IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 26, 2020, accepted November 23, 2020, date of publication December 15, 2020,
date of current version December 31, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3045071

Resistive Crossbar-Aware Neural Network Design and Optimization

MUHAMMAD ABDULLAH HANIF[1,*], (Graduate Student Member, IEEE), ADITYA MANGLIK[1,2,*], (Student Member, IEEE), AND MUHAMMAD SHAFIQUE[3], (Senior Member, IEEE)

[1]Faculty of Informatics, Technische Universität Wien (TU Wien), 1040 Vienna, Austria
[2]Birla Institute of Technology & Science, Pilani, Pilani 333031, India
[3]Division of Engineering, New York University Abu Dhabi (NYUAD), Abu Dhabi, United Arab Emirates

Corresponding author: Muhammad Abdullah Hanif (muhammad.hanif@tuwien.ac.at)

*Muhammad Abdullah Hanif and Aditya Manglik contributed equally to this work.

ABSTRACT Recent research in Non-Volatile Memory (NVM) and Processing-in-Memory (PIM) technologies has proposed low energy PIM-based system designs for high-performance neural network inference. Simultaneously, there is a tremendous thrust in neural network architecture research, primarily targeted towards task-specific accuracy improvements. Despite the enormous potential of a PIM-based compute paradigm, most hardware proposals adopt a *one-accelerator-fits-all-networks* approach, bleeding performance across all verticals. The overarching goal for this work is to improve the throughput and power efficiency of convolutional neural networks on resistive crossbar-based microarchitectures. To this end, we demonstrate why, how, and where to prune contemporary neural networks for superior exploitation of the crossbar's underlying parallelism model. Further, we present the first crossbar-aware neural network design principles for discovering novel crossbar-amenable network architectures. Our third contribution includes simple yet efficient hardware optimizations to boost energy & area efficiency for modern deep neural networks and ensembles. Finally, we combine these ideas towards our fourth contribution, *CrossNet*, a novel network architecture family which improves computational efficiency by 19.06× and power efficiency by 4.16× over state-of-the-art designs.

INDEX TERMS Processing-in-memory, memristor, crossbar, resistive RAM, ReRAM, neural network, pruning, ensemble, DNN, deep neural networks, CNN, convolutional neural networks, CrossNet, neuromorphic computing, in-memory computing, efficiency, performance, energy consumption, design, principles, optimization.

I. INTRODUCTION

Neural networks have achieved breakthroughs in a wide range of hard computation problems, from image recognition to speech translation [1]. State-of-the-art networks rely on massive parameter count ($\geq 10^8$) and manually designed architectures (with $10^1 - 10^3$ layers) to surpass prior benchmarks and beat human-level performance [2], [3]. However, these algorithms demand enormous memory ($\geq 10^9$ Byte), and operational cost ($\geq 10^9$ operations for each input), scaling commensurately with ever-growing datasets and network depth [4]–[7].

To quantify the hardware requirements of modern deep learning workloads, we analyzed the results of the prized *ILSVRC Image Classification and Segmentation* challenge from 2012 till date.[1] Contrary to popular perception, *neural network ensembles* (a cluster of networks) dominated the challenge, consistently outperforming singular model submissions. For example, in the 2014 winning submission, 7 models were combined as an ensemble, improving the error rate by 3.45% over a single model. Interestingly, the runner-up VGGNet (8.43% error, 103 million parameters [8]) performed better than the GoogleNet [9] model (10.07% error, 6.7 million parameters) in the single-model comparison. Further, for the same batch size and inference hardware, GoogleNet_v1 performed 3.29× faster than VGG-16 (128.16 *ms*). From these observations, we deduce:

The associate editor coordinating the review of this manuscript and approving it for publication was Javed Iqbal.

[1]https://image-net.org/challenges/LSVRC/

1) An effective **ensembling strategy** was the winning trick for the GoogleNet submission. Singular models fared poorly in almost all aspects of the challenge.
2) Computation costs can be improved by **aligning the network design space with the underlying hardware's parallelism model**. Even though the single VGGNet model offers better accuracy, the *performance-per-parameter* for GoogleNet is higher by *two orders of magnitude (13.55 accuracy-points/million-parameters against 0.65 accuracy-points/million-parameters)*.

The **key take-away** from these deductions is that it is possible to achieve both higher accuracies and lower computation costs by considering the target hardware's parallelism model along with the network architecture search space. Applying these lessons in practice, we analyze and present *the first convolutional neural network design optimizations in the context of resistive crossbar microarchitectures*.

Recently there has been a surge in the research community's interest towards specialized accelerator-based architectures [10]–[13] and innovative compute-paradigms (such as *Processing-in-Memory (PIM)*). Memristor-based crossbar microarchitectures have been demonstrated to be more efficient than GPUs by up-to three orders of magnitude [14], [15]. Large resistive crossbars[2] (for example, 1024 × 1024 or more) offer lower static power costs due to a high number of operations per cycle. However, these designs *suffer from practical challenges such as high ADC precision requirement, wire IR drop and large write currents* [16]. On the contrary, using small crossbars (for example, 64 × 64 or less) for computing large dot products poses the *spillover problem*. This is tackled by breaking up the input vectors across different crossbars and combining results of a single dot product via Shift-&-Add units. As an illustrative example, unrolling and mapping 64 5 × 5 × 3 filters on 64 × 64 crossbars. This requires two crossbar units to perform the MAC operation correctly per input vector. Figure 1 demonstrates how 82.81% space is wasted in the second unit (figure not to scale). Extending this observation across an *ever-increasing number of layers required by modern deep networks*, we observe tremendous wastage of expensive chip area. This may be attributed to the following observations:

1) Multiple crossbars must co-ordinate to calculate a single output. This is inefficient as each crossbar can produce independent results in the same cycle. Such synchronization barriers impede throughput and degrade effective parallelism. (comparing Figure 1 against Figure 2)
2) High-precision global Shift & Add units are needed to accumulate the result from multiple crossbars, which increases peripheral area & energy costs.
3) When we consider mapping modern network architectures with split branches (InceptionNet [9], TreeNet [17]) and skip connections (DenseNet [18], ResNet [2]), there is no work in the direction of exploiting the crossbar architecture's underlying parallelism towards eliminating duplicate memory accesses for these shared activations.

[2]detailed background in Section II.D

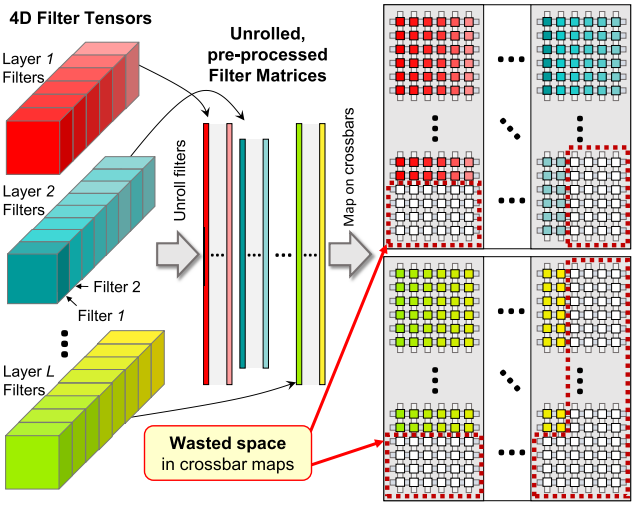


FIGURE 1. A small subset of weights of a set of filters occupying only a part of a crossbar (*spillover effect*). The remaining part of the crossbar cannot be used in parallel to ensure accurate MVM operation. Different colors show different layers and different shades correspond to different filters in the figure.

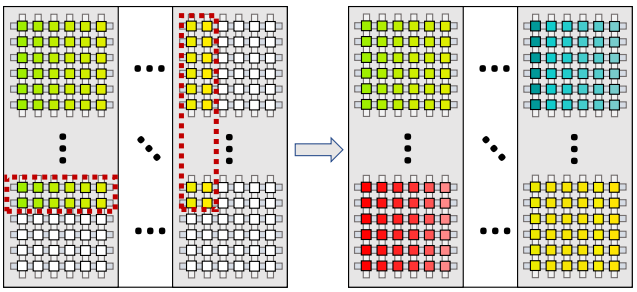


FIGURE 2. Algorithm 1 eliminates the excess columns, while Algorithm 2 eliminates spillover. This improves occupancy ratio from 0.44 to 1.0.

These problems motivate us towards aligning the underlying hardware's parallelism model with the network data flow. We believe this is the key to practical improvements in computational efficiency.

Our Novel Contributions: This paper serves to demonstrate the merits of a hardware-software co-design perspective over the one-size-fits-all approach by both hardware architects and network designers. To this end, we present the following novel contributions (see Figure 4):

1) We demonstrate why, how, and where to prune contemporary neural network architectures for better exploitation of the crossbar's underlying parallelism model using **novel pruning algorithms**. (Section III)
2) We present **innovative hardware optimizations** for crossbar peripherals to improve the energy and area efficiency for previously neglected *heterogeneous convolution filters and ensembles*. (Section IV)

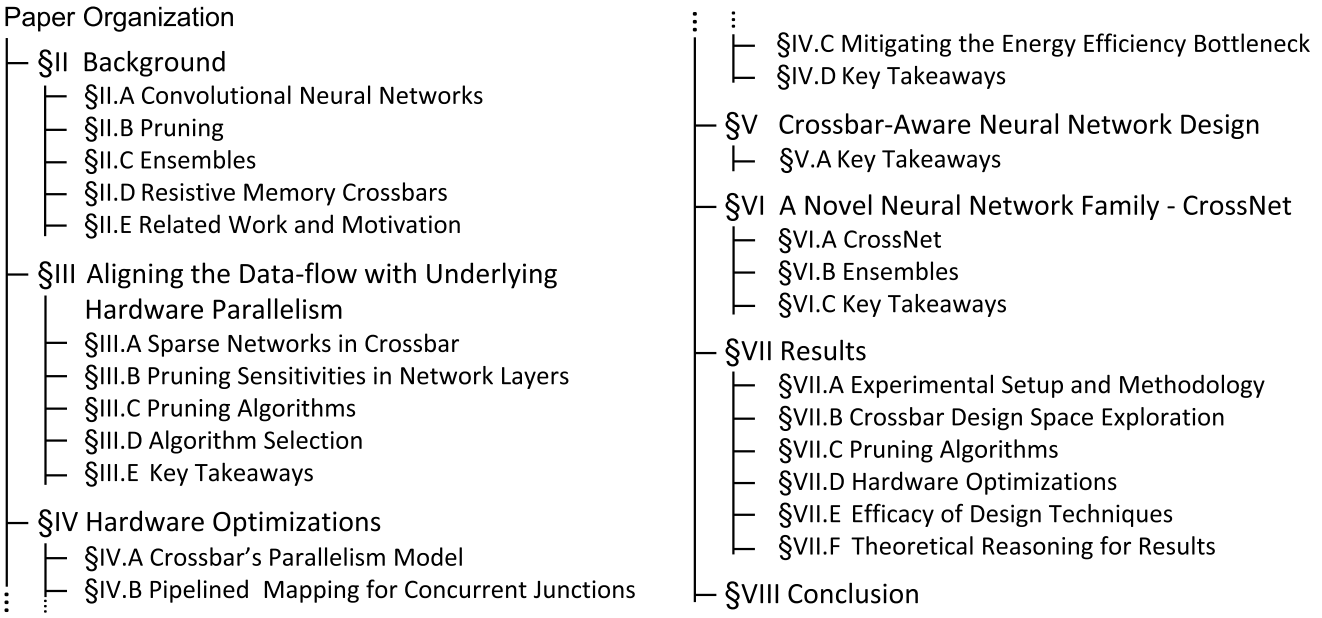


FIGURE 3. Paper organization.

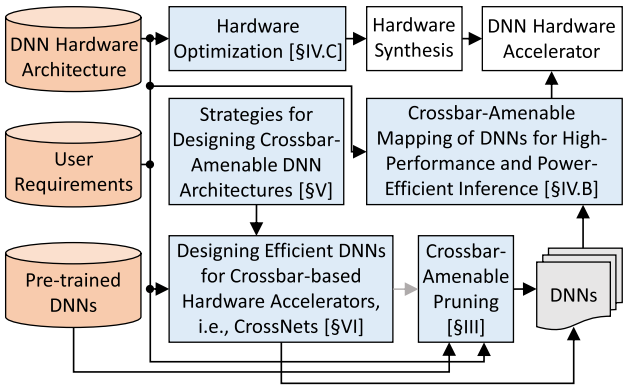


FIGURE 4. System overview where our contributions are highlighted in blue.

3) We present the first resistive **crossbar-aware design principles for network architecture search**. These principles are targeted towards practical performance improvements without specialized hardware support [19]. (Section V)
4) We utilize these principles to develop a *new family of neural networks*, called **CrossNet**, which improves on the best-in-class network's computational efficiency by 19.06× and power efficiency by 4.16×. (Section VI)
5) We develop case-studies for these ideas in the context of latest developments in the neural network architecture research and **quantify their efficacy across a diverse test-bench**. (Section VII)

To comprehensively evaluate the potential of each idea, the article is organized in independent sections with key takeaways listed at the end to ease the reader's task. Please refer to Figure 3 for the organization of the upcoming sections.

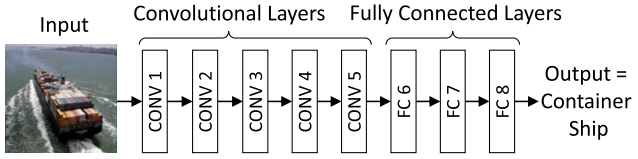


FIGURE 5. Illustration of a deep convolutional neural network.

Figure 4 demonstrates one potential flow for combining the presented ideas.

II. BACKGROUND

In this section, we present the necessary background required to understand the concepts presented in the technical sections of the paper. This section also introduces the key terms and variables. Section II-A introduces convolutional neural networks and highlights some of the key challenges faced in deploying high-accuracy DNNs in resource-constrained systems. Section II-B discusses DNN pruning and its two major categories. Section II-C presents the concept of DNN ensemble for high accuracy scenarios. Section II-D gives an overview of resistive memory crossbars and their use for DNN inference. Finally, Section II-E highlights the key related works and the motivation behind this work.

