Fast Resilient-Aware Data Layout Organization for Resistive Computing Systems

Baogang Zhang*, M. G. Sarwar Murshed†, Faraz Hussain‡ and Rickard Ewetz§
*§Department of Electrical and Computer Engineering, University of Central Florida, Orlando FL, USA
†‡Department of Electrical and Computer Engineering, Clarkson University, Potsdam NY, USA
*baogang.zhang@knights.ucf.edu, †murshem@clarkson.edu, ‡fhussain@clarkson.edu, §rickard.ewetz@ucf.edu

***Abstract*—Resistive computing systems (RCSs) are projected to be leveraged as inference engines for Deep Neural Networks (DNNs). Unfortunately, limited device yield due to immature fabrication processes may severely degrade the DNN's classification accuracy. The arising solution is to leverage resilient-aware data layout organization techniques to mask the defects using the neural network weights. However, current techniques are too slow to be practical for real-world applications. In this paper, we propose a framework for fast resilient-aware data layout organization to enable large DNNs to be deployed on RCSs with defects. The framework contains three speed-up mechanisms: i) sparse defect indexing, ii) weight range characterization, and a iii) linear programming formulation. The first two techniques aim to quickly compute the errors introduced by various data to hardware assignments (or data layout organizations). The third technique aims to swiftly select the data layout organization that results in the smallest amount of errors. The experimental results demonstrate that the proposed framework is capable of achieving software level classification accuracy in resistive hardware without any use of retraining. Compared with the previous work, the run-time is reduced with 89% on the average.**

I. Introduction

Acceleration of DNNs using resistive computing systems (RCSs) has recently attracted significant interest due to their capability of natively performing energy-efficient multiply-and-accumulate (MAC) operations, which is the dominating computation within DNNs. Moreover, data movement is greatly reduced as the computation is performed in-memory, which circumvents the von-Neumann bottleneck. Nevertheless, RCSs are vulnerable to variations and non-ideal effects that may lead to system malfunction.

The performance of RCSs consisting of resistive crossbar arrays (RCAs) is impacted by non-zero array parasitics, non-linear device temperature variations, resistance drift and limited device yield. Many recent studies aim to improve the robustness of RCSs to the aforementioned issues [6], [10], [13], [17], [1], [12], [2], [11], [4], [18]. Among these challenges, limited device yield may be the most important, as only a few device defects can render an entire RCS system non-functional [12]. A resistive device that is stuck to the maximum conductance (stuck-on) or minimum conductance (stuck-off) is called a device defect or stuck-at-fault. Techniques to mitigate the negative impact of defects are based on hardware-aware training [2], [11], [12], [15], [4], digital compensation [4], utilizing redundant hardware [2], [5], [16], [18], and data layout organization [2], [11], [15], [18]. Hardware-aware training aims to train the weights of a DNN to mimic the defects in the hardware. However, hardware-aware training requires full access to the training data. Digital compensation is based on compensating for the defects using a digital co-processor, which introduces massive overheads. Techniques based on redundant hardware aim to compensate for defects by representing each weight using multiple parallel resistive devices. Resilient-aware data layout organization techniques attempt to mask the defects by modifying the data to hardware assignment, i.e., large (small) weights are assigned to devices stuck-on (stuck-off). In [2], [11], data layout organization was performed by permuting rows and columns in a weight matrix using routers. In [15], it was observed that data layout organization in DNNs can be performed by reordering the neurons in layers, which avoids the use of routers that introduce hardware overhead. Recent work on data layout organization is based on formulating and solving an assignment problem to find the data layout organization (or ordering of the neurons) in each layer that minimizes the weight errors (or cost) introduced by the defects [18]. The limitation is that the run-time of the technique is prohibitively long for real-world applications. In particular, the run-time of the resilient-aware data layout organization is 68 hours for a sixteen-layer convolutional neural network (CNN).

In this paper, we propose a framework to perform fast resilient-aware data layout organization to enable DNNs to be deployed on RCSs with defects. The framework reduces the run-time of the state-of-the-art data layout organization techniques using three speed-up mechanisms. A sparse defect indexing technique and a weight range technique are used to quickly compute the error cost of alternative data layout organizations. Specifically, the sparse defect indexing technique reduces the run-time by avoiding to compute "zero" cost of assigning weights to non-defective devices. The weight range technique is based on pre-characterizing the weight range that can be realized using hardware redundancy instead of computing the weight range dynamically for every weight. To swiftly select the data layout organization with the minimal total cost, a linear programming formulation is proposed. The proposed LP formulation solves the assignment problem faster than the Hungarian algorithm used in [18], [2]. The experi-

Research was supported in part by NSF awards CCF-1755825 and CNS-1908471.

2159-3477/20/$31.00 ©2020 IEEE
DOI 10.1109/ISVLSI49217.2020.00023

mental results show that the proposed framework is capable of achieving software level classification accuracy in resistive hardware. Compared with the state-of-the-art techniques, the run-time of the data layout organization for a 16-layer CNN is reduced from 68 hours to 7 hours.

The remainder of this paper is organized as follows: Section II gives the preliminaries; Section III reviews the data layout organization. The proposed speed-up techniques and the methodology are respectively given in Section IV and Section V; Section VI presents the evaluation of the proposed framework and Section VII concludes this paper.

II. Preliminaries

In this section, we introduce the background of deploying DNNs to RCSs for inference, the impact of device defects, and the problem definition.

DNN deployed on RCSs: A DNN consists of L layers of neurons are connected together by synapses. A DNN classifies an input image into one of multiple output categories by passing it to the first layer and recording the output from the last layer. Let x_l and y_l denote the input and output of neurons in layer l. In each layer, the output y_l is obtained by a MAC operation, $y_l = W_l x_l$, where W_l is the weight matrix connecting the neurons in layer l to the neurons in layer $l+1$. The output y_l is passed to a non-linear activation function $\sigma(.)$ and converted to the input x_{l+1} of the next layer, i.e., $x_{l+1} = \sigma(y_l)$. The software classification accuracy is equal to the ratio between the number of correctly classified input images and the number of total input images.

A DNN deployed on a RCS is shown in Figure 1. When a DNN is deployed on a RCS, each of the weight matrix W_l is assigned to an RCA_l (or a grid of RCAs), as shown in the middle of Figure 1. The RCAs perform MAC operations extremely energy-efficiently. However, any defects in the RCAs introduce errors when performing the MAC operations.

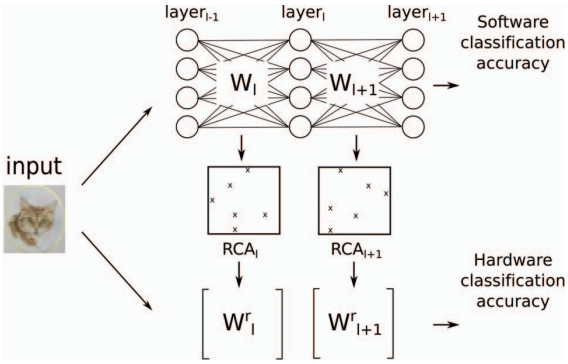


Fig. 1. DNN inference deployment on RCAs of RCSs.

MAC using RCAs with defects: Let G^d denotes a matrix with the device defects for an RCA. G^d can be determined using the technique in [1], [7], [14]. An RCA with defects G^d realizes an matrix W^r instead of the matrix W. Each element w^r_k in W^r is obtained, as follows:

$$w^r_k = \begin{cases} W_{max}, & \text{stuck-on,} \\ w_k, & \text{non-defective,} \\ W_{min}, & \text{stuck-off.} \end{cases} \quad (1)$$

where w_k is a weigh in weight matrix W. W_{max} and W_{min} are the maximum value and minimum value of weight matrix W, respectively. Given a DNN and the defects for each RCA, the classification accuracy in hardware is computed by evaluating the DNN using the weight matrices W^r instead of W, which is shown at the bottom of Figure 1.

Problem definition: Data layout organization is based on the observation that if the order of two neurons (in layer l of a DNN) are permuted, the network is functionally equivalent in software if the corresponding rows in W_l and columns in W_{l+1} are exchanged. However, the reordering of neurons will change the assignment of the weight matrices to the RCAs, which results in different classification accuracy in hardware. Consequently, the data layout organization problem consists of finding the ordering of the neurons in each layer that maximizes the classification accuracy in hardware.

III. Data layout organization in [18]

In this section, we review the data layout organizations in [18], which is an extension of the techniques in [2], [11], [15]. The technique is based on reordering the neurons in each layer, while minimizing a cost metric that measures the difference between the weight matrix W and the realized weight matrix W^r. The cost metric is introduced in Section III-A. The reordering of the neurons in a layer is performed by first computing the cost of various candidate data to hardware assignments (or data layout organizations). Next, the data layout organization that minimizes the cost is selected by solving an assignment problem, which is detailed in Section III-B. The run-time limitation of [18] is analyzed in Section III-C.

A. The Error Cost(EC) metric

Previous studies introduce a cost metric to measure the difference between the weight matrix W_l and the realized weight matrix W^r_l [2], [11], [15], [16], [18]. In [18], the weighted square error metric was used to compute the assignment cost, which is defined as follows:

$$EC = c_l \cdot \sum_{w_k \in W_l} (w_k - w^r_k)^2 \quad (2)$$

where EC is the error cost. c_l is the ratio of the number of times a weight matrix is used per image and the number weights in the matrix. w_k and w^r_k are the weights in the weight matrix W_l and the realized weight in matrix W^r_l, respectively.

B. Cost matrix computation and assignment problem

The reordering of the neurons in a layer can be viewed as the problem of assigning each of the neurons to a hardware location, which is illustrated in Figure 2 (a). Determining the mapping that minimizes the cost in Eq (2) can be formulated as an assignment problem, which requires a cost matrix C to computed. c_{ij} in C denotes the cost of assigning neuron i to location j. This involves computing the cost of assigning row i in W_l to row j in an RCA$_l$ and column i in W_{l+1} to column j in an RCA$_{l+1}$. Next, the Hungarian algorithm is used to find the neuron to hardware assignment that minimizes the cost in Eq (2) based on the cost matrix C.

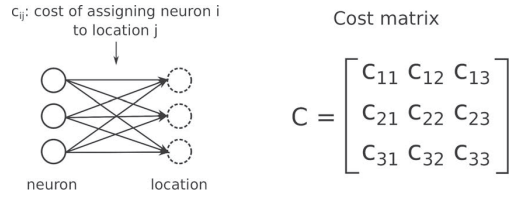


Fig. 2. (a) Formulation and (b) Cost matrix of the assignment problem.

C. Run-time Limitation of Data Layout Organization in [18]

When the resilient-aware data layout organization in [18] is applied to two 16-layer CNNs, it can be observed that the run-time is 1.7 hours and 67.9 hours, which is illustrated in Figure 3 (a). The run-time is longer for CNN-16b because it has been optimized for throughput. Clearly, the run-time is too long to be practical for real-world applications, which motivates the work in this paper. To identify the run-time bottleneck, we profile the run-time of the data layout organization of CNN-16b in Figure 3 (b). The figure shows that 96.9% of the run-time is consumed by computing the cost matrix of the assignment problem and 2.8% of the run-time is consumed by solving the assignment problem. Therefore, the speed-up techniques in the paper are focused on reducing the run-time of these two steps.

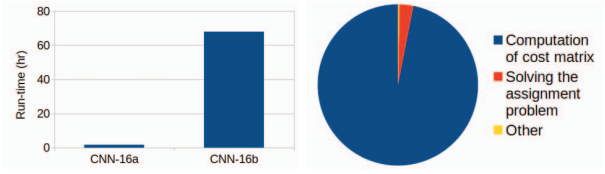


Fig. 3. Data layout organization run-time break down of a 16-layer network.

IV. Proposed Speed-up Techniques

In this section, we present the details of the three proposed speed-up techniques.

A. The sparse defect indexing technique

The sparse defect indexing technique aims to speed-up the computation of the cost matrix. The key insight of the technique is that the cost of assigning weight to non-defective device is equal to zero. Consequently, it is expected that the run-time can be significantly reduced by only computing the cost of assigning weights to defective devices.

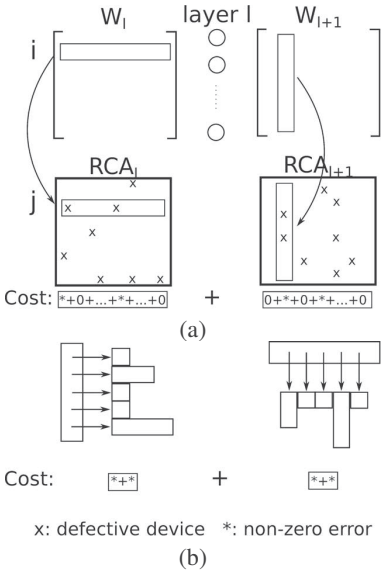


Fig. 4. (a) The cost computation of data to hardware and (b) the proposed sparse data structure for cost computation.

The computation of c_{ij} in the cost matrix C is obtained by respectively mapping row i in W_l to row j in an RCA$_l$ and column i in W_{l+1} to column j in an RCA$_{l+1}$, which is illustrated in Figure 4. It can be observed that many of the costs are equal to zero as the number of non-defective devices outnumbers the number of defective devices. To only compute the cost of assigning weights to defective devices, we introduce two adjacency matrices to store the location of the defects within each RCA. One stores the defect locations in a row-oriented fashion and the other stores the defect locations in a column oriented fashion. Consequently, when the cost of assigning row i in W_l to row j in an RCA$_l$ is computed, the framework iterates over the elements in the row-oriented data structure to only compute the cost of assigning weight to the defective devices. Similarly, the column-oriented data structure is used to compute the cost of assigning column i in W_{l+1} to column j in an RCA$_{l+1}$. The use of the defect indexing results in that the computation number of computing costs is proportional to the number of defective devices instead of the total number of devices.

B. The weight range characterization technique

The weight range technique aims to speed-up the computation of the cost matrix when more than one resistive device is used to realize each weight. The main idea is to pre-characterize the weight value range that can be realized by a set of parallel devices, which avoids dynamically computing the weight range every time a weight is assigned to the devices.

When R parallel resistive devices are used to realize a single weight, the realized weight w_k^r is computed in three steps [16], as follows:

Step 1: For the R parallel devices, count the number of device stuck-on d^H and device stuck-off d^L.

Step 2: Convert the d^H and d^L into an weight value range $[w_{min}, w_{max}]$, as follows:

$$\begin{aligned} w_{min} &= (d^H \cdot W_{max} + (R - d^H) \cdot W_{min})/R \\ w_{max} &= (d^L \cdot W_{min} + (R - d^L) \cdot W_{max})/R \end{aligned} \tag{3}$$

where w_{min} and w_{max} are the minimum and maximum of the weight range. W_{min} and W_{max} are the minimum and maximum values of matrix W_l, respectively.

Step 3: Compute the realized weight w_k^r based on the weight value range $[w_{min}, w_{max}]$, as follows:

$$w_k^r = \begin{cases} w_{min}, & w_k < w_{min}, \\ w_k, & w_{min} \leq w_k \leq w_{max}, \\ w_{max}, & w_k > w_{max} \end{cases} \tag{4}$$

Using w_k^r, the cost can be evaluated quickly using Eq (2).

When formulating the cost matrix in Section III-B, we observe that the algorithm evaluates mapping many different weights to the same parallel resistive devices. For each weight, the three steps are repeated and the time complexity is $O(R)$ based on the first step. The process is illustrated at the top of Figure 5. However, we observe that the first two steps are independent of the specific weight. Consequently, there exists an opportunity to pre-characterize the weight range using a one-time expensive initialization phase. Next, each realized weight can be computed fast and efficiently based on the weight range characterization, as shown at bottom of Figure 5.

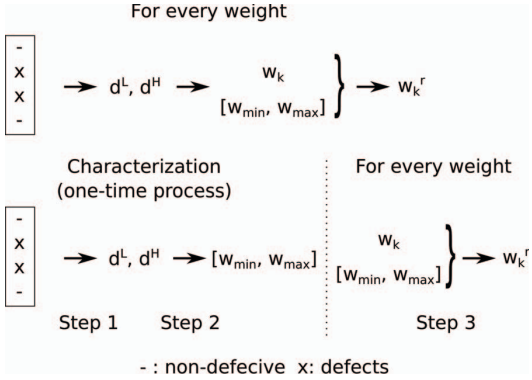


Fig. 5. An example of the weight range characterization technique of using $R = 4$ devices for a single weight.

The time complexity analysis of assigning N weights to R parallel resistive devices on the RCA is shown in Table I. Without the weight range characterization, the time complexity of the N assignments is $O(NR)$. When the one-time characterization is utilized, the time complexity is reduced to $O(N+R)$. Clearly, the technique is particularly effective when a higher redundancy factors are used to compensate for many defects in the hardware.

TABLE I
TIME COMPLEXITY ANALYSIS OF ASSIGNING N WEIGHTS TO R PARALLEL RESISTIVE DEVICES IN AN RCA.

Weight range mechanism	Time complexity			
	Step 1	Step 2	Step 3	Total
Without characterization	$O(NR)$	$O(N)$	$O(N)$	$\boldsymbol{O(NR)}$
With characterization	$O(R)$	$O(1)$	$O(N)$	$\boldsymbol{O(N+R)}$

C. The LP formulation technique

The assignment problem can be solved using the Hungarian algorithm or an LP formulation. Previous studies have used the Hungarian algorithm. We empirically observe that the use of an LP formulation results in shorter run-time due to the structure of the problem. The LP is formulated, as follows:

$$\min \sum_{i \in M} \sum_{j \in N} c_{ij} x_{ij}, \tag{5}$$

$$\sum_{j \in N} x_{ij} = 1, \forall i \in M \tag{6}$$

$$\sum_{i \in M} x_{ij} = 1, \forall j \in N \tag{7}$$

where c_{ij} is the cost of assigning neuron i to location j, i.e., entry (i, j) in the cost matrix C. $x_{ij} = \{0, 1\}$ is a binary variable that denotes if neuron i is assigned to location j. The objective function in Eq (5) minimizes the total cost of the data layout organization. The constraints in Eq (6) and (7) ensure that each neuron is assigned to one and only one location.

V. METHODOLOGY

The flow of the proposed framework for fast data layout organization is shown in Figure 6. The input to the framework is the weight matrices of a neural network with L layers, a RCA for each weight matrix, and a redundancy factor R indicating the number of parallel resistive devices that are used to realize each weight. The output from the framework is the order of the neurons in each layer of the neural network. The neuron order is expected to reduce the cost in Eq (2) and therefore improve the classification accuracy in hardware.

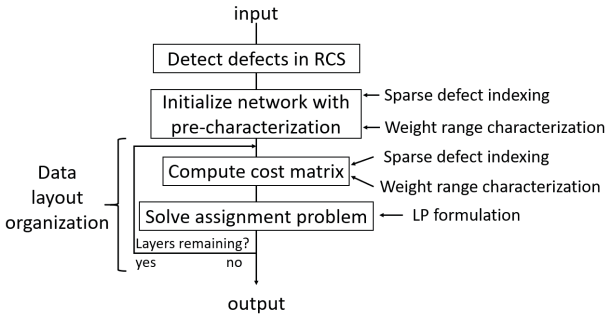


Fig. 6. Flow of the proposed framework.

First, the defective devices in each RCA is detected using the technique in [1], [7], [14]. In the initialization phase, the row-oriented and column-oriented data structures are constructed to facilitate the sparse indexing technique. Moreover,

the value range for each location for each set of R parallel devices is pre-characterized. Next, the neuron permutation is iteratively applied to each layer of the neural network from the first layer to the last layer. In each layer, the neuron permutation is performed by computing a cost matrix and solving an assignment problem, as explained in Section III-B. The cost matrix is quickly computed using the proposed sparse defect indexing and weight range characterization techniques in Section IV-A and IV-B, respectively. Next, the assignment problem is swiftly solved using the proposed LP formulation in Section IV-C.

VI. Experimental evaluation

The speed-up techniques in the proposed framework are implemented using C++ and the experiments are performed on a 3.4GHz×8 core machine with 32GiB of memory. The neural networks on MNIST [9] and CIFAR-10 [8] are trained using Keras [3] and TensorFlow on a NVIDIA Tesla K80 GPU.

TABLE II
Properties of the evaluated neural networks.

Network	Dataset	Software accuracy	Layers	Weights	Through -put
MLP-4	MNIST	98.35%	4	545000	1
MLP-6	MNIST	98.43%	6	774000	1
CNN-7	CIFAR-10	75.03%	7	1250144	1024
CNN-16a	CIFAR-10	93.45%	16	14977728	1024
CNN-16b	CIFAR-10	93.45%	16	54586368	32

We evaluate the performance of the framework using two MLPs trained on MNIST and three CNNs trained on CIFAR-10. The properties of the evaluated neural networks on both the MNIST and CIFAR-10 datasets are shown in Table II.

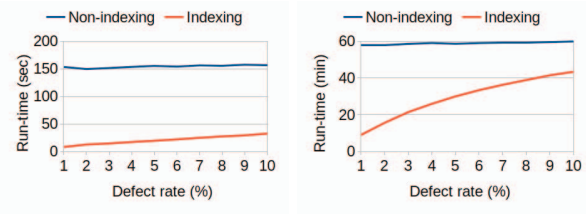
The framework is evaluated in terms of the normalized classification accuracy, is equal to the hardware accuracy divided by the software accuracy. The software classification accuracy is shown in the table. The classification accuracy in hardware is obtained by introducing 10% defects into the RCAs and evaluating the DNN classification accuracy using the realized weight matrices using Eq (1).

The baseline method with no optimization is labeled '-'. The technique of utilizing redundant hardware in [16] is denoted 'H'. The use of both redundant hardware and data layout organization in [18] is denoted 'HD'. The 'HD' method extended with the sparse defect indexing technique is called 'HD-I'. 'HD-IC' is the 'HD-I' method integrated with the weight range characterization technique. 'HD-ICL' is the 'HD-IC' technique extended with the proposed LP formulation.

In Section VI-A, we present the evaluation of the individual speed-up techniques. In Section VI-B, we compare our framework with the methods from previous studies.

A. Evaluation of the speed-up techniques

1) Evaluation of sparse defect indexing: The sparse defect indexing technique is evaluated in Figure 7. The figure shows the run-time of the framework with and without the sparse defect indexing technique with different defect rates on both the MNIST and CIFAR-10 datasets.



(a) MLP-6 (b) CNN-16a

Fig. 7. Evaluation of sparse defect indexing with respect to defect rate on (a) MNIST and (b) CIFAR-10 dataset.

The figure shows that the run-time without sparse defect indexing is proportional to the number of devices in the DNN. The sparse defect indexing technique significantly reduces the run-time, as the run-time is proportional to the number of defective devices. With the defect rate increasing from 1% to 10%, the normalized indexing run-time increases from 5.1% to 20.6% on the MNIST and from 15.5% to 72.4% on the CIFAR-10, respectively. The run-time is reduced by only computing the cost at the recorded locations instead of computing the cost at each location of the RCAs, as the cost introduced by non-defective devices is equal to zero.

2) Evaluation of the weight range characterization: The weight range characterization technique is evaluated in Figure 8. The figure shows that the run-time is proportional to R when the the weight range characterization technique is not used. In contrast, the run-time is almost constant when the weight range characterization is applied. The speed-up stems from that the time-complexity of computing w_k^r is reduced from $O(R)$ to $O(1)$.

(a) (b)

Fig. 8. Effectiveness of the weight range characterization technique with respect to different redundancy factor R on (a) MLP-6 and (b) CNN-16a.

3) Evaluation of the LP formulation: The linear programming formulation technique is evaluated and compared with the Hungarian algorithm, as shown in Figure 9. It can be observed that the run-time of the Hungarian algorithm greatly increase as the cost matrix dimension increase.

B. Comparison with related studies

In Table III, we present the comparison of the proposed framework with previous studies in terms of normalized accuracy, the cost in Eq (2), and run-time. The comparison is performed on both the MNIST and CIFAR-10 datasets using networks with 10% defect rate. The redundancy factor is set such that the 'HD' method achieves 99% normalized accuracy. Specifically, a redundancy factor R equal to 4, 4, 8, 4, 8

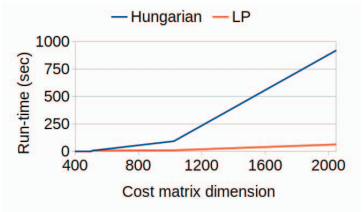


Fig. 9. Evaluation of the solver run-time vs. problem dimension.

is used for MLP-4, MLP-6, CNN-7, CNN-16a, CNN-16b, respectively.

When no optimization is applied, the normalized accuracy is unacceptably low. The technique of utilizing hardware redundancy (the 'H' method in [16]) greatly improves the accuracy. Compared with the 'H' method, the 'HD' method improves the average normalized classification accuracy with 1.8% without any hardware overhead. It is easy to understand that the normalized accuracy is improved because the cost in Eq (2) is improved with 52.6%. However, the run-time is too long to be practical for the larger neural networks. In particular, it can be observed that the run-time for CNN-16a and CNN-16b is 1.7 hours and 67.9 hours, respectively.

TABLE III
COMPARISON OF THE PROPOSED FRAMEWORK WITH RELATED STUDIES.

Network (dataset)	Work	Method	cost in Eq (2)	Norm. accuracy	Run-time (sec)
MLP-4 (MNIST)		-	39.0	22.0	3
	[16]	H	6.8	97.8	4
	[18]	HD	0.1	99.9	176
	Ours	HD-I	0.1	99.9	69
	Ours	HD-IC	0.1	99.9	18
	Ours	HD-ICL	0.1	99.9	18
MLP-6 (MNIST)		-	48.4	22.9	4
	[16]	H	7.3	98.4	5
	[18]	HD	0.1	100.0	259
	Ours	HD-I	0.1	100.0	100
	Ours	HD-IC	0.1	100.0	27
	Ours	HD-ICL	0.1	100.0	27
CNN-7 (CIFAR-10)		-	10050.3	13.3	2
	[16]	H	97.8	97.2	7
	[18]	HD	7.9	99.9	686
	Ours	HD-I	7.9	99.9	668
	Ours	HD-IC	7.9	99.9	71
	Ours	HD-ICL	7.9	99.9	71
CNN-16a (CIFAR-10)		-	2931020.0	10.7	25
	[16]	H	5821.1	96.9	64
	[18]	HD	1911.8	99.3	5969
	Ours	HD-I	1911.8	99.3	5051
	Ours	HD-IC	1911.8	99.3	943
	Ours	HD-ICL	1911.8	99.3	939
CNN-16b (CIFAR-10)		-	90413.3	7.5	166
	[16]	H	24.3	99.5	770
	[18]	HD	0.3	99.9	244572
	Ours	HD-I	0.3	99.9	95725
	Ours	HD-IC	0.3	99.9	28905
	Ours	HD-ICL	0.3	99.9	25248
Norm.		-	61011.4	(-84.5%) 0.15	0.01
	[16]	H	47.4	(-1.8%) 0.98	0.01
	[18]	HD	**1.00**	**1.00**	1.00
	Ours	HD-I	**1.00**	**1.00**	0.60
	Ours	HD-IC	**1.00**	**1.00**	0.12
	Ours	HD-ICL	**1.00**	**1.00**	**0.11**

Compared with the 'HD' method, the proposed 'HD-I', 'HD-IC', and 'HD-ICL' methods result in the exact same cost and normalized classification accuracy. This is expected because the speed-up techniques only avoid redundant computation when computing the cost metric or solves the assignment problem faster, i.e., the exact same data to hardware assignment (or data layout organization) is obtained. Compared with the 'HD' method, the 'HD-I', 'HD-IC', and 'HD-ICL' method respectively reduces the run-time with 40%, 88%, and 89%. The improvements in run-time are not surprising as the 'HD-I' method avoids a significant amount of redundant "zero" cost computation and the weight characterization reduces the time complexity of utilizing redundant hardware. The LP formulation slightly reduces the run-time for the large neural networks. In summary, the proposed framework reduces the average run-time of the state-of-the-art data layout organization without degrading the performance in resistive hardware.

VII. Conclusion and Future Work

In this paper, we have proposed a framework to speed-up the data layout organization for deploying DNNs on RCSs. The results show that the proposed framework is capable of achieving software level classification accuracy while reducing up to 89% of the run-time. Future study will explore specific customized techniques for large neural networks to balance the trade-off between the run-time and performance of the data layout organization.

References

[1] C. Y. Chen et al. RRAM defect modeling and failure analysis based on march test and a novel squeeze-search scheme. *IEEE Transactions on Computers*, 64(1):180–190, 2015.
[2] L. Chen et al. Accelerator-friendly neural-network training: Learning variations and defects in RRAM crossbar. DATE, pages 19–24, 2017.
[3] F. Chollet et al. Keras. https://keras.io, 2015.
[4] Z. He et al. Noise injection adaption: End-to-end reram crossbar non-ideal effect adaption for neural network mapping. DAC, page 57, 2019.
[5] M. Hu et al. Memristor crossbar-based neuromorphic computing system: A case study. *IEEE TNNLS*, pages 1864–1878, 2014.
[6] M. Hu et al. Dot-product engine for neuromorphic computing: Programming 1T1M crossbar to accelerate matrix-vector multiplication. DAC, pages 1–6, 2016.
[7] S. Kannan et al. Modeling, detection, and diagnosis of faults in multilevel memristor memories. *IEEE TCAD*, 34(5):822–834, 2015.
[8] A. Krizhevsky et al. Cifar-10 (canadian institute for advanced research).
[9] Y. LeCun et al. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, 2, 2010.
[10] B. Liu et al. Reduction and IR-drop compensations techniques for reliable neuromorphic computing systems. ICCAD, pages 63–70, 2014.
[11] B. Liu et al. Vortex: Variation-aware training for memristor X-bar. DAC, pages 1–6, 2015.
[12] C. Liu et al. Rescuing memristor-based neuromorphic design with high defects. DAC, pages 87:1–87:6, 2017.
[13] X. Liu et al. Harmonica: A framework of heterogeneous computing systems with memristor-based neuromorphic computing accelerators. *IEEE Tran. on Circuits and Systems I: Regular Papers*, 63(5), 2016.
[14] A. van de Goor and Y. Zorian. Effective march algorithms for testing single-order addressed memories. In *Proc. European Conference on Design Automation with the European Event in ASIC Design*, pages 499–505, 1993.
[15] L. Xia et al. Fault-tolerant training with on-line fault detection for RRAM-based neural computing systems. DAC'17, pages 1–6, 2017.
[16] L. Xia et al. Stuck-at fault tolerance in RRAM computing systems. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 8(1):102–115, 2018.
[17] B. Yan et al. A closed-loop design to enhance weight stability of memristor based neural network chips. ICCAD, pages 541–548, 2017.
[18] B. Zhang et al. Handling stuck-at-fault defects using matrix transformation for robust inference of dnns. *IEEE TCAD*, 2019.