SynthNet: A High-throughput yet Energy-efficient Combinational Logic Neural Network

Tianen Chen, Taylor Kemp*, and Younghyun Kim
University of Wisconsin–Madison
{tianen.chen, tkemp, younghyun.kim}@wisc.edu

2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC) | 978-1-6654-2135-5/22/$31.00 ©2022 IEEE | DOI: 10.1109/ASP-DAC52403.2022.9712554

Abstract—**In combinational logic neural networks (CLNNs), neurons are realized as combinational logic circuits or look-up tables (LUTs). They make make extremely low-latency inference possible by performing the computation with pure hardware without loading weights from the memory. The high throughput, however, is powered by massively parallel logic circuits or LUTs and hence comes with high area occupancy and high energy consumption. We present SynthNet, a novel CLNN design method that effectively identifies and keeps only the sublogics that play a critical role in the accuracy and remove those which do not contribute to improving the accuracy. It captures the abundant redundancy in NNs that can be exploited only in CLNNs, and thereby dramatically reduces the energy consumption of CLNNs with minimal accuracy degradation. We prove the efficacy of SynthNet on the CIFAR-10 dataset, maintaining a competitive accuracy while successfully replacing layers of a VGG-style network which traditionally uses memory-based floating point operations with combinational logic. Experimental results suggest our design can reduce energy-consumption of CLNNs more than 90% compared to the state-of-the-art design.**

I. Introduction

Reducing memory access is the core of realizing fast and efficient neural networks (NNs). In conventional neural processing element (NPE)-based NNs, frequent multiply-and-accumulate (MAC) operations incur heavy memory access overhead for fetching weight parameters and storing intermediate outputs, which is the primary source of latency and energy consumption [1]. An emerging hardware-oriented solution to this challenge is combinational logic NN (CLNN), where the inputs and outputs of the neurons are binarized and represented as *arbitrary Boolean functions* mapped to combinational logic circuits or look-up tables (LUTs) [2]–[6]. The evaluation of such hardware does not involve any memory access other than fetching the inputs and storing the final outputs, and hence is extremely faster than equivalent binarized MAC on NPEs. It makes CLNNs attractive and suitable for low-latency, fixed-function applications, such as in-sensor inference [2] and network intrusion detection [5].

The low latency of CLNNs, however, is powered by massively parallel hardware resources that cost significant area occupancy and hence energy consumption. The key to the realization of area- and energy-efficient CLNNs is to exploit its intrinsic redundancy and error resilience like in other NN optimization methods (e.g., pruning and quantization) in every design step including logic-level optimization. Unfortunately, conventional logic optimization methods that strictly preserve original input-to-output mapping are not able to capture and exploit the redundancy and error resilience, and thus are far from effective when optimizing CLNNs. As recognized in [7], this gap between machine learning and logic optimization is yet to be resolved, and should be addressed for more widespread adoption of CLNNs.

In this paper, we propose a novel CLNN design method called SynthNet for bridging this gap and pushing the limit of CLNN adoption for high-throughput and low-power applications. Our techniques improve upon the scalability of CLNNs by proposing minimization techniques that allow for large scale networks to be compressed to synthesizable circuits. SynthNet exploits the intrinsic error resilience of NNs in order to selectively remove or replace Boolean mapping functions and thereby significantly reduce the logic circuit size. By judiciously *over-minimizing* the truth tables of neuron mapping functions based on the significance of each mapping with the awareness of neuron activation properties, the circuit implementation of the resultant truth tables is reduced by orders-of-magnitude than that of the original truth tables. This design method also boosts the accuracy by suppressing inference errors induced by random output mapped to input combinations unseen during training, which is a unique hazard in CLNNs.

The contributions of the paper are summarized as follows:

- We analyze activation of neurons realized as a logic circuit and identify opportunities to further minimize the logic size beyond what traditional logic optimization methods can achieve, in order to fully exploit the error resilience of NNs.
- Based on the analysis, we present two very effective CLNN optimization techniques: i) synthesis-aware pruning and ii) input-driven neural logic minimization. We explore the energy-accuracy trade-offs of the logic optimization and present an efficient and more scalable design framework to determine the optimal implementation that meets a given accuracy constraint.
- We evaluate SynthNet for a CLNN using the CIFAR-10 dataset [8]. Our method reduces the energy consumption per image by 90–99% compared to a systolic array-based architecture, while maintaining 82% accuracy, yet to be achieved by CLNN-only implementations on the CIFAR-10 dataset.

*This work was done while the author was at the University of Wisconsin–Madison. The author is currently at Facebook (taylorkemp@fb.com).

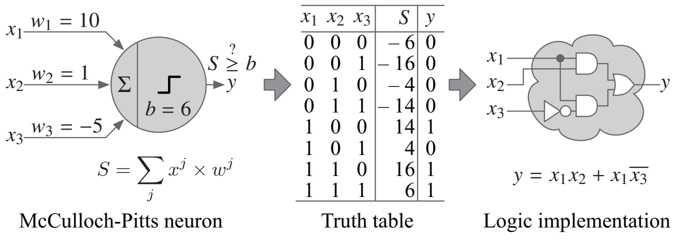

Fig. 1. Combinational logic implementation of a McCulloch-Pitts neuron.

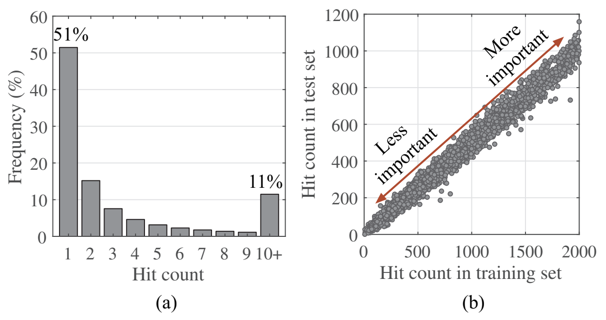

Fig. 2. (a) Histogram of row hit counts greater than 0 in the training set. (b) Correlation between row hit counts between the training set and the test set. (Range limited for better visualization.)

II. Related Work

Researchers have proposed various methods to save energy in neural processing by exploiting its intrinsic error resilience, sparsity, and massive parallelism at different levels of the system stack, but the common goal has been reducing memory access. Examples include quantization [9], pruning [10], [11], and model compression [12]. Processing-in-memory (PIM) also reduces memory access by performing analog computing in or near specially designed memory without fetching the weights into the NPE [1], [13].

CLNNs approach the same goal as PIM—performing computations where the weights are stored—from the opposite direction by embedding the weights in the fully digital processor in the form of logic circuits or LUTs. The feasibility of realization of NNs as combinational logic has been proven in recent work [2]–[4], but the problem of scaling the large logic size is yet to be addressed. It is mainly because the error resilience of NNs has not been exploited during logic minimization and synthesis, resulting in overly precise combinational logic and thereby leaving the potential of energy saving largely untapped.

III. Background: Combinational McCulloch-Pitts Neural Networks

The McCulloch-Pitts neuron model, which has binary inputs and a binary output, is an ideal model for CLNN implementation since combinational logic can implement any arbitrary binary mapping. A McCulloch-Pitts neuron's function can is defined as follows:

$$y = \begin{cases} 1 & \text{if } \sum_j x^j w^j \geq b \\ 0 & \text{otherwise} \end{cases}, \tag{1}$$

where y is the output of the neuron, x^j and w^j respectively are the j-th input and weight, and b is the bias of the neuron. Unlike binarized NNs that binarize everything including weights (e.g., XNOR-based NNs), w_j can be a high-precision floating-point value, which are desirable for high accuracy [4]. Boolean logic circuit is suitable for implementing this neuron model because the inputs and output are binary and their mapping requires the ability to realize arbitrary Boolean functions.

A trained McCulloch-Pitts neuron can be implemented as a logic circuit as illustrated in Fig. 1. If the number of inputs is small, outputs can be defined for all possible input combinations based on (1) to build the truth table of a completely specified function (CSF) with no don't-care (DC) output. In practical NNs, however, the number of inputs is much grater than three, making it impossible to enumerate outputs for all input combinations. Alternatively, defining the Boolean function as an incompletely specified function (ISF), where the output is specified only for a subset of input combinations and the rest are left as DC, can greatly reduce the enumerated outputs since only a small subset of input combinations are seen during training and inference [4]. Finally, logic minimization and synthesis is followed to implement the ISFs as logic circuits.