A. CONVOLUTIONAL NEURAL NETWORKS

A typical convolutional network is composed of several convolutional (CONV) layers and Fully Connected (FC) layers; see Figure 5. A convolutional layer (see Figure 6) can be represented by the equation:

$$A_k^{out}(x,y) = \sigma\Big(\sum_{j=0}^{N_i-1}\sum_{a=0}^{K_x-1}\sum_{b=0}^{K_y-1} A_j^{in}(x+a, y+b) * K_k(j,a,b)\Big) \quad (1)$$

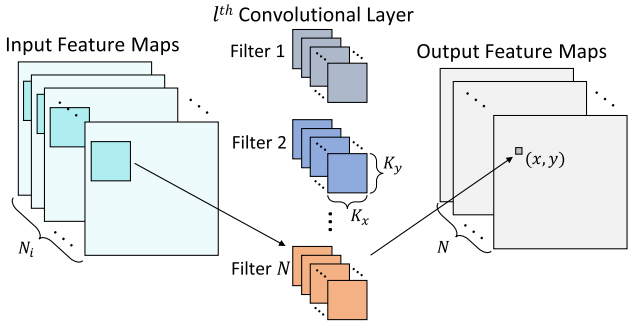


FIGURE 6. Illustration of a convolutional layer.

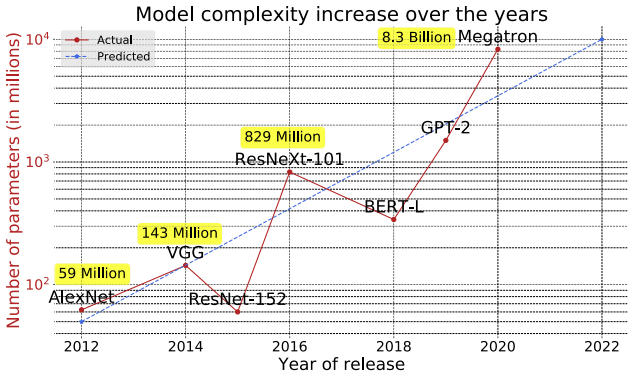


FIGURE 7. Models like Megatron [3] are challenging to train and deploy even in cloud clusters with theoretically *infinite scaling capacities*.

K_k denotes the k^{th} tensor of dimensions $K_x \times K_y \times N_i$, A_j^{in} denotes the j^{th} input feature map, and j, a, b represent the convolution operation indices. Equation 1 has a space and time complexity of $O(N^3)$. Comparing data and network sizes (1.5 MB for 224 × 224 color image and 240 MB for AlexNet weights) with modern cache capacities (12 MB L3, Intel Core i7 8700K), we observe that the computation is bottlenecked by memory accesses. High costs of data movement [20], high degree polynomial run-time and space complexity restricts deep learning to cloud-based services, entailing enormous communication costs amid growing data privacy concerns [21], [22]. Recent language models such as BERT-L [23], Megatron [3], and GPT-3 [24] (Figure 7) have saturated the limits of available GPU hardware, and yet promise significant growth in network size.

For decreasing the service latency, it is imperative to optimize or parallelize the computation load. Fortunately, the main operation in deployment use-case, network inference, is embarrassingly parallel within the context of a layer and for each input cycle [25]. DaDianNao [11] exploits this observation for context-switch based per-layer processing.

B. PRUNING

Sparsifying networks via pruning is a well-known technique used to decrease the computation load (see Figure 8), usually at the cost of accuracy [26]–[29]. Multiplying dense matrices (activations maps) with sparse tensors (pruned network), also known as sparse-MVM, needs significant effort in

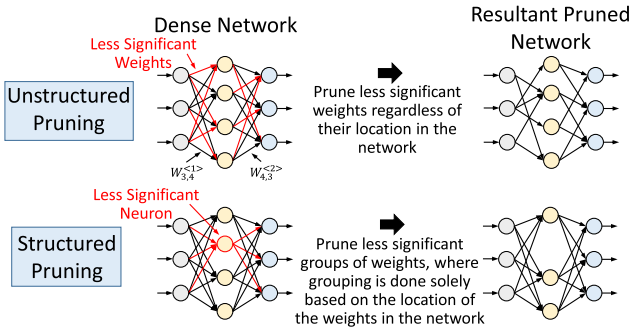


FIGURE 8. Structured vs. unstructured pruning.

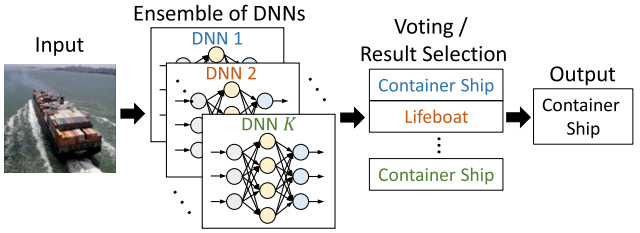


FIGURE 9. Neural network ensemble.

decoding the input sparse matrix representations [30] and encoding the outputs. This overhead nullifies any significant latency advantages obtained from sparse-MVM over GEMV. Consequently, **unstructured sparsity** [19], [31], [32] fails to deliver practical latency improvements without specialized hardware support [13]. For example, [33] observed that pruning 89% of AlexNet weights [19] counter-intuitively increased the CPU execution time by 25%. This observation motivates us towards reducing the model size without sacrificing data locality.

Structured sparsity clusters weight pruning by vectors, filters, and layers [34], [35]. Spatial locality is preserved at the cost of lower pruning flexibility & reduced compression ratios [36], [37]. It retains the dense matrix multiplication structure, which translates to practical reductions in processing time [33], [38].

C. ENSEMBLES

Neural network ensembles (see Figure 9) [39] breakdown the complex decision boundary into smaller sub-problems (i.e., *divide and conquer paradigm*). They were among the earliest innovations used to achieve breakthroughs in human-level performance for computer vision challenges [40], [41]. Primary advantages include smoothening the decision boundary, eliminating outliers, and advancing correct classification rates beyond individual capacity [17], [42], [43].

A Committee Machine averages the output of several members, whereas an ensemble (or a mixture-of-experts) utilizes a gating network that probabilistically weighs the outputs from several specialized networks towards the final result [44]. Figure 10 demonstrates ensembling method performance across varying datasets and network architectures. The work in [17] observed that random weight initialization provides

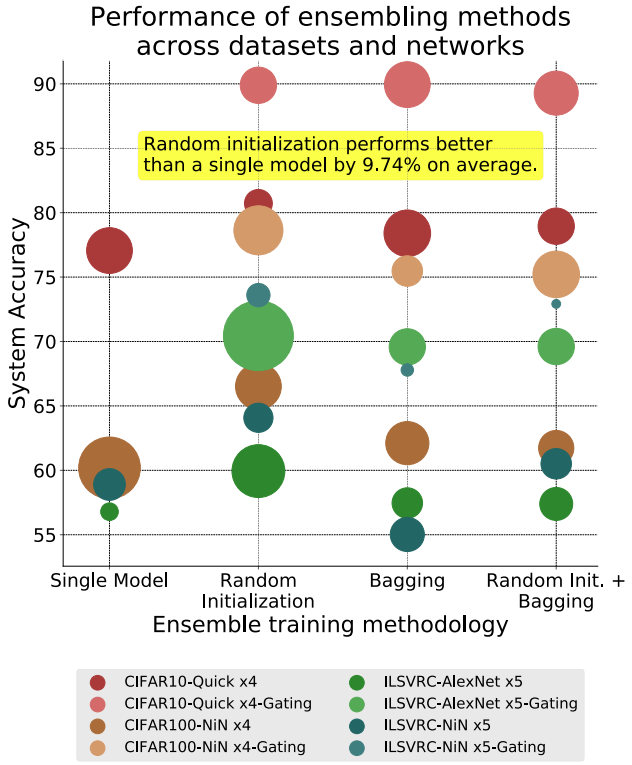


FIGURE 10. **Circle sizes represent deviation values in system accuracy measurement. ILSVRC-AlexNet-x5-Gating indicates 5 AlexNet architecture networks trained on ILSVRC dataset, and final result computed via a gating network.**

maximum performance improvements (4.58%) as compared to bagging (0.125%) and combination of bagging + random initialization (1.41%).

Ensembles are heavily favored in high-accuracy scenarios (for example, medical imaging analysis), where single networks cannot achieve the desired reliability, accuracy, and fault tolerance [45]–[49].

D. RESISTIVE MEMORY CROSSBARS

Considering the challenges in DRAM scaling [50], multiple memory technologies are being explored, including PCM, STT-RAM, ReRAM, etc. [51]–[53]. Among the Non-Volatile Memory candidates, *memristor-based Resistive RAM (ReRAM)* offers high endurance (up to 10^{10} cycles), fast switching and low read/write energy [54]–[56]. High-density crossbar microarchitectures with memristor substrate have demonstrated natural amenability for high-throughput GEMV multiplication [57], [58]. Figure 1 illustrates the process of unrolling convolution filters as they are mapped on crossbars. The activations are delivered via DAC on wordlines and summed currents obtained on the bitlines (see Figure 11). The final results are digitized via ADC and a network of Shift-&-Add units. Memristive devices can be built from a wide variety of substrates and device isolation/access mechanisms [59]–[63]. However, the strategies and algorithms presented in this paper are

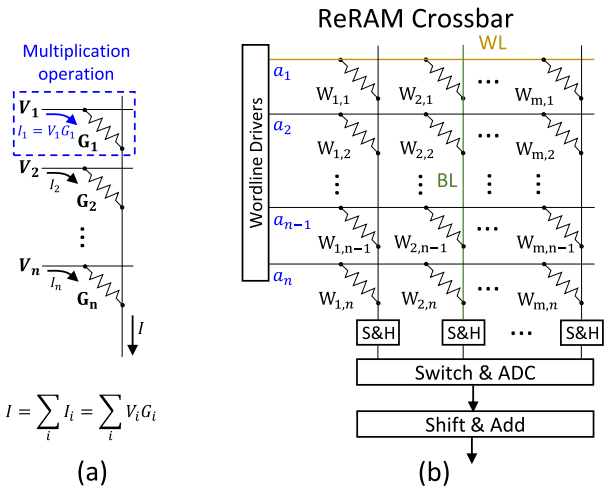


FIGURE 11. **(a) Working principle of resistive crossbars. (b) Crossbar architecture.**

generalized for resistive crossbar microarchitecture, and agnostic to the underlying device-level specifications.

E. RELATED WORK AND MOTIVATION

DaDianNao [11] is a notable CMOS-based neural network accelerator design among others [64]–[68]. SCNN [12] and Cnvlutin [69] propose sparse neural network accelerators, targeting unstructured sparsity and dynamic activation-zero eliminations. State-of-the-art NVM-based designs, including ISAAC [15] and PRIME [70], improve throughput for a *single network's* inference cycles, while PipeLayer [71] and TIME [72] accelerate network training process. ReCom [73] and SNrram [74] present NVM based accelerators focusing on unstructured sparsity. TraNNsformer [75] proposes a spectral clustering scheme to mitigate the fragmentation challenge in unstructured pruning but focuses only on Fully Connected layers. Other proposals exploiting crossbar microarchitectures include [76]–[78]. Previous works [19], [33] have shown that most networks are significantly over-parameterized and may be pruned safely without sacrificing accuracy. However, no previous paper has focused on developing an understanding of the vast pruning search space, especially towards aligning existing network designs with the resistive crossbar's parallelism model. Despite significant literature on neural network *design techniques* [79], [80], none of the previous works have explored developing these ideas in the context of resistive crossbar's parallelism model. Further, previous proposals have overlooked massive deployment costs of neural network ensembles, which reduces the practicability and industrial viability of their proposition. *These problems stem from a one-design-fits-all approach, ignoring practical concerns for the sake of generalization.* To the best of author's knowledge, there is no prior work in the direction of *guiding network architecture search to improve data-flow on crossbar hardware*. For instance, none of the above works discuss how to exploit pruning to improve power efficiency on resistive crossbars, nor do they consider

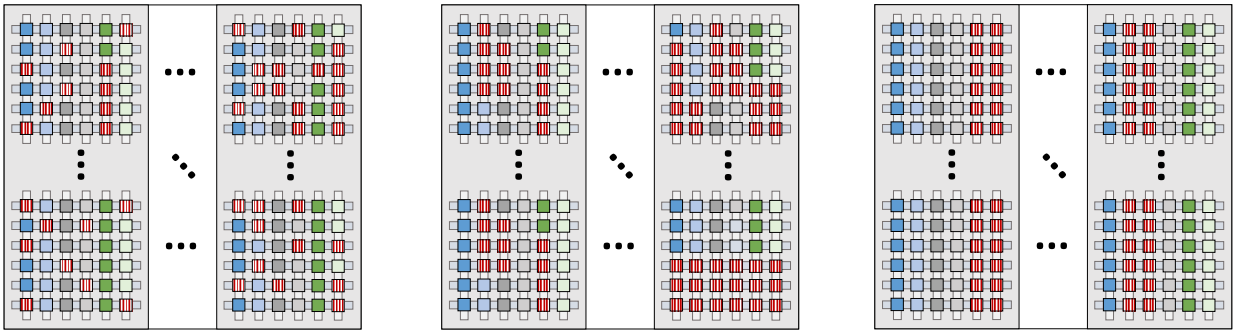


FIGURE 12. Examples of mapping of sparse neural networks in a crossbar. Blue, green and grey cells represent sparse tensors with the empty cells shown in red. Left-to-Right: pruning increases from connection-based, vector-based to filter-based.

deploying ensembles on their hardware for a realistic test-bench.

III. ALIGNING THE DATA-FLOW WITH UNDERLYING HARDWARE PARALLELISM

This section demonstrates how a network must be pruned to improve alignment with the crossbar's underlying parallelism model to obtain a significant reduction in the number of crossbars needed to map a DNN. Sections III-A and III-B highlight the key challenges and opportunities in using pruning to improve the efficiency of DNN inference on crossbar-based hardware. Based on the highlighted challenges and opportunities, Section III-C presents our pruning algorithms. Section III-D defines a method for choosing the appropriate pruning algorithm, followed by key takeaways in Section III-E.

A. SPARSE NETWORKS IN CROSSBAR

Figure 12 demonstrates convolutional tensors mapped on a group of crossbars after applying different pruning techniques. Starting from left, the first crossbar set represents mapping for a layer with non-structural pruning of connections [19]. The zeroed-out weights (highlighted in red verticals) have no impact on the final result, nor do they improve the crossbar area efficiency (zero-weight values do not contribute anything to the output). Due to the non-uniform nature of this sparsity granularity, precision of the final result cannot be determined in advance. This forces full-precision processing by peripherals to accommodate for the worst-case scenario, preventing clock-gating based reductions in energy or latency. Although this granularity achieves the highest compression ratios for fully connected layers [31], the insignificant reduction in storage footprint when mapped on crossbars restraints practical improvements in area and energy efficiency.

The middle crossbar set exhibits a vector-based weight pruning strategy [81], with contiguous blocks of zeroed-out weights. If such sparse blocks are spread out across different locations in the layer (illustrated in upper-left, upper-right and bottom-left crossbars), this technique offers no improvement over the previous case. However, if all filters enforce sparse blocks in the same location for a given layer (e.g. bottom-right crossbar), the corresponding unrolled tensor can eliminate the zeroed-out connections and directly reduce the storage requirements (Figure 2). Given the static computation graph for neural networks is known at compile tile, off-chip memory accesses for the irrelevant sections of activation maps can be removed from the data fetching queue. Hence, aligning the pruning process for all filters collectively at the block level allows for improving the area and energy efficiency without sacrificing accuracy.

The third (right-most) crossbar set in Figure 12 depicts layers mapped after pruning entire filters [82], [83]. This eliminates complete columns from the crossbar mapping. Reducing the number of columns lowers the number of ADC sampling cycles needed to process results from all targets in a crossbar, which may be used to lower the ADC sampling rate (decreasing power proportionately) [84], [85]. This further generates space to map more filters in the given crossbar (from the same or different layers). In the case of different layers, it is important to note that the *concurrently mapped filters* should ideally have common input feature maps (similar to filters from the same layer). In the case of different inputs for different columns, the corresponding activation maps may be time-multiplexed for correct target computation. Even in this scenario, filter pruning allows for a denser mapping, trading throughput with storage efficiency (especially on embedded devices or resource-constrained platforms).

These observations motivate us to exploit ***vector-based and filter-based pruning algorithms*** *for aligning the data-flow with the underlying hardware's parallelism.*

B. PRUNING SENSITIVITES IN NETWORK LAYERS

Figure 13 displays accuracy loss of different layers in the VGG-16 network upon pruning. Shallow layers (conv1, conv2, conv3) are noticeably sensitive to pruning but have fewer filters due to large and thin activation maps. Deeper layers tend to be significantly overparameterized and may be pruned without significant accuracy drop penalties. This can be understood as follows: As we go deeper, the network searches for a very large number of features (in overparameterized layers) against smaller activation maps (information filtered from previous layers). *Filter pruning*

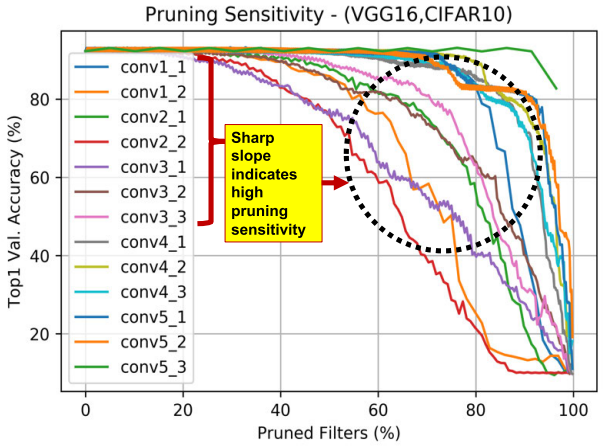


FIGURE 13. Deeper layers (flat slope, upper right corner) offer high pruning potential with minimal accuracy loss.

Algorithm 1 Structural Filter Pruning

Result: ConvNet with pruned filters
Input: Trained ConvNet, δ, α;
initialization: Split train-valid dataset
while *Accuracy loss* $< \delta$ **do**
  for *pruning_iteration i* **do**
    rank filters by batch normalization constants in each layer
    eliminate filters below α;
    generate compact model;
  end
  fine-tuning to recover accuracy
end

enforces concentration of these features in fewer parameters by eliminating the excess filters.

C. PRUNING ALGORITHMS

1) INTER-FILTER SPARSITY

Observing Figure 13, we propose a filter-pruning algorithm based on state-of-the-art research to eliminate filters with low impact on network performance [82], [83], [86].

Optimization objective for network training is given as:

$$L = \sum_{(x,y)} Loss\ Function(f(x, W), y) + \lambda \times \sum_{\gamma \in B.N.} g(\gamma) \quad (2)$$

x, y, W represent input, ground-truth and network weights respectively in Equation 2.

Summary: Algorithm 1 minimizes the output scaling batch normalization constant (γ) associated with each layer using the regularizer in Equation 2. δ represents the tolerable accuracy loss (dataset dependent), and α represents the filter pruning threshold hyper-parameter (determines the aggressiveness of pruning). When a specific filter's output scaling constants fall below the threshold, the filter is forcibly set to zero and eliminated from the layer. The fine-tuning process continues with a new copy of the model without this filter and proceeds to recover the accuracy loss.

Algorithm 2 Perception Field Pruning

Result: ConvNet with pruned receptive fields
Input: Trained ConvNet, θ, ϵ, δ
initialization: Split dataset into train-validate
while *While weight groups were pruned in last* 5 *epochs* **do**
  SGD with regularizer (3)
  if *norm of weight group* $< \epsilon$ **then**
    eliminate group from tensor
  end
  if *accuracy loss* $< \delta$ **then**
    increase θ
  else
    decrease θ
  end
  fine-tune to recover accuracy
end

Strength: The algorithm scales as $O(Layers)$ with network depth, orders of magnitude better than $O(Weights)$ scaling followed by [19]. For example, most networks usually have 10^2 layers vs 10^9 weights.

We hypothesize that incorporating the filter pruning process within network training process by including the crossbar parameters within the loss function may provide for highly targeted improvements in efficiency, but at the cost of code portability. However, this is left for future work.

2) INTRA-FILTER SPARSITY

Perception field-based structured pruning [87] offers opportunity to reduce unrolled filter dimensions (Figure 2, Equation 6). We observe that this may be utilized to eliminate spillover while minimizing accuracy degradation.

Truncated $l_{2,1}$ norm regularizer for unified weight-group sparsity across filters is given as:

$$\omega_{2,1}^{T}(K) = \lambda \times min \sum_{a,b,j} \Big(\sqrt{\sum_{t=1}^{T} K(a,b,j,t)^2}, \theta \Big) \quad (3)$$

In Equation 3, K represents the filter[3] and λ denotes the weight-decay hyper-parameter. $l_{2,1}$ norm is favored over the group lasso due to higher computational efficiency [81], [88], [89].

Summary: Algorithm 2 details the perception field pruning algorithm. θ, ϵ, δ are hyper-parameters representing the intensity of field pruning, threshold for setting weight-groups to 0, and tolerable accuracy loss, respectively. Pruned weights are eliminated from the network using sparsity masks for each layer. The performance of the algorithm was generally observed to be insensitive to the choices of θ & ϵ. δ must be tuned based on the dataset.

Strength: It has been observed that high-entropy weights in receptive fields tend to cluster in almost circular shapes upon pruning, similar to Figure 14 [87]. This is an interesting observation which correlates with findings from the

[3] same as described in Section II.A

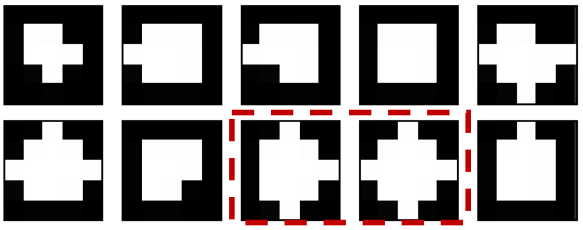


FIGURE 14. Sparse receptive fields tend to cluster in *roughly circular shapes*, mimicking biological visual receptors. Non-zero weights shown in white.

biological visual cortex [90], and may be attributed to nature's preference for circular feature detectors (an area-minimizing shape) against sharp-edged square fields used by artificial neural networks (due to underlying computation scheme). However, in case of very small filters (1×1), the algorithm might not offer any compression. In our analysis, we observed that such filters constitute less than 0.1% of the network parameters.

A natural question arises regarding the initial costs involved in effectively exploiting these optimizations. Arguments in favor include:

1) Figure 1 and 2 demonstrates how filter-pruning and field-pruning algorithms improve crossbar utilization significantly (detailed results in Section VII.D).
2) Both pruning algorithms are directly applicable to all contemporary and future convolution-based architectures, and scalable across network width and depth. [34]
3) The algorithms enable networks to exploit crossbar's parallelism model effectively and do not mandate any specialized hardware support to obtain practical speedups, unlike prior proposals [13].
4) Initial network mapping on crossbar expedites very high write energy, which is directly reduced by these optimizations [91].
5) These recommendations are orthogonal to hardware innovations and software optimizations presented in prior works [92]–[94], which ensures future relevance of these ideas and algorithms.

We do not drive the pruning processes to the maximum-possible sparsity achievable by these algorithms. This is because the objective is to maximize crossbar amenability while using as few pruning iterations as possible. This eliminates diminishing returns in the optimization process.

D. ALGORITHM SELECTION

The crossbar *occupancy ratio* quantifies the density of non-redundant weights in the occupied crossbars.

$$Occupancy\ Ratio = \frac{Number\ of\ occupied\ junctions}{Total\ available\ junctions} \quad (4)$$

Consider the example of a convolutional layer with 128 $3 \times 3 \times 64$ filters, and a 128×128 dimension crossbar. Unrolling these filters, the mapping dimensions are 576×128, which need 5 crossbars for computation. Mapping without pruning wastes 50% of the row space in the last crossbar, yielding 0.9 occupancy ratio. Using the perception field pruning strategy, pruning a single weight group yields new dimensions of 512×128, which can be *perfectly* mapped on 4 crossbars, improving area-efficiency by 20%.

Alternately, using the filter pruning strategy, 8 filters must be pruned from the previous layer to obtain new dimensions of 504×128. This yields a map of 4 crossbars, with 10% improvement in occupancy ratio ($= 0.99$). It can be observed that both strategies may be exploited to reduce the unrolled tensor dimensions. To enable balancing between these algorithms efficiently, we define a new hyper-parameter Ω, which can be set for each layer in the network.

$$\Omega = \frac{Row\ Occupancy\ Ratio}{Column\ Occupancy\ Ratio} \quad (5)$$

A low row occupancy ratio implies large filter dimensions spilling over into neighboring units. In this case, Equation 5 indicates Ω sits closer to 0, which prioritizes perception field pruning. In case of low column occupancy ratio, Ω gets biased towards higher values, prioritizing filter pruning. We combine these observations in algorithm 3.

In our experiments, we observed that Ω ranges from 1 to 1000, with a theoretical upper limit of 10^6. Ω must be adjusted for every network and dataset. It would be interesting to consider integrating it as a *learnable hyper-parameter* during the network training itself, paving the way for dynamic network pruning techniques given crossbar specifications [33], [95]. However, this needs further exploration and is left for future work.

Algorithm 3 Crossbar-Aware Network Pruning

Result: Pruned ConvNet with higher occupancy ratio
Input: Trained ConvNet, δ, α, θ, ϵ, δ
initialization: Split dataset into train-validate
1. Determine the number of crossbars needed for each layer in the original network.
2. Identify layers with low occupancy ratios and concurrent junctions (using algorithm IV-B).
3. Apply both pruning algorithms separately to generate sensitivity curves similar to Figure 13. This is used to identify the vulnerable weight groups and layers for pruning.
4. In case of low Ω, prefer perception field pruning (algorithm 2).
5. In case of high Ω, prefer filter pruning (algorithm 1).

E. KEY TAKEAWAYS

We demonstrate that:

1) Structured pruning is critical for aligning neural network computation with the resistive crossbar's parallelism model.
2) We present a filter-pruning algorithm to alleviate the spillover challenge across crossbar columns. Detailed results are discussed in Section VII-C.

3) We present a convolutional-filter field pruning algorithm to alleviate the spillover challenge across crossbar rows. Detailed results are discussed in Section VII-C.
4) We demonstrate how to pick the appropriate algorithm given practical constraints.

IV. HARDWARE OPTIMIZATIONS

This section explains the hardware optimizations which improve the energy/power efficiency of DNN inference. Section IV-A explains how computations of a convolutional layer can be parallelized on crossbar-based hardware. Section IV-B presents our pipelining strategy to improve the computational throughput. Section IV-C exploits the lack of weight sharing in *FC* layers to reduce the hardware cost by increasing the number of crossbars per ADC. In the end, Section IV-D presents the key takeaways.

A. CROSSBAR'S PARALLELISM MODEL

Prior crossbar-based acceleration proposals [15], [70] rely on the observation that one set of input values can be delivered via wordlines across *all columns (filters) within the same cycle*. This allows for mapping a stack of multiple filters across columns and calculating MVM output in a single cycle. Based on this idea, prior works advocate a weight-stationary computation model [4] for mapping a layer's multiple channels on the crossbar columns. Further, each resistive crossbar acts as an independent computation unit. Prior works utilize this observation for pipelining sequential layers across dedicated crossbar units. Each unit receives partial inputs to start processing it's outputs, and delivers these outputs to the next crossbar in the pipeline [15].

B. PIPELINED MAPPING FOR CONCURRENT JUNCTIONS

Based on the parallelism model, we develop the *Concurrent Junction Identification and Mapping algorithm* to maximize crossbar occupancy_ratio and computational throughput. We first define the mapping algorithm and corresponding assumptions, and then explain our methodology and rationale. We model the neural network computation data-flow as a static, directed acyclic graph (DAG). A vertex is defined as a computation node, with each node representing a single channel of the current convolution layer. In this graph, we input an image at the source vertex (*start layer*, 3 input vertices for 3 channels), and traverse the graph based on the operations defined at each edge to obtain an output value at the sink vertex (*output layer*, vertices equal to number of output classes). We define *concurrent junctions* as a vertex at which there are multiple outgoing edges in the graph. In terms of data-flow, this is equivalent to a neural network junction in which the current activation map is delivered to multiple filters in the next layer.

We exploit the column-level parallelism by observing that *residual connections (in ResNet) and split branches (in InceptionNet, TreeNet) also represent concurrent junctions*. These concurrent junctions offer similar column-level parallelism opportunities. We also observe that ensembles can be represented via connected components in this computation graph, and the input layer represents a concurrent junction for the same. *These observations were ignored by prior works, a gap which this work seals.*

Formally, concurrent junctions can be identified via a Breadth-First Search (BFS) on the graph (vertices sharing the same processing number). BFS is used to group vertices into concurrent junctions which can be mapped on the same crossbar's columns. Algorithm IV-B describes the implementation details. The group identifier is used to group vertices to be mapped collectively. A group identifier of -1 indicates no group.

Strength: These optimizations enable elimination of repeat memory accesses by mapping concurrent junctions as channels on the same crossbar, and higher utilization for the available ADCs. The superior occupancy ratios enables higher area-efficiency by mapping a larger number of networks on the same chip.

Summary: Prior works [15] map the network on crossbars sequentially on a per-layer basis. They exploit the column-level parallelism by mapping the channels from the same layer on the same crossbar. However, they fail to consider residual and split branches in the parallelism model, which leads to redundant off-chip memory access, lowering computational efficiency due to poor ADC utilization and small occupancy ratios. Step 1 in Algorithm IV-B solves this problem.

Next, we focus on the challenge of mapping these concurrent junctions while balancing area and energy constraints. Each convolutional layer in the network depends on the prior layer's outputs to start processing (Figure 5). In case of deep networks, these dependencies can be exponential, especially when considering convolution stride > 1. Prior works manually identify these dependencies and duplicate the crossbars with high dependency to ensure constant throughput in deeper layers [15]. Here, we propose *the first algorithmic approach for solving this problem via a* ***Topological sort*** on the input network graph (Step 2 in Algorithm IV-B). This approach allows us to solve the concurrent junction and pipelining problem in a single step instead of the manual approach of previous proposal.

C. MITIGATING THE ENERGY EFFICIENCY BOTTLENECK

1) HARDWARE CHALLENGES

Analog-digital inter-conversion (via DAC and ADC) (Figure 16) costs 8× the actual computation energy (performed in the memristor array, see Figure 15) in current crossbar architectures.

2) WORKLOAD CHALLENGES

Table 2 in Section VII demonstrates how Fully Connected (FC) layers are responsible for occupying the maximum number of crossbars in a convolutional network mapping. Prior works have shown that it is possible to

Algorithm 4 Concurrent Junction Identification and Mapping

Result: Pipelined crossbar mapping which accounts for all concurrent junctions in the network computation graph

Input: Input Neural Network in DAG representation

Step 1: Breadth-First Search to identify concurrent junctions

Initialization: Instantiate each vertex's group_identifier $= -1$, push the start vertex on the vertex_queue, update the start vertex's group_identifier $= 0$, instantiate the vertex_counter $= 1$.

while *queue has pending vertices* **do**

1. Pop the vertex at the top of queue, start processing it.
2. For each outbound edge, add the outbound vertex to vertex_queue. Assign the current vertex_counter to this vertex's group_identifier.
3. Increment vertex_counter by 1.

if *any vertex is unexplored (group_identifier* $== -1$*)* **then**

Add vertex to vertex_queue

```
// ensures all graph components
are covered
```

end

end

Step 2: Topological Sort to balance the pipeline between vertex_groups

Initialization: Instantiate each vertex's sequence_identifier $= -1$, push the start vertex on the vertex_queue, update the start vertex's sequence_identifier $= 0$.

while *queue has pending vertices* **do**

1. Pop the vertex at the top of queue, start processing it.
2. For each outbound edge

if *vertex already visited (sequence_identifier* $! = -1$*)* **then**

Sequence_identifier = current vertex's sequence_identifier + 1.

Recurse on this vertex until no more neighbours exist.

end

if *any vertex is unexplored (sequence_identifier* $== -1$*)* **then**

Add vertex to vertex_queue

```
// ensures all graph components
are covered
```

end

end

Step 3: Mapping pipelined vertex_groups on crossbar

while *sequence_identifier* $! =$ *end* **do**

1. Assign each filter from the vertex_group progressively on crossbar's columns.
2. In case of large strides, duplicate the dependency crossbars for maintaining throughput in deeper layers.
3. In case of more columns needed than available, saturate current tile's available crossbars before moving to next tile.

end

reduce this requirement by pruning such layers [19], [34], [96]. Unlike convolutional layers, there is no weight-sharing

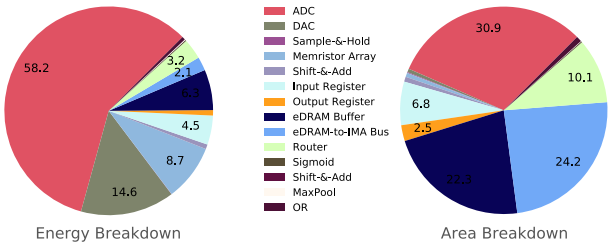


FIGURE 15. Understanding the per-component cost of analog compute in ISAAC [15] microarchitecture.

in FC layers, which nullifies the energy savings due to weight-reuse (per input cycle) in crossbars. Moreover, the *all-to-all communication bottleneck* prohibits processing with partial outputs from the previous layer. As an illustrative example, consider the *FC7 layer* in VGG16 with 4096×4096 dimensions, mapped on 128×128 2-bit crossbars. 8192 crossbars are required for calculating the result for one input in a single cycle. A naive strategy to reduce crossbar storage requirements would be matrix tiling, overwriting the corresponding tiles with the required FC weights as inputs become available. As an example, if we dedicate only 256 crossbars for this layer, 32 context switches would be needed to compute the output. However, this approach is infeasible due to the enormous differences in current read/write capabilities for underlying devices.[4]

3) OPTIMIZATION

ISAAC utilizes a ratio of 1 ADC/crossbar to maintain the digital output pipeline for every cycle. Considering the enormous energy cost for writing new weights in crossbar on-the-fly against ADC power/area consumption (Figure 15), we propose *increasing the number of crossbars per ADC* in specially dedicated *DNN_Tiles*. This distributes the computation load across multiple ADC cycles in a pipelined fashion. The input data buffer requirements can be reduced and aligned with ADC availability for the particular crossbar with corresponding weights. In the example noted above, using an 8 crossbar/ADC ratio reduces buffering requirements per output cycle by 8×, per IMA power by 58%, area by 65.8%, and produces a more balanced pipeline at the cost of 8 extra cycles. More importantly, this technique provides for easier manipulation of chip's power consumption without relying on power gating the ADCs.

D. KEY TAKEAWAYS

We demonstrate that:

1) The crossbar's computation model is based on independent column-based computation units. Hence, aligning the network's MAC computations with column sizes improves efficiency.
2) The crossbar's data-flow model is based on common vector inputs to the wordline, which are shared across

[4]memristor write cycle costs three orders of magnitude higher energy and one order of magnitude higher latency than read cycle [91]

all columns. Hence, the network filters that share common activations must be aligned across the same crossbar's columns to reduce data movement.

3) Large fully connected layers worsen the energy efficiency for crossbar designs. This problem can be solved by reducing the number of ADCs allocated for computing FC layers.

V. CROSSBAR-AWARE NEURAL NETWORK DESIGN

Considering the aforementioned problems including high number of crossbars needed per network, low occupancy ratio extending to spillover, and peripherals throttling the energy efficiency, designing novel *crossbar-aware network architectures* can tremendously increase the computation efficiency with minimal changes to the hardware architecture. Surveying state-of-the-art neural network architectures [97]–[104], we propose the following design strategies targeting convolutional neural network design for resistive crossbars:

1) STRATEGY 1: SMALL PERCEPTION FIELDS

Crossbar column size is bottlenecked by ADC precision and bitline current draw constraints. Large columns offer poor energy and area efficiency, hence, smaller column sizes are preferred.

$$\textit{Unrolled filter parameters} = K_x \times K_y \times N_i \tag{6}$$

Equation 6 represents the unrolled vector length needed to map a convolutional kernel on the crossbar column. Smaller filters offer a quadratic reduction in required row capacity (5×5 has $3\times$ more parameters than 3×3 for a fixed number of input channels). Prior works [2], [8], [105], [105] have demonstrated that stacks of smaller filters can approximate a large filter without losing the information capacity. For example, a stack of three 3×3 filters has the same representation capacity as one 7×7 filter, with almost half the parameter cost. Furthermore, larger perception fields have been demonstrated to be significantly redundant towards the fringes (Figure 14) [87].

2) STRATEGY 2: DECOMPOSING CONVOLUTION FOR CROSSBARS

$$\textit{Output datapoints} = \left\lfloor \frac{A_x^{in} + 2 \times padding - K_x}{stride} + 1 \right\rfloor^2 \times N_o \tag{7}$$

Equation 7 formula is used to calculate the number of calculations needed to compute the output for the convolution operation. It may be observed that decomposing the perception fields from $K_x \times K_y$ to $K \times 1$ (*spatial-separable convolution* [99], [105]) reduces network's parameter costs and cycle count from quadratic (K^2) to linear ($2K$) scaling, as compared to standard 3D convolution computation.

Depth-wise convolution [100] breaks down Equation 7 in 2 stages - depth-wise ($K_x \times K_y \times 1$) and point-wise ($1 \times 1 \times N_o$) [106], [107] convolutions. For illustration, consider a $12 \times 12 \times 3$ activation map and $5 \times 5 \times 3$ filter, generating $8 \times 8 \times 256$ output. Standard convolution costs $1,228,800$ multiplications, which is reduced to $4800 + 49152 = 53952 = 4.4\%$ of original $1,228,800$ multiplications by depth-wise format (Equation 7). Notably, it is further possible to create a pipeline between the 2 stages, *drastically reducing the inter-layer buffering requirement and cycle count.*

3) STRATEGY 3: SMALLER CONVOLUTION STRIDE

Prior works [15], [70], [71] focus on maintaining a constant output throughput by balancing the data pipeline between the layers. The key observation is that this pipeline model advocates duplicating shallow layer crossbars to meet the succeeding layer's input buffer requirements. Let S_x and S_y be the convolution strides in x & y direction over the activation maps. To prevent data non-availability stall in the L_x layer, the previous layer's (L_{x-1}) crossbars must be duplicated by $S_x \times S_y$. As an illustrative example, consider a network with 5 identical sequential convolution layers. If all layers perform convolution with a stride of 2 in both directions, crossbars assigned for layer 1 must be replicated 1024 times to produce results from the final layer in every cycle.

Based on this observation, we propose enforcing low stride values (1, 2) in deeper layers. This can help in easily eliminating data non-availability stalls. Further, multiple network design research groups [2], [105] have reported higher accuracy and low parameter count by avoiding premature down-sampling of the information flow towards the deeper layers, *a key feature of low stride values*. Some groups [108] also report balancing this down-sampling process via the MaxPooling layers (which do not utilize crossbars). It should be noted that larger strides in pooling layers are easily manageable as they depend on digital peripherals.

4) STRATEGY 4: 1×1 FILTERS TO COMPRESS DEEP FILTERS

Equation 6 demonstrates that N_i (filter depth) directly inflences the final filter size mapped on crossbar column.

Equation 7 demonstrates that a large number of filters in the current layer proportionally increases the next layer's input dimension ($\times N_o$).

To solve this problem, most modern network designs [9], [108] utilize 1×1 filters (introduced in [109]) to compress information from multiple channels into a single map. This can significantly reduce the output dimensions and cycle count for deeper layers.

5) STRATEGY 5: ELIMINATE/ZZZZZ/WWWWW/PRUNE FULLY CONNECTED LAYERS

The fully connected (FC) layers tend to concentrate the weight density in most designs [4]. With an all-to-all communication requirement, minimal weight-sharing and enormous matrix dimensions ($7\times7\times512*4096$ FC-1 layer in

VGG-16 needs 50176 128 × 128 2-bit crossbars), these layers are a bottleneck for storage efficiency.

Modern network architectures [9], [108], [109] eschew FC layers in favor of *global average pooling*, which boosts computation performance and also increases the accuracy slightly. If they cannot be eliminated, pruning such layers yields almost immediate improvements in storage efficiency and cycle count. However, unstructural pruning advocated in [19], [75] mandates dedicated hardware support for effective speedup.

Focusing on conv5 layer in Figure 13, we observe that pruning entire filters from the last convolutional layer can dramatically reduce the matrix dimensions, while preserving data locality (pockets of dense connections corresponding to remaining layers). For example, pruning merely 50% filters in the last convolutional layer of VGG16 cascades as a direct 50% reduction in area requirement. This is well below (55%) the total number of filters which can be pruned from this layer.

Although these strategies look relatively straightforward, they have far-reaching consequences for the hardware-software co-design search space:

1) These ideas provide constraints on the network design search space. This is important for applying state-of-the-art *reinforcement learning and Network Architecture Search (NAS)* based techniques for automatically discovering hardware-aware network designs. [110]–[113].
2) As software requirements get easier to estimate, hardware architects can focus on optimizing ISA, error correction codes, compilers and tackling the precision challenges, while offering higher performance guarantees [114]–[116].

A. KEY TAKEAWAYS

We demonstrate that:

1) Resistive crossbars offer a different computation model which was not considered as part of the design process of contemporary neural networks.
2) We identify the shortcomings observed in existing hardware and software. Based on this analysis, we propose *the first crossbar-aware design strategies for optimizing network performance*.

VI. A NOVEL NEURAL NETWORK FAMILY - CrossNet

In this section, we utilize the proposed strategies to explain the design choices for a crossbar-aware network architecture and propose the CrossNet family of networks.

A. CrossNet

We began the quest for a novel crossbar-aware network architecture by modifying the SqueezeNet architecture [108], intending to improve the design towards higher hardware amenability and/or task accuracy. To improve the accuracy, we first tried to increase the number of parameters in the network by reducing input channel compression. If the number of input channels for all expansion layers are increased by 4×, Top-5 accuracy improves to 85.3%, at the cost of 2.39× extra crossbars. Interestingly, the occupancy ratio also improves to 0.882 (32.45% increase). To understand the impact of 1 × 1 and 3 × 3 filters on task accuracy and hardware performance, we replaced all 3 × 3 filters in-place by 1 × 1 filters. The accuracy dropped to 76.3%, occupancy ratio reduced by 4.57%, and the number of crossbars decreased by 30%. Several such optimization rounds of the channel squeeze and expansion parameters, network depth, and application of the aforementioned crossbar-aware network design techniques led to the novel architecture family, which we titled as *CrossNet*.

The first variant, **CrossNet-A**, achieves SqueezeNet-equivalent accuracy, with 2× lesser parameters, 16.85% improved power efficiency (defined as MACs / area for unit energy consumption), and 2.5× higher computational efficiency (defined as MACs / number of cycles). The network achieves the highest computational efficiency (19.06× against VGG11) at the lowest storage cost (149.5× lower against VGG11) among all bench networks. To further improve accuracy, we increased the activation map dimensions and the number of channels, which resulted in the CrossNet-B architecture.

CrossNet-B achieves accuracy closer to GoogleNet (88.2%), with 2.89× higher accuracy/million-parameters. Unlike the CrossNet-A, CrossNet-B has double the number of input channels in most layers. Further, it achieves high-accuracy with a 58.93× lower parameter count than VGG-11, effectively offering 58× higher accuracy/million-parameters. A notable aspect is that the CrossNet designs utilize a single FC layer before final output from the Softmax classifier. To reduce the input dimensions to the FC layer, the input channels are average-pooled into a single channel (*Strategy 5*), which tremendously reduces the storage requirement (1000× lower than VGG family). *Extensive focus on the strategies noted above significantly reduced the search space while enabling the architecture to achieve the highest computational and power efficiency among all network benchmarks.*

We present comprehensive results for the improvements obtained by these designs over contemporary networks (Section VII.E), along with hardware mapping results and network specifications for both CrossNet-A and CrossNet-B.

B. ENSEMBLES

Drawing upon the deductions from ILSVRC-2014 winners (Section I), we hypothesize that it should be possible to further improve CrossNet's performance with simple ensembling strategies. To validate this hypothesis, we experiment by making an ensemble of 4 units and attempt to map the same on-chip.

The ensemble utilized naive averaging as final output, and no special modifications were required in hardware to support this design. Based on Figure 10's observations, Random Initialization was observed to provide maximum performance

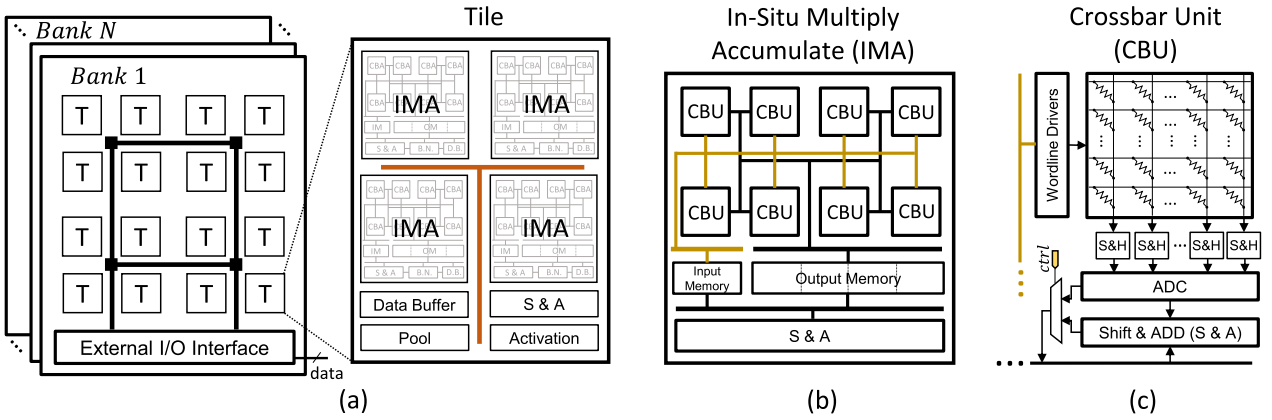


FIGURE 16. Architecture of ReRAM crossbar-based DNN hardware.

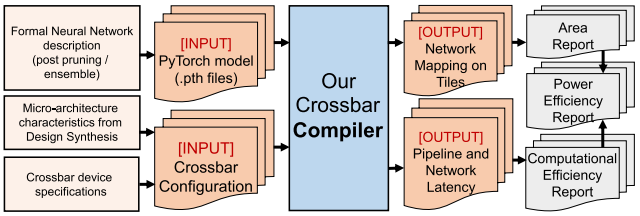


FIGURE 17. Experimental design flow.

improvements and was selected as the ensembling method in this case.

We observed that it was easier to improve the task accuracy with an ensemble of models trained for fewer epochs, as compared to training a single model for a large number of epochs. This may be understood as a parallelization of exploration of the network's training search space, with simplified linear scaling of compute costs. The ensemble's performance was initially worse than individual models. However, after 5 epochs, the performance improved and surpassed the individual model's accuracy. Upon continued training, the 60-epoch ensemble's performance surpassed a 200-epoch single model.

C. KEY TAKEAWAYS

We demonstrate that:

1) The proposed CrossNet family of neural networks significantly outperforms prior network designs in computational and power efficiency.
2) The performance improvements are well understood, and attributed to aligning the neural network architecture with the crossbar's parallelism model.
3) We consider ensemble systems as first-class candidates for practical deployments and include them as part of the testing conditions. The Results Section VII.F quantifies the improvements obtained.

VII. RESULTS

A. EXPERIMENTAL SETUP AND METHODOLOGY

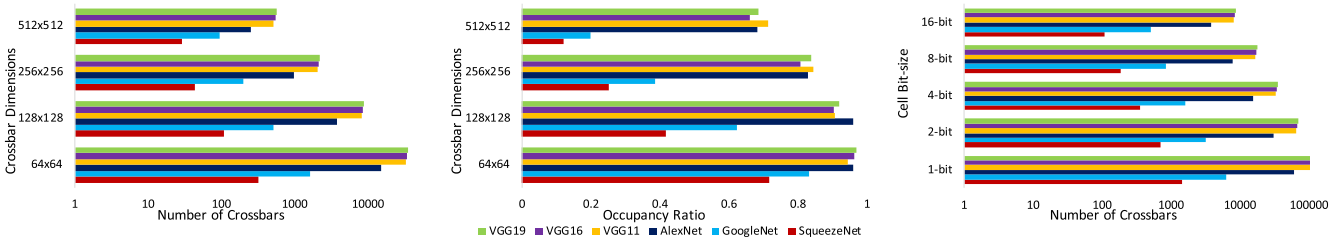
Figure 17 explains the methodology used for this paper. Neural Network definitions were utilized from PyTorch's model library.[5] The networks were trained on a *NVIDIA Tesla K80* GPU via the Google Colab platform. For ensemble training, we chose the *CIFAR-100* dataset for experimentation as it represents a sweet spot in terms of complexity (100 output classes, 50000 training examples, 10000 test examples) and training time (240 seconds per epoch). Training was performed using the *Adam optimizer* with the following hyper-parameters: $learning\_rate = 0.1$, $learning\_rate\_decay = 0.02$, $num\_epochs = 200$, $batch\_size = 128$, $momentum = 0.9$, $weight\_decay = 5e-4$.

Considering the lack of open source tools to map network workloads comprehensively on crossbar microarchitectures, we wrote a compiler for mapping a given network configuration on the crossbar microarchitecture (Figure 17). We open-source the compiler code for the benefit of the research community at the code repository.[6]

This work primarily targets improvements in data-flow characteristics at the fundamental level of the resistive crossbar PIM platform. Owing to the fast-paced changes in underlying memristor cell development (bit-size, endurance, isolation mechanisms etc. [117]–[120]) and immature MLC fabrication technology [121]–[123], we assume a generalized crossbar system (Figure 16) as the fundamental computation unit. This enables us to validate the ideas and strategies independent of device characteristics. Further, this system allows us to verify how these ideas will scale with future developments in resistive memory technology. Our compiler accepts a *crossbar_configuration* file along with Pytorch model file and generates the corresponding network maps.

Table 1 details the power and area values for all microarchitecture components in this system. For crossbar latency, energy and area, we utilize [15], [59], chosen primarily due to the clarity of approach and reproducibility. For handling the precision challenge, we utilize a 16-bit computation model, spread out across 8 columns of 2-bit cells in a 128 × 128 crossbar array [59]. 1-bit DAC and 8-bit ADC are used for

[5]https://github.com/pytorch/vision/tree/master/torchvision/models
[6]https://github.com/adityamanglik/NeuralNetworkCrossbarCompiler



(a) Impact of crossbar dimensions on area (b) Crossbar dimensions and occupancy ratio (c) Cell bit-size vs. number of crossbars

FIGURE 18. Logarithmic scale for number of crossbars highlights immense storage requirements, addressed by our strategies and algorithms.

TABLE 1. Energy and area for different microarchitecture components [15].

Component	Specification	Power[mW]	Area[mm^2]
IMA Properties			
ADC	8x8bits, 1.28 Gsps	16	0.0096
DAC	128x1bit, 8 units	4	0.00017
Sample+Hold	128x8	.01	0.00004
Memristor Array	128x128, 8 units	2.4	0.0002
Shift+Add	4	0.2	0.00024
Input Buffer	2KB	1.24	0.0021
Output Buffer	256B	0.23	0.00077
Tile Properties			
IMAs Total	12	289	0.157
eDRAM Buffer	64KB	20.7	0.083
eDRAM-IMA Bus	384	7	0.09
Router	8	42	0.151
Sigmoid	2	.52	0.0006
Shift+Add	1	.05	0.00006
MaxPool	1	.4	0.00024
Output Register	3 KB	1.68	0.0032
1 Tile Total		330	0.372

analog-digital signal inter-conversion. The clock speed for computations is 10 MHz, and the ADC operates at 1.28 Gsps.

B. CROSSBAR DESIGN SPACE EXPLORATION

Using the diverse network bench (Table 2), we performed a design space exploration to determine the ideal crossbar system. An oracle system (ideal scenario) should offer highest possible occupancy ratios (maximum 1.0) while minimizing the compute area (determined by crossbar dimensions and cell bit-size), for any network architecture. Due to the aforementioned challenges associated with large crossbars (high IR drop and high read/write voltage requirements), we constrained the upper-limit of crossbar size as 512 × 512. Furthermore, multiple works [124]–[126] have reported neural network accuracy to be unaffected when reducing computation precision to 16-bit. This allowed us to place an upper-limit of 16-bit on the MLC bit-size (Figure 18(c)). For lower bit-sizes, the computation is spread across $16/w$ columns, where w represents the cell bit-size (this has no effect on output accuracy, as noted in [15]).

Analyzing Figure 18 (a) and (b), we observe that 128×128 crossbar size is suitable for minimizing the area requirement for large networks, while maintaining high occupancy ratios. Interestingly, 64 × 64 crossbars offered higher occupancy ratios on average, but were deemed impractical due to unfeasible area requirements (VGG19 needed 280576 crossbars, translating to 5506.3 mm^2 area). On the basis of these observations from the design sweep, we concluded 128×128 size as the best-fit. Figure 18 (c) demonstrates the variations in the number of crossbars for different networks across cell bit-size (addressability). Increasing the cell addressability by one order of magnitude inversely reduces the number of crossbars by 93.72%, but the power and area costs associated with corresponding high-precision peripherals and analog-digital inter-conversion circuits[7] severely limit throughput improvements. In our design space exploration, $w = 2$ emerged as a sweet spot in terms of balancing the area and energy trade-offs for the test bench.

1) COMPARISON WITH CPU AND GPU

Using this configuration, AlexNet [127] can be mapped on 30474 (2-bit) crossbars. Single inference pass for one image costs 3025 cycles. For a batch of 16 images, our reference system requires 4.8 *ms*. For comparison, the *NVIDIA GTX 1080* GPU requires 7 ms. for the same batch. We infer that our system is 31.43% faster than the GPU, at a 16× lower clock speed. This speedup can be explained by the massively parallel compute capabilities and weight-stationary data layout in crossbar-based microarchitectures.

The VGG-19 [8] network requires an enormous 70168 crossbars & 50176 cycles, costing 80.28 ms for the same 16-image batch. We now compare to an *Intel Dual Xeon E5-2630 v3* CPU, which consumes 3609.78 ms. In this case, our system is 44.96× faster at 320× lower clock rate. Essentially, performing the multiply-and-accumulate operation via analog computation in a single column read, and processing multiple filters in the same cycle leads to high throughput for such GEMV dominated algorithms.

C. PRUNING ALGORITHMS

Figure 19 demonstrates normalized layer-wise area reduction obtained by channel pruning. We observe that, on average, pruning reduces the number of input channels by 81.2% across convolutional layers. This translates to 91.68% average reduction in the number of crossbars needed to map the network. Pruning the last convolution layer reduces the fully connected layer dimensions by 92.35%. Further, This reduction does not sacrifice the dense structure of the matrix

[7] 16-bit DAC and ≥ 26-bit ADC are needed for 16-bit MLC cells

TABLE 2. Network maps obtained by our compiler for 128 × 128 crossbar with 2-bit computation model. Cycles and Computational Efficiency have been normalized as these values are microarchitecture dependent.

Network	Crossbars 2-bit, Conv	Crossbars 2-bit, FC	Total Crossbars	Occupancy Ratio	Top-5 Accuarcy (Single model)	Parameters (Millions)	Accuracy per-million Parameters	Pipeline cycles (Normalized)	Computational Efficiency (Normalized)	Power Efficiency (Normalized)
AlexNet	1834	28640	30474	**0.9905**	80.3	62.378	1.2873	**1x**	15.97x	1.92x
VGG11	4508	**60384**	64892	0.9075	91.9	132.86	0.6743	16.59x	1x	2x
VGG16	7192	**60384**	67576	0.9377	92.8	138.357	0.6707	16.59x	1.09x	2.17x
VGG19	9784	**60384**	**70168**	0.9475	**92.9**	**143.667**	0.6466	16.59x	1.057x	2.11x
GoogleNet	3130	-	3130	0.9167	92.1	6.797	13.5487	4.30x	1.99x	1x
SqueezeNet	707	-	707	0.6659	80.3	1.236	64.9939	4.22x	7.60x	3.56x
CrossNet-A	**371**	63	434	0.3767	80.3	0.528	**152.2**	4.07x	**19.06x**	**4.16x**
CrossNet-B	1158	63	1221	0.7797	88.2	2.2545	39.12	4.15x	8.96x	3.86x

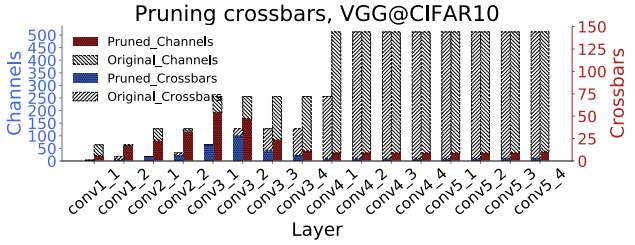


FIGURE 19. Impact of network pruning on the channels and crossbars units occupied by each layer of the VGG-16 network.

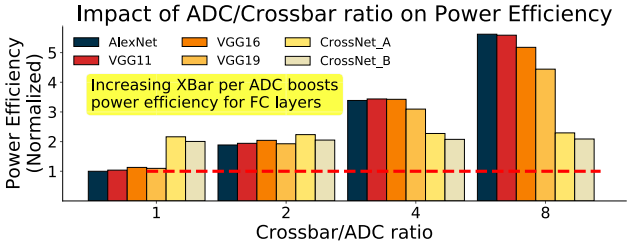


FIGURE 20. DNN_Tiles improve power efficiency tremendously for fully connected computations.

as the occupancy ratio for the remainder crossbars remains high (96.98%), corroborating *Strategy 5*. An interesting trend to note is the performance of the algorithm as we increase the network's depth. Initial layers offer less pruning viability due to small tensor dimensions (attributable to the fact that shallow layers extract high entropy features. Eliminating such filters causes a sharp accuracy drop (also seen in Figure 13)). In contrast, deeper layers offer high compression rates. Compounded with large tensor dimensions, these layers offer highest storage efficiency improvements.

In our reference system, the VGG19 conv2_1 layer has $\Omega = 0.11$. In this case, algorithm 3 favors perception field pruning which eliminates fringe weight groups and reduces storage requirement by 40%. Further, it eliminates 50, 176 input data loads at zero accuracy cost. For testing algorithm 3, we forcibly bias $\Omega = 1$ for the same layer. In this case, algorithm 3 enforces filter pruning strategy, eliminating 35.16% channels and 33.33% crossbars. However, occupancy ratio decreases by 13.56%. Input data movement is unaffected by this pruning, justifying the algorithm's original choice for field pruning.

1) COMPARISON WITH PRIOR WORK

Training neural networks with *stochastic gradient descent* using *back-propagation* scales by $O(3 \times N^3) \times (X/m)$, where N is the number of weight matrices, X is the number of training examples, and m is the mini-batch size.

The spectral clustering scheme proposed by *TraNNsformer* [75] costs $O(N^3)$ run-time additionally per filter (pushing training costs as high as $O(N^6)$), and is executed multiple times per training epoch. Such expensive optimization strategies are impractical for even medium-sized networks, far more so for large networks and ensembles. Our filter pruning algorithm prunes the networks in $O(Layer)$ additional cost per layer, per pruning epoch, a much more scalable and practicable approach.

D. HARDWARE OPTIMIZATIONS

1) CONCURRENT JUNCTION MAPPING ALGORITHM

Using the algorithm IV-B for SqueezeNet [108] reduces the number of crossbars by 20.51%, improving the occupancy ratio by 25.8%. However, it also increases the number of cycles by 3.042×, effectively decreasing the computational efficiency by 58.64%. However, overall power efficiency improves by 28%. We hypothesize that such dense mappings can be especially useful for field devices with constrained power envelopes, low memory/low-cost deployments, and edge computing scenarios (for IoT).

2) DNN_TILES

Figure 20 demonstrates power efficiency improvements obtained as we sweep the number of crossbars assigned per ADC. GoogleNet and SqueezeNet networks were not considered for this experiment due to absence of fully connected layers. For the stock configuration (leftmost set), CrossNet family offers maximum performance per Watt, thanks to a network architecture designed to maximize power efficiency for the given hardware. AlexNet and the VGG family have poor power efficiency on account of poor utilization of the available ADCs in the FC layer crossbars. However, as we increase the number of crossbars/ADC, these networks significantly improve the energy efficiency with minimal latency increment (increases by less than 1% of overall network's latency). Further, CrossNet family's minimal use

of FC layers results in insignificant improvements in energy efficiency across the sweep, corroborating Strategy 5.

E. EFFICACY OF DESIGN TECHNIQUES

Table 2 details results for the benchmark neural networks. To quantify the impact of our crossbar-aware network design techniques (Section V), we performed an ablation study with the network test bench. We observe that small perception fields help reduce the VGGNet's input layer ($3 \times 3 \times 3$) requirements by 66% against AlexNet ($11 \times 11 \times 3$), in line with *Strategy 1*. The VGG architectures require 60384 crossbars for the FC layers alone, which represents $86.05-93.05\%$ of total storage requirement across different variations (11-19 layers deep). For comparison, GoogleNet's complete ensemble with 7 networks (ILSVRC-2014's winning submission) requires only 21910 crossbars. This gap illustrates the challenges of FC layer's storage requirement on crossbar architectures. Interestingly, AlexNet and VGG have superior occupancy ratios, ascribed to an extremely uniform architecture for both networks, compared to the heterogeneous filters and information flow splits in the GoogleNet. AlexNet has the highest occupancy ratio and a low cycle count, which enables it to have the second-highest computational efficiency (only 19.35% lower than CrossNet, discovered 7 years in the future).

The Inception module uses heterogeneous small filters (1×1, 3×3, 3×1 and 1×3 etc.) in the same layer (similar to Strategy 1 noted above). GoogleNet uses the global average pooling layer instead of fully connected layers (in line with Strategy 5). The authors observe that average pooling improves classification performance by 0.6% as compared to FC layers. However, the original paper [9] mentions the use of a single fully connected layer for transfer learning towards different computer-vision tasks using a pre-trained network. As this layer has no significance for the original network's performance, we ignore it in our analysis. InceptionNet v1 also utilizes Local Response Normalization (LRN) layers, which are no longer considered important for the network's performance [128]. Future iterations of this architecture (v2, v3, v4) [105] eliminate these layers. Hence, we safely ignore them in our analysis.

The SqueezeNet [108] network utilizes Fire modules comprising of squeeze layers (1×1 convolutions, similar to Strategy 1 dicussed in Section III.C) and expand layers (3×3 convolutions). This is an interesting design choice as the expand layers are forced to use fewer channels due to the preceding squeeze layers (in line with Strategy 4). The network uses a stride of 1 for all convolutional layers (Strategy 3) and stride of 2 for max pooling, primarily for down-sampling activation maps. By replacing the Fully Connected layer with Global Average Pooling (Strategy 5), the architecture achieves AlexNet (60 million parameters) equivalent accuracy with only 1.25 million parameters. SqueezeNet achieves the second-highest Accuracy-per-million-Parameters and second-lowest crossbar requirements by compressing input channels (using 1×1 squeeze layers) before feeding into larger filters (*Strategy 4*). The last convolutional layer occupies merely 37 crossbars against 101 for GoogleNet and 144 for VGG11, attributable to slower down-sampling of information flow (*Strategy 2 and 3*). However, the Squeeze layers have low occupancy ratios (due to 1×1 filters) as compared to Expand layers, which reduces the overall average for the network. Considering a pipelined execution model, inter-layer activation-map buffering requirements are the lowest in SqueeezeNet (normalized to $1\times$ for apples-to-apples comparison), closely followed by CrossNet ($1.38 - 2.07\times$), GoogleNet ($3.8\times$) and lastly the VGG family ($15.23\times$), exemplifying *Strategy 3*.

1) CrossNet ANALYSIS

CrossNet-A is a deep architecture with 105 convolution layers, comprised almost entirely of 1×1 filters. It achieves Top-5 accuracy of 80.3%, which is on par with AlexNet. However, it occupies only 1.4% of AlexNet's area, uses $118.14\times$ fewer parameters and offers 19.35% higher computational efficiency. Despite poorer than average occupancy ratios and a large cycle count, CrossNet-A obtains highest power efficiency due to the efficient use of the minimized parameter count, resulting in the lowest MAC costs per input among all benchmark networks. Similar to CrossNet-A, CrossNet-B architecture also has 105 layers with residual connections. It significantly improves on the occupancy ratio (by $2.06\times$) due to larger filters. Larger storage requirements lower the power efficiency, attributable to Strategy-1. In our experiments, CrossNet-B was favorable only when task accuracy was critical, otherwise CrossNet-A was the more efficient model.

2) ENSEMBLE & SPLIT NETWORK PERFORMANCE

To test improvements by the concurrent layer mapping strategy, we mapped the ensemble of 4 CrossNet-A networks on a single chip. The ensemble can be mapped using 969 crossbars using algorithm IV-B. For comparison, a sequential mapping scheme would cost 1484 crossbars (34.7% lower storage requirement). Owing to the concurrent mapping, the storage requirement for 4 networks is only $2.33\times$ higher than a single network. We observe that occupancy ratio increases from 0.3767 to 0.8616 ($2.29\times$ higher). Processing multiple networks simultaneously increases the power requirement of chip, but reduced area (on account of higher occupancy ratio) improves the power efficiency by $2.75\times$. The computational efficiency for the ensemble is $1.79\times$ higher than a single model.

We further compare the ensemble's observations with a **TreeNet** [17] based architecture, with the network split point observed before *Convolution*64 layer. Essentially, the information flow is split and duplicated at this point, transforming into a multi-channel stream (for reference, the split is at the input layer in an ensemble). This design requires 537 crossbars for processing, with an occupancy ratio of 55.2%. Compared to an equivalent ensemble of 2

networks, power efficiency for this implementation is higher by 59.20%.

F. THEORETICAL REASONING FOR RESULTS

In the conventional load-store paradigm, a MAC operation costs 3 memory accesses (i.e., $O(3N)$ space complexity), and scales by $O(2N)$ time complexity. To process F different filters (F decomposes to $N_X \times N_y$ for convolution in x and y direction), the costs increase to $O(F \times 3N)$ and $O(F \times 2N)$ for space and time respectively.

Memristor-based PIM architecture reduces the space complexity to $O(F \times 2N)$ by storing the filter weights *in-memory*. The MAC operation is now performed in a single cycle (Figure 11 (a)), decreasing the time complexity to $O(F \times 2)$. Please note, the hidden constants in these run-times have increased considerably due to the addition of peripheral components associated with mixed-signal processing. However, the overall design still yields higher computation efficiency due to massive parallelism [15].

In this paper, *we propose optimizations that further reduce the space complexity to $O(2N)$, yielding higher energy efficiency due to reduced data movement.* For F filters, we obtain $1 - \frac{1}{F}$ normalized reduction in main memory accesses. Theoretically, a higher value of F inversely reduces energy costs, but due to the crossbar's physical limitations (write disturbance, wire parasitics, sneak current, etc.), practical improvements do not scale linearly [91], [129], [130].

VIII. CONCLUSION

This work represents the first step in the direction of understanding and quantifying the existing challenges in resistive crossbar-aware neural network design and optimization. We present the first pruning algorithms for CNNs on crossbars, improving crossbar amenability by 40%. We also explore various facets of optimizing neural network processing at scale while improving energy (2.75×) and area efficiency ($66 - 93.05\%$, network dependent), especially for ensemble systems. Further, the paper presents novel strategies to quickly design high-performance architectures without resource-intensive hyper-parameter sweep. Finally, the paper defines the CrossNet family, designed for maximizing crossbar-based performance (19.06× higher computational efficiency and 4.16× improved power efficiency).

ACKNOWLEDGMENT

(The authors acknowledge the TU Wien Bibliothek for Open Access Fee support through its Open Access Funding Program.)

REFERENCES

[1] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *J. Mach. Learn. Res.*, vol. 12 pp. 2493–2537, Aug. 2011.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[3] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-LM: Training multi-billion parameter language models using model parallelism," 2019, *arXiv:1909.08053*. [Online]. Available: http://arxiv.org/abs/1909.08053

[4] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec. 2017.

[5] J. Deng, "Imagenet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Dec. 2009, pp. 248–255.

[6] A. Boroumand, "Google workloads for consumer devices: Mitigating data movement bottlenecks," in *Proc. Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, 2018, pp. 316–331.

[7] M. Shafique, M. Naseer, T. Theocharides, C. Kyrkou, O. Mutlu, L. Orosa, and J. Choi, "Robust machine learning systems: Challenges,Current trends, perspectives, and the road ahead," *IEEE Des. Test. Comput.*, vol. 37, no. 2, pp. 30–57, Apr. 2020.

[8] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*. [Online]. Available: http://arxiv.org/abs/1409.1556

[9] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 1–9.

[10] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 367–379.

[11] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam, "DaDianNao: A machine-learning supercomputer," in *Proc. 47th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Dec. 2014, pp. 609–622.

[12] A. Parashar, "Scnn: An accelerator for compressed-sparse convolutional neural networks," *SIGARCH Comput. Archit. News*, vol. 45, no. 2, pp. 27–40, 2017.

[13] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "EIE: Efficient inference engine on compressed deep neural network," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 243–254.

[14] J. J. Yang, D. B. Strukov, and D. R. Stewart, "Memristive devices for computing," *Nature Nanotechnol.*, vol. 8, no. 13, pp. 13–24, Dec. 2012, doi: 10.1038%2Fnnano.2012.240.

[15] A. Shafiee, "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *ACM SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 14–26, 2016.

[16] S. Yu, P.-Y. Chen, Y. Cao, L. Xia, Y. Wang, and H. Wu, "Scaling-up resistive synaptic arrays for neuro-inspired architecture: Challenges and prospect," in *IEDM Tech. Dig.*, Dec. 2015, pp. 3–17.

[17] S. Lee, S. Purushwalkam, M. Cogswell, D. Crandall, and D. Batra, "Why m heads are better than one: Training a diverse ensemble of deep networks," 2015, *arXiv:1511.06314*. [Online]. Available: http://arxiv.org/abs/1511.06314

[18] F. Iandola, M. Moskewicz, S. Karayev, R. Girshick, T. Darrell, and K. Keutzer, "DenseNet: Implementing efficient ConvNet descriptor pyramids," 2014, *arXiv:1404.1869*. [Online]. Available: http://arxiv.org/abs/1404.1869

[19] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," 2015, *arXiv:1510.00149*. [Online]. Available: http://arxiv.org/abs/1510.00149

[20] O. Mutlu, S. Ghose, J. Gómez-Luna, and R. Ausavarungnirun, "Processing data where it makes sense: Enabling in-memory computation," *Microprocessors Microsyst.*, vol. 67, pp. 28–41, Jun. 2019.

[21] W. Chan, N. Jaitly, Q. Le, and O. Vinyals, "Listen, attend and spell: A neural network for large vocabulary conversational speech recognition," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Mar. 2016, pp. 4960–4964.

[22] N. P. Jouppi, "In-datacenter performance analysis of a tensor processing unit," in *Proc. ACM/IEEE 44th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2017, pp. 1–12.

[23] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018, *arXiv:1810.04805*. [Online]. Available: http://arxiv.org/abs/1810.04805

[24] T. B. Brown, "Language models are few-shot learners," 2020, *arXiv:2005.14165*. [Online]. Available: http://arxiv.org/abs/2005.14165

[25] J. Albericio, "Bit-pragmatic deep neural network computing," in *Proc. 50th Annu. Int. Symp. Microarchit.*, 2017, pp. 1–5, doi: 10.1145%2F3123939.3123982.

[26] A. Marchisio, M. A. Hanif, M. Martina, and M. Shafique, "PruNet: Class-blind pruning method for deep neural networks," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2018, pp. 1–8.

[27] B. Hassibi and D. G. Stork, "Second order derivatives for network pruning: Optimal brain surgeon," in *Proc. Adv. neural Inf. Process. Syst.*, 1993, pp. 164–171.

[28] Z. Yang, "Deep fried convnets," in *Proc. Int. Conf. Comput. Vis.*, Dec. 2015, pp. 1476–1483, doi: 10.1109%2Ficcv.2015.173.

[29] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," 2018, *arXiv:1810.05270*. [Online]. Available: http://arxiv.org/abs/1810.05270

[30] M. Zhu and S. Gupta, "To prune, or not to prune: Exploring the efficacy of pruning for model compression," 2017, *arXiv:1710.01878*. [Online]. Available: http://arxiv.org/abs/1710.01878

[31] S. Han, "Learning both weights and connections for efficient neural network," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 1135–1143.

[32] R. Yu, A. Li, C.-F. Chen, J.-H. Lai, V. I. Morariu, X. Han, M. Gao, C.-Y. Lin, and L. S. Davis, "NISP: Pruning networks using neuron importance score propagation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 9194–9203, doi: 10.1109/CVPR.2018.00958.

[33] J. Yu, A. Lukefahr, D. Palframan, G. Dasika, R. Das, and S. Mahlke, "Scalpel: Customizing DNN pruning to the underlying hardware parallelism," in *Proc. 44th Annu. Int. Symp. Comput. Archit.*, Jun. 2017, pp. 548–560.

[34] H. Mao, S. Han, J. Pool, W. Li, X. Liu, Y. Wang, and W. J. Dally, "Exploring the regularity of sparse structure in convolutional neural networks," 2017, *arXiv:1705.08922*. [Online]. Available: http://arxiv.org/abs/1705.08922

[35] S. Anwar, K. Hwang, and W. Sung, "Structured pruning of deep convolutional neural networks," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 13, no. 3, pp. 1–18, Feb. 2017, doi: 10.1145%2F3005348.

[36] S. Lin, "Toward compact convnets via structure-sparsity regularized filter pruning," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 2, pp. 574–588, Feb. 2019, doi: 10.1109%2Ftnnls.2019.2906563.

[37] J. Lin, "Runtime neural pruning," in *Proc. Adv. Neural Inf. Process. Syst.*, I. Guyon, Eds. Red Hook, NY, USA: Curran Associates, 2017, pp. 2181–2191. [Online]. Available: http://papers.nips.cc/paper/6813-runtime-neural-pruning.pdf

[38] Y. Gong, L. Liu, M. Yang, and L. Bourdev, "Compressing deep convolutional networks using vector quantization," 2014, *arXiv:1412.6115*. [Online]. Available: http://arxiv.org/abs/1412.6115

[39] H. D. Navone, P. F. Verdes, P. M. Granitto, and H. A. Ceccatto, "Selecting diverse members of neural network ensembles," in *Proc. 6th Brazilian Symp. Neural Netw.*, 1996, pp. 535–541.

[40] D. Ciresan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2012, pp. 3642–3649.

[41] F. Agostinelli, M. R. Anderson, and H. Lee, "Adaptive multi-column deep neural networks with application to robust image denoising," in *Proc. Adv. Neural Inf. Process. Syst.*, 2013, pp. 1493–1501.

[42] G. Brown, "Diversity in neural network ensembles," Ph.D. dissertation, School Comput. Sci., Univ. Birmingham, Birmingham, U.K., Jan. 2004.

[43] A. Lazarevic and Z. Obradovic, "Effective pruning of neural network classifier ensembles," in *Proc. Int. Joint Conf. Neural Netw. Process.*, 2001, pp. 796–801.

[44] M. Su and M. Basu, "Gating improves neural network performance," in *Proc. Int. Joint Conf. Neural Netw. Process.*, 2001, pp. 2159–2164.

[45] Z.-H. Zhou, Y. Jiang, Y.-B. Yang, and S.-F. Chen, "Lung cancer cell identification based on artificial neural network ensembles," *Artif. Intell. Med.*, vol. 24, no. 1, pp. 25–36, Jan. 2002.

[46] Z.-H. Zhou and Y. Jiang, "Medical diagnosis with c4.5 rule preceded by artificial neural network ensemble," *IEEE Trans. Inf. Technol. Biomed.*, vol. 7, no. 1, pp. 37–42, Mar. 2003.

[47] J. Jiang, P. Trundle, and J. Ren, "Medical image analysis with artificial neural networks," *Comput. Med. Imag. Graph.*, vol. 34, no. 8, pp. 617–631, Dec. 2010.

[48] Q. K. Al-Shayea, "Artificial neural networks in medical diagnosis," *Int. J. Comput. Sci.*, vol. 8, no. 2, pp. 150–154, 2011.

[49] P. Cunningham, J. Carney, and S. Jacob, "Stability problems with artificial neural networks and the ensemble solution," *Artif. Intell. Med.*, vol. 20, no. 3, pp. 217–225, Nov. 2000.

[50] O. Mutlu, "Memory scaling: A systems architecture perspective," in *Proc. 5th IEEE Int. Memory Workshop*, May 2013, pp. 21–25.

[51] B. C. Lee, "Architecting phase change memory as a scalable dram alternative," in *Proc. 36th Annu. Int. Symp. Comput. Archit.*, 2009, pp. 1–5, doi: 10.1145%2F1555754.1555758.

[52] C. W. Smullen, V. Mohan, A. Nigam, S. Gurumurthi, and M. R. Stan, "Relaxing non-volatility for fast and energy-efficient STT-RAM caches," in *Proc. IEEE 17th Int. Symp. High Perform. Comput. Archit.*, Feb. 2011, pp. 50–61, doi: 10.1109/HPCA.2011.5749716.

[53] G. W. Burr, R. M. Shelby, S. Sidler, C. Nolfo, J. Jang, I. Boybat, R. S. Shenoy, P. Narayanan, K. Virwani, E. U. Giacometti, B. N. Kurdi, and H. Hwang, "Experimental demonstration and tolerancing of a large-scale neural network (165 000 synapses) using phase-change memory as the synaptic weight element," *IEEE Trans. Electron Devices*, vol. 62, no. 11, pp. 3498–3507, Nov. 2015, doi: 10.1109%2Fted.2015.2439635.

[54] M.-J. Lee, "A fast, high-endurance and scalable non-volatile memory device made from asymmetric ta 2 o 5- x/tao 2- x bilayer structures," *Nature Mater.*, vol. 10, no. 8, p. 625, 2011.

[55] H. Dong Lee, "Integration of 4F2 selector-less crossbar array 2Mb ReRAM based on transition metal oxides for high density memory applications," in *Proc. Symp. VLSI Technol. (VLSIT)*, Jun. 2012, pp. 151–152, doi: 10.1109/VLSIT.2012.6242506.

[56] H. Y. Lee, Y. S. Chen, P. S. Chen, P. Y. Gu, Y. Y. Hsu, S. M. Wang, W. H. Liu, C. H. Tsai, S. S. Sheu, P. C. Chiang, W. P. Lin, C. H. Lin, W. S. Chen, F. T. Chen, C. H. Lien, and M.-J. Tsai, "Evidence and solution of over-RESET problem for HfOX based resistive memory with sub-ns switching speed and high endurance," in *Proc. Int. Electron Devices Meeting*, Dec. 2010, pp. 19.7.1–19.7.4, doi: 10.1109/IEDM.2010.5703395.

[57] L. Gao, P.-Y. Chen, and S. Yu, "Demonstration of convolution kernel operation on resistive cross-point array," *IEEE Electron Device Lett.*, vol. 37, no. 7, pp. 870–873, Jul. 2016.

[58] B. Govoreanu, "10×00D7;10nm2 Hf/HfOx crossbar resistive RAM with excellent performance, reliability and low-energy operation," in *IEDM Tech. Dig.*, Dec. 2011, pp. 6–31.

[59] M. Hu, "Dot-product engine for neuromorphic computing," in *Proc. 53rd Annu. Des. Autom. Conf.*, 2016, pp. 1–6, doi: 10.1145%2F2897937.2898010.

[60] T.-Y. Liu, "A 130.7 mm^2 2-layer 32-gb reram memory device in 24-nm technology," *IEEE J. Solid-State Circuits*, vol. 49, no. 1, pp. 140–153, Aug. 2013.

[61] A. Kawahara, R. Azuma, Y. Ikeda, K. Kawai, Y. Katoh, Y. Hayakawa, K. Tsuji, S. Yoneda, A. Himeno, K. Shimakawa, T. Takagi, T. Mikawa, and K. Aono, "An 8 mb multi-layered cross-point ReRAM macro with 443 MB/s write throughput," *IEEE J. Solid-State Circuits*, vol. 48, no. 1, pp. 178–185, Jan. 2013.

[62] R. Fackenthal, M. Kitagawa, W. Otsuka, K. Prall, D. Mills, K. Tsutsui, J. Javanifard, K. Tedrow, T. Tsushima, Y. Shibahara, and G. Hush, "19.7 a 16Gb ReRAM with 200MB/s write and 1GB/s read in 27nm technology," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2014, pp. 338–339.

[63] S. Yu, Y. Wu, and H.-S. P. Wong, "Investigating the switching dynamics and multilevel capability of bipolar metal oxide resistive switching memory," *Appl. Phys. Lett.*, vol. 98, no. 10, Mar. 2011, Art. no. 103514, doi: 10.1063%2F1.3564883.

[64] M. Abdullah Hanif, R. Vidya Wicaksana Putra, M. Tanvir, R. Hafi, S. Rehman, and M. Shafique, "MPNA: A massively-parallel neural array accelerator with dataflow optimization for convolutional neural networks," 2018, *arXiv:1810.12910*. [Online]. Available: http://arxiv.org/abs/1810.12910

[65] M. A. Hanif, F. Khalid, and M. Shafique, "CANN: Curable approximations for high-performance deep neural network accelerators," in *Proc. 56th Annu. Des. Autom. Conf.*, Jun. 2019, pp. 1–6.

[66] B. Reagen, "Minerva," *ACM SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 267–278, Jun. 2016, doi: 10.1145%2F3007787.3001165.

[67] C. Gao, "DeltaRNN: A power-efficient recurrent neural network accelerator," in *Proc. Int. Symp. Field-Program. Gate Arrays*, 2018, pp. 21–30, doi: 10.1145%2F3174243.3174261.

[68] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H.-J. Yoo, "UNPU: A 50.6TOPS/W unified deep neural network accelerator with 1b-to-16b fully-variable weight bit-precision," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2018, pp. 218–220, doi: 10.1109/ISSCC.2018.8310262.

