. W. Leong, M. Jahre, and K. A. Vissers, "FINN: A framework for fast, scalable binarized neural network inference," in *Proc. Int. Symp. on Field-Programmable Gate Arrays*, 2017, pp. 65–74.

[5] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. of the 32nd Int. Conf. on Machine Learning, ICML*, 2015, pp. 448–456.

[6] S. Liang, S. Yin, L. Liu, W. Luk, and S. Wei, "FP-BNN: binarized neural network on FPGA," *Neurocomputing*, vol. 275, pp. 1072–1086, 2018.

[7] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Advances in Neural Information Processing Systems 29: Conf. on Neural Information Processing Systems*, 2016.