EMAT: An Efficient Multi-Task Architecture for Transfer Learning using ReRAM

Fan Chen, Hai Li
Department of Electrical and Computer Engineering, Duke University, Durham, NC, USA
{fan.chen,hai.li}@duke.edu

ABSTRACT

Transfer learning has demonstrated a great success recently towards general supervised learning to mitigate expensive training efforts. However, existing neural network accelerators have been proven inefficient in executing transfer learning by failing to accommodate the layer-wise heterogeneity in computation and memory requirements. In this work, we propose EMAT—an efficient multi-task architecture for transfer learning built on resistive memory (ReRAM) technology. EMAT utilizes the energy-efficiency of ReRAM arrays for matrix-vector multiplication and realizes a hierarchical reconfigurable design with heterogeneous computation components to incorporate the data patterns in transfer learning. Compared to the GPU platform, EMAT can perform averagely 120× performance speedup and 87× energy saving. EMAT also obtains 2.5× speedup compared to the-state-of-the-art CMOS accelerator.

CCS CONCEPTS

• **Hardware → Hardware accelerators**;

KEYWORDS

ReRAM, transfer learning, accelerator

1 INTRODUCTION

Deep convolutional neural networks (CNNs) have become a pervasive approach in a broad range of modern application domains involving computer vision [15], natural language processing [8] and text processing [25]. The state-of-the-art accuracy of CNNs, however, comes at the cost of significant computational and memory resources. Normally, the computation and memory intensive CNN training is offloaded on powerful GPUs in the cloud. The developed models, on the other hand, are deployed at the edge devices (e.g., IoT or mobile) to perform the testing function.

In real applications, however, data is essentially dynamic. It would be desirable to have the intelligent edge devices capable of adaptively learning and tuning its parameters to achieve an ideal accuracy. For instance, in wearable applications that monitor the health of users, it would be desirable to adapt the CNN models locally rather than sending the self-made health data back to the cloud due to significant data movement overhead and privacy issues. For other applications such as robots, drones and autonomous vehicles, the statically trained models could not efficiently handle various environmental conditions. However, the large data transmission latency of sending huge environmental data to the cloud for incremental training is unacceptable. More importantly, many real-life scenarios essentially desire the real-time execution of multiple tasks and dynamic adaptation capability [13]. Nevertheless, it is extremely challenging to perform learning in end devices because of their stringent computing resources and tight power budget.

Recently, transfer learning [19, 20] is presented to reduce the expensive training efforts. It is a new general supervised training framework that re-utilizes a previously trained neural network in one *source domain* and adapts (or transfers) it to a different domain with distinct feature space and distribution (*target domain*). By having the convolutional layers (CONVs) fixed while training the fully-connected layers (FCs) only, transfer learning could dramatically speed up training in the target task (or domain) [3, 9, 24]. Moreover, recent studies [5, 7, 21, 22] have demonstrated the use of emerging ReRAM for implementing neural network accelerators, benefiting from its computation and storage capabilities. However, it is challenging to deploy transfer learning on existing hardware platforms. PRIME [7] and ISAAC [21] only support CNN testing. PipeLayer [22] and ReGAN [5] are proposed to support training of CNNs and generative adversarial networks, respectively. These two schemes, however, are both homogeneous designs, which didn't exploit the heterogeneity of CONVs and FCs. In this work, we present a ReRAM-based architecture for accelerating transfer learning that can adaptively accommodate the end devices' requirement. Our contributions in this work include:

- We analyze the general training procedure in transfer learning and accordingly present EMAT—an architecture that directly leverages ReRAM arrays to perform computation. In EMAT, the subsystems are specialized for the computation and storage characteristics of CONVs and FCs, respectively, making EMAT a scalable architecture.
- To optimize EMAT, we propose a novel time-multiplexed design for efficiently executing multi-tasks that share the trained CONVs.
- We evaluate EMAT and compared it against baseline CMOS architecture and GPU platform. Our experimental results show that EMAT can perform averagely 120× performance speedup and 87× energy saving against GPU platform. EMAT also obtains 2.5× speedup compared to its CMOS counterpart.

