EcoFlow: Efficient Convolutional Dataflows for Low-Power Neural Network Accelerators

EcoFlow: Efficient Convolutional Dataflows for Low-Power Neural Network Accelerators

Lois Orosa, Skanda Koppula, Yaman Umuroglu, Konstantinos Kanellopoulos, Juan Gómez-Luna, Michaela Blott, Kees Vissers, Onur Mutlu

Abstract—Dilated and transposed convolutions are widely used in modern convolutional neural networks (CNNs). These kernels are used extensively during CNN training and inference of applications such as image segmentation and high-resolution image generation. Although these kernels have grown in popularity, they stress current compute systems due to their high memory intensity, exascale compute demands, and large energy consumption.

We find that commonly-used low-power CNN inference accelerators based on spatial architectures are *not* optimized for both of these convolutional kernels. Dilated and transposed convolutions introduce significant zero padding when mapped to the underlying spatial architecture, significantly degrading performance and energy efficiency. Existing approaches that address this issue require significant design changes to the otherwise simple, efficient, and well-adopted architectures used to compute direct convolutions.

To address this challenge, we propose EcoFlow, a new set of dataflows and mapping algorithms for dilated and transposed convolutions. These algorithms are tailored to execute efficiently on existing low-cost, small-scale spatial architectures and requires minimal changes to the network-on-chip of existing accelerators. At its core, EcoFlow eliminates zero padding through careful dataflow orchestration and data mapping tailored to the spatial architecture. EcoFlow enables flexible and high-performance transpose and dilated convolutions on architectures that are otherwise optimized for CNN inference.

We evaluate the efficiency of our dataflows on CNN training workloads and Generative Adversarial Network (GAN) training workloads. Experiments in our new cycle-accurate spatial architecture simulator show that EcoFlow 1) reduces end-to-end CNN training time between 7-85%, and 2) improves end-to-end GAN training performance between 29-42%, compared to state-of-the-art CNN inference accelerators.

[Open-Source Artifact] We *open-source* both our Spatial Architecture Simulator for Machine Learning (SASiML) and the SASiML compiler to help enable the development of new dataflows and high-accuracy simulation environments for new spatial architectures and dataflows. This can be freely found at https://github.com/CMU-SAFARI/sasiml.

Index Terms—Neural Network Accelerators, Dataflow, Machine Learning, Hardware/Software Co-Design, Neural Network Training, Generative Adversarial Networks, Deep Learning.

1 Introduction

Deep convolutional neural networks (CNNs) have been widely adopted to solve hard problems in computer vision, natural language understanding, speech processing, medical applications, and more [1–37]. Transposed and dilated convolutions are the two key workhorses used to train CNNs, and run a variety of other deep learning models [38]. For example, both kernels are employed in applications requiring significant upsampling or downsampling to process high-resolution media such as image generation (using Generative Adversarial Networks (GANs) and Variational Auto-encoders (VAEs) [7, 39]), image super-resolution [40–42], and image segmentation [43, 44]. Additionally, more emerging machine learning works in text-to-speech generation [45], speech recognition [46], and audio synthesis [47] use dilated convolutions. Other experimental machine learning models, such as hierarchical capsule networks [48] and dilated residual networks [49] for improved image modeling, use both these convolution types.

- *Lois Orosa, Konstantinos Kanellopoulos, Juan Gómez-Luna and Onur Mutlu are with ETH Zurich.*
- *Skanda Koppula is with DeepMind.*
- *Yaman Umuroglu, Michaela Blott, and Kees Vissers are with Xilinx.*

Meanwhile, specialized architectures for CNN inference have gained traction to support the demand for low-cost deep learning on a variety of devices [50, 51]: Internet-of-Things devices (IoT) [52–55], phones, wearables [56], servers, and various embedded electronics [57, 58]. While these works demonstrate efficient execution of direct convolutions (i.e., regular or 'standard' convolutions), we find that existing dataflows for transposed and dilated convolutions are poorly tailored for these architectures, causing significant bottlenecks for emerging edge workloads that use transpose and dilated convolutions. Despite this issue, these workloads are of growing interest to manufacturers, because they can enable: (1) on-device model training for improved user data privacy [59–61], (2) high-resolution image generation critical for augmented reality [62, 63], (3) real-time speech recognition and generation [42, 45], and many other applications employing dilated and transposed convolutions [40, 41, 46, 47, 49, 64–72, 72–77].

To address this issue, we introduce *EcoFlow*, a new set of dataflows and data mappings designed to efficiently perform transposed and dilated convolutions on low-cost, small-scale spatial architectures that are already widely in-use for regular CNN inference. We identify key bottlenecks introduced by these operations, originating from padding and zero-insertions required to up- and downsample feature maps. EcoFlow circumvents these bottlenecks by meticulously orchestrating the data mapping and dataflows onto the target spatial architecture. By eliminating unnecessary operations, EcoFlow achieves significant improvements in performance and energy consumption, with minimal changes to the spatial array of a common CNN inference accelerator.