In this work, the straight-through estimator in the form of the hard hyperbolic tangent (tanh) linear activation function is used as in [9], in order for the binarized neurons to be able to successfully update gradients within the back-propagation algorithm when the derivative is zero everywhere. We substitute the classical dropout regularization technique with a random binarization probability. Activations are binarized with a certain specified probability. Stochastically binarizing activations ensures our gradients update during back-propagation, derived from the variant of dropout in [9].

IV. Motivation: Unexploited Error Resilience

As mentioned above, conventional *precise* logic minimization and synthesis does not take advantage of the high error resilience of CLNNs. It leads to two limitations (and opportunities) in terms of energy efficiency and accuracy.

A. Non-uniform Input Combination Frequency

When input-to-output mapping does not *always* have to be precise, which is the case for CLNNs, *approximate* implementation of Boolean functions allows sharing not only exactly equivalent sub-circuits but also near-equivalent sub-circuits, resulting in a smaller logic size and thus high energy efficiency that precise implementation cannot achieve. For the approximate implementation of combinational neuron, we should exploit the property that the probability distributions of neuron inputs and outputs are not uniform.

Let us consider the VGG-like architecture in [14] of six convolutional layers followed by three fully-connected layers. As an example, to implement the second convolutional layer (3×3 convolution, 20 input channels, 20 output channels) as a logic circuit, we would need to build 20 truth tables of $3 \times 3 \times$

$20 = 180$ inputs (i.e., 2^{180} input combinations) of one output each. After building a truth table with 10,000 images from the training set, out of the 2^{180} rows, output is specified (as either 0 or 1) for only 1.1×10^6 rows on average across 20 output channels, and the output for the rest of the rows remains unspecified (DC). This corresponds to only $7.4 \times 10^{-47}\%$ of total maximum possible rows. More importantly, some rows are seen more frequently than other rows, implying that not all rows are equally important. As a metric of the importance of rows, we define *hit count*, HC, which refers to the number of occurrences that the row's input combination is seen during training or inference. Fig. 2(a) shows the histogram of the row hit counts greater than 0 (i.e., excluding DC rows with $HC = 0$) after training. We can see that 51% of the rows are hit (i.e., the corresponding input is seen) only once, and only 11% of the rows are hit 10 or more times. Furthermore, there exists a very high correlation between the hit counts of the training set and the test set (10,000 images) as shown in Fig. 2(b), which suggests that logic optimization based on training set will work as well for inference.

B. Accuracy Loss Due to Unspecified Outputs

The input combinations defined in the ISFs from the training set cover the most of the input combinations of the test set, but not all. In the same example above, about 14% of the input combinations of the test set are not seen in the training set. Since the output for such input combinations is set to DC in the ISFs, an output that violates (1) may be mapped during synthesis, which becomes a source of accuracy loss. This is a unique hazard of CLNNs that does not exist in NPE-based NNs where outputs for unseen inputs are still correctly computed based on the loaded weights.

In order to mitigate the problem, we need to minimize the accuracy loss due to unspecified outputs. This could be achieved by increasing the possibility that the output is specified for given input combinations, i.e., increasing the *hit rate*, which is defined as the percentage of the input combinations that are seen during both the training and inference. However, specifying more output for unseen rows is not a viable option since it will increase the logic complexity. Rather, introducing DCs in the inputs will increase the hit rate because the total number of unseen input combinations is reduced, and as a result more *generalized* truth tables will be generated. This is similar to pruning of conventional NNs in that it requires a judicious choice of inputs to be ignored. In CLNNs, this is an opportunity for boosting accuracy by preventing *unknown outputs* during training (which does not happen in conventional NN training).

V. Design Optimization of CLNNs

Based on the intuitions discussed in Section IV, we present a design optimization method of CLNNs called SynthNet, focusing on logic minimization. Specifically, we propose two complementary techniques, synthesis-aware pruning and input-driven neural logic minimization, to address the above-mentioned limitations and exploit the error resilience of CLNNs for improving energy efficiency.

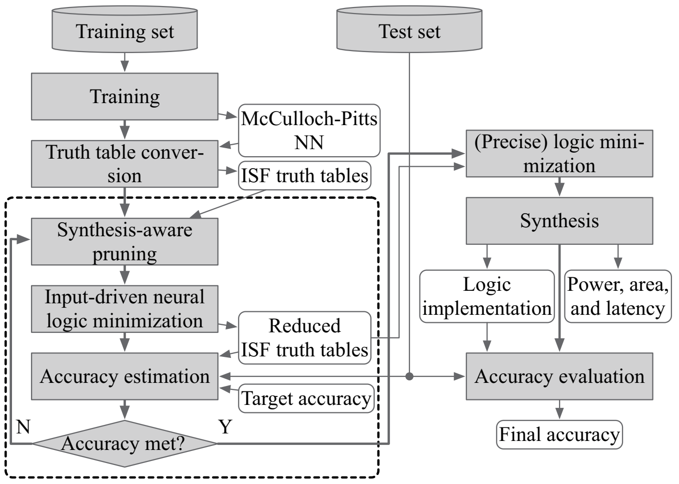

Fig. 3. SynthNet's fully automated design optimization of CLNNs. The dashed box represents the territory of the proposed logic optimization.

A. Design Flow

The overall design flow is presented in Fig. 3. We first train a McCulloch-Pitts NN and convert target layers into ISF truth tables. Our CLNN optimization is performed before the conventional logic minimization and synthesis of the truth tables, as highlighted by the dashed box in the figure. We split the multi-input multi-output truth tables into multi-input single-output truth tables to be optimized independently. The truth tables are then sent through our logic optimization procedure composed of iterative synthesis-aware pruning and input-driven neural logic minimization until it reaches a given target accuracy. Accuracy is evaluated on the test set using the reduced truth tables. Finally, the reduced truth tables are implemented as logic circuits through conventional logic minimization and synthesis. The synthesized circuit is evaluated for hardware metrics, and the test set is applied on the circuit to get the actual accuracy. The following two subsections respectively describe the synthesis-aware pruning and the input-driven neural logic minimization, followed by the integration of both in the design flow.

B. Synthesis-aware Pruning

Pruning, in general, removes low-magnitude weights that contribute little to the model output. In CLNNs, weight pruning is equivalent to removing an input from the truth table, or placing DC on the input to be removed. This serves two benefits. First, the truth table size (hence the circuit complexity) decreases exponentially as the number of inputs decreases. Second, the hit rate increases because some DC outputs, which would have mapped to a random output, are now specified based on remaining more significant weights, leading to accuracy improvement. There is a point of diminishing returns in accuracy improvement because beyond a certain point, accuracy loss due to over-generalization becomes greater than the accuracy gain due to the reduction of DC outputs.

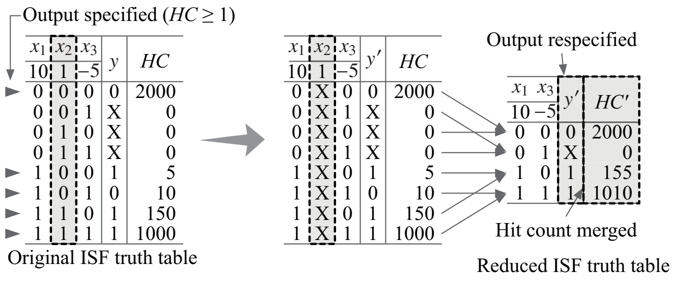

Fig. 4. Synthesis-aware pruning. In this example, $\Delta_p = 33.3\%$, and hence x_2 is removed.

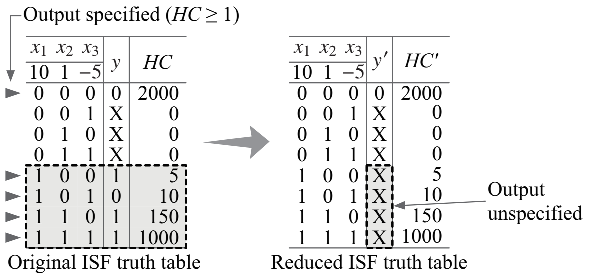

Fig. 5. Input-driven logic minimization (row dropping). In this example, $\Delta_h = 40\%$, and output for four input combinations, 100, 101, 110, and 111, are unspecified.