[69] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos, "Cnvlutin: Ineffectual-neuron-free deep neural network computing," *ACM SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 1–13, Oct. 2016.
[70] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "PRIME: A novel Processing-in-Memory architecture for neural network computation in ReRAM-based main memory," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 27–39.
[71] L. Song, X. Qian, H. Li, and Y. Chen, "PipeLayer: A pipelined ReRAM-based accelerator for deep learning," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2017, pp. 541–552.
[72] M. Cheng, "Time: A training-in-memory architecture for memristor-based deep neural networks," in *Proc. 54th ACM/EDAC/IEEE Des. Automat. Conf.*, Jun. 2017, pp. 1–6.
[73] H. Ji, L. Song, L. Jiang, H. Li, and Y. Chen, "ReCom: An efficient resistive accelerator for compressed deep neural networks," in *Proc. Des., Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2018, pp. 237–240.
[74] P. Wang, Y. Ji, C. Hong, Y. Lyu, D. Wang, and Y. Xie, "SNrram: An efficient sparse neural network computation architecture based on resistive random-access memory," in *Proc. 55th ACM/ESDA/IEEE Des. Autom. Conf. (DAC)*, Jun. 2018, p. 106.
[75] A. Ankit, A. Sengupta, and K. Roy, "TraNNsformer: Neural network transformation for memristive crossbar based neuromorphic system design," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des. (ICCAD)*, Nov. 2017, pp. 533–540.
[76] X. Liu, Q. Wu, J. Yang, M. Mao, B. Liu, H. Li, Y. Chen, B. Li, Y. Wang, H. Jiang, and M. Barnell, "RENO: A high-efficient reconfigurable neuromorphic computing accelerator design," in *Proc. 52nd Annu. Des. Autom. Conf.*, 2015, pp. 1–6.
[77] L. Xia, "Technological exploration of RRAM crossbar array for matrix-vector multiplication," *J. Comput. Sci. Technol.*, vol. 31, no. 1, pp. 3–19, Jan. 2016, doi: 10.1007%2Fs11390-016-1608-8.
[78] M. S. Tarkov, "Mapping weight matrix of a neural network's layer onto memristor crossbar," *Opt. Memory Neural Netw.*, vol. 24, no. 2, pp. 109–115, Apr. 2015, doi: 10.3103%2Fs1060992x15020125.
[79] D. Teney, P. Anderson, X. He, and A. V. D. Hengel, "Tips and tricks for visual question answering: Learnings from the 2017 challenge," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 4223–4232.
[80] L. N. Smith and N. Topin, "Deep convolutional neural network design patterns," 2016, *arXiv:1611.00847*. [Online]. Available: http://arxiv.org/abs/1611.00847
[81] W. Wen, "Learning structured sparsity in deep neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 2074–2082.
[82] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, Apr. 2017.
[83] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 2736–2744.
[84] L. Kull, T. Toifl, M. Schmatz, P. Andrea Francese, C. Menolfi, M. Brändli, M. Kossel, T. Morf, T. Meyer Andersen, and Y. Leblebici, "A 3.1 mW 8b 1.2 GS/s single-channel asynchronous SAR ADC with alternate comparators for enhanced speed in 32 nm digital SOI CMOS," *IEEE J. Solid-State Circuits*, vol. 48, no. 12, pp. 3049–3058, Dec. 2013, doi: 10.1109/JSSC.2013.2279571.
[85] B. Murmann. (2016). *Adc Performance Survey 1997-2016*. [Online]. Available: http://www.stanford.edu/murmann/adcsurvey.html
[86] J.-H. Luo, J. Wu, and W. Lin, "ThiNet: A filter level pruning method for deep neural network compression," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 5068–5076, doi: 10.1109/ICCV.2017.541.
[87] V. Lebedev and V. Lempitsky, "Fast ConvNets using group-wise brain damage," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 2554–2564.
[88] J. Wang, C. Xu, X. Yang, and J. M. Zurada, "A novel pruning algorithm for smoothing feedforward neural networks based on group lasso method," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 5, pp. 2012–2024, 2018.
[89] T. Ochiai, S. Matsuda, H. Watanabe, and S. Katagiri, "Automatic node selection for deep neural networks using group lasso regularization," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Mar. 2017, pp. 5485–5489.
[90] C. D. Gilbert and T. N. Wiesel, "Receptive field dynamics in adult primary visual cortex," *Nature*, vol. 356, no. 6365, p. 150, 1992.
[91] C. Xu, D. Niu, N. Muralimanohar, R. Balasubramonian, T. Zhang, S. Yu, and Y. Xie, "Overcoming the challenges of crossbar resistive memory architectures," in *Proc. IEEE 21st Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2015, pp. 476–488.
[92] F. Alibart, L. Gao, B. D. Hoskins, and D. B. Strukov, "High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm," *Nanotechnology*, vol. 23, no. 7, pp. 075201-1–075201-20, 2012, doi: 10.1088%2F0957-4484%2F23%2F7%2F075201.
[93] M. Hu, "Hardware realization of BSB recall function using memristor crossbar arrays," in *Proc. 49th Annu. Des. Autom. Conf.*, 2012, pp. 498–503, doi: 10.1145%2F2228360.2228448.
[94] D. Niu, "Design trade-offs for high density cross-point resistive memory," in *Proc. Int. Symp. Low Power Electron. Des.*, 2012, pp. 209–214, doi: 10.1145%2F2333660.2333712.
[95] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient DNNs," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 1379–1387.
[96] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang, "Network trimming: A data-driven neuron pruning approach towards efficient deep architectures," 2016, *arXiv:1607.03250*. [Online]. Available: http://arxiv.org/abs/1607.03250
[97] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 6848–6856.
[98] G. Huang, S. Liu, L. V. D. Maaten, and K. Q. Weinberger, "CondenseNet: An efficient DenseNet using learned group convolutions," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 2752–2761.
[99] A. Gholami, K. Kwon, B. Wu, Z. Tai, X. Yue, P. Jin, S. Zhao, and K. Keutzer, "SqueezeNext: hardware-aware neural network design," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jun. 2018, pp. 1638–1647.
[100] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*. [Online]. Available: http://arxiv.org/abs/1704.04861
[101] N. Ma, "Shufflenet V2: Practical guidelines for efficient CNN architecture design," in *The Eur. Conf. Comput. Vis. (ECCV)*, Sep. 2018, pp. 116–131.
[102] J. Redmon. (2016). *Darknet: Open Source Neural Networks in C*. [Online]. Available: http://pjreddie.com/darknet/
[103] C. Szegedy, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Proc. AAAI Conf. Artif. Intell.*, 2017, pp. 1–5
[104] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 1251–1258.
[105] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 2818–2826.
[106] Y. Ioannou, D. Robertson, R. Cipolla, and A. Criminisi, "Deep roots: Improving CNN efficiency with hierarchical filter groups," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 1231–1240.
[107] S. Xie, R. Girshick, P. Dollar, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 1492–1500.
[108] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size," 2016, *arXiv:1602.07360*. [Online]. Available: http://arxiv.org/abs/1602.07360
[109] M. Lin, Q. Chen, and S. Yan, "Network in network," 2013, *arXiv:1312.4400*. [Online]. Available: http://arxiv.org/abs/1312.4400
[110] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "MnasNet: Platform-aware neural architecture search for mobile," 2018, *arXiv:1807.11626*. [Online]. Available: http://arxiv.org/abs/1807.11626
[111] Y. He, "Amc: Automl for model compression and acceleration on mobile devices," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 784–800.
[112] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013, *arXiv:1312.5602*. [Online]. Available: http://arxiv.org/abs/1312.5602
[113] H. Cai, L. Zhu, and S. Han, "ProxylessNAS: Direct neural architecture search on target task and hardware," 2018, *arXiv:1812.00332*. [Online]. Available: http://arxiv.org/abs/1812.00332

[114] S. Liu, Z. Du, J. Tao, D. Han, T. Luo, Y. Xie, Y. Chen, and T. Chen, "Cambricon: An instruction set architecture for neural networks," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 393–405.
[115] A. Ankit, I. E. Hajj, S. R. Chalamalasetti, G. Ndu, M. Foltin, R. S. Williams, P. Faraboschi, W.-M.-W. Hwu, J. P. Strachan, K. Roy, and D. S. Milojicic, "PUMA: A programmable ultra-efficient memristor-based accelerator for machine learning inference," in *Proc. Int. Conf. Archit. Support for Program. Lang. Oper. Syst.*, Apr. 2019, pp. 715–731.
[116] Y. Ji, Y. Zhang, W. Chen, and Y. Xie, "Bridge the gap between neural networks and neuromorphic hardware with a neural network compiler," in *Proc. 23rd Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, Mar. 2018, pp. 448–460.
[117] P. Yao, H. Wu, B. Gao, J. Tang, Q. Zhang, W. Zhang, J. J. Yang, and H. Qian, "Fully hardware-implemented memristor convolutional neural network," *Nature*, vol. 577, no. 7792, pp. 641–646, Jan. 2020.
[118] A. Kumar, M. Das, V. Garg, B. S. Sengar, M. T. Htay, S. Kumar, A. Kranti, and S. Mukherjee, "Forming-free high-endurance Al/ZnO/Al memristor fabricated by dual ion beam sputtering," *Appl. Phys. Lett.*, vol. 110, no. 25, Jun. 2017, Art. no. 253509.
[119] B. J. Murdoch, D. G. McCulloch, R. Ganesan, D. R. McKenzie, M. M. M. Bilek, and J. G. Partridge, "Memristor and selector devices fabricated from HfO_{2-x} Nx," *Appl. Phys. Lett.*, vol. 108, no. 14, Apr. 2016, Art. no. 143504.
[120] F. Budiman, "Recent progress on fabrication of memristor and transistor-based neuromorphic devices for high signal processing speed with low power consumption," *Jpn. J. Appl. Phys.*, vol. 57, nos. 2–3, 2018, Art. no. 03EA06.
[121] S. Pi, "Memristor crossbar arrays with 6-nm half-pitch and 2-nm critical dimension," *Nature Nanotechnol.*, vol. 14, no. 1, p. 35, 2019.
[122] H. Bao, N. Wang, H. Wu, Z. Song, and B. Bao, "Bi-stability in an improved memristor-based third-order wien-bridge oscillator," *IETE Tech. Rev.*, vol. 36, no. 2, pp. 109–116, Mar. 2019.
[123] Z. Wang, "Reinforcement learning with analogue memristor arrays," *Nature Electron.*, vol. 2, no. 3, p. 115, 2019.
[124] D. Zhang, J. Yang, D. Ye, and G. Hua, "Lq-nets: Learned quantization for highly accurate and compact deep neural networks," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 365–382.
[125] P. Wang, Q. Hu, Y. Zhang, C. Zhang, Y. Liu, and J. Cheng, "Two-step quantization for low-bit neural networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 4376–4384.
[126] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized convolutional neural networks for mobile devices," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 4820–4828.
[127] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.
[128] Y. Wu and K. He, "Group normalization," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, Sep. 2018, pp. 3–19.
[129] S. Yu and H.-S.-P. Wong, "A phenomenological model for the reset mechanism of metal oxide RRAM," *IEEE Electron Device Lett.*, vol. 31, no. 12, pp. 1455–1457, Dec. 2010.
[130] S. Choi, Y. Yang, and W. Lu, "Random telegraph noise and resistance switching analysis of oxide based resistive memory," *Nanoscale*, vol. 6, no. 1, pp. 400–404, May 2014, doi: 10.1039%2Fc3nr05016e.

MUHAMMAD ABDULLAH HANIF (Graduate Student Member, IEEE) received the B.Sc. degree in electronic engineering from Ghulam Ishaq Khan Institute of Engineering Sciences and Technology (GIKI), Pakistan, and the M.Sc. degree in electrical engineering with a specialization in digital systems and signal processing (DSSP) from the School of Electrical Engineering and Computer Science (SEECS), National University of Sciences and Technology (NUST), Islamabad, Pakistan. He is currently pursuing the Ph.D. degree with the Faculty of Informatics, Technische Universität Wien (TU Wien), Vienna, Austria. In the past, he worked as a Research Associate with the Vision Processing (VISpro) Lab, Information Technology University (ITU), Pakistan, and as a Lab Engineer with GIKI. He was a recipient of President's Gold Medal for his outstanding academic performance during his M.S. degree.

ADITYA MANGLIK (Student Member, IEEE) received the B.E. degree (Hons.) in electrical and electronics engineering from the Birla Institute of Technology & Sciences, Pilani, India. His research interests include design and critical analysis of high-performance computing (HPC) systems, focusing on exascale computation platforms, and next-generation workloads, such as ubiquitous machine learning and bio-informatics.

MUHAMMAD SHAFIQUE (Senior Member, IEEE) received the Ph.D. degree in computer science from the Karlsruhe Institute of Technology (KIT), Germany, in 2011.

Afterwards, he established and led a highly recognized research group at KIT for several years as well as conducted impactful research and development activities in Pakistan. In October 2016, he joined the Institute of Computer Engineering, Faculty of Informatics, Technische Universität Wien (TU Wien), Vienna, Austria, as a Full Professor of Computer Architecture and Robust, Energy-Efficient Technologies. Since September 2020, he has been with the Division of Engineering, New York University Abu Dhabi (NYUAD), United Arab Emirates. He is currently a Global Network Faculty with the NYU Tandon School of Engineering, USA. His research interests include brain-inspired computing, AI & machine learning hardware and system-level design, energy-efficient systems, robust computing, hardware security, emerging technologies, FPGAs, MPSoCs, and embedded systems. His research interests also include cross-layer analysis, modeling, design, and optimization of computing and memory systems. The researched technologies and tools are deployed in application use cases from the Internet-of-Things (IoT), smart cyber-physical systems (CPS), and ICT for development (ICT4D) domains. He has given several keynotes, invited talks, and tutorials, as well as organized many special sessions at premier venues. He has served as the PC chair, a track chair, and a PC member for several prestigious IEEE/ACM conferences. He holds one U.S. patent has (co)authored six books, more than book chapters, and over 200 articles in premier journals and conferences. He received the 2015 ACM/SIGDA Outstanding New Faculty Award, the AI 2000 Chip Technology Most Influential Scholar Award in 2020, six gold medals, and several best paper awards and nominations at prestigious conferences.