2 PRELIMINARY

2.1 Convolutional Neural Networks

Figure 1 shows an example of a typical CNN, which consists of three types of essential components—convolutional, fully-connected and pooling layers. Cascaded CONVs are usually considered as a generic

ICCAD '18, November 5–8, 2018, San Diego, CA, USA

ACM ISBN 978-1-4503-5950-4/18/11…$15.00
https://doi.org/10.1145/3240765.3240805

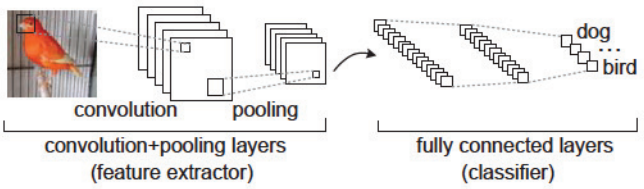

Figure 1: A typical CNN.

extractor of high- or mid-level image representation [19]. Pooling layers are used in an interleaved fashion with CONVs to reduce the feature map size and the computation. FCs are located at the final stage of a CNN as a classifier, which applies a linear transformation on the input feature vector.

In supervised learning, the weights and bias of a neural network are determined through the training on labeled dataset. The trained parameters are then deployed to run testing functions. In a testing phase, an input data is fed into the network and flows through layers consecutively by performing a forward function:

$$\boldsymbol{y} = \boldsymbol{W} \cdot \boldsymbol{x} + \boldsymbol{b} \text{ and } z = h(\boldsymbol{y}), \tag{1}$$

where the output neuron vector z is determined by the input neuron vector $\boldsymbol{x}$, the weight matrix of connections $\boldsymbol{W}$ and the bias vector $\boldsymbol{b}$. Usually, $h(\cdot)$ is a non-linear activation function.

In a training phase, data first flows along the same forward direction as in the testing phase, followed by a series of operations in the backward direction. A *cost function* $\mathcal{J}$ is defined to quantitatively evaluate the difference between the network's real output and its expected output. The error for layer l is defined as $\delta_l = \frac{\partial J}{\partial b_l}$, which can be back-propagated (BP) to the previous layer as [16]:

$$\delta_{l-1} = W_l^T \cdot \delta_l \cdot h'(\boldsymbol{y}_l), \tag{2}$$

The error and input activation are multiplied layer-wise to find the gradient of weights (∇W). The updated weight (W_{ji}) and bias (b_{ji}) are:

$$W_{ji,l} \leftarrow W_{ji,l} - \eta \cdot z_{i,l-1} \cdot \delta_{j,l}, \tag{3}$$

$$b_{ji,l} \leftarrow b_{ji,l} - \eta \cdot \delta_{j,l}, \tag{4}$$

where η is the learning rate and $\delta_{j,l}$ is the error back propagated from the node j in layer l. $z_{i,l-1}$ is the input of the node i in layer $l-1$.

2.2 Transfer Learning

In many applications, training a CNN requires significant computing resources and huge amount of weight updating iterations. Moreover, traditional supervised machine learning methods are based on an assumption that the training and testing data are drawn from the same feature space and have the same distribution. This task-specific training method, however, implies a poor scalability of CNNs as a model needs to be trained from scratch on the newly collected data if the feature space changes. In many real world applications, however, it is expensive or even impossible to collect the sufficient new training data and rebuild the models. To tackle the above challenges, transfer the knowledge learned from a source domain to a new task domain emerges as a more promising approach.

The transferability of CNN models is based on the key observation that CONVs located in the early stage often act as a feature extractor by performing common functions, such as recognition,

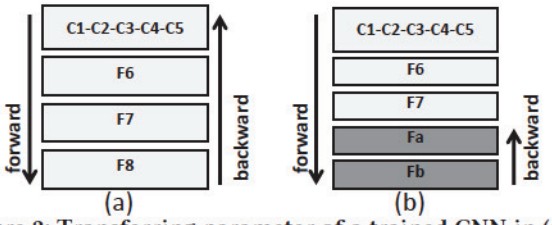

Figure 2: Transferring parameter of a trained CNN in (a) to (b) for different target task.

localization and detection [15, 20]. Normally, it takes a two-stage process for transferring parameters of a CNN. First, a network is trained on the source task with a large amount of available labeled data as illustrated in Figure 2(a). The pre-trained parameters of internal layers including CONVs and the first a few FCs (e.g., C1~C5, F6, F7 in the given example) can be directly transferred to the *target* tasks. Specifically, one or several adaptation layers (e.g., Fa, Fb) are added to form a new network as shown in Figure 2(b). This network only needs to be trained with a small amount of labeled data specified for the *target* task. When training the new network, the transfered layers are fixed and execute the feed-forward function; while the newly added FCs will be trained through back propagation. As can be seen that transfer learning can dramatically simplify the training procedure by limiting the complex BP operations and reducing the training samples.

2.3 ReRAM Basics

Figure 3(a) depicts the three-layer structure of a ReRAM cell: an oxide layer sandwiched between a top electrode and a bottom electrode layer. A ReRAM cell can be switched between a low resistance state (LRS, logic "1") and a high resistance state (HRS, logic "0") by applying an external write voltage with appropriate pulse width and magnitude. Figure 3(b) shows a simple ReRAM crossbar array structure, which has the smallest cell size of $4F^2$, where F denotes the technology feature size. Recent studies have demonstrated that ReRAM-based architecture performs in-situ matrix-vector multiplication [14] and is effective for accelerating neural network computation [5, 7, 22]. As shown in the figure, an input vector can be represented by the input signals to the wordlines (WL) of the array. An element of the matrix is programmed as the conductance of the corresponding cell in the crossbar array. The current flowing to the end of each bitline (BL), therefore, can be viewed as the result of the matrix-vector multiplication. For a large matrix that can not fit in a single array, the inputs and outputs shall be partitioned into multiple arrays. The output of each array is a partial sum, which is

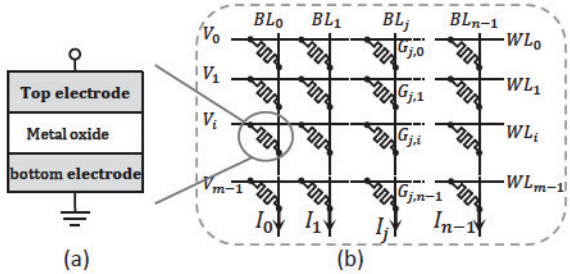

Figure 3: The ReRAM basics. (a) The cell structure. (b) Mapping matrix-vector multiplication to ReRAM crossbar array.

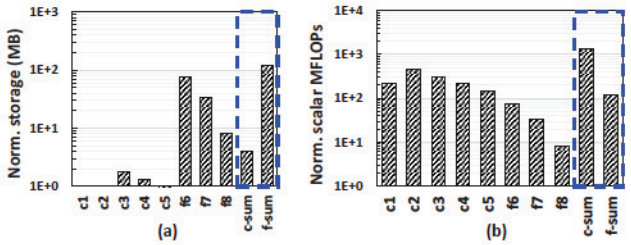

Figure 4: Evaluation of CONV/FC layers. (a) Storage requirement. (b) MFLOPs.

collected horizontally and summed vertically to generate the actual results.

3 CNN WORKLOAD ANALYSIS

We note that heterogeneous computing phases are involved in transfer learning: (i) the pre-trained CONVs perform the forward inference function for the feature extraction; and (ii) the trainable FCs deploy both the forward and backward functions. To explore the computational characteristics and challenges in transfer learning, we first quantify the computation characteristic of CONVs and FCs by using AlexNet [15] as a representative example. AlexNet is a famous CNN which comprises of 5 CONVs (C1-C5), 3 pooling layers and 3 FCs. We ran AlexNet on ImageNet and summarizes each layer's memory space and computation requirement (in million floating point operations per second, or MFLOPs) in Figure 4. Note that the $y-$axis in the figure is in logarithmic scale. And the rightmost two bars respectively represent the total demands of CONVs and FCs.

Clearly, the heterogeneity in layer topology naturally translates into different computing and memory requirements: FCs consume a significant fraction (30× more than CONV layers) in data storage (or bandwidth) resources, but contribute only a small portion of the total computations. CONVs, in contrast, account for the majority of the MFLOPs (92%). Our proposed EMAT architecture takes advantages of the above heterogeneity in layers topologies, the details of which shall be elaborated in the following section.

4 EMAT ARCHITECTURE

We propose EMAT—an efficient multi-task architecture for transfer learning using ReRAM, the design details of which is described in this section. We will start with how to map neural networks to CNN crossbar arrays to better support transfer learning. Furthermore, to exploit the underlying heterogeneity in computation and memory requirements of CONVs and FCs, two types of processing cells, namely CONV-CELL and FC-CELL, are proposed. At the end, we present the overall EMAT architecture and the optimization for multi-task training.

4.1 ReRAM Design for Transfer Learning

Feed forward function. Figure 5 illustrates the feed forward computation execution of CONVs and FCs on ReRAM arrays. In this case, the CONV and FC both have 9 input neurons and 4 output neurons. Each neuron in a FC is connected to all the neurons in the other layer. Thus, the total number of parameters in this simple example is $9 \times 4 = 36$. The neurons in a CONV, however, are only connected to a small region of the adjacent layer. The weights are shared among input neurons, while the input neurons are reused

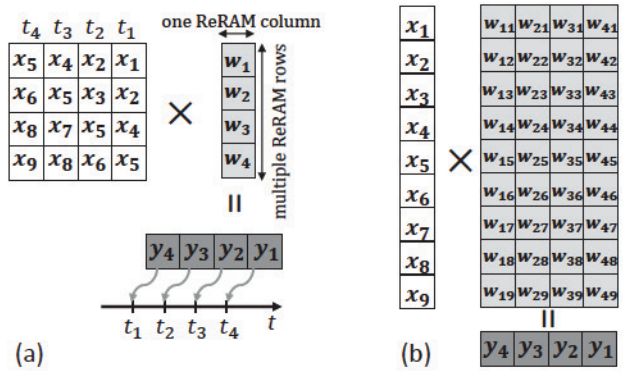

Figure 5: The feed-forward execution on ReRAM arrays: (a) CONV function; (b) FC function.

as denoted in the figure. Hence, the number of weights decreases to only 4. For larger networks, the parameter sharing scheme significantly reduces the number of parameters from $O(n^2)$ to $O(1)$.

In practice, the weight elements of the connection matrix are first programmed as conductance values of the corresponding cells in the crossbar array (the light gray part in the figure) under a certain programming voltage. Input neuron vector X is prepared according to the computation and then supplied as the input voltage sequences to the wordlines of ReRAM array. The output current at the end of each bitline are collected as the value of the output neuron (the dark gray part in the figure) without including the non-linear activation function if any.

Back propagation function. As a learning algorithm, the back propagation (BP) function is essential but the hardware support is more difficult. Equations (2)-(4) show that there are three steps in a BP process, including deviation calculation, error back-propagation and parameter updating. We use the host CPU to calculate the deviation of $\delta_l = \frac{\partial J}{\partial b_l}$ for high data accuracy. The error back-propagation step is composed of a matrix-vector multiplication ${W_l}^T \cdot \delta_l$ and a vector-vector multiplication $\delta_l \cdot h'(y_l)$. The former can be implemented on ReRAM arrays by following the same mapping method in the feed-forward function, while for the vector-vector multiplication, we reserve a special pre-processing function unit in the ReRAM buffer subarray (see Section 4.2). The parameter update step includes a matrix-vector multiplication to calculate $z_{i,l-1} \cdot \delta_{j,l}$, a scalar multiplication to apply the learning rate η and a subtraction to compute the updated weights and bias. Similarly, the matrix-vector multiplications can be realized through ReRAM arrays, while the other two require special function units to support.

Improving parallelism in execution. A naïve implementation by strictly following the structure in Figure 5 results in unbalanced processing latency in CONVs and FCs: the CONV takes 4 computing cycles while FC only need 1 cycle to complete the execution, though they have the same number of input/output neurons. A possible solution for the scenario is to duplicate a few ReRAM arrays so that the long operation can be partitioned and executed in parallel. For instance, making four copies of the ReRAM for CONV in Figure 5(a) and assigning each input column to one array can complete its operations in just one cycle. Essentially, the parallelism can be enhanced at the cost of hardware resource. We can further utilize the trade-offs between system performance and hardware