We improve on several prior works that propose specialized accelerators that target specifically either transposed convolutions [78–81], dilated convolutions [80, 81], or general sparsity [82–89]. We generalize, simplify, and significantly reduce the required architectural changes to support exactly the structured sparsity of these convolutional kernels. Our design goal is to avoid highly-specialized accelerator architectures that are markedly different from common and well-understood spatial architectures (i.e., a matrix of processing elements working in a systolic array fashion) optimized for direct convolutions. Re-use of existing hardware architectural designs permits lower testing and manufacturing costs. EcoFlow could also inspire the optimization of dataflows for spatial architectures designed to accelerate other applications such as genome sequence analysis [90–96].

We make the following key contributions:

- We propose EcoFlow, a new set of dataflows and data mappings that enable efficient execution of transpose and dilated convolutions on CNN inference accelerators by introducing minimal hardware changes (Section 4).
- We develop a cycle-accurate spatial architecture simulator to evaluate EcoFlow. Our architectural simulator includes TPU [97], Eyeriss [50], and EcoFlow models, and it supports efficient execution of transposed, dilated, and direct convolutions (Section 5).
- We comprehensively evaluate the performance and energy efficiency of EcoFlow. Our evaluation shows that EcoFlow: 1) reduces end-to-end CNN training time between 7-85%, and 2) improves end-to-end GAN training performance between 29-42%, compared to state-of-the-art CNN inference accelerators.

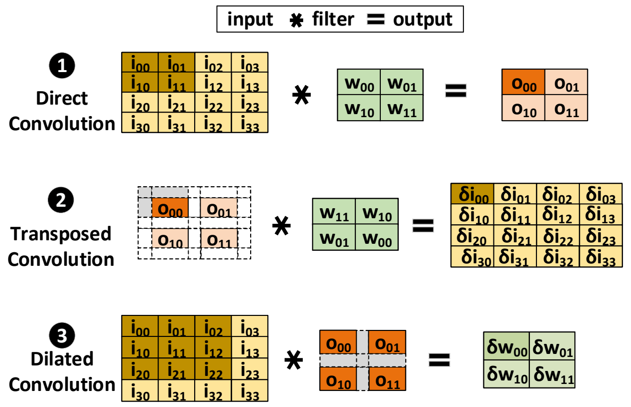
2 Background

A deep convolutional neural network (CNN) is a neural network with one or more convolutional layers. A convolutional layer in a CNN applies a sliding filter to a 2D or 3D matrix that represents the input image or intermediate layer input. The input and output matrices to a convolutional layer are referred to as the *input feature map (ifmap)* and *output feature map (ofmap)*, respectively. The *filter* (or *kernel*) is the sliding filter that is applied to the ifmap to calculate the ofmap. In each convolutional layer, there are usually multiple filters applied in parallel to the ifmap, producing multiple output matrices that compose the ofmap. The stride of a convolution refers to the size of the step that the convolutional filter takes while sliding through the ifmap. The core operation of a CNN is a multiply-and-accumulate (MAC) operation. Modern CNNs may have up to 10^{18} MACs performed during one forward evaluation [98]. Most of these MAC operations are performed in the convolutional layers [99].

Inference is the production phase where the CNN classifies unknown images. Before performing inference in production, the network is trained with and algorithm called backpropagation, that includes the calculation of input gradients and filter gradients. For a detailed treatment of CNN operation and gradient computation, we refer the reader to [4, 100–110].

2.1 Different Types of Convolutions in CNNs

Figure 1 illustrates the three main types of convolutions we can find in convolutional neural networks. This examples shows the case for the CNN training phase of a convolutional neural network, where we find direct convolutions in the forward pass, and transposed and dilated convolutions in the backward pass.



input ✱ filter = output
1 Direct Convolution
i00 i01 i02 i03 i10 i11 i12 i13 i20 i21 i22 i23 i30 i31 i32 i33
W00 W01 W10 W11
O00 O01 O10 O11
2 Transposed Convolution
O00 O01 O10 O11
W11 W10 W01 W00
δi00 δi01 δi02 δi03 δi10 δi11 δi12 δi13 δi20 δi21 δi22 δi23 δi30 δi31 δi32 δi33
3 Dilated Convolution
i00 i01 i02 i03 i10 i11 i12 i13 i20 i21 i22 i23 i30 i31 i32 i33
O00 O01 O10 O11
δW00 δW01 δW10 δW11

Fig. 1: Different types of convolutions with an example 4x4 input, 2x2 filter, and stride 2 used in the CNN training phase.

2.1.1 Direct Convolution

A direct convolution (also known as convