CompAct: On-chip Compression of Activations for Low Power Systolic Array Based CNN Acceleration

JEFF (JUN) ZHANG and PARUL RAJ, New York University
SHUAYB ZARAR and AMOL AMBARDEKAR, Microsoft
SIDDHARTH GARG, New York University

This paper addresses the design of systolic array (SA) based convolutional neural network (CNN) accelerators for mobile and embedded domains. On- and off-chip memory accesses to the large activation inputs (sometimes called feature maps) of CNN layers contribute significantly to total energy consumption for such accelerators; while prior has proposed off-chip compression, activations are still stored on-chip in uncompressed form, requiring either large on-chip activation buffers or slow and energy-hungry off-chip accesses. In this paper, we propose CompAct, a new architecture that enables *on-chip* compression of activations for SA based CNN accelerators. CompAct is built around several key ideas. First, CompAct identifies an SA schedule that has nearly regular access patterns, enabling the use of a modified run-length coding scheme (RLC). Second, CompAct improves compression ratio of the RLC scheme using Sparse-RLC in later CNN layers and Lossy-RLC in earlier layers. Finally, CompAct proposes look-ahead snoozing that operates synergistically with RLC to reduce the leakage energy of activation buffers. Based on detailed synthesis results, we show that CompAct enables up to 62% reduction in activation buffer energy, and 34% reduction in total chip energy.

CCS Concepts: • **Computer systems organization** → **Systolic arrays**; **Neural networks**; **Data flow architectures**; **Embedded hardware**; • **Hardware** → **Chip-level power issues**;

Additional Key Words and Phrases: Deep neural networks, systolic arrays, low-power design

ACM Reference format:
Jeff (Jun) Zhang, Parul Raj, Shuayb Zarar, Amol Ambardekar, and Siddharth Garg. 2019. CompAct: On-chip Compression of Activations for Low Power Systolic Array Based CNN Acceleration. *ACM Trans. Embed. Comput. Syst.* 18, 5s, Article 47 (October 2019), 24 pages.
https://doi.org/10.1145/3358178

1 INTRODUCTION

Deep neural networks (DNN) provide best-in-class accuracy for a range of machine learning tasks including text, speech, image and video [18, 21, 32]. DNNs are composed of multiple layers of computation. Each layer performs a linear transformation on its inputs followed by a non-linear activation such as a sigmoid or rectified linear unit (ReLU). The outputs of each layer, or equivalently

This article appears as part of the ESWEEK-TECS special issue and was presented in the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES) 2019.

Authors' addresses: J. (Jun) Zhang, P. Raj, and S. Garg, New York University, 370 Jay Street, Brooklyn, NY, 11201; emails: {jeffjunzhang, pr1498, sg175}@nyu.edu; S. Zarar and A. Ambardekar, Microsoft, 1 Microsoft Way, Redmond, WA, 98052; emails: {shuayb, amolamb}@microsoft.com.

1539-9087/2019/10-ART47 $15.00
https://doi.org/10.1145/3358178

the inputs of each subsequent layer, are referred to as *activations*. DNNs that use convolutions as linear transforms are referred to as convolutional neural networks (CNNs). Modern CNN architectures can have tens to hundreds of layers. Each layer is composed of multiple filters whose parameters (or *weights*) are learnt during a training process that is based on gradient descent.

State-of-the-art CNNs perform billions of operations on large matrices or tensors representing the CNN's activations and weights, thus resulting in high computational and memory footprint. To mitigate these issues, there is growing interest in the design of hardware accelerators to speed-up CNN inference from both industry and academia [4, 5, 9, 17, 27, 30].[1] Among the many accelerator architectures that have been proposed, *systolic array* (SA) based CNN acceleration is one of the frequently used architectural solutions; examples from industry include the Google Tensor Processing Unit (TPU) [17] for server applications and ARM's Trillium [6] targeting the embedded/mobile domain.

SAs are tightly coupled 2-D grids of multiply-and-accumulate (MAC) units that can perform matrix multiplications using only nearest neighbour communication. Activations and weights are streamed through the array in a precisely synchronized manner, such that each activation gets multiplied by its corresponding weight(s) and the resulting partial product(s) are appropriately summed to produce an output. Compared to competing architectural solutions, SAs (i) obviate the need for complex on-chip routing and buffering; and, importantly (ii) amortize the energy costs of reading weights and activations from memory over multiple MAC operations.

Paper Motivation. Our motivation is the observation that for low-power SA accelerators (targeted towards embedded/mobile applications), the energy cost of accessing memory is a significant fraction of total energy consumption. This is because memory accesses are amortized over fewer MAC operations; for instance, a 64×64 SA similar to the ARM Trillium has 4× fewer MAC operations/access compared to the 256×256 Google TPU. Indeed, as observed in Section 4.2 memory accesses contribute up to 65% of total energy for a 64×64 SA.

Memory access energy comes from two sources: energy of on-chip SRAM accesses and off-chip DRAM accesses. Architectures like the TPU (and others [11]) choose to store all activations in on-chip SRAMs; the goal is to minimize the performance and energy penalty of costly off-chip DRAM accesses. In these architectures, the outputs of each layer are written back into the on-chip activation buffer instead of being written off-chip and loaded again from DRAM when the next layer is executed. However, state-of-the-art CNNs have large activations (3.2MB for the largest conv layer of VGGNet [33]) requiring large activation buffers; the TPU, for example, has a 24MB activation buffer.[2]

An alternative is to use a smaller activation buffer, but at the expense of slower, energy hungry DRAM accesses; architectures like Eyeriss adopt this strategy [5]. To reduce DRAM energy cost and bandwidth, Eyeriss [5] uses run length encoding (RLC), a technique that compresses contiguous sequences of the same value to one value and the number of repeated repetitions ("runs"), to compress/decompress data to/from DRAM. Nonetheless frequent DRAM accesses impose an energy and performance penalty.

Paper Contributions. In this paper, we propose a new SA based CNN accelerator, CompAct, that seeks to achieve the best of both worlds: like the TPU architecture, CompAct seeks to store activations on chip (as much possible), but using only small on-chip activation buffers. CompAct achieves this using on-chip $\underline{\text{com}}$pression of $\underline{\text{act}}$ivations, i.e., activations are stored in compressed

[1]Note that accelerators for CNN training have also been designed, but this paper focuses specifically on CNN inference accelerators.
[2]Although the designers note that this buffer is over-provisioned.

format even in the on-chip activation buffers. All previous SA based architectures of which we are aware store activations in uncompressed form on chip. On-chip compression of activations for SAs is challenging for two reasons. (1) SAs are *precisely synchronized*; a new activation must be fed to each row of the systolic array in every clock cycle. Any read stalls cause the *entire* SA to stall, resulting in large performance penalty. This obviates the use of complex variable length coding schemes. (2) Read access patterns from the activation buffer are typically complex (or scattered) (at least for the convolutional layers), obviating naive streaming compression schemes like RLC. To the best of our knowledge, CompAct is the *first SA based CNN accelerator that uses on-chip compression for activations.* CompAct is built around several key ideas.

- **RLC-Aware Scheduling.** CompAct schedules the execution of convolutional layers such that read accesses from the activation buffer are *nearly* regular (as described in Section 3.1), and proposes a modified RLC scheme (described in Section 3.2) that enables on-chip compression of activations.
- **RLC Optimizations.** We propose two additional optimizations, described in Section 3.3, to the modified RLC scheme that further increase the compression ratio. The first optimization uses *Sparse-RLC*, a variant of RLC that only compresses runs of zeros, for later layers in the CNN that are typically highly sparse. The second optimization is a novel *Lossy-RLC* scheme that provides even greater compression ratios at the expense of limited loss in classification accuracy. An energy-aware greedy algorithm (see Algorithm 2) determines the optimal (lossy) compression ratio for each layer.
- **Look-Ahead Snoozing.** CompAct proposes look-ahead snoozing (LAS) (Section 3.4), a leakage energy reduction technique based on the observation that RLC allows exact identification of when the activation buffer will be accessed next; thus the buffer can be proactively placed in drowsy (or snooze) mode and woken-up in advance, yielding significant reductions in leakage power.

Based on detailed simulations and synthesis results, we show that when employed synergistically, the proposed techniques reduce activation buffer energy by up to 62% and yield up to 34% reduction in total chip energy for the AlexNet and VGG-16 CNN benchmarks.

2 BACKGROUND

In this section, we describe CNN inference mathematically and show how SAs can be used to accelerate it.

2.1 CNN Inference

A CNN consists of L stacked layers of computation; typically the first several layers are convolutional while the last few layers are fully-connected. The input of a convolutional layer l is a *tensor* of activations $A^l \in \mathbb{R}^{N_l \times N_l \times C_l}$ that is convolved with a tensor of weights $W^l \in \mathbb{R}^{F_l \times F_l \times C_l \times C_{l+1}}$ to yield an output tensor $Y^{l+1} \in \mathbb{R}^{N_{l+1} \times N_{l+1} \times C_{l+1}}$. The weight tensor can be thought of as C_{l+1} 3-D *filters* of dimensionality $F_l \times F_l \times C_l$. The convolution operation can be written as:

$$y^{l+1}_{i,j,k} = \sum_{p=0}^{F_l-1} \sum_{q=0}^{F_l-1} \sum_{r=0}^{C_l-1} a^l_{i+p,j+q,r} \times w^l_{p,q,r,k}. \tag{1}$$

Essentially, as shown in Figure 1, each filter strides across the activation tensor, producing one element of the output tensor for each stride and yielding one channel of the output tensor. (For simplicity, this discussion has assumed a stride of 1; in general, the filter in Equation 1 moves across the input tensor with a stride of S.) Y^{l+1} is then passed through an element-wise

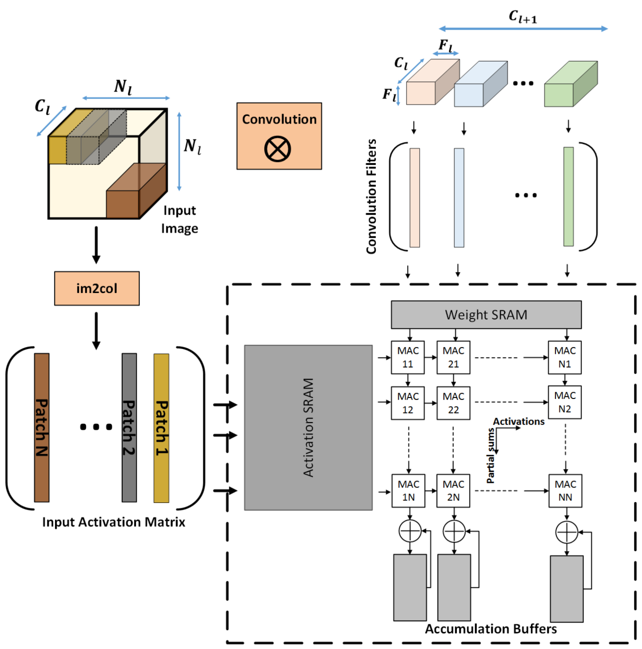


Fig. 1. Illustration of a CNN's convolutional layer and representation as a matrix multiplication using an `im2col` transformation. Also shown is the architecture of a baseline weight-stationary SA based on the TPU.

non-linearity, $\phi(.)$, for example, the ReLU or a sigmoid function, to yield activations of the next layer $A^{l+1} \in \mathbb{R}^{N_{l+1} \times N_{l+1} \times C_{l+1}}$, *i.e.*, $a^{l+1}_{i,j,k} = \phi(y^{l+1}_{i,j,k})$. The computational cost of a layer l's convolution is $O(N^2_{l+1} F^2_l C_l C_{l+1})$; typically, N_{l+1}, C_l and C_{l+1} are large, resulting in high computational costs.

Convolutions as matrix multiplication. Convolutions can also be expressed as matrix multiplication by using `im2col` transform, as shown in Figure 1. Each row of the 2-D weight matrix, W^{2D} (layer index dropped for simplicity), corresponds to one of the C_{l+1} filters, that is completely unrolled into a row vector. Correspondingly, each column of the 2-D activation matrix, A^{2D}, contains a *patch* of the input activation tensor. A patch is a 3-D volume of the same dimensions as a filter unrolled into a column vector. Each row-column vector dot product outputs one element of the output tensor.

The `im2col` transform is commonly used in software libraries for CNN inference and training [34, 37] since it enables the use of highly optimized numerical libraries for matrix multiplication. Similarly, since SAs are highly efficient for matrix multiplications, prior work [12, 22] has advocated for its use in hardware acceleration as well. This strawman strategy is described next, followed by a discussion of its shortcomings.

2.2 SA-Based Acceleration of CNNs

In this section we describe how an SA can be used to accelerate matrix multiplication; specifically, to multiply 2-D weight matrix W^{2D}, with 2-D activation matrix A^{2D}.

Figure 1 shows a block-diagram of an SA modeled on the TPU. The core of the SA is a grid of MAC units, that only connect to their neighbors. The SA also has three SRAMs that provide inputs to the SA core and store outputs. These are: (1) **activation buffer**: stores elements of A^{2D}, each column of A^{2D} maps to a column of the activation buffer; (2) **weight buffer**: stores elements of W^{2D}, each row of W^{2D} maps to a column of the weight buffer; (3) **accumulation buffer**: stores the outputs of matrix multiplication. Note that if the dimensions of A^{2D} and/or W^{2D} are large relative to the SA core, outputs are computed over several rounds or phases; in this case, the accumulation buffer stores and accumulates partial results from each round [22].

However, for now, assume that the SA is large enough to hold the entire weight matrix, and that the activation matrix can be fully stored in the activation SRAM. We now describe how $W^{2D} \times A^{2D}$ can be accomplished using a *weight stationary* SA like the TPU. In a weight stationary architecture, the weight matrix is first loaded into the SA's MAC units and remain stationary throughout a round of computation; Activations are then streamed through the SA, starting with the first SRAM row. The second row lags behind the first by one clock cycle, and the third behind the second by one clock cycle and so on. As we will see, this precise synchronization in which each row lags behind the preceding one by exactly one clock cycle enables the activations to arrive at each MAC at just the right time. Any stall or change in the schedule will result in incorrect computations.

In the first clock cycle, a_{11}^{2D} and w_{11}^{2D} are multiplied and the partial product is passed downwards. $a_{12}^{2D} \times w_{12}^{2D}$ is added to the partial product in the next clock and so on until the first element of the output matrix is computed by the first column of the SA. Thus, the first column of SA computes all elements of the first row of the output matrix. In the meantime, the second column of SA receives activations from the first column and computes the second row of the output matrix, and so on. The outputs of a layer are stored in the accumulation buffer. These outputs are passed through a pooling and ReLU activation (not pictured in Figure 1) and fed back into the activation buffer from where the next layer's processing proceeds.

Drawbacks of `im2col` *implementation.* On the one hand, the baseline `im2col` implementation described above is conceptually simple. Nonetheless, it has a major drawback as it relates to our goal of reducing the area and energy costs of the on-chip activation buffer: the size of the 2-D activation matrix A^{2D} is F^2 times *larger* than the raw 3-D activation tensor A. Assuming even small $F = 3$ filter size, this results in 9× blow-up in the size of the accumulation buffer (assuming, as the TPU does, that all activations stay on chip).

One can argue that the memory blow-up of `im2col` can be contained somewhat using compression. Indeed, since activations are fetched from the activation buffer in streaming fashion for the `im2col` implementation, columns of A^{2D} can be compressed using simple RLC. However, in our experiments, the *best-case* compression ratio we achieved was 8.5×, which implies that even *after* compression, the `im2col` implementation results in a net increase in memory requirement. As described next, CompAct adopts a different strategy: activations are stored in the activation buffer in raw 3-D tensor form, and compressed to significantly reduce memory requirements.

3 COMPACT DESIGN

We now describe in detail the new architectural innovations that we propose as parts of CompAct.

3.1 Compression-Aware Scheduling

As we noted in Section 2, a naive `im2col` mapping results in memory blow-up; instead, our starting point is to explore mapping/scheduling strategies in which activations are stored in raw tensor format; however, as we will see, doing so might introduce irregular read access patterns from the activation buffer.

We start with a mapping, as shown in Figure 2, in which each input channel maps to a different row of the activation buffer, and each filter maps to a different column. Looking at the first column, we note that weights from "pipe" spanning all three channels of the first filter are loaded into the column; a corresponding pipe of activations, also spanning all three channels, is streamed in from the activation buffer in a synchronized manner such that the column computes the dot product of

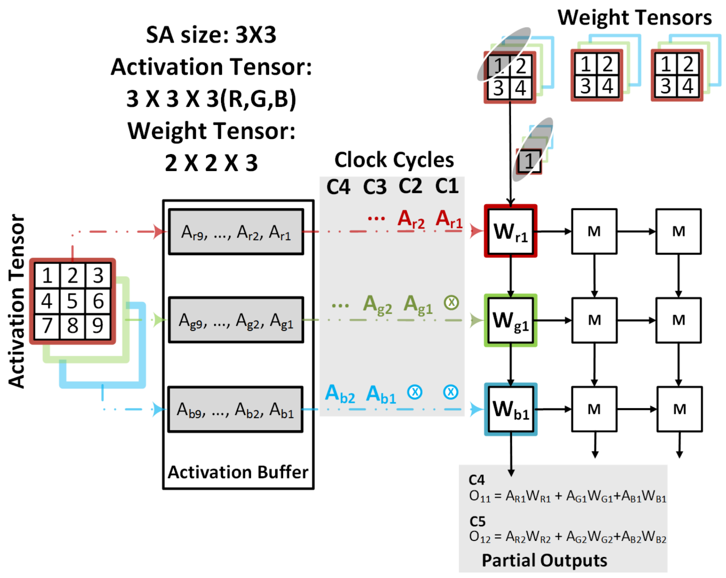


Fig. 2. Baseline mapping strategy used in compact.

the two pipes. The dot product is a partial sum for the first element of the output activation tensor and stored in the accumulation buffer (not pictured). This mapping strategy has been used in prior work [40, 41] and, from publicly available information [29], is also used in the TPU.

Now consider how a convolution operation is scheduled given this mapping. Before proceedings, we note that each column of the accumulation buffer holds data from a different output channel. However, we note that the size of the accumulation buffer is typically limited since it holds data at higher resolution than activations and weights.

Consider, without loss of generality, the first column of the SA. We will also assume, for now, that the number of input channels equals the number of SA rows. The first step loads a *pipe* of $1 \times 1 \times C_l$ weights from the first filter into the SA column. Activations are now streamed into the SA as per one of two schedules, both described in [29].

(1) *Rasterized schedule:* as shown in Figure 3(a), that generates a $Q \times Q$ patch of the first output channel by repeating the schedule for each pipe of the input filter.[3] To generate the next patch of $Q \times Q$ outputs, a similar rasterized schedule is used over a window of inputs that overlaps with the previous one. Note that in each round, *the order in which activations are fetched is different.* In Figure 2, for instance, the order of accesses from the red channel would be $A_{r1} \rightarrow A_{r2} \rightarrow A_{r4} \rightarrow A_{r5}$, at which point the first output would be computed, followed by $A_{r2} \rightarrow A_{r3} \rightarrow A_{r5} \rightarrow A_{r6}$ to generate the next output. Observe that both A_{r3} and A_{r4} need to be accessed after A_{r2}. Since RLC schemes typically only allow sequential accesses, this complicates the design of such a scheme for the rasterized schedule.

(2) *Row-major schedule:* as shown in Figure 3(b), that generates M rows of the first output channel (where M depends on the size of the accumulation buffer). As before, the schedule for the next M rows overlaps with the first one, but *accesses within a row are always sequential.* This enables us to craft a simple RLC scheme for the row major schedule.

Figure 4 shows equivalent software code for the row major schedule that is actually implemented. The comments in the code indicate how each operation in the code is mapped to

[3] As noted in [29], this schedule performs some extra computations.

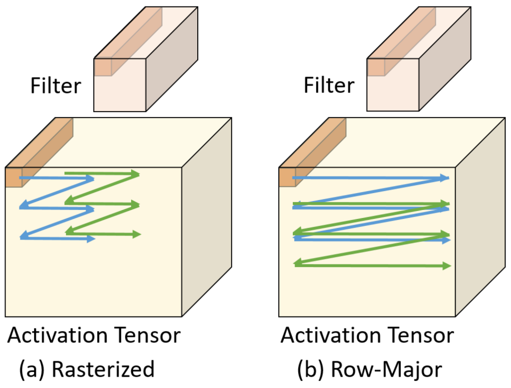


Fig. 3. Illustration of (a) rasterized and (b) row major schedules. The rasterized schedule computes a square patch of the output tensor at a time, while the row major schedule computes a subset of rows of the output tensor.

```
//Output filters mapped to columns on the SA
for (i = 0; i < Cout; i++){
  //For each pipe of a filter
  for (j = 0; j < F; j++){
    for (k = 0; k < F; k++){
      //Compute first M rows of the output tensor
      for (s = j; s < M+j; s++){
        for (t = 0; t < N; t++){
          //Dot product of input tensor and filter pipes
          //Occurs over rows of SA
          prod = 0;
          for (p = 0; p < Cin; p++)
            prod + = w[j][k][p][i] * a[s][t][p];
          //Performed in accumulation buffer
          if((t-k>=0) && (t-k<=N-F))
            y[s-j][t-k][i] + = prod;
    }}}}}
```

Fig. 4. Code representing the row major schedule. The code outputs the first M rows of the output tensor. Note that the schedule results in a small number of redundant computations that are dropped by the accumulator.

hardware. We make two observations: (i) M contiguous rows of the activation tensor are read at any given time; and (ii) assuming no padding, the schedule involves a small number of redundant computations that are filtered out by the accumulation buffer. An RLC scheme for the proposed row major schedule is described next.

3.2 On-chip RLC for Row-Major Scheduling

The row major schedule described above suggests the following RLC scheme: for each channel of the input activation tensor, CompAct compresses each row of channel separately, and concatenates the compressed rows into a single vector. This vector is then loaded into the corresponding row of the activation buffer, as shown via the example in Figure 5.

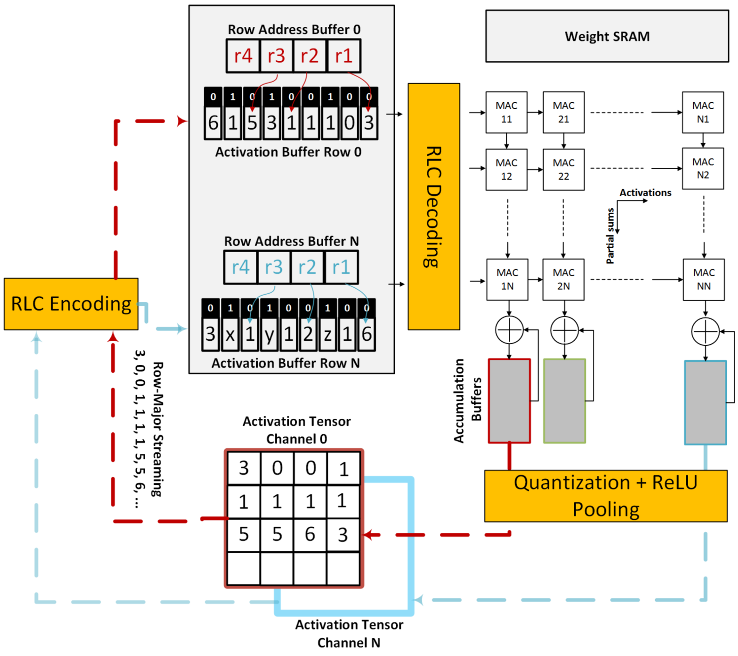


Fig. 5. Illustration of the proposed CompAct architecture. The RLC encoding and decoding blocks ensure that activations are never stored on-chip in uncompressed form.

The RLC scheme adds a single *indicator bit* per entry in the activation buffer that indicates whether the entry in the activation buffer is an activation or a run. A run entry indicates the number of times the previous activation value is repeated. In practice, the overheads of the indicator bit is roughly 10% for both area and energy, relative to the baseline activation buffer, assuming 8-bit activations.

Finally, note that the row major schedule can be implemented if any subset of M successive rows can be read from the activation buffer. To allow for this, we allocate a separate N-entry **row address buffer** that stores the starting address for each row in a channel of the input activation tensor. The size of this buffer is $N \log R$ bits, where N is the maximum spatial dimension of any activation tensor and R is the number of entries in an activation buffer row. For VGG-16, $N = 256$, which requires an extra storage of $512B$ per row of the activation buffer, and a total extra storage cost of only $32KB$ for a 64×64 systolic array. Further, the row activation buffer is infrequently accessed (relative to the activation buffer).

Algorithm 1 shows the RLC scheme that CompAct uses. The takes as input an $N \times N$ input, A, that represents a channel of the input activation tensor and a threshold θ whose relevance will become clear shortly; for now we can assume $\theta = 0$. The algorithm outputs: (1) a variable length vector s that represents the compression of A, and (2) a vector r of length N that represents the entries of the row address buffer. Element s_i of s is a tuple (c_i, v_i), where c_i is a bit that indicates if entry v_i is an activation or a run.

Decoder logic. The RLC decoder block shown in Figure 5 can now fetch any subset of M contiguous rows from the activation buffer by looking up the corresponding starting address in the row address buffer, and then reading sequentially from the activation buffer until it has read a total of $M \times N$ activation values. Note that the decoder decompresses data before feeding it into the SA, such that the SA sees the exact same input stream as it would for the baseline design. For example, when the decoder decodes a value 0 stored with run length 5, it outputs 5 zeros in sequence. The decoder adds a single cycle latency but does not incur a performance overhead since it outputs one activation per clock cycle to the systolic array.

Encoder logic. Recall that the row major schedule generates M contiguous rows per channel in the accumulation buffer. In the baseline design (without RLC), as long as $M > P$, (P is the size of the subsequent pooling filter), the re-quantization, ReLU and pooling logic can write back data from each accumulation buffer to the corresponding row of the activation buffer in *sequential row major order.* The RLC encoder intercepts the stream of data between the accumulation and activation buffers and uses Algorithm 1 to store RLC compressed activations in the activation buffer. Note that the first element of each row from an activation channel always gets encoded as a value in our row-major scheduling.

ALGORITHM 1: CompAct RLC Scheme.

```
Data: Activation matrix $A_{N\times N}$, Threshold $\theta$ of a layer.
Result: Compressed vector $s = (c, v)$, Row address buffer $r$.
$indx \leftarrow 0$;
for $i \in [0, N-1]$ do
    $r_i \leftarrow indx$;
    $v_{indx} \leftarrow a_{i0}$;
    $c_{indx} \leftarrow 0$;
    $run \leftarrow 0$;
    for $j \in [1, N-1]$ do
        if $|a_{ij} - v_{indx}| \leq \theta$ then
            if $c_{indx} = 0$ then
                $indx \leftarrow indx + 1$;
                $c_{indx} \leftarrow 1$;
                $run \leftarrow 1$;
            end
            else
                $run \leftarrow run + 1$;
            end
        end
        else
            if $run > 0$ then
                $v_{indx} \leftarrow run$;
            end
            $indx \leftarrow indx + 1$;
            $c_{indx} \leftarrow 0$;
            $v_{indx} \leftarrow a_{ij}$;
        end
    end
end
```

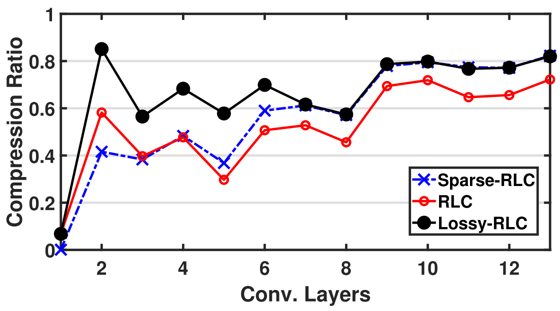


Fig. 6. Illustration of RLC and Sparse-RLC compression ratios on VGG-16.

3.3 RLC Optimizations

We now discuss two further optimizations that CompAct performs on the baseline RLC scheme to further improve compression ratio. These are: (1) Sparse-RLC: a variant of RLC that only compresses runs of 0 activations; and (2) Lossy-RLC. We describe these next.

Sparse-RLC. Our first optimization is based on the observation that later layers in the network have sparse activations [26], that is, most activations are zeros. Consequently, when using RLC, most runs in these layers are runs of zeros. Our baseline scheme uses *two* entries to store a run, one for the value and the other for the run length. However, if we only compress runs of zeros, the value of the run does not need to be stored and larger compression ratios can be achieved for sparse layers. We call this scheme Sparse-RLC. Empirically, we find that for VGG-16, RLC typically provides equal or up to 1.5× higher compression ratio (compared to Sparse-RLC) for earlier layers (Layers 1–4) in the network, while Sparse-RLC provides up to 1.3× greater compression ratio for later layers (Layers 6–13). This is illustrated in Figure 6.

In practice, CompAct switches between RLC and Sparse-RLC modes from layer to layer, depending on which provides higher compression. The choice of which scheme is used for each layer is made *offline* based on training data. For example, in Figure 6, RLC is used for Layer 1 and 2, while Sparse-RLC is used for all other layers.

Lossy-RLC. The second optimization is a lossy-RLC scheme that increases compression ratio at the expense of a tolerable loss in classification accuracy, as specified by the designer. The Lossy-RLC scheme is parameterized by a threshold parameter θ. Instead of terminating runs as soon as an activation different from run's value is encountered, we instead only terminate runs when we encounter an activation that is different from the run's value by at least $\theta + 1$ (by this token, the baseline RLC scheme is equivalent to Lossy-RLC with $\theta = 0$). Note that Algorithm 1 is already described in terms of the general Lossy-RLC case.

CompAct sets the threshold for each layer differently offline (i.e., during training) using a greedy, *energy-aware* algorithm described in Algorithm 2. Given a budget t on the maximum tolerable reduction in classification accuracy, the greedy algorithm iteratively picks the current layer with the *highest* energy consumption (in line 3 where $En_l(CNN_{\vec{\theta}})$ refers to the energy consumption of the l^{th} layer of the CNN with threshold vector $\vec{\theta}$), increases its threshold by 1 (in line 4) and tests the CNN's classification accuracy on the validation set with this new vector of thresholds (in line 2, where $Acc(CNN_{\vec{\theta}})$ refers to the validation accuracy of the CNN with threshold vector $\vec{\theta}$). The algorithm iterates till the classification accuracy drops below the budget. The algorithm then reverts to the last point before which the accuracy dropped beyond the budget (line 6), and returns this vector of thresholds.

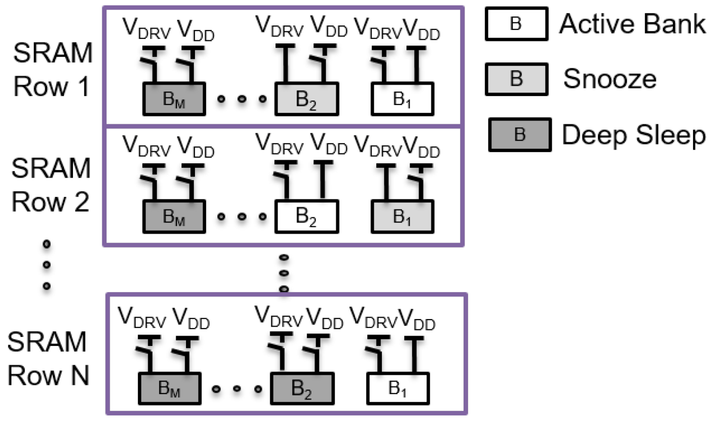


Fig. 7. Banked activation SRAM architecture with two power gating transistors for each bank. A bank can be either in active (holds valid data, currently being accessed), snooze (holds valid data, currently unaccessed), or deep sleep mode (does not hold data) modes.

Figure 6 also plots the compression ratio obtained on VGG-16 using Lossy-RLC with a 2% drop in accuracy; we observe up to a 1.4× increase in compression ratio, which is especially prominent for earlier layers in the network (more details on the experimental setup are noted in Section 4.1). Note that since the latter layers contribute relatively little to energy, the greedy algorithm does not perform lossy compression for these layers.

ALGORITHM 2: Greedy, Energy-Aware Lossy-RLC Threshold Selection.

```
   Data: Trained $L$ layer $CNN$; tolerable accuracy loss, $t$.
   Result: $L$-dimension vector $\vec{\theta}$ of compression thresholds for CNN layers.
1  $\vec{\theta} \leftarrow [0, 0, \dots, 0]$;
2  while $Acc(CNN_{\vec{\theta}_{base}}) - Acc(CNN_{\vec{\theta}}) < t$ do
3      $l \leftarrow \arg\max_{l \in [1,L]} En_l(CNN_{\vec{\theta}})$;
4      $\vec{\theta}_l \leftarrow \vec{\theta}_l + 1$;
5  end
6  $\vec{\theta}_l \leftarrow \vec{\theta}_l - 1$;
7  return $\vec{\theta}$;
```

3.4 Look-Ahead Snoozing (LAS)

RLC reduces the number of accesses to the activation buffer, thus reducing its dynamic energy; however, a significant fraction of energy consumption of large SRAMs comes from leakage power.

As a starting point, we assume that each row of the activation buffer is sub-banked into M banks, as shown in Figure 7. Each bank is connected to two power gating transistors, enabling it to be put in one of three modes:

- **Active mode:** the SRAM bank is connected to nominal voltage V_{DD} and consumes leakage energy E_{act}.
- **Snooze mode:** the SRAM bank is connected to a data retention voltage (DRV) voltage $V_{DRV} < V_{DD}$ that reduces a bank's leakage power but without any data loss and consumes leakage energy $E_{snz} < E_{act}$.

- **Deep sleep:** both power gating transistors are OFF, resulting in maximal savings in leakage power, but at the expense of data loss. A bank in deep sleep consumes leakage energy $E_{slp} < E_{snz}$.

CompAct can leverage the benefits of RLC to place a greater number of SRAM banks, i.e., those that are completely unused in any layer, in deep sleep mode. Note that the baseline design does this too (because some layers require lesser memory than others), but is able to place fewer banks in deep sleep mode than CompAct.

Second, CompAct leverages the fact that SRAM accesses occur in regular fashion, i.e., each bank is accessed for a number of clock cycles after which the next bank is accessed and so on, to put all banks that hold valid activations but are currently not accessed in snooze mode. Therefore, *only* one bank, the currently accessed one, is in active mode in any cycle. For fair comparison, we assume that this technique is also used by the baseline design.

Leveraging these two approaches, the total leakage energy consumption of the baseline scheme (without compression) per SRAM row is $E_{base} = E_{act} + (M-1)E_{snz}$ (assuming the activation memory is utilized). Assume a sparsity factor of α_i for row i, the corresponding leakage energy of CompAct will be

$$E_{CompAct} = E_{act} + (1-\alpha)(M-1)E_{snz} + \alpha(M-1)E_{slp}.$$

Now, we discuss LAS, a leakage saving technique that is possible only for CompAct and operates synergistically with our RLC scheme. Note that even the active SRAM bank is unaccessed between runs. Because we store the run length, we *know* up-front in which cycle the bank will be next accessed. Let C_{wkp} be the number of clock cycles required to wake-up a bank; we can put an active bank in snooze mode when it is unaccessed and put it back in active mode C_{wkp} cycles before it is next accessed.

An ideal implementation of LAS that can put a bank in snooze mode even if it is unaccessed for a single cycle will have a leakage energy of

$$E_{CompAct+LAS} = \alpha E_{act} + (1-\alpha)ME_{snz} + \alpha(M-1)E_{slp}.$$

That is, CompAct additionally reduces the leakage power consumed by active banks. In practice, the actual leakage power savings depend on the value of C_{wkp} and the energy overheads to transition from snooze to active mode. These overheads are discussed in Section 4.2.

3.5 System-Level Considerations

Here we describe how CompAct operates at the system level, taking into account practical considerations that were not addressed in the discussion thus far.

Reducing size of the activation buffer. Our description of CompAct has implicitly assumed thus far that the activation buffer is sized such that activations do not need to be written off-chip [17]. In practice, we evaluate CompAct with much smaller activation buffer sizes. If there is no space in the activation buffer to store a layer's outputs, the outputs are written to off-chip DRAM, in raw format for the baseline SA and in compressed form for CompAct. When the subsequent layer executes, its input activation tensors are fetched from off-chip DRAM.

In addition to reducing the number of accesses from the activation buffer, CompAct also reduces number of off-chip accesses when the activation buffer sizes are small. That is, when both input and output activations are compressed, there is a chance that they fit in the limited on-chip activation buffer even when the uncompressed activations do not. In the worst case, of course, CompAct also goes off-chip to store compressed activations if required.

Provisioning weight and accumulation buffers. CompAct does not target energy or area reductions for accumulation and weight buffer; these buffers consume relatively little area for the baseline SA (as also in the TPU). Specifically, we provision the weight buffer so it can store the weights of at least one filter per column of the SA. Weights are loaded from off-chip DRAM when required, and are not compressed (weight compression is orthogonal to the proposed approach and will only further increase the relative benefits of CompAct versus the baseline).

The accumulation buffer is sized such that each column of the buffer can hold at least P rows from *any* layer, where P is the size of the largest pooling filter. Recall the our row major schedule produces outputs row by row, which must then be pooled to produce outputs for the next layer. With these constraints, the weight and accumulation buffers are still small, and consume between 5%–12% of chip area.

Batching. Finally, note that CNN accelerators typically benefit from batching, i.e., computing on multiple inputs in parallel, since it allows the overheads of fetching weights from DRAM to be amortized over the batch. On the flip side lower batch sizes do provide the benefit of lower latency which might be important for real-time applications. We were able to run AlexNet with a batch size of 4 while still keeping the size of the activation buffer relatively small. However for a larger network like VGG-16, even a batch size of 1 results in very large activation buffers. Nonetheless, there is still an opportunity for optimization, i.e., the later layers of VGG-16 have smaller activations than its earlier layers and can be batched. More specifically, the first l ($l < L$) layers execute with a batch size of 1 while the last $L - l$ layers run with a batch size of B, where B is determined based on the number of *uncompressed* activations that can fit in the activation buffer. As mentioned in the TPU paper [17], the first l layers execute B times and generate a batch of B inputs for the final layers. Finally, we note that by compressing activations, CompAct might enable the use of larger batch sizes compared to the baseline. However, we have conservatively assumed that the batch sizes are the same for both for the fairest comparison.

4 EMPIRICAL EVALUATION

We now present our empirical evaluation of CompAct compared to a baseline scheme modeled on the TPU architecture that does not use any on-chip compression.

4.1 Experimental Setup

CNN Benchmarks. We evaluate CompAct on two CNN benchmarks, AlexNet [20] and VGG-16, both for image recognition using the ImageNet dataset [7]. AlexNet has 5 convolution layers and VGG-16 has 13. We note that for CNNs, convolution layers account more than 90% of the computational load and therefore dominate energy consumption [2, 26, 28]. Table 1 shows the parameters of the two CNN architectures. The weights and activations of both CNNs are quantized to 8-bits for energy efficient hardware implementation, as is standard practice.

Hardware Parameters. Our focus in this paper is on CNN accelerators for mobile and embedded computing. For instance, ARM recently announced an SA based CNN accelerator that operates within a 1.5W power budget [3]. To operate under a tight power budget, we evaluate on small 32×32 and 64×64 SAs.

We evaluate CompAct with four different CNN accelerator configurations, one for AlexNet and three for VGG-16. For all four configurations, the sizes of the SA and on-chip weight and accumulation buffers are shown in Table 2.

Table 1. Details of the CNNs Used and Accuracy of 32-bit and 8-bit Implementations for the Entire Test Set

	Benchmarks	**Accuracy (%)**	
Name	**Architecture**	**32-bit**	**8-bit**
AlexNet [21]	L1-L2 (Conv): $(224, 224, 3) \times (27, 27, 64) \times (13, 13, 192)$	Top 5	Top 5
	L3-L5 (Conv): $(13, 13, 384) \times (13, 13, 256) \times (6, 6, 256)$		
	L6-L8 (FC): $4096 \times 4096 \times 1000$	79.066	78.598
VGG-16 [33]	L1-L2 (Conv): $(224, 224, 3) \times (224, 224, 64) \times (112, 112, 64)$	Top 5	Top 5
	L3-L5 (Conv): $(112, 112, 128) \times (56, 56, 128) \times (56, 56, 256)$		
	L6-L8 (Conv): $(56, 56, 256) \times (28, 28, 256) \times (28, 28, 512)$		
	L9-L11 (Conv): $(28, 28, 512) \times (14, 14, 512) \times (14, 14, 512)$	90.382	90.118
	L12-L13 (Conv): $(14, 14, 512) \times (7, 7, 512)$		
	L14-L16 (FC): $4096 \times 4096 \times 1000$		

- The AlexNet accelerator has a $512KB$ activation buffer; this buffer size is sufficient to ensure that even uncompressed activations remain on chip with a batch size of 4, while accounting for 52% of the total chip area.
- We explore three configurations of the VGG-16 accelerator with *large* (L), *medium* (M) and *small* (S) activation buffer sizes. VGG-L has a $6.4MB$ activation buffer sized to ensure that all activations remain on chip, but the activation buffer occupies 80% of the total chip area. Therefore, this architecture may not be suitable for embedded applications. VGG-M and VGG-S have more reasonably sized activation buffers, of size $1.6MB$ and $0.8MB$, respectively, that represent 51% and 34% of chip area.

The chips are designed in 45 nm technology node. We assume a $V_{DD} = 1V$, a $V_{DRV} = 0.7V$ (which is conservative) [35] and a target frequency of 500 MHz. We evaluate the following techniques:

- **Base**: Does not perform any compression of values in the activation SRAM. To reduce leakage power, the baseline architecture puts unused banks in sleep mode, and unaccessed banks in snooze mode.
- **Base+RLC**: like Eyeriss, uses RLC to compress activations when they are written off-chip, but stores activations on-chip in uncompressed form.
- **CompAct**: our proposed CompAct architecture that always keeps activations in compressed format, both on- and off-chip. CompAct includes $Compact_{Lossless}$, compressing the activations without any approximation; and $Compact_{Lossy:t\%}$, allowing upto t% CNN classification accuracy drop during the compression. In our experimental results we allow t to be up to 2%.

Energy/Power Estimation. We implemented CompAct along with its RLC coding/decoding logic in synthesizable Verilog, and synthesized with Nangate FreePDK 45 *nm* standard cell library using the Cadence Genus synthesis tool to obtain power estimates for the logic components. Our energy and power estimates for all memory components are based on HP CACTI 7.0 using 45 *nm* technology node and a low power process [25]. We also obtain the wake-up time and wake-up energy overheads from CACTI 7.0 using its power gating feature.

Table 2 reports the parameters for two specific chip designs, one for AlexNet and the other for VGG-16. For a fair **iso-area** comparison, we use the same hardware parameters for both baseline and Compact. The peak power consumption of the AlexNet chip is 0.7 W with a peak performance

Table 2. Baseline and CompAct Design Parameters

Components	AlexNet	VGG-16
SA	Parameter	
	Size: 32X32	Size: 64X64
	Area (mm^2): 2.06	Area (mm^2): 8.22
	MAC OP dynamic energy (pJ): 1.32 MAC leakage energy (pJ): 0.03 Cycle Time (ns): 2	
RLC Logic Ctrol	Dynamic energy per OP (pJ): 0.17; Leakage per OP (pJ): 0.006	
Activation Buffer	Capacity (KB): 512 Area (mm^2): 3.46 Banks: 1 Output bits: 9 Dynamic R/W energy (pJ): 11.2/5.8 Bank Leakage (pJ): 8.9 Snooze Leakage (pJ): 0.89 Wake-up Energy (pJ): 5 Wake-up time (ns): 0.36	Capacity (MB): S: 0.8 M: 1.6 L: 6.4 Area (mm^2): S: 5.35 M: 11.4 L: 46.0 Banks: S: 1 M: 2 L: 8 Output bits: 9 Dynamic R/W energy (pJ): S: 7.7/4.5 M: 10.8/6.6 L: 17.6/13.3 Bank Leakage (pJ): S: (5.14) M: (6.85) L: (6.99) Snooze Leakage (pJ): 0.7 Wake-up Energy (pJ): 9.8 Wake-up time (ns): 0.56
Row Address Buffer	Capacity (KB): 16.3 Area (mm^2): 0.128 Output bits: 16 Dynamic R/W energy (pJ): 0.9/2	Capacity (KB): 32 Area (mm^2): 0.256 Output bits: 16 Dynamic R/W energy (pJ): 0.9/2
Accumulation Buffer	Capacity (KB): 43 Area(mm^2): 0.35 Output bits: 24 Dynamic R/W energy (pJ): 2.16/3.2 Leakage (pJ): 0.314	Capacity (KB): 86 Area(mm^2): 0.71 Output bits: 24 Dynamic R/W energy (pJ): 2.16/3.27 Leakage (pJ): 0.32
Weight Buffer	Capacity (KB): 112 Area(mm^2): 0.82 Dynamic R/W energy (pJ): 2.96/2.6 Leakage (pJ): 1	Capacity (KB): 295 Area(mm^2): 2.18 Dynamic R/W energy (pJ): 4.4/3.19 Leakage (pJ): 1.5

The SRAM access energies are reported on a per row/column basis.

of 128 GOPS and that of the VGG-16 chip is 2.5 W with a peak performance of 500 GOPS. All our results are based on an in-house cycle accurate simulator for TPU operation.

Simulation Methodology. Based on the synthesized results in Table 2, we built up a cycle-accurate model for all the components shown in Figure 5. All the overheads including the encoding/decoding logic, wake-up energy costs are all built in. We scheduled the benchmark CNNs on the systolic array as described in Section 3.1, and simulated the execution all CNN layers, by accounting the data movements within and between CNN layers. We used a set of training data to determine compression thresholds, and a separate validation data for test.

Table 3. Details of Compact Coding Scheme and Threshold $\vec{\theta}$ for Each CNN (r Represents RLC, and s is Sparse-RLC)

		L1	**L2**	**L3**	**L4**	**L5**	**L6**	**L7-L13**
AlexNet	coding	r	r	s	s	s	-	-
Lossy: 2%	$\vec{\theta}$	0	1	2	1	1	-	-
VGG-16	coding	r	r	s	s	s	s	s
Lossy: 1%	$\vec{\theta}$	0	1	1	1	1	1	0
VGG-16	coding	r	r	s	s	s	s	s
Lossy: 2%	$\vec{\theta}$	0	2	1	1	1	1	0

4.2 CompAct Energy Savings

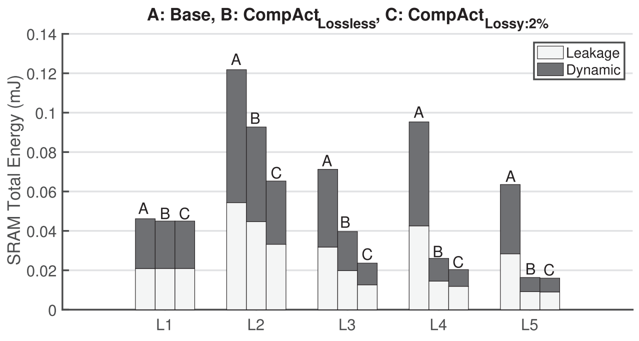
In this section, we compare the proposed CompAct design against the Base and Base+RLC. We first report the obtained saving in the leakage, dynamic and total energy for the activation buffers alone, and then report the total system-wide energy savings; note that although CompAct does not (and is not intended to) reduce the power consumed by the SA logic, weight buffers or accumulation buffers, the saving in activation buffer and off-chip energy translate into significant savings in total chip energy. In our experiments, we present data for both $CompAct_{Lossless}$ and $CompAct_{Lossy:2\%}$. Details of coding scheme and threshold $\vec{\theta}$ used by *Compact* are in Table 3.

Activation Buffer Energy Reduction. We begin by evaluating CompAct energy savings for the activation buffer only. Figure 8(a) and Figure 8(b) plots the activation buffer energy costs, broken down into dynamic and leakage energy, for *each* layer of AlexNet and the VGG-S chips, respectively (the results for VGG-L and VGG-M chips are qualitatively similar and are not shown due to space limitations). We note that any savings in leakage power consumption accrue directly from the proposed LAS scheme, since LAS is able to additionally turn off the active bank during long runs.

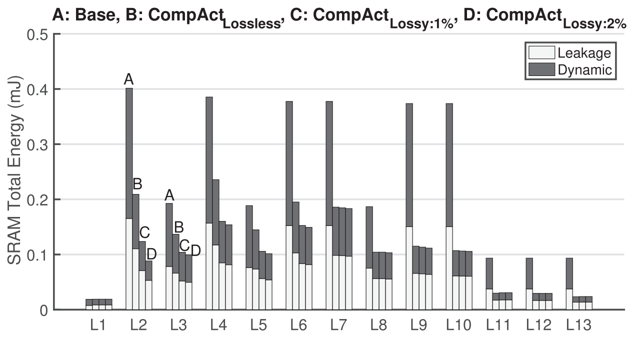
We compare only Base with CompAct since Base+RLC has the same activation buffer costs as Base. Several observations are in order. (1) Across the board, CompAct (both lossless and lossy) reduces both dynamic and leakage energy compared to Base. This is true for both AlexNet and VGG-S. The reductions for each layer depend largely on the compression ratio for dynamic energy, and the length of runs for leakage energy (since the wake-up energy costs are amortized). (2) The reductions for $CompAct_{Lossy:2\%}$ are greater than those for $CompAct_{Lossless}$. (3) Summed across all layers, $CompAct_{Lossy:2\%}$ ($CompAct_{Lossless}$) achieves 63% (50%) reduction in dynamic energy, 50.74% (38.67%) reduction in leakage energy and 57.22% (44.8%) reduction in static+dynamic energy for AlexNet. Figure 9 plots the activation buffer energy summed over all layers for the VGG-S, VGG-M and VGG-L chips. For VGG-S 72% (61%), 49% (36.7%) and 62% (51.3%) reductions in dynamic, leakage and dynamic+leakage energy are achieved by $CompAct_{Lossy:2\%}$ ($CompAct_{Lossless}$).

Accuracy Vs. Energy Trade-offs. The data above demonstrate a trade-off between accuracy and energy savings for lossless ($t = 0\%$) and lossy (with $t = 2\%$) compression accuracy. To further illustrate the impact of varying t, Figure 10 for VGG-16 plots compression ratio vs accuracy for different values of t. From the plot, we observe that accuracy drops gradually till $t = 1\%$ ($Compact_{Lossy:1\%}$) but more significantly thereafter. The per-layer and total activation buffer energy savings for $Compact_{Lossy:1\%}$ are shown in Figure 8(b) and Figure 9 illustrating the energy-accuracy trade-offs achieved by CompAct.

System-level Energy Reduction. Figure 11 plots the system-wide energy per layer of AlexNet, while Figure 12 shows the same information for VGG-L, VGG-M and VGG-S. The data for



(a) AlexNet with 512 KB Activation Buffer.



(b) VGG-S with 0.8 MB Activation Buffer.

Fig. 8. Per layer activation buffer energy reduction.

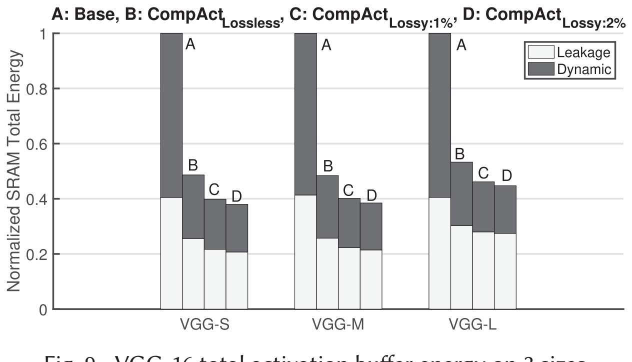


Fig. 9. VGG-16 total activation buffer energy on 3 sizes.

CompAct in all cases correspond to our more aggressive *CompAct*$_{Lossy:2\%}$. The energy costs are broken down by contributions by the SA core logic, on-chip buffers and DRAM access costs. We observe that AlexNet's memory access costs are dominated by activation buffer energy, which CompAct reduces significantly, and off-chip (DRAM) weight accesses, that CompAct does not target. Overall, CompAct reduces total chip energy across all layers by 22% compared to both Base and Base+RLC which have the same energy costs for AlexNet.

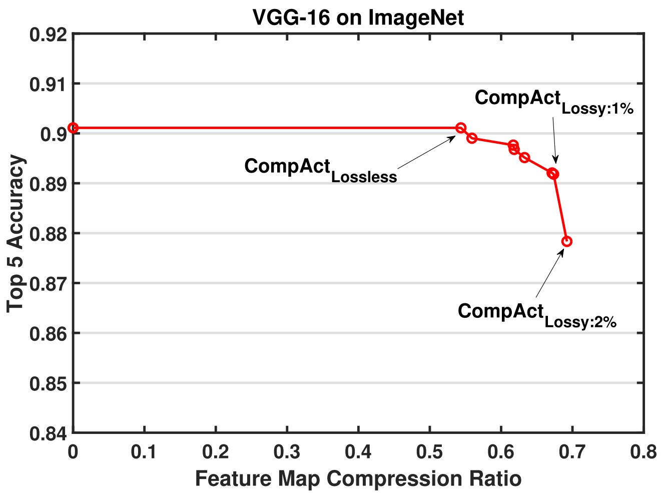


Fig. 10. VGG-16 accuracy vs compact tradeoff.

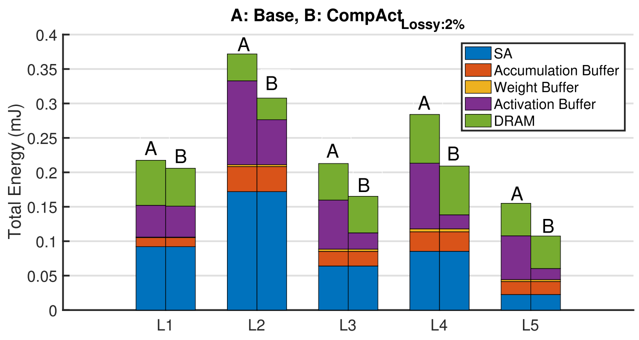
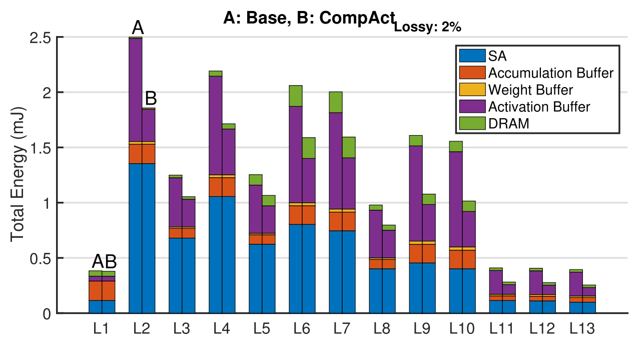


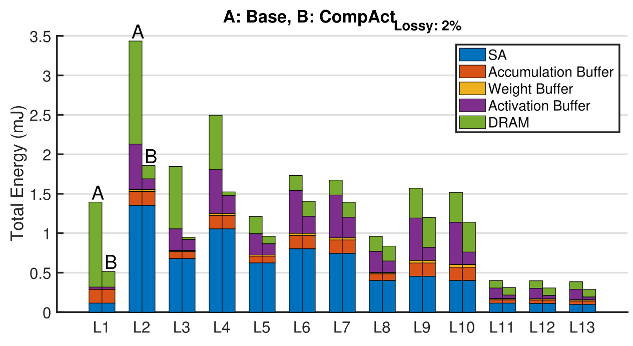
Fig. 11. AlexNet per-layer system-level energy.

Several interesting observations can be made for VGG-16 in Figure 12, in particular as it relates to decreasing activation buffer sizes. (1) For the largest activation buffer size, VGG-L, memory energy costs are dominated by activation buffer accesses (all DRAM access are due to weights); the benefits of CompAct therefore accrue from the reductions in dynamic and leakage power of the activation buffer. (2) For VGG-M and VGG-S, the contribution of DRAM energy increases. For the earlier layers (such as L2) these costs are dominated by storing and loading activations from DRAM — CompAct dramatically reduces the DRAM costs of L2 because after on-chip compression, activations typically fit within the activation buffer. In the few cases that CompAct has to go off-chip, it fetches compressed activations (as does Base+RLC, which is not pictured). (3) For the latter layers of VGG-S (L8, L9, L10), the DRAM costs go up due to repeated weight accesses. CompAct does not reduce these costs; however, orthogonal weight compression techniques might be of use here.

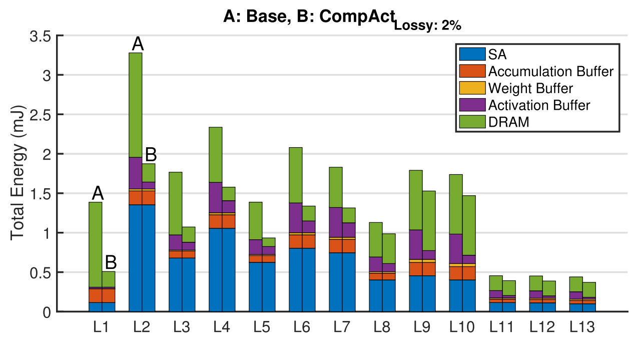
Figure 13 shows the chip-wide energy costs of VGG-S, VGG-M and VGG-L for Base, Base+RLC and CompAct. We make several observations. (1) For VGG-S, Base+RLC reduces DRAM costs relative to Base but has the same activation buffer energy, resulting in a 18% energy reduction. CompAct *further* reduces both the DRAM and activation buffer energy (the latter more than the former), providing 33% energy reduction over Base. (2) VGG-M has similar trends, except that CompAct



(a) Activation buffer size 6.4 MB.



(b) Activation buffer size 1.6 MB.



(c) Activation buffer size 0.8 MB.

Fig. 12. VGG-16 per layer system-level energy.

significantly cuts both the activation buffer *and* DRAM energy relative to Base+RLC. CompAct provides 34% and 26% energy reduction over Base and Base+RLC, respectively. (3) For VGG-L, the energy costs of the activation buffer are significant, while DRAM costs are small. Here, Base and Base+RLC have similar energy. CompAct provides a 24% energy saving over Base+RLC.

4.3 Comparison with Channel Pruning

Another approach to reducing activation buffer accesses is channel pruning, which has been widely studied as a CNN model compression strategy [15, 23, 24, 31, 36, 38].

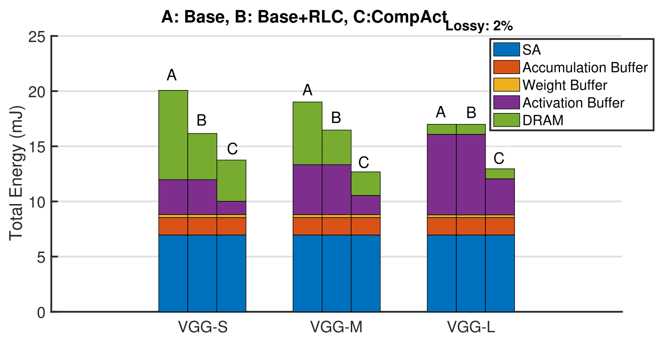
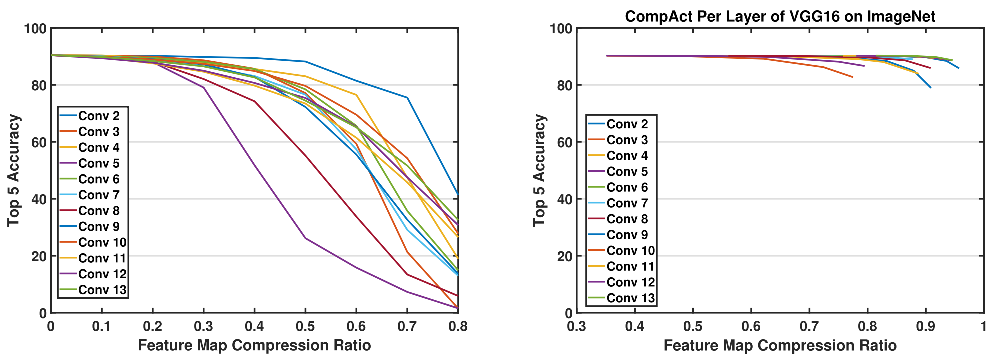


Fig. 13. VGG-16 total energy on 3 activation buffer sizes.



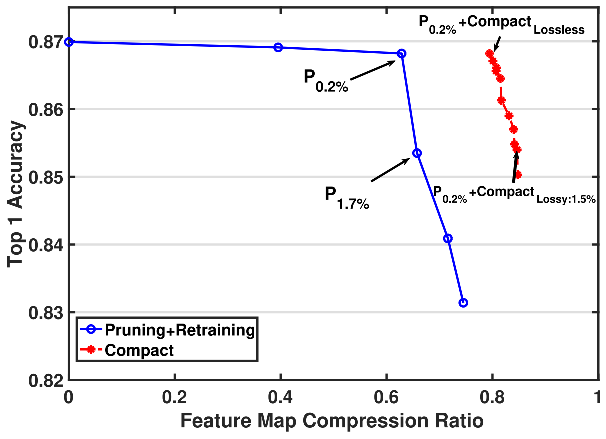
(a) Filter Pruning (b) Compact

Fig. 14. VGG-16 Per Layer Filter Pruning (without re-training) Vs Compact on ImageNet.

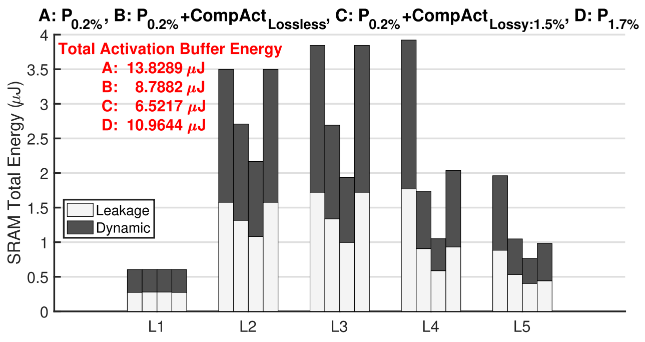
At the outset, we note that $CompAct_{Lossless}$ can always be applied on top of any pruned CNN, resulting in further energy reductions *without* accuracy loss. As we have seen in the results so far, $CompAct_{Lossless}$ itself provides significant activation buffer energy savings. We now show that $CompAct_{Lossy}$ can be used *synergistically* with channel pruning to achieve even greater savings.

Before beginning, we first note that channel pruning *without* retraining results in *significant* drop in accuracy even with limited amounts of pruning. Figure 14 shows accuracy versus per-layer compression ratio using pruning (Figure 14(a)) or $CompAct_{Lossy}$ (Figure 14(b)) on each layer individually — pruning without retraining can achieve at most 50% compression ratio beyond which accuracy starts dropping sharply, while $CompAct_{Lossy}$ typically enables > 50% compression. This is consistent with data reported in other papers [15, 23]. Thus a fair comparison with pruning requires the pruned networks to be retrained — unfortunately, retraining a single pruned network for ImageNet would take more than 40 hours on 4 GPUs, limiting our ability to search for optimally pruned networks. Thus, for this comparison, we use the smaller CIFAR-10 dataset, which is commonly used in prior work for exploring accuracy vs. energy trade-offs of pruning [15, 23, 24, 31].

Comparison with Pruning+Retraining. In order to compare with the pruning with Retraining, we use a small Cifar-10 dataset to speed up the re-training process. *Note here, we only apply training on Pruning, but not on the Compact.* Figure 15(a) shows the accuracy versus compression ratio (in terms of the total size of the CNN's activation layers) for five pruned versions of AlexNet obtained via the L1-norm pruning method [23, 24]. Beyond a point (shown as AlexNet $P_{0.2\%}$ with only



(a) Compression Ratio



(b) Activation Buffer Energy

Fig. 15. AlexNet Pruning vs Compact on Cifar-10.

0.2% accuracy drop), we see sharp drop in accuracy if channels are further pruned. We now apply $CompAct_{Lossless}$ and $CompAct_{Lossy:t\%}$ for several values of $t \leq 2\%$ on top of $P_{0.2\%}$ and plot the accuracy versus compression ratio also in Figure 15(a). Now, we compare two solutions with exactly the same accuracy loss: (1) $P_{1.7\%}$, that uses pruning alone, with a net accuracy drop relative to the unpruned of 1.7%, and (2) $P_{0.2\%} + CompAct_{Lossy:1.5\%}$ which has the same net accuracy drop relative to the unpruned. Figure 15(b) shows the activation buffer energy consumption of these two solutions for the AlexNet chip in Table 2, along with that of $P_{0.2\%}$ and $P_{0.2\%} + CompAct_{Lossless}$. Allowing for a 0.2% accuracy drop, $P_{0.2\%} + CompAct_{Lossless}$ provides 38% savings over pruning ($P_{0.2\%}$) alone. Allowing for a 1.7% accuracy drop, synergistic pruning plus lossy CompAct ($P_{0.2\%}$ + $CompAct_{Lossy:1.5\%}$) provides 41% savings over pruning alone ($P_{1.7\%}$). Although our focus is on reducing activation buffer energy, these savings translate roughly 20% reductions in system-wide energy consumption. We repeated this experiment for VGG; Figure 16 shows pareto curves of accuracy versus energy for pruning alone, pruning+$CompAct_{Lossless}$ and pruning+$CompAct_{Lossy}$ — we note again that at *iso-accuracy*, pruning+$CompAct_{Lossy}$ results in the greatest energy savings. At the points highlighted in the figure (all with 1.5% accuracy drop), pruning+$CompAct_{Lossless}$ has 53% lower energy than pruning alone; pruning+$CompAct_{Lossy}$ provides a further 17% reduction in energy.

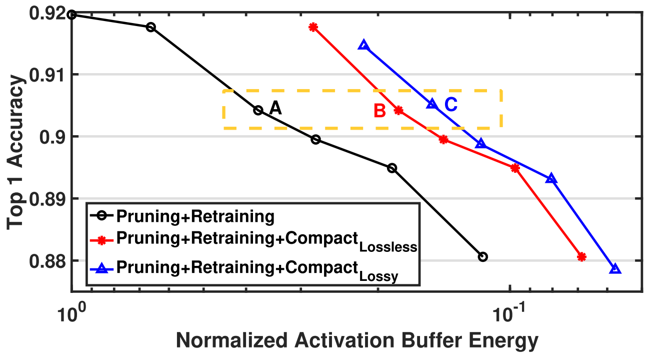


Fig. 16. VGG-16 Pruning+Compact on Cifar-10.

5 RELATED WORK

There have been several attempts in the past to design compression schemes for CNN acceleration, including schemes that target weight/filter compression, activation compression or both. Weight/filter compression techniques such as CNN/DNN pruning [13, 36, 39] are orthogonal to the goal of the CompAct, that focuses on compressing activations. Techniques such as cirCNN [8] place constraints on the weight matrices that allow for compact representations, but again do not explicitly compress activations. For the weight-stationary SA architecture that we adopt as a baseline, the dominant SRAM block (in terms of area and energy) is the activation SRAM [17], which is the compression target for CompAct. Note also that weight compression can be performed *offline*, while activations needs to be compressed online. Further, we expect that deploying any of the aforementioned weight compression approaches, CompAct's relative benefits over the baseline would improve, since activations would account for an even greater fraction of total chip-wide energy.

Techniques that compress activations have only been explored thus far in the context of loosely-coupled accelerators. The EIE architecture uses run-length encoding (RLC) for *off-chip* compression, but stores data in uncompressed format [14]. More relevant is the SCNN [26] approach that uses sparse coding for *on-chip activation compression*. However, SCNN implements a highly specialized architecture that uses on-chip compression to reduce the number of MAC operations performed; that is, SCNN stores the indices and values of all non-zero activations and weights, and operates on these elements in an irregular fashion. In contrast, to maintain precise synchrony for an SA architecture, *all* activations and weights, including zeros, must be streamed through the SA in regular order. Thus, the SCNN compression scheme cannot be easily or directly applied to SA based acceleration. Furthermore, it is important to note from Figure 6, that RLC provides far greater compression ratio than sparse coding for earlier layers in the CNNs that we experimented with.

Previous work has exploited sparsity to reduce compute energy by skipping or bypassing MACs with zero inputs [1, 5, 19]. Our baseline design also uses the same approach (we do not claim this as a new contribution). For loosely couple accelerators, sparse coding also improves performance [26] by skipping compute cycles. However, because precise synchrony has to be maintained, this is not possible for the TPU. As we show, the primary benefit of sparse coding in this context is to reduce the dynamic/leakage energy of activation SRAMs. Because of their low design complexity (and other advantages as explained in [6]), several recent works have performed design space exploration studies of SA based accelerators. This includes work on optimizing the SA architecture including the shape of the SA core and buffer sizes [16, 42], optimized memory schedules [10], and techniques to reduce power consumption of the SA core [40]. None of these papers have so far performed on-chip compression of activations.

6 CONCLUSION AND FUTURE WORK

In this paper, we have presented and evaluated CompAct, the *first* architecture that uses *on-chip compression* in the context of tightly-coupled SA based CNN accelerators. CompAct is motivated by the observations that memory energy is a major contributor to on-chip energy consumption for SAs based accelerators, especially for mobile and embedded accelerators with small SAs. CompAct introduces several novel ideas to enable on-chip compression of activations; (i) CompAct uses a row major schedule that has nearly regular access patterns from the activation buffer, and implements a modified RLC scheme that exploits this access patterns; (ii) CompAct further increases compression ratio using Sparse-RLC in later layers in the network and Lossy-RLC in earlier layers; and (iii) CompAct reduces leakage power using a lookahead snooze scheme that places activation buffers in sleep mode during long runs. Leveraging these ideas, CompAct is able to achieve up to 62% reduction in activation buffer energy, and up to 34% savings in total chip energy for chips with 32×32 and 64×64 SAs for AlexNet and VGG-16 CNNs. As future work, we will explore the use of RLC coding for modified SA based architectures, like the one proposed by [22], that are especially designed for CNNs like MobileNet.

ACKNOWLEDGMENTS

This work is performed when the first author interned at Microsoft Research, and is supported in part by an National Science Foundation CAREER Award. The authors would like to thank the anonymous reviewers for their time, suggestions, and valuable feedback.

REFERENCES

[1] Jorge Albericio et al. 2016. Cnvlutin: Ineffectual-neuron-free deep neural network computing. In *Processdings of ACM/IEEE ISCA*. 1–13.
[2] Manoj Alwani et al. 2016. Fused-layer CNN accelerators. In *IEEE/ACM MICRO*. 1–12.
[3] ARM. 2018. PROJECT TRILLIUM@ONLINE. https://www.arm.com/products/silicon-ip-cpu/machine-learning/project-trillium
[4] Srimat Chakradhar et al. 2010. A dynamically configurable coprocessor for convolutional neural networks. In *ACM Computer Architecture News*, Vol. 38. 247–257.
[5] Yu-Hsin Chen et al. 2017. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE JSSC* 52, 1 (2017), 127–138.
[6] Reetuparna Das and Tushar Krishna. [n.d.]. DNN Accelerator Architecture – SIMD or Systolic? https://www.sigarch.org/dnn-accelerator-architecture-simd-or-systolic/. Accessed: 2019-04-27.
[7] Jia Deng et al. 2009. Imagenet: A large-scale hierarchical image database. In *IEEE CVPR*. 248–255.
[8] Caiwen Ding, Siyu Liao, Yanzhi Wang, Zhe Li, Ning Liu, Youwei Zhuo, Chao Wang, Xuehai Qian, Yu Bai, Geng Yuan, et al. 2017. CirCNN: Accelerating and compressing deep neural networks using Block-CirculantWeight matrices. *arXiv preprint arXiv:1708.08917* (2017).
[9] Zidong Du et al. 2015. ShiDianNao: Shifting vision processing closer to the sensor. In *ACM SIGARCH Computer Architecture News*, Vol. 43. 92–104.
[10] Li et al. 2018. SmartShuttle: Optimizing off-chip memory accesses for deep learning accelerators. In *2018 IEEE DATE*. 343–348.
[11] Mingyu Gao, Xuan Yang, Jing Pu, Mark Horowitz, and Christos Kozyrakis. 2019. Tangram: Optimized coarse-grained dataflow for scalable NN accelerators.
[12] Yijin Guan et al. 2017. FP-DNN: An automated framework for mapping deep neural networks onto FPGAs with RTL-HLS hybrid templates. In *2017 IEEE FCCM*. 152–159.
[13] Song Han et al. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).
[14] Song Han et al. 2016. EIE: Efficient inference engine on compressed deep neural network. In *Proceedings of IEEE ISCA*. 243–254.
[15] Muhammad Abdullah Hanif, Alberto Marchisio, Tabasher Arif, Rehan Hafiz, Semeen Rehman, and Muhammad Shafique. 2018. X-DNNs: Systematic cross-layer approximations for energy-efficient deep neural networks. *Journal of Low Power Electronics* 14, 4 (2018), 520–534.

[16] Weiwen Jiang, Edwin Hsing-Mean Sha, Qingfeng Zhuge, Lei Yang, Xianzhang Chen, and Jingtong Hu. 2018. Heterogeneous fpga-based cost-optimal design for timing-constrained cnns. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37, 11 (2018), 2542–2554.

[17] Norman Jouppi et al. 2017. In-datacenter performance analysis of a tensor processing unit. *arXiv preprint arXiv:1704.04760* (2017).

[18] Andrej Karpathy et al. 2014. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE CVPR*. 1725–1732.

[19] Dongyoung Kim et al. 2017. ZeNA: Zero-aware neural network accelerator. *IEEE Design Test* (2017).

[20] Alex Krizhevsky. 2014. One weird trick for parallelizing convolutional neural networks. *arXiv preprint arXiv:1404.5997* (2014).

[21] Alex Krizhevsky et al. 2012. Imagenet classification with deep convolutional neural networks. In *NIPS*. 1097–1105.

[22] HT Kung et al. 2019. Packing sparse convolutional neural networks for efficient systolic array implementations: Column combining under joint optimization. In *24th ASPLOS*. ACM, 821–834.

[23] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. 2016. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710* (2016).

[24] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. 2018. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270* (2018).

[25] Naveen Muralimanohar, Rajeev Balasubramonian, and Norman P. Jouppi. 2009. CACTI 6.0: A tool to model large caches. *HP Laboratories* (2009), 22–31.

[26] Angshuman Parashar et al. 2017. SCNN: An accelerator for compressed-sparse convolutional neural networks. *SIGARCH Comput. Archit. News* 45, 2 (June 2017).

[27] Seongwook Park et al. 2015. 93TOPS/W scalable deep learning/inference processor with tetra-parallel MIMD architecture for big-data applications. In *IEEE ISSCC*.

[28] Atul Rahman et al. 2016. Efficient FPGA acceleration of convolutional neural networks using logical-3D compute array. In *IEEE DATE*. 1393–1398.

[29] Jonathan Ross and Gregory Michael Thorson. 2017. Rotating data for neural network computations. US Patent 9,805,303.

[30] Murugan Sankaradas et al. 2009. A massively parallel coprocessor for convolutional neural networks. In *IEEE ASAP 2009*. 53–60.

[31] Syed Shakib Sarwar, Gopalakrishnan Srinivasan, Bing Han, Parami Wijesinghe, Akhilesh Jaiswal, Priyadarshini Panda, Anand Raghunathan, and Kaushik Roy. 2018. Energy efficient neural computing: A study of cross-layer approximations. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 8, 4 (2018), 796–809.

[32] Jürgen Schmidhuber. 2015. Deep learning in neural networks: An overview. *Neural networks* 61 (2015), 85–117.

[33] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[34] Aravind Vasudevan et al., Andrew Anderson, and David Gregg. 2017. Parallel multi channel convolution using general matrix multiplication. In *IEEE 28th ASAP*. 19–24.

[35] Elena I Vatajelu and Joan Figueras. 2011. Statistical analysis of 6T SRAM data retention voltage under process variation. In *14th IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems*. IEEE, 365–370.

[36] Wei Wen et al. 2016. Learning structured sparsity in deep neural networks. In *NIPS*. 2074–2082.

[37] Keiji Yanai, Ryosuke Tanno, and Koichi Okamoto. 2016. Efficient mobile implementation of a cnn-based object recognition system. In *Proceedings of the 24th ACM international conference on Multimedia*. ACM, 362–366.

[38] Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. 2017. Designing energy-efficient convolutional neural networks using energy-aware pruning. In *Proceedings of the IEEE CVPR*. 5687–5695.

[39] Jiecao Yu et al. 2017. Scalpel: Customizing DNN pruning to the underlying hardware parallelism. In *Proceedings of ACM ISCA*. 548–560.

[40] Jeff Zhang, Kartheek Rangineni, Zahra Ghodsi, and Siddharth Garg. 2018. Thundervolt: Enabling aggressive voltage underscaling and timing error resilience for energy efficient deep learning accelerators. In *ACM 55th DAC*. 19.

[41] Jeff Jun Zhang, Tianyu Gu, Kanad Basu, and Siddharth Garg. 2018. Analyzing and mitigating the impact of permanent faults on a systolic array based neural network accelerator. In *2018 IEEE 36th VLSI Test Symposium (VTS)*. IEEE, 1–6.

[42] Chen Zhang et al. 2015. Optimizing fpga-based accelerator design for deep convolutional neural networks. In *Proceedings of the 2015 ACM FPGA*. ACM, 161–170.

Received April 2019; revised June 2019; accepted July 2019