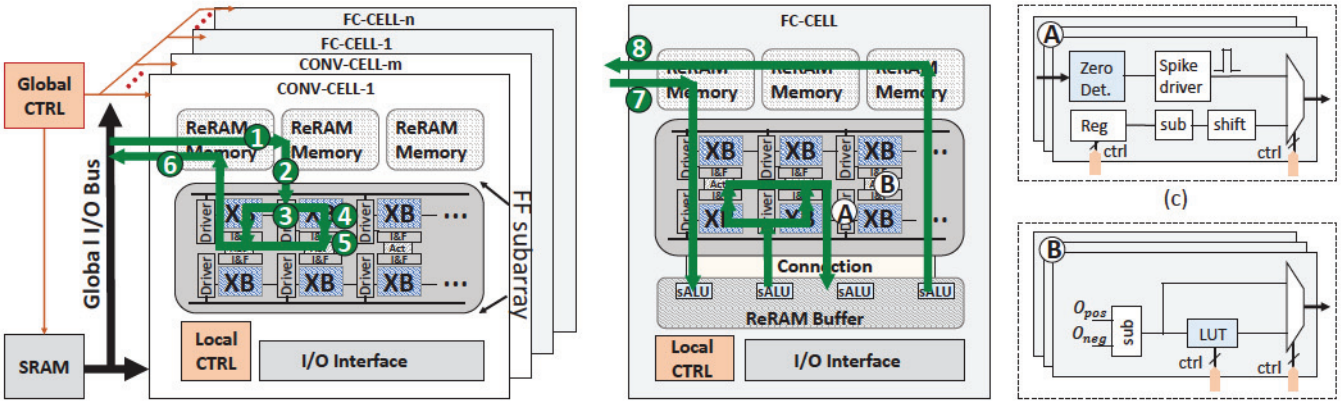

Figure 6: (a) The EMAT architecture overview and CONV-CELL design; (b) FC-CELL design; (c) wordline driver module; and (d) activation function module.

requirement to support multiple CNN applications in EMAT, which shall be explained in Section 4.4.

4.2 Heterogeneous Computing Cells

We propose two types of computing components, CONV-CELL and FC-CELL, for EMAT architecture. Each type is specifically designed to optimize the inference in CONVs and the training in FCs required in transfer learning procedure (see Section 2.2). As illustrated in Figure 6(a & b), both CONV-CELL and FC-CELL have two key components: ReRAM memory sub-arrays and full function sub-arrays (FF). The memory subarrays are the same as conventional memory subarrays for data storage. An FF subarray can be configured in either computation or storage modes: in computation mode, it executes matrix-vector multiplications; while in memory mode, it is used as memory subarrays to save data. Detailed array designs are similar as [7].

We aim to support both computation and storage in FF subarrays with minimum overheads in area and energy consumption. To achieve the purpose, we replace the power-hungry analog-digital/digital-analog converters (ADC/DAC) [7, 21] with the spike-based scheme such as [22]. The spike-based design requires two new circuit components—*spike driver* and *integrate-and-fire circuit* (IFC). The spike driver is used to convert an input signal to a sequence of weighed spikes, while the IFC integrates input currents and generates output spikes. Here, we omit the working mechanism and design details of the spike-based system, which can be found in many prior studies.

Compared to the CONV-CELL, the design of FC-CELL shall satisfy two unique requirements: the memory-intensive operations as discussed in Section 3 and the support of training with more complex operations explained in Section 2.2. Thus, in our design, a portion of ReRAM arrays are dedicated as buffer subarrays. We particularly place the buffer subarrays close to the FF subarrays and connect them with high-bandwidth interconnect. As such, the intermediate data produced by FF subarrays can be transferred to and saved in buffer subarrays and input to FF subarrays in later cycles. Moreover, we add *special function unit* (sALU) into the buffer subarray to support the training of FC layers. sALU provides three types of functions for data pre- and post-processing: (i) vector-vector multiplication used in error back-propagation; (ii) a scalar multiplication; and (iii) an abstractor for weights updates.

In addition to the memory subarrays, FF subarrays, buffer subarrays and sALU as the major components in CONV-CELL and FC-CELL designs, *wordline driver* and *activation function* are also important elements.

Figure 6(c) depicts the scheme of *wordline driver*. In the design, we include a *zero detect* module in front of the *spike driver*. This is based on the observation that a large portion of input activations in a conventional layer are zeros [12]. So zero detect module can help reduce the amount of computations. *Spike driver* is used to generate weighted input spikes. It is worth noting that the spike driver also serves as the write driver to tune the weights programmed in the ReRAM array. We deploy the *sub* and *shift* circuits to realize the subtraction and division operations in batch normalization.

The design of *activation function* is shown in Figure 6(d). The computation results from the positive and negative subarrays are first merged at the subtractor (sub), and then sent to the configurable look up table (LUT) to realize the activation function. The LUT can be bypassed in certain scenarios, e.g., when a large matrix is mapped to multiple crossbar arrays.

4.3 EMAT Overview and Dataflow

Figure 6(a) illustrates the hierarchical organization of the proposed EMAT architecture. It is a scalable extension of CNN-CELLs and FC-CELLs. The communication between different computing cells is realized by a *Global I/O Bus* that connects to an SRAM. In this case, data transfer between different computing cells go through the SRAM. A global controller orchestrates the data transfers between computing cells in training and testing based on the computing configurations (e.g., batch size). The execution of a CNN application on the EMAT architecture can be realized in a pipelined fashion, thanks to the data independence in CNN layer-wise computation [22].

❶ Once the input are latched from SRAM into the ReRAM subarry in CNN-CELL, the FF subarrys start the computation.

❷ When the first input vector is sent to the ReRAM crossbar, the wordline driver can continue processing the next input vector.

❸ The input are multiplied element-wisely with the two crossbar with positive and negative weights, respectively.

❹ Integrate-and-Fire circuit collects a current on the bitline and produces output pulse representing results.

❺ The activation function units perform non-linear functions on results.

❻ When the CNN-CELL completes the computation, the extracted features are output to and stored in SRAM.

❼ The FC-CELL fetches input from SRAM and follows a similar intra-cell computing flow.

❽ The results are write back to the SRAM when the processing of one image is finished or when the buffer subarray is full.

EMAT is a spatially scalable architecture. It specializes the forward CONV and backward FC layers across distinct processing cells to leverage the key compute and data access patterns in transfer learning. Additionally, the reconfigurability of the architecture provides significant flexibility for mapping a given network topology. For instance, using more cells for mapping a larger CNN or duplicating cells for a single CNN to improve parallelism.

4.4 Multi-Task Optimization

Many end devices, e.g., mobile devices, need support many real-time applications concurrently [13]. It requires not only the dynamic adaptation capability but also multi-task support. The flexibility offered by the EMAT architecture enables the operation of multiple CNN tasks by sharing the same feature extractor in a time-multiplex fashion.

Figure 7(a) illustrates the typical pipelined execution for a single task, in which a new input can enter the pipeline every cycle within a batch. Note that in this work, we train all the FCs in target domain instead of only the adaptation layers for high accuracy. After a pipeline filling period, one output is produced per cycle, providing a very high execution efficiency. Given the computation-intensive nature of CNNs as discussed in Section 3, switching from one task to another will have to take a long wait time if following the single-application working flow. To address this problem, we aggressively exploit the hardware-performance trade-off discussed in Section 3. Our key idea is to re-utilize the same feature extractor for multiple tasks. A more powerful CNN-CELL by duplicating copies of ReRAM arrays with the same weight configuration can be realized to process feature extraction in a shorter time. Figure 7(b) presents the case when we duplicate the naïve design for four times. The processing time of a given layer would finish 4× faster than the orginal setup. Multiple tasks occupy the shared CNN-CELL in a time-multiplexed fashion. Each FC-CELL fetches the extracted feature buffered in the SRAM through *Global I/O bus*.

To sum up, multiple FC-CELLs dedicated to different tasks work in parallel. Data transfer occurs serially through the shared bus across multiple FC-CELLs.

5 EXPERIMENT METHODOLOGY

Workload. All the experiments in this work are run on classification tasks. We take ImageNet with 14.2M images in 21.8K indexed synsets as the *source* domain and Pascal VOC 2007 [11] datasets as the *target* domain in transfer learning. Pascal VOC 2007 is selected as it has significant differences in image statistics compared to ImageNet. To test the efficiency on multi-task scenario, we include MNIST [17] and Animals on the Web [4] datasets together as the multiple *target* domains.

Network models. AlexNet and VGG-16 have been widely used in transfer learning studies. Thus we choose these two popular large-scale CNN networks and adopted the same typologies and hyper-parameters as the references [15, 23].

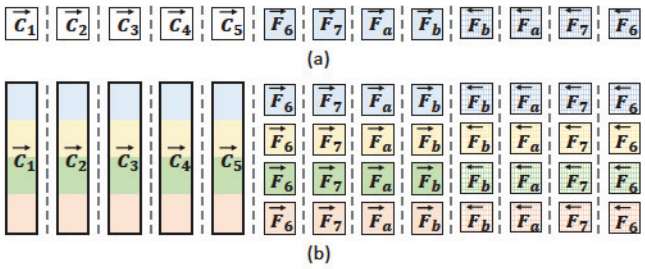

Figure 7: (a) Pipeline based design. (b) Multiplexed design for multi-task support.

Evaluation platform. We pre-train AlexNet and VGG-16 on ImageNet on GPU by Tensorflow [1]. The parameters of our GPU platform are shown in Table 1. The training procedure periodically evaluates the cross-entropy objective function on a subset of the training set and on a validation set. The learning rates are divided by 10 until the training cross-entropy is stabilized. We then stop training after 2 iterations.

We built a ReRAM simulator based on NVSim [10] for EMAT evaluation. The read/write latency is set as 29.31ns/50.88ns per spike, and the read/write energy is 1.08pJ/3.91nJ per spike [22]. The default data resolution on EMAT is 16-bit. Without loss of generality, we use a resistance range of $20K\Omega \sim 200K\Omega$ for ReRAM cells. The resistance ranges provides 16 levels (or 4-bit resolution) for weight-discretization. To obtain the 16-bit data operation, we group every four arrays and add shifted 4-bit outputs. The ReRAM crossbar array size is 64, that is, 64 rows and 64 columns per array as that in [2]. The input SRAM memory was modeled using CACTI [18]. We compare the transfer training to *target* tasks on the EMAT architecture against (1) the GPU system and (2) a CMOS counterpart based on Dadiannao [6].

6 EVALUATION

Performance. Figure 8 compares the execution speedup obtained in transfer learning when considering only a single *target* task. The runtime is measured and normalized to the one when a pre-trained AlexNet learns the MNIST dataset on CPU (Intel E5-2630 v3 8-core). The results show that the speedup of EMAT increases 90× at least, 150× at most and 120× on average compared to the GPU platform. The geometric mean speedup to the CMOS counterpart is 2.5×.

The results show that, EMAT achieves higher speedup when the target task is complex, for example, VGG-16 on VOC 2007 classification. It is simply because the long training time benefits more from the pipelined design as the computations and data movements are performed in a highly parallel manner.

Table 1: Configuration of the GPU platform.

Memory	128 GB
Storage	1 TB
Graphic Card	NVIDIA Geforce GTX 1080
Architecture	Pascal
CUDA Cores	2560
Base Clock	1607 MHz
Compute Capability	6.1
Graphic Memory	8 GB GDDR5X
Memory Bandwidth	320 GB/s
CUDA Version	8

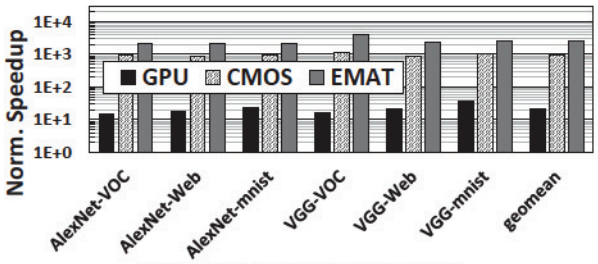

Figure 8: Normalized speedup.

Energy Consumption. Figure 9 shows the energy consumption of all the applications of interest. Again, all the numbers are normalized to the CPU energy consumption when learning MNIST on AlexNet. For the similar reason, the energy saving of EMAT increases with the complexity of target tasks. Thus, the VGG-16 on VOC 2007 provides the most energy saving as 115×. AlexNet on VOC 2007 offers the second most savings as 110×. Compared with GPU platforms, EMAT provides averagely 87× energy reduction. The results show that, the energy efficiency of EMAT increases by 1.9× at least, 30× at most and 8× on average compared with CMOS platform. This is mainly due to the large size SRAM input memory. For small- or middle-scale networks that can fit within a small number or even single computing cells, the energy of EMAT would decrease significantly due to the decreased access to the global SRAM memory .

Impact of ReRAM array sizes. We also explore the design space of different ReRAM array sizes and show the results in Figure 10. An intuitive thought is that as array size increases, the overhead on peripheral circuitry reduces and thereby the overall energy consumption decreases. Interestingly, it is not always the case: an increase in energy consumption is observed when the ReRAM array size enlarges from 64×64 to 128×128. This is because of the decrease in array utilization ratio due to the sparse connectivity in CNNs.

Multi-task operation. For EMAT to support concurrent CNN classifications, we tested EMAT with four concurrent applications running complex CNNs, e.g., ImageNet, VOC 2007, Animal on the Web and MNIST. Results show that with a 4 MB SRAM and 30 FPS, EMAT would be able to perform 4-task simultaneously. With large SRAM capacity, EMAT can support more concurrent applications as large memory stores more intermediate feature and reduces spatial conflicts.

7 CONCLUSIONS

In this paper, we present a ReRAM-based accelerator, EMAT, to accelerate transfer learning for low power end devices. Two computing components are specialized for the computational and storage characteristics of CONV and FC layers. We also introduce a novel time-multiplexed training flow for executing multi-tasks. The experimental results show that EMAT can achieve a high speedup and significant energy saving compared to the GPU platform.

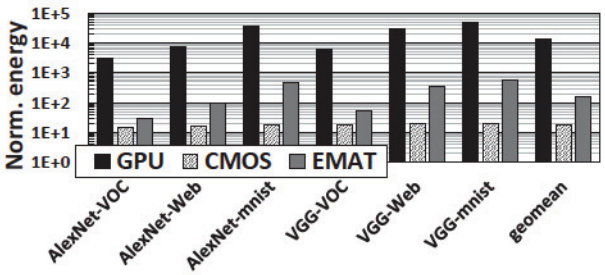

Figure 9: Normalized energy saving.

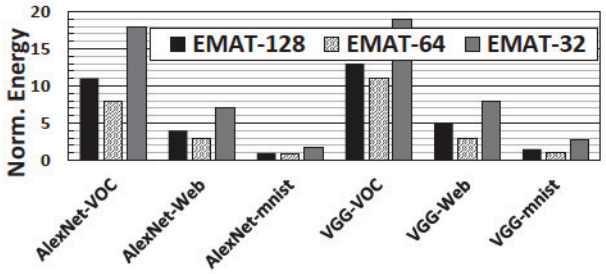

Figure 10: The EMAT energy consumption comparison when varying ReRAM crossbar array size.

ACKNOWLEDGMENT

This work was supported in part by NSF 1725456, DOE DE-SC0018064 and AFRL FA8750-18-2-0057. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of grant agencies or their contractors.

REFERENCES

[1] M. Abadi *et. al*, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems," *ArXiv e-prints*, 2016.
[2] A. Ankit *et. al*, "RESPARC: A Reconfigurable and Energy-Efficient Architecture with Memristive Crossbars for Deep Spiking Neural Networks," in *DAC*, 2017.
[3] Y. Aytar and A. Zisserman, "Tabula rasa: Model transfer for object category detection," in *ICCV*, 2011.
[4] T. L. Berg and D. A. Forsyth, "Animals on the Web," in *CVPR*, 2006.
[5] F. Chen, *et al.*, "ReGAN: A pipelined ReRAM-based accelerator for generative adversarial networks," in *ASP-DAC*, 2018.
[6] Y. Chen *el. al*, "DaDianNao: A Machine-Learning Supercomputer," in *MICRO*, 2014.
[7] P. Chi *et. al*, "PRIME: A Novel Processing-in-memory Architecture for Neural Network Computation in ReRAM-based Main Memory," in *ISCA*, 2016.
[8] R. Collobert *et. al*, "A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning," in *ICML*, 2008.
[9] R. Collobert *et. al*, "Natural Language Processing (almost) from Scratch," *arXiv*, 2011.
[10] X. Dong, *et al.*, "NVSim: A Circuit-Level Performance, Energy, and Area Model for Emerging Nonvolatile Memory," *IEEE TCAD*, 2012.
[11] M. Everingham *et. al*, "The Pascal Visual Object Classes Challenge: A Retrospective," *International Journal of Computer Vision*, 2015.
[12] S. Han *et. al*, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding," *ICLR*, 2016.
[13] S. Han *et. al*, "MCDNN: An Approximation-Based Execution Framework for Deep Stream Processing Under Resource Constraints," in *MobiSys*, 2016.
[14] M. Hu, *et al.*, "Memristor Crossbar-Based Neuromorphic Computing System: A Case Study," *IEEE Transactions on Neural Networks and Learning Systems*, 2014.
[15] A. Krizhevsky, *et al.*, "ImageNet Classification with Deep Convolutional Neural Networks," in *NIPS*, 2012.
[16] Y. LeCun, "A Theoretical Framework for Back-Propagation," , 1988.
[17] Y. LeCun, "The MNIST database of handwritten digits," , 1998.
[18] N. Muralimanohar, *et al.*, "Optimizing NUCA Organizations and Wiring Alternatives for Large Caches with CACTI 6.0," in *MICRO*, 2007.
[19] M. Oquab, *et al.*, "Learning and Transferring Mid-level Image Representations Using Convolutional Neural Networks," in *ICCV*, 2014.
[20] S. J. Pan and Q. Yang, "A Survey on Transfer Learning," *IEEE Transactions on Knowledge and Data Engineering*, 2010.
[21] A. Shafiee, *et al.*, "ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars," in *ISCA*, 2016.
[22] L. Song, *et al.*, "PipeLayer: A Pipelined ReRAM-Based Accelerator for Deep Learning," in *HPCA*, 2017.
[23] S. *et. al*, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *ArXiv e-prints*, 2015.
[24] T. Tommasi, *et al.*, "Safety in numbers: Learning categories from few examples with multi model knowledge transfer," in *ICCV*, 2010.
[25] X. Zhang *et. al*, "Text Understanding from Scratch," *ArXiv e-prints*, February 2015.