TinyADC: Peripheral Circuit-aware Weight Pruning Framework for Mixed-signal DNN Accelerators

Geng Yuan*[1], Payman Behnam*[2], Yuxuan Cai[1], Ali Shafiee[3], Jingyan Fu[4], Zhiheng Liao[4], Zhengang Li[1], Xiaolong Ma[1], Jieren Deng[5], Jinhui Wang[6], Mahdi Bojnordi[7], Yanzhi Wang[1], Caiwen Ding[5]

[1]*Northeastern University,* [2]*Georgia Institute of Technology,* [3]*Samsung,* [4]*North Dakota State University,*
[5]*University of Connecticut,* [6]*University of South Alabama,* [7]*University of Utah*

[1]{yuan.geng, cai.yuxu, li.zhen, ma.xiaol, yanz.wang}@northeastern.edu,
[2]payman.behnam@gatech.edu [3]ali.shafiee@samsung.com [4]{jingyan.fu, zhiheng.liao}@ndsu.edu,
[5] {jieren.deng, caiwen.ding}@uconn.edu, [6]jwang@southalabama.edu, [7]bojnordi@cs.utah.edu

***Abstract*—As the number of weight parameters in deep neural networks (DNNs) continues growing, the demand for ultra-efficient DNN accelerators has motivated research on non-traditional architectures with emerging technologies. Resistive Random-Access Memory (ReRAM) crossbar has been utilized to perform in-situ matrix-vector multiplication of DNNs. DNN weight pruning techniques have also been applied to ReRAM-based mixed-signal DNN accelerators, focusing on reducing weight storage and accelerating computation. However, the existing works capture very few peripheral circuits features such as Analog to Digital converters (ADCs) during the neural network design. Unfortunately, ADCs have become the main part of power consumption and area cost of current mixed-signal accelerators, and the large overhead of these peripheral circuits is not solved efficiently.**

To address this problem, we propose a novel weight pruning framework for ReRAM-based mixed-signal DNN accelerators, named TINYADC, which effectively reduces the required bits for ADC resolution and hence the overall area and power consumption of the accelerator without introducing any computational inaccuracy. Compared to state-of-the-art pruning work on the ImageNet dataset, TINYADC achieves 3.5× and 2.9× power and area reduction, respectively. TINYADC framework optimizes the throughput of state-of-the-art architecture design by 29% and 40% in terms of the throughput per unit of millimeter square and watt ($\frac{GOPs}{s \times mm^2}$ and $\frac{GOPs}{w}$), respectively.

I. INTRODUCTION

In recent years, DNNs have achieved remarkable progress across many fields such as computer vision [1], natural language processing [2]. However, as Moore's Law is coming to an end, the massive memory and computational required by state-of-the-art DNN models have been burdening traditional Von-Neumann architecture. To mitigate the intensive computation and memory storage of DNN models, the crossbar arrays using ReRAM device have been widely investigated. The main reason is that crossbar arrays perform matrix-vector multiplication (the most computing intensive operation in DNNs) in the analog domain and solve systems of linear equations in $O(1)$ time complexity [3]. As a result, ReRAM-based mixed-signal DNN accelerators are developed for high efficient DNN computations [4]–[6].

On the other hand, weight pruning techniques have also been applied to ReRAM-based DNN accelerators to effectively reduce the DNN model storage and computations [7]–[10]. However, by taking low power and high integration density advantages of the ReRAM devices, in the recent development of mixed-signal DNN accelerators, the major portion of the area and power consumption comes from the peripheral circuits such as ADCs. For example, in [5], more than 51% and 31% of the area and power overhead of a tile is the ADC usage. The hardware cost of ADCs can even go beyond 90% in some cases [11]. Unfortunately, such a high percentage of ADC makes current weight pruning techniques inefficient to reduce the overall hardware costs of the DNN accelerators, even with a high weight pruning rate, i.e., the bottleneck is not storage but ADC peripheral circuitry. The current weight pruning techniques are peripheral circuit oblivious and do not considers the costs of ADCs during neural network design. Thus, a peripheral circuit-aware solution is highly desired, which can effectively reduce the hardware costs of both model storage and peripheral circuits while maintaining the DNN model accuracy.

To address the challenge, we propose TINYADC, a weight pruning framework considering the required ADC resolution during pruning process. TINYADC can effectively and systematically reduce the area and power consumption cost on both the model storage and the ADC circuit, while a high model accuracy can be preserved. We summarize the contributions as follows:

- We propose a novel pruning scheme, named column proportional pruning, which reduces the required bits for ADC resolution during DNN inference. Smaller ADCs can be used to replace the original ADCs without introducing any computational inaccuracy.
- Our framework combines structured pruning schemes with our proposed column proportional pruning in a crossbar size-aware fashion. As a two-pronged solution, our combined pruning scheme reduces both the cost of individual ADC and the number of ADCs (and crossbars). The overall hardware costs of the accelerator are further reduced.
- Compared to state-of-the-art pruning works, TINYADC achieves a higher pruning rate while maintaining higher accuracy. Moreover, lower area and power consumption and higher throughput are achieved when TINYADC framework is applied to ReRAM-based mixed-signal DNN accelerators.

*These Authors contributed equally.

978-3-9819263-5-4/DATE21/©2021 EDAA

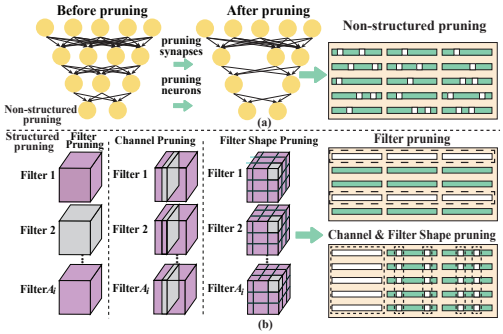

Fig. 1. (a) Non-structured pruning and (b) structured pruning.

II. Background and Motivation

A. DNN Weight Pruning

There are two most trendy pruning schemes: Non-structured pruning and structured pruning.

1) **Non-structured Pruning:** The non-structured pruning methods [12], [13] aim to heuristically prune the redundant weights on arbitrary locations, as shown in Fig. 1(a). It has high flexibility on selecting desired pruning locations, which can generally maintain the accuracy under a high pruning rate. However, due to its irregular weight distribution, in a ReRAM-based mixed-signal accelerator, the pruned weights (zero weights) still need to be mapped on the ReRAM crossbars and cannot lead to any hardware reduction or acceleration.

2) **Structured Pruning:** Structured pruning is proposed to structurally remove entire filters, channels or filter shapes from the weight matrix [7], [14], as shown in Fig. 1(b). By taking advantage of the regular shapes of the pruned weight matrices, structured pruning avoids introducing extra indices to indicate the pruned locations and becomes more hardware-friendly. However, due to the coarse pruning granularity used in structured pruning, a considerable accuracy drop can be observed under a high pruning rate. This drawback limits the hardware reduction and acceleration achieved by structured pruning.

B. ADC in ReRAM-based Mixed-signal DNN Accelerator

In ReRAM-based mixed-signal DNN accelerators, the weights in a DNN layer (especially for the later layers which contain a huge number of weights) are usually stored in multiple crossbar arrays. Generally, each crossbar array will have its own ADC(s), and each ADC will be shared by some or all the columns within the same crossbar array. Assuming v input bits are shifted into crossbar arrays per cycle, w weight bits are stored per ReRAM cell on the crossbar arrays, and r rows per each crossbar array are activated. The required number of bits for ADC is computed as follows:

$$ADC_{bits} = \begin{cases} v + w + log(r), & \text{if } v > 1 \ \& \ w > 1 \\ v + w + log(r) - 1, & \text{otherwise.} \end{cases} \quad (1)$$

For example, if the crossbar array has 8 activated rows, 1-bit DAC is used, and we use a 2-bit multi-level ReRAM cell (MLC), then a 5-bit ADC is required to ensure computational accuracy. The area and power consumption of ADCs is growing almost exponentially by adding each 1-bit precision [15]. This makes ADCs the main contributor of hardware overhead in mixed-signal DNN accelerators.

C. Motivations

Weight pruning techniques have been demonstrated as an effective method to reduce the DNN model size for less computation and memory intensity. However, both of the non-structured pruning and structured pruning have shortcomings that prevent them from being the perfect solution for mixed-signal DNN accelerator designs. Moreover, since the major portion of the area and power consumption becomes the peripheral circuits such as ADCs, the overall power and area reduction of accelerators are limited by only reducing the weight storage. Thus, we raise two design questions:

- **Question 1:** Is there a desired pruning scheme that takes both the advantage of non-structured pruning and structured pruning and effectively reduces the hardware costs of peripheral circuits in ReRAM-based accelerators?
- **Question 2:** How does the desired pruning scheme compare with existed schemes? Are they able to work collaboratively?

III. Framework Design

A. ADC-aware Column Proportional Pruning

To answer the first question in Section II-C, we propose an ADC-aware pruning scheme, named column proportional pruning, which targets on reducing the required bits for ADC resolution and hence the area and power consumption of ReRAM-based crossbars. Column proportional pruning is an intermediate pruning scheme takes both the high model accuracy advantage of non-structured pruning and the hardware-friendliness advantage of structured pruning.

In column proportional pruning, we fix the number of non-zero weights on each column of a crossbar array, while we do not specify the locations for those non-zero weights. At the same time, we require the number of non-zero weights the same for all columns. Fig. 2 demonstrates an example of column proportional pruning with the crossbar array size of 8×8 and assumes 1-bit DAC and 2-bit MLC ReRAM cells are used. In this case, blocks of 8×8 weights from the DNN weight matrix will be mapped on to the 8×8 ReRAM crossbar arrays. If a 4× column proportional pruning is applied, only two non-zero weights will be allowed in each column of weight matrix and the remaining weights will be pruned to zero during the offline training process. Then, the column proportional pruned weights will be mapped to crossbar arrays for inference. Without pruning, all eight weights (ReRAM cells) on each column are non-zero, which indicates all eight rows of the crossbar are activated and need to participate in the computation. Thus, a minimum of 5-bit ADC is required to avoid computational inaccuracy, as we mentioned in Section II-B. On the contrary, with the column proportional pruning, six weights (ReRAM cells) on each column are pruned to zero, which can be considered as deactivating those six rows during the computation regardless of the input data. Thus, a 3-bit ADC can be used to replace the 5-bit ADC without introducing any computational inaccuracy. By reducing two bits resolution of ADCs, the power and area of each ADC can be significantly reduced.

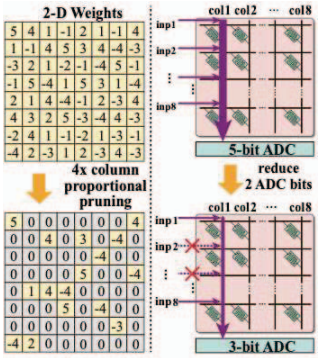

Fig. 2. Column proportional pruning using 8×8 crossbar array.

The high model accuracy of column proportional pruning is attributed to the structural flexibility. Since we do not restrict the position of the non-zero weights in each column, it provides high flexibility of searching the desired pruning locations.

The hardware-friendliness is achieved by avoiding the use of the indices during the computation. By limiting the number of non-zero weights in each column, it ensures the number of activated rows calculated by the ADC, thus we do not need to identify the exact locations of pruned weights.

B. Dynamic Regularization-based Optimization using ADMM

To obtain DNN models with our proposed column proportional pruning feature, we incorporate alternating direction method of multipliers (ADMM) [13] during the offline training process. ADMM is an optimization technique that has been introduced to transform the DNN weight pruning problems into an optimization problem of dynamically updating the regularization terms restricted by the designated constraint sets (i.e., pruning with specific dimensions or with any desired weight matrix shapes). It decomposes an original DNN weight pruning problem into two sub-problems, where one can be solved by standard stochastic gradient descent (SGD) while the other one can be solved by iteratively updating regularization terms with ADMM steps.

We formulate the weight pruning problem in an N-layer DNN as:

$$\begin{aligned} \underset{\{\mathbf{W}_i\},\{\mathbf{b}_i\}}{\text{min}} \quad & \mathcal{L}\big(\{\mathbf{W}_i\}_{i=1}^N, \{\mathbf{b}_i\}_{i=1}^N\big), \\ \text{subject to} \quad & \mathbf{W}_i \in \mathbf{S}_i,\ i = 1, \ldots, N, \end{aligned} \tag{2}$$

where $\mathbf{W}_i$ and $\mathbf{b}_i$ represent the weights and biases in $i-th$ layer. $\mathcal{L}$ is the loss function with respect to $\mathbf{W}_i$ and $\mathbf{b}_i$. $\mathbf{S}_i$ is the constraint of the remaining weights satisfying the requirement of **column proportional pruning**.

The weights of a convolutional layer are represented in a 4-D tensor format $\mathbf{W}_i \in \mathbb{R}^{w\times h\times c\times f}$, where w, h, c, f, are the weight kernel width, kernel height, number of channels, and number of filters, respectively. In ReRAM-based mixed-signal DNN accelerator, the column proportional pruning constraint is based on the crossbar array. A large layer needs several crossbar arrays to accommodate all the weights, thus we divide the weights into several blocks, where the block size is the same as the size of crossbar arrays. Each block of weights will be mapped to one corresponding crossbar array. Assume the weights in i-th are divided into b_i blocks, and crossbar array size is $m \times n$. Given the value of l_i (number of non-zero values in each column of crossbar arrays in i-th layer), the constraint in the i-th convolutional layer becomes $\mathbf{W}_i \in \mathbf{S}_i := \{\mathbf{W_i} \mid \forall col_{\alpha,\beta} :\ card(col_{\alpha,\beta}) \le l_i,\ for\ \alpha \in 1, \ldots, n,\ \beta \in 1, \ldots, b_i\}$, where $card(\cdot)$ returns the number of non-zero weights in $col_{\alpha,\beta}$, meaning the α-th column in β-th block.

Since the problem (2) with constraint cannot be directly solved by classic stochastic gradient descent (SGD) methods as original DNN training, we utilize the ADMM regularization to reforge and separate the problem into two sub-problems, then solve them iteratively.

First, we incorporate the constraint $\mathbf{S}_i$ by using indicator function, which is

$$g_i(\mathbf{W}_i) = \begin{cases} 0 & \text{if } \mathbf{W}_i \in \mathbf{S}_i, \\ +\infty & \text{otherwise.} \end{cases}$$

We reformulate problem (2) as follows:

$$\begin{aligned} \underset{\{\mathbf{W}_i\},\{\mathbf{b}_i\}}{\text{minimize}} \quad & f\big(\{\mathbf{W}_i\}_{i=1}^N, \{\mathbf{b}_i\}_{i=1}^N\big) + \sum_{i=1}^N g_i(\mathbf{Z}_i), \\ \text{subject to} \quad & \mathbf{W}_i = \mathbf{Z}_i,\ i = 1, \ldots, N, \end{aligned} \tag{3}$$

where $\mathbf{Z}_i$ is an auxiliary variable. With the formation of augmented Lagrangian [16], problem (3) can be decomposed into two subproblems (4) and (5),

$$\underset{\{\mathbf{W}_i\},\{\mathbf{b}_i\}}{\text{minimize}} \quad f\big(\{\mathbf{W}_i\}_{i=1}^N, \{\mathbf{b}_i\}_{i=1}^N\big) + \sum_{i=1}^N \frac{\rho_i}{2} \|\mathbf{W}_i - \mathbf{Z}_i^t + \mathbf{U}_i^t\|_F^2, \tag{4}$$

$$\underset{\{\mathbf{Z}_i\}}{\text{minimize}} \quad \sum_{i=1}^N g_i(\mathbf{Z}_i) + \sum_{i=1}^N \frac{\rho_i}{2} \|\mathbf{W}_i^{t+1} - \mathbf{Z}_i + \mathbf{U}_i^t\|_F^2, \tag{5}$$

where $\mathbf{U}_i$ denotes dual variable and t is the iteration index. The positive scalars ρ_i is a penalty hyperparameter for the $L2$ regularization. The first subproblem can be solved by classic SGD, and the solution for the second subproblem is given by

$$\mathbf{Z}_i^{t+1} = \prod_{\mathbf{X}_i} (\mathbf{W}_i^{t+1} + \mathbf{U}_i^t), \tag{6}$$

where $\prod_{\mathbf{X}_i}(\star)$ is Euclidean projection to $\mathbf{X}_i \in \mathbf{S_i}$, thereby weight matrices are column proportional pruned. These two subproblems will be iteratively solved and we update $\mathbf{U}_i$ in each iteration by $\mathbf{U}_i^t := \mathbf{U}_i^{t-1} + \mathbf{W}_i^t - \mathbf{Z}_i^t$ until convergence. Here, we have the pruned model with the weights satisfying the column proportional constraint, and can be used to map to the ReRAM crossbar arrays.

C. ReRAM Crossbar Mapping Scheme

When mapping a large number of DNN weights onto the ReRAM crossbar arrays, multiple crossbar arrays are needed to accommodate all the weights from one layer. For instance, as shown in Fig. 3, we first flatten the weights to form a 2-D weight matrix, which each column of the 2-D weight matrix contains all the weights from one filter. The first column on the 2-D weight matrix (marked as yellow in Fig. 3 top) represents the weights from $filter\ 1$. If we choose the ReRAM crossbar arrays with the size of $m\_rows \times n\_columns$, the 2-D weight matrix is split into multiple blocks, with the total size of $m\_rows \times n\_columns$. Since the column proportional pruning is applied during the training process, the weights in each block column satisfy the column proportional constraint

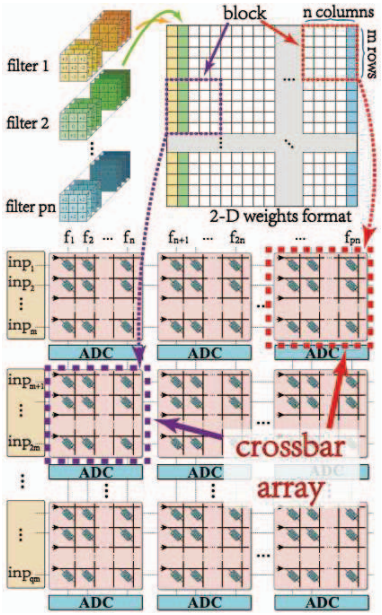

Fig. 3. ReRAM crossbar mapping scheme.

(which only allows l_s fixed non-zero weights). The weights in each block (including the zero weights) will be mapped onto the corresponding ReRAM crossbar array. Note that if the number of columns/rows cannot be divided by the block column/row size, then additional crossbar arrays are needed for those weights on the remaining columns/rows.

Due to the high hardware overhead of sense amplifiers of high ReRAM MLCs and also reliability issues, using more than 2-3 ReRAM bit cells are not practical. As a result, multiple ReRAM cells are used jointly to represent one weight.

D. Exploration of Combined Pruning

To answer the second question in Section II-C, we manage to combine structured pruning with our proposed column proportional pruning. Our column proportional pruning can effectively reduce the power and area costs of each ADC, while the structured pruning reduces the number of crossbar arrays and their associated peripheral circuits. Thus, the combined pruning can be used as a two-pronged solution, reducing both the cost of each ADC and the number of crossbar arrays with their peripheral circuits. Eventually, the overall hardware costs of the accelerator are further reduced.

Two types of structured pruning methods are used in TINYADC, which are filter pruning and filter-shape pruning [7]. Pruning a filter can be considered as removing the weights of one entire column from the 2-D weights format in Fig. 3, while pruning a filter-shape represents removing the weights on one entire row. TINYADC uses an efficient structure pruning by considering the crossbar array size during the training process. In each DNN layer, we only prune the number of filters/filter-shapes that equals the multiples of the column/row size of the crossbar array. For example, if the crossbar array size is 128×128, then the possible numbers of pruned filters/filter-shapes are 128, 256, 384, 512 and so on. In this way, after removing the pruned weights, the remaining weights can still reform a dense matrix with the size of the multiples of crossbar array size. As the result, the structural pruned weights can be fully converted to the crossbar array reductions.

In combined pruning, the filter pruning can be applied before or after the column proportional pruning. However, the filter-shape pruning can only be applied before the column proportional pruning. Because the reformed weight matrix may violate the column proportional constraint if the filter-shape pruning is applied after the column proportional pruning. Compared to only using the column proportional pruning, the pruning rate of column proportional pruning may be reduced to maintain accuracy when it is combined with the structured pruning.

IV. EVALUATION

A. Experimental Setup

We evaluate our framework on the CIFAR-10 and CIFAR-100 dataset using ResNet18, ResNet50, and VGG16, respectively. For large dataset, we test on the ImageNet dataset using ResNet18. Our sparse models are trained on a server with four NVIDIA RTX-2080Ti GPUs using PyTorch.

We utilize a developed in-house tool at 32nm named NVCACTI to model the area and power of all crossbar arrays, buffers, sense amplifiers and on-chip interconnects [17]. NVCACTI builds a unified framework that can models both volatile and non-volatile memories with multi-banking properties and efficient wiring model. We use the VTEAM ReRAM model [18] to set the parameters of crossbar arrays and conservatively consider a 10% process variation during evaluations. To make a fair and consistent comparison, we compute the power, and area of the same ADC [19] with different resolutions. To do so, we scale down the area, and power the memory, clock, and *vref* buffer linearly, and the capacitive DAC exponentially. In our design, 2-bit MLC ReRAM and 128×128 crossbar arrays [5] are used, and all the results are normalized to a non-pruned design.

B. Evaluation on Column Proportional Pruning

1) ***Pruning Rate & Accuracy****:* Table I shows the results of model accuracy under different column proportional pruning rate on various networks and datasets. The *"CP pruning"* column in the table represents the *column proportional pruning rate*, which equals to $\frac{crossbar\ array\ column\ size}{non-zero\ weights\ per\ column}$. For example, with the column size of 128, 32× column proportional pruning leads to only 4 non-zero weights remained on each column, and all other 124 weights are pruned to zero. The *"ADC Bits Reduction"* column represents the reduction of required bits of ADC resolution compared with the non-pruned design which uses crossbar arrays of size 128×128 and 8-bit ADCs. The results in Table I refer to applying a uniform pruning rate to all the convolutional layers except the first layer. Thus, the ADC bits reduction is evenly applied to all ADCs except for the first layer. On the CIFAR-10 dataset, TINYADC achieves up to 64× and 32× column proportional pruning rate without accuracy degradation on ResNet18 and VGG16 respectively. This will lead to a 6-bits and 5-bits reduction on ADCs' required bits without introducing computational inaccuracy.

TABLE I
ACCURACY UNDER DIFFERENT COLUMN PROPORTIONAL PRUNING RATE ON DIFFERENT DATASETS AND NETWORKS

Method	Original Acc. (%)	CP pruning	Final Acc. (%)	ADC Reduction
		CIFAR10		
TinyADC ResNet18	94.14	16×	94.51	-4 bits
		32×	94.22	-5 bits
		64×	**94.17**	**-6 bits**
TinyADC VGG16	93.70	16×	93.95	-4 bits
		32×	**93.74**	**-5 bits**
		64×	92.41	-6 bits
		CIFAR100		
TinyADC ResNet18	76.01	8×	76.13	-3 bits
		16×	76.08	-4 bits
		32×	**75.98**	**-5 bits**
TinyADC ResNet50	77.85	8×	78.13	-3 bits
		16×	78.01	-4 bits
		32×	**77.87**	**-5 bits**
TinyADC VGG16	74.62	8×	74.93	-3 bits
		16×	**74.60**	**-4 bits**
		32×	74.23	-5 bits
		ImageNet		
TinyADC ResNet18	89.07	2×	89.09	-1 bits
		4×	**88.64**	**-2 bits**

* The TinyADC results are based on column proportional pruning.

On the ImageNet dataset, we test our column proportional pruning on ResNet18 and achieves 2× pruning rate without any accuracy degradation, where there is a minor accuracy degradation when we apply a 4× pruning rate. Since the classification task on the ImageNet is much complex than on the CIFAR-10, it is expected that the pruning rate on the ImageNet is much lower than on the CIFAR-10.

*2) **Area & Power**:* Fig. 4 demonstrate the estimated power and area results of accelerator designs that runs a specific network over a specific dataset. The results belong to the best column proportional pruning rate selected from Table I (i.e., bold rows) and they are normalized to the non-pruned baseline results. Herein, we design separate accelerators to be able to show the effectiveness of our column proportional pruning scheme on different datasets and networks in terms of area and power reduction. As the results show, 37% power reduction and 22% area reduction are achieved on ImageNet using ResNet18, while up to 62% power reduction and 45% area reduction are achieved on the CIFAR-10.

C. Evaluation on Combined Pruning

*1) **Pruning Rate & Accuracy**:* Table II shows the comparison results between several previous works and TINYADC (without and with combining the structured pruning). Our proposed column proportional pruning scheme and combined pruning scheme consistently outperforms the reference works in terms of pruning rate and accuracy on all datasets and networks. Our results also indicate that the structured pruning can be combined with our column proportional pruning. By combining the structured pruning, the overall pruning rate of our TINYADC can be significantly improved with minor accuracy degradation. However, to maintain the accuracy, we have to make a trade-off between the pruning rate of two pruning schemes.

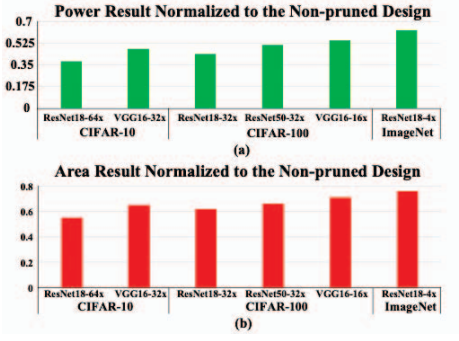

Fig. 4. (a) Power and (b) area results normalized to non-pruned design.

TABLE II
COMPARISON BETWEEN REFERENCE WORKS AND TINYADC WITH COLUMN PROPORTIONAL PRUNING AND COMBINED PRUNING ON DIFFERENT DATASETS AND NETWORKS

Network	Method	Original Acc. (%)	Structured Pruning	CP Pruning	Overall Pruning	Final Acc. (%)	Crossbar Reduction	ADC Bits Reduction
				CIFAR10				
ResNet18	Ultra-Effifcient [7]	94.14	20.88×	–	20.88×	93.20	-89.31%	–
	TinyButAcc [8]	94.14	59.84×	–	59.84×	93.20	-93.12%	–
	TinyADC w/o SP [1]	94.14	–	64×	64×	94.17	–	-6 bits
	TinyADC	94.14	7.5×	16×	120×	93.92	-86.38%	-4 bits
VGG16	Ultra-Effifcient [7]	93.70	29.81×	–	29.81×	93.36	-91.46%	–
	TinyButAcc [8]	93.70	44.67×	–	44.67×	93.36	-93.46%	–
	TinyADC w/o SP	93.70	–	32×	32×	93.74	–	-5 bits
	TinyADC	93.70	7.63×	8×	61×	93.64	-86.81%	-3 bits
				CIFAR100				
ResNet18	N2N [20]	72.22	4.64× [2]	–	4.64× [2]	68.01	–	–
	TinyADC w/o SP	76.01	–	32×	32×	75.98	–	-5 bits
	TinyADC	76.01	1.6×	16×	25×	75.79	-36.90%	-4 bits
ResNet50	TinyADC w/o SP	77.85	–	32×	32×	77.87	–	-5 bits
	TinyADC	77.85	2.06×	32×	65.92×	76.80	-51.45%	-5 bits
VGG16	SSL [21]	73.16	2.6×	–	2.6×	73.18	-49.89%	–
	Decorralation [22]	73.16	3.9×	–	3.9×	73.21	-51.88%	–
	TinyADC w/o SP	74.62	–	16×	16×	74.60	–	-4 bits
	TinyADC	74.62	1.78×	16×	28.48×	74.03	-43.90%	-4 bits
				ImageNet				
ResNet18	DCP [23]	88.98	2×	–	2×	87.60	-49.49%	–
	DCP [23]	88.98	3.3×	–	3.3×	85.68	-68.81%	–
	TinyADC w/o SP	89.07	–	4×	4×	88.64	–	-2 bits
	TinyADC	89.07	2.3×	2×	4.6×	88.38	-54.63%	-1 bits

[1] *TinyADC w/o SP* stands for TinyADC without combining structured pruning.
[2] The pruning rate for N2N [20] is based on non-structured pruning, which cannot achieve crossbar reduction.

*2) **Area & Power**:* We also compare the area and power of the combined pruning method with existing baselines [1]. Like the previous section, we consider for all of the baselines and TINYADC and also for each network/dataset, a separate architecture can be designed to show how much each different technique can ameliorate the area and power. Fig. 5 shows the area and power comparison results of TinyADC using combined pruning with different baseline designs over different neural networks and datasets. All the results are normalized to a non-pruned design. Compared to non-pruned design, for the CIFAR-10 using ResNet18, TINYADC achieves 15× and 12× reduction on power and area, respectively. The baseline work TinyButAcc [8] also achieves similar area reduction using structured pruning but with non-negligible accuracy drop. TINYADC performs more obvious advantages on more complex dataset CIFAR-100 and ImageNet. Compared to work Decorralation [22], TINYADC saves 21% more area and saves 38% more power on the CIFAR-100 using VGG16. On the ImageNet using ResNet18, compared to non-pruned design, TINYADC achieves 3.5× and 2.9× reduction on power and area, respectively, with 0.69% top-5 accuracy degradation. In comparison, DCP [23] only achieves 2× reduction on both

[1]Some of the baselines didn't provide the results for some networks and/or baseline. Hence, we were not able to provide the area/power results for them

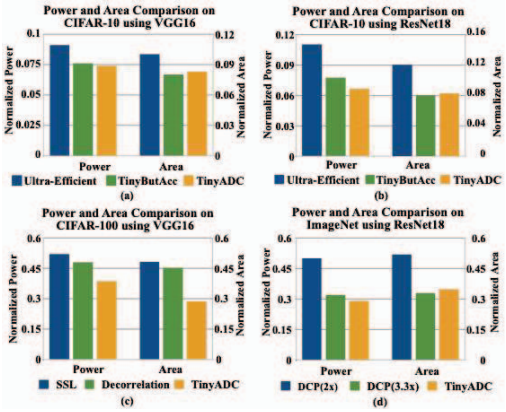

Fig. 5. (a) Power and (b) area results normalized to non-pruned design.

TABLE III
COMPARISON THE PEAK THROUGHPUT OF DIFFERENT ARCHITECTURES.

Architecture	$\frac{GOPs}{s \times mm^2}$	$\frac{GOPs}{W}$
DaDianNao [24]	63.46	286.4
TPU [25]	40.88	301.91
PUMA [6]	338.76	497.25
ISAAC [5]	478.95	627.5
TinyADC(ISAAC)	**621.19**	**879.1**

power and area but with a 1.38% top-5 accuracy drop. When DCP has a similar power and area reduction with TINYADC, a 3.3% top-5 accuracy drop is observed.

D. Throughput Analysis

We optimize the state-of-the-art architecture design by adopting our TINYADC framework. Here, we consider a reconfigurable design that can run all types of tested datasets and networks on it. Thus, the worst case scenario (i.e., ImageNet using ResNet18) is evaluated. Using smaller ADCs will lead to fewer and smaller intermediate results compared with the baseline architecture. Consequently, buffers, sample&hold and shift-and-add logics are smaller and faster. Moreover, by reducing the ADC resolution, designers are able to select smaller ADCs with higher frequency or use more ADCs per crossbar. As the result shows in Table III, optimized by TINYADC, the throughput of ISAAC can be significantly improved by 29% and 40% in terms of $\frac{GOPs}{s \times mm^2}$ and $\frac{GOPs}{w}$, respectively.

E. Fault Tolerance Analysis

Interestingly, our proposed TINYADC framework also improves the fault tolerance of ReRAM crossbars. Besides the power saving and area reduction that is the same benefit with the structured pruning method, our proposed column proportional pruning intentionally holds a large amount of zero weights on ReRAM crossbars. This provides TINYADC a qualified tolerance for the Stuck-At-0 (SA0) Fault of ReRAM-based crossbar arrays. By using the ReRAM SA0 failure model in [26] with 5%, 10%, and 15% overall Stuck-At Fault rate, on the complex dataset (i.e., ImageNet), TINYADC framework demonstrates higher reliability property, i.e., the accuracy drop of TINYADC is 0.5%, 1.8%, and 3.9% lower than that of DCP (with 3.3× pruning rate).

V. CONCLUSION

In this paper, we propose TINYADC, a peripheral circuit-aware pruning framework for ReRAM-based mixed-signal DNN accelerators. TINYADC can effectively reduce the required bits for ADC resolution and hence the overall area and power consumption of the accelerator without introducing any computational inaccuracy.

ACKNOWLEDGEMENT

This work is funded by the National Science Foundation Awards CCF-1937500 and CNS-1909172.

REFERENCES

[1] I. Goodfellow *et al.*, *Deep learning*. MIT press, 2016.
[2] J. Devlin *et al.*, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of NAACL-HLT*, 2019.
[3] G. Yuan *et al.*, "Memristor crossbar-based ultra-efficient next-generation baseband processors," in *IEEE MWSCAS*, 2017.
[4] L. Song *et al.*, "Pipelayer: A pipelined reram-based accelerator for deep learning," in *2017 IEEE International Symposium on HPCA*, 2019.
[5] A. Shafiee *et al.*, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *Proceedings of ISCA*, 2016.
[6] A. Ankit *et al.*, "Puma: A programmable ultra-efficient memristor-based accelerator for machine learning inference," in *ASPLOS*, 2019.
[7] G. Yuan *et al.*, "An ultra-efficient memristor-based dnn framework with structured weight pruning and quantization using admm," in *IEEE/ACM ISLPED*, 2019.
[8] X. Ma *et al.*, "Tiny but accurate: A pruned, quantized and optimized memristor crossbar framework for ultra efficient dnn implementation," *2020 25th ASP-DAC*, 2020.
[9] X. Ma, S. Lin *et al.*, "Non-structured dnn weight pruning – is it beneficial in any platform?" 2019.
[10] C. Ding *et al.*, "Structured weight matrices-based hardware accelerators in deep neural networks: Fpgas and asics," in *Great Lakes Symposium on VLSI*, 2018.
[11] M. Imani *et al.*, "Floatpim: In-memory acceleration of deep neural network training with high precision," in *ISCA*, 2019.
[12] S. Han *et al.*, "Learning both weights and connections for efficient neural network," in *Advances in NeurIPS*, 2015.
[13] T. Zhang *et al.*, "A systematic dnn weight pruning framework using alternating direction method of multipliers," in *ECCV*, 2018.
[14] Y. He *et al.*, "Soft filter pruning for accelerating deep convolutional neural networks," in *Proceedings of the 27th IJCAI*, 2018.
[15] B. Murmann, "Adc performance survey 1997-2019,[online]. available: http://web.stanford.edu/ murmann/adcsurvey.html."
[16] S. Boyd *et al.*, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine Learning*, 2011.
[17] P. Behnam, A. P. Chowdhury, and M. N. Bojnordi, "R-cache: A highly set-associative in-package cache using memristive arrays," in *2018 IEEE 36th International Conference on Computer Design (ICCD)*. IEEE, 2018, pp. 423–430.
[18] S. Kvatinsky *et al.*, "Vteam: A general model for voltage-controlled memristors," *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2015.
[19] C.-H. Chan *et al.*, "16.4 a 5mw 7b 2.4 gs/s 1-then-2b/cycle sar adc with background offset calibration," in *2017 IEEE ISSCC*, 2017.
[20] A. Ashok *et al.*, "N2n learning: Network to network compression via policy gradient reinforcement learning," 2017.
[21] W. Wen *et al.*, "Learning structured sparsity in deep neural networks," in *NeurIPS*, 2016.
[22] X. Zhu *et al.*, "Improving deep neural network sparsity through decorrelation regularization," in *Proceedings of the 27th IJCAI*, 2018.
[23] Z. Zhuang *et al.*, "Discrimination-aware channel pruning for deep neural networks," 2018.
[24] Y. Chen *et al.*, "Dadiannao: A machine-learning supercomputer," in *Proceedings of the 47th Annual IEEE/ACM MICRO*, 2014.
[25] Google supercharges machine learning tasks with TPU custom chip, https://cloudplatform.googleblog.com/2016/05/Google-supercharges-machine-learning-tasks-with-custom-chip.html.
[26] C. Chen *et al.*, "Rram defect modeling and failure analysis based on march test and a novel squeeze-search scheme," *IEEE Transactions on Computers*, 2015.