Specifically, for a given ISF truth table, we gradually remove inputs (introduce DCs) beginning with the corresponding lowest-magnitude weights. We denote the percentage of removed inputs by *pruning degree* Δ_p. For example, in the 3×3 convolutional layer where there are 20 input channels, we can remove up to 180 inputs. If $\Delta_p = 90\%$, 162 lowest-weight inputs will be removed, reducing the maximum of input combinations from 2^{180} to 2^{18} . The input removal results in multiple rows with different outputs mapping to the same row with the same output in the reduced truth table. In order to determine the new output, we take a weighted average of the inputs mapped together under the DCs to take the importance of each row into account for the new output y' as follows:

$$y' = \text{round}\left(\frac{\sum_{i \in I} y_i \times HC_i}{\sum_{i \in I} HC_i}\right), \tag{2}$$

where I is the set of inputs that are merged, y_i is the output and HC_i is the hit count of the i-th row that is merged. The hit count of the new rows, HC' is the sum of the hit counts of the merged rows. That is,

$$HC' = \sum_{i \in I} HC_i. \tag{3}$$

Consider an example shown in Fig. 4. The second input, x_2, has the lowest weight of 1, so we consider pruning it. Pruning a single input will cause pairs of inputs to respectively map to a single input. For example, consider the two input combinations $x_1x_2x_3 = 101$ and $x_1x_2x_3 = 111$. In the original truth table, their outputs differ as 0 and 1, respectively, but in the reduced truth table, they both map to $x_1x_3 = 11$, and the new output is round$\left(\frac{0\times10+1\times1000}{10+1000}\right) = 1$. The hit count of the new rows is now $10 + 1000 = 1010$.

Two input combinations $x_1x_2x_3 = 000$ and $x_1x_2x_3 = 010$ show how pruning improves accuracy. The first input combination, 000, is seen during training, and its output is specified as 0 in the original truth table. On the other hand, the second input combination, 010, is not seen during training, and its output is not specified. If this DC output is mapped to 1 during synthesis, it would become a source of inference error because it violates (1). Pruning x_2 effectively specifies the output of 010 as 0, which is the correct output if it had been seen during training.

C. Input-driven Neural Logic Minimization

As discussed in Section IV-A, the hit counts of the rows varies significantly, and the majority of input combinations appear only a few times during training. These low-hit count rows contribute little to the CLNN accuracy, as compared to high-hit count rows, and can be removed from the ISF truth table as if they have not appeared during training. This is done by unspecifiying the output, i.e., making 0 or 1 to DC, and the row is called *dropped* from the table. As we drop seldom-hit inputs and introducing more DC outputs, we can reduce the complexity of the synthesized circuit.

Specifically, our input-driven neural logic minimization, or simply *row dropping*, unspecifies the output for the rows with the lowest non-zero hit counts until the total hit counts of dropped rows reaches a *row dropping degree* Δ_h. We do not set the hit count of the dropped rows to zero because the hit counts should be preserved for making decisions in the following iterations of pruning. Fig. 5 shows an example of row dropping for $\Delta_h = 40\%$. Since the total hit count is 3165, we drop low-hit count rows until the sum of the hit counts of the dropped row reaches 1266. In this case, four rows $x_1x_2x_3 = 100$ $x_1x_2x_3 = 101$, $x_1x_2x_3 = 110$, and $x_1x_2x_3 = 111$ have the lowest non-zero hit counts, whose sum is $5+10+150+1000 = 1165$. Therefore, the four rows are dropped by unspecifying their output, but their hit counts, 1165 in total, are preserved.

D. Pruning and Row Dropping Thresholds

Determining the two thresholds, pruning degree Δ_p and row dropping degree Δ_h, can be time-consuming since the design space is large, and logic synthesis for accuracy evaluation takes a long time. We propose a variant of coordinate descent to determine the two thresholds to minimize the logic size while meeting an accuracy constraint c without time-consuming exhaustive search. Also, we estimate the accuracy of the resulting NN using the reduced ISF truth table without synthesizing it.

The threshold optimization procedure is described in Algorithm 1. Initially, Δ_p is set to 0% and Δ_h is set to 0%, i.e., no pruning and no row dropping is applied. For larger networks, we can start Δ_h and Δ_p at a higher number. In the PRUNE procedure, we first search along the space of Δ_p by gradually increasing it by a granularity of δ_p until the accuracy reaches the maximum. During this procedure, new accuracy ACC$(\Delta_p^{new}, \Delta_h)$ is compared to the previous step's accuracy,

Algorithm 1 CLNN logic optimization

```
1: procedure OPTIMIZE($c$)
2:     initialize $\delta_p$ and $\delta_h$
3:     let $\Delta_p^{old} = 0,\ \Delta_h^{old} = 0,\ \Delta_p^{new} = \delta_p,\ \Delta_h^{new} = \delta_h$
4:     while $\Delta_p^{new} > \Delta_p^{old}$ or $\Delta_h^{new} > \Delta_h^{old}$ do
5:         decrease $\delta_p$ and $\delta_h$
6:         $\Delta_p^{old} = \Delta_p^{new},\ \Delta_h^{old} = \Delta_h^{new}$
7:         $\Delta_p^{new} = \text{PRUNE}(\Delta_p^{old}, \delta_p, \Delta_h^{old})$
8:         $\Delta_h^{new} = \text{ROWDROP}(\Delta_h^{old}, \delta_h, \Delta_p^{new}, c)$
9:     end while
10:    return $\Delta_p^{old},\ \Delta_h^{old}$
11: end procedure
12: procedure PRUNE($\Delta_p^{old}$, $\delta_p$, $\Delta_h$)
13:    while $\text{ACC}(\Delta_p^{new}, \Delta_h) > \text{ACC}(\Delta_p^{old}, \Delta_h)$ do
14:        $\Delta_p^{old} = \Delta_p^{new},\ \Delta_p^{new} = \Delta_p^{old} + \delta_p$
15:    end while
16:    return $\Delta_p^{old}$
17: end procedure
18: procedure ROWDROP($\Delta_h^{old}$, $\delta_h$, $\Delta_p$, $c$)
19:    while $\text{ACC}(\Delta_p, \Delta_h^{new}) > c$ do
20:        $\Delta_h^{old} = \Delta_h^{new},\ \Delta_h^{new} = \Delta_h^{old} + \delta_{hit}$
21:    end while
22:    return $\Delta_h^{old}$
23: end procedure
```

ACC$(\Delta_p^{old}, \Delta_h)$. At the maximum accuracy, we fix Δ_p and move on to the ROWDROP procedure where we gradually increase Δ_h by a granularity of δ_h until the accuracy hits the user-defined accuracy constraint c. This is repeated until increasing Δ_p further does not improve accuracy and increasing Δ_h further causes the accuracy to drop below c. To reduce computational complexity, we start at a coarse granularity for the initial search and gradually increase the granularity. As a result, the coordinate descent algorithm's iterative procedure repeats the two-dimensional search and returns the optimal Δ_p and Δ_h that satisfy the given accuracy constraint.

In each iteration during coordinate descent for pruning, ACC() function estimates the accuracy of a candidate NN, which would require time-consuming synthesis and simulation of its logic implementation. In order to reduce execution time, we estimate the accuracy by keeping the ISFs as lookup tables. Instead of time-consuming synthesis and simulation, these lookup tables can be used to quickly estimate the output of the circuit. Because the truth tables are incomplete, some table lookups may fail when an input with unspecified output is encountered. Upon the completion of Algorithm 1 for each layer, all layers are integrated and synthesized for the evaluation of accuracy and power consumption.

VI. Experiments

In this section, we demonstrate the efficacy of SYNTHNET for energy-efficient implementation of CLNN.

A. Experimental Setup

1) NN model and dataset: We consider a small CNN model that is suitable for low-latency in-sensor image recognition as shown in Table I. Note that, while small, this network is the deepest and most computationally complex network yet to be integrated with ultra-low latency CLNNs. The model is trained on the CIFAR-10 image dataset using Google Tensorflow. Since making early layers energy-efficient is more effective [15], we apply the proposed method to the second and third convolutional layers. However, the applicability is not limited to any specific layer type and is applicable to any binary-input binary-output layers. The other layers are binarized as well and processed by NPEs.

TABLE I
VGG-like NN architecture with 32×32 input map used in the experiments.

Layer	Type	Max pool 2×2	Dropout
1	Binarized 3×3 Conv (3,20)		
2	Binarized 3×3 Conv (20,20)	✓	✓
3	Binarized 3×3 Conv (20,40)		
4	Binarized 3×3 Conv (40,40)	✓	✓
5	Binarized 3×3 Conv (40,80)		
6	Binarized 3×3 Conv (80,80)	✓	✓
7	Fully connected (80,1024)		✓
8	Fully connected (1024,1024)		✓
9	Fully connected (1024,10)		

We use batch normalization after every layer, a batch size of 128, and the last 5000 samples of the training set as a validation set with the test error rate reported on the best validation accuracy after 200 epochs, similar to [9]. We do not retrain on the validation set and use only the images in the training set, excluding the validation set, to build the truth tables. Therefore, the truth tables are built on a completely separate set of images than the images on which we test the classification accuracy. All accuracy numbers reported in this section are obtained by simulating synthesized circuits, not estimated by the ACC() function.

2) Hardware evaluation: All results for CLNN are synthesized with Synopsys Design Compiler using the TSMC 45 nm library. Every circuit representing individual truth tables needs cycles according to the height $\times$ width so that the whole input is convolved. For example, the input to Layer 2 is 32×32, thus it takes 1024 cycles of the combinational logic to convolve the entire input. Note that one can exchange latency for area by instantiating more combinational circuits to reduce the number of cycles. As the baseline, we use SCALE-Sim [16] and DRAMPower [17] to estimate the energy consumption for memory accesses in systolic array-based architecture with the parameters in [18].

B. Pruning and Row Dropping Results

Figs. 6 and 7 show the change in hit rate and classification accuracy for varying Δ_p and Δ_h, respectively. In Fig. 6, we can see that hit rate increases as Δ_p increases. As a result, as discussed in Section IV-B, the accuracy increases up to 86.4% at $\Delta_p = 83.3\%$ in Layer 2 and 86.9% at $\Delta_p = 89.4\%$ in Layer 3, which is found by Algorithm 1. Beyond these Δ_p, combinational circuitry is coalescing too many inputs and making the output too generic. Figs. 7(a) and 7(b) shows that the hit rate decreases as Δ_h increases for Layers 2 and 3, while Δ_p is set to the respective optimal value. As a

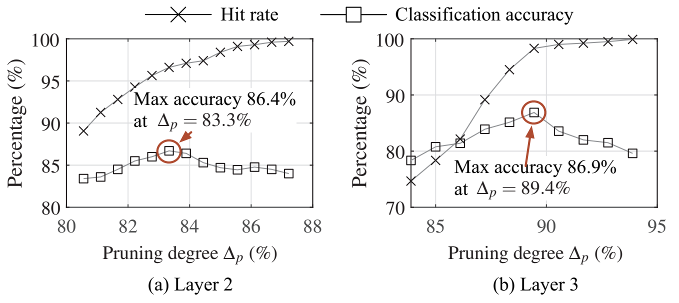

Fig. 6. Synthesis-aware pruning with a varying pruning threshold Δ_p on (a) Layer 2 and (b) Layer 3.

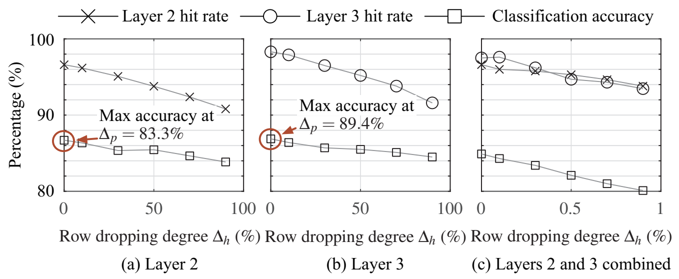

Fig. 7. Input-driven neural logic minimization with a varying row dropping threshold Δ_h on (a) Layer 2, (b) Layer 3, and (c) Layer 2 and 3 combined.

result, the classification accuracy also drops, but only by a few percent even at $\Delta_h = 90\%$, i.e., when only 10% of the input combinations from training are preserved. Fig. 7(c) shows the hit rate of Layers 2 and 3 and the accuracy after combining both layers with the same Δ_h. As a result of row dropping in subsequent layers, the accuracy is slightly decreased, but it is still as high as 80% even when 90% of the rows are dropped in each layer.

C. Energy Efficiency Comparison

Finally, we evaluate the CLNN in comparison to a conventional systolic array-based NN estimated using SCALE-Sim and DRAMPower. We set the target accuracy of our design to 82%, which is achieved at $\Delta_h = 50\%$ (see Fig. 7(c)). Pruning degree Δ_p is set to the respective optimal, 83.3% for Layer 2 and 89.4% for Layer 3. Compared to the systolic array-based NN architecture, the proposed CLNN consumes only a fraction of energy because there is no energy consumption for fetching weights from off-chip memory. We can also observe the greatest power reduction for layers with smaller input size but larger output channel width. For example, Layer 3 is convolving over a 16×16 input map after a max pooling layer, which means less cycles are needed to perform the full convolution. However, the weight fetching DRAM accesses' expenditure is much higher, regardless of input size, and is only dependent upon the width of the input and output channels. Therefore, layers with smaller input map sizes and larger width experience the greatest reduction in power consumption.

TABLE II
SYNTHESIS RESULTS OF TARGET LAYERS AND COMPARISON TO SCALE-SIM.

Layer	CLNN			SCALE-Sim	Energy
	Energy per cycle (pJ)	Cycles per image	Energy per image (nJ)	energy per image (nJ)	reduction per image
2	541	1024	553	5682	90.3%
3	249	256	64	11,360	99.4%

VII. Conclusions

CLNNs are a promising approach to reduce the energy overhead of memory access in ultra low-latency NNs, but the large logic size has limited their application at scale in area- and energy-constrained applications. We presented a design optimization method for the energy-efficient implementation of CLNNs allowing for scalable synthesize of deeper NNs. Exploiting the error-resilient nature of NNs, we proposed two techniques to over-simplify the input-to-output mapping of neurons and a coordinate descent-based optimization algorithm to determine the optimal level of simplification. Compared to the conventional systolic array-based NN, more than 90% power reduction is demonstrated per layer with an accuracy of 82% on CIFAR-10, yet to achieved by previous CLNNs.

Acknowledgement

This work was supported by the National Science Foundation under award CNS-1845469.

References

[1] P. Chi *et al.*, "PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory," in *ISCA*, 2016.
[2] M. Rusci *et al.*, "Design automation for binarized neural networks: A quantum leap opportunity?" in *ISCAS*, 2018.
[3] C.-C. Chi *et al.*, "Logic synthesis of binarized neural networks for efficient circuit implementation," in *ICCAD*, 2018.
[4] M. Nazemi *et al.*, "Energy-efficient, low-latency realization of neural networks through Boolean logic minimization," in *ASP-DAC*, 2019.
[5] Y. Umuroglu *et al.*, "LogicNets: co-designed neural networks and circuits for extreme-throughput applications," in *FPL*, 2020.
[6] E. Wang *et al.*, "LUTNet: Rethinking inference in FPGA soft logic," in *FCCM*, 2019.
[7] S. Rai *et al.*, "Logic synthesis meets machine learning: Trading exactness for generalization," in *DATE*, 2021.
[8] Y. LeCun *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, 1998.
[9] I. Hubara *et al.*, "Binarized neural networks," in *NIPS*, 2016.
[10] G. Castellano *et al.*, "An iterative pruning algorithm for feedforward neural networks," *IEEE Transactions on Neural Networks*, 1997.
[11] H. Li *et al.*, "Pruning filters for efficient convnets," *arXiv preprint arXiv:1608.08710*, 2016.
[12] S. Han *et al.*, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.
[13] M. Cheng *et al.*, "TIME: A training-in-memory architecture for memristor-based deep neural networks," in *DAC*, 2017.
[14] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *ICLR*, 2014.
[15] P. Panda *et al.*, "Conditional deep learning for energy-efficient and enhanced pattern recognition," in *DATE*, 2016.
[16] A. Samajdar *et al.*, "SCALE-Sim: Systolic CNN accelerator simulator," *arXiv preprint arXiv:1811.02883*, 2018.
[17] K. Chandrasekar *et al.*, "Improved power modeling of DDR SDRAMs," in *Euromicro DSD*, 2011.
[18] Y.-H. Chen *et al.*, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *JSSC*, 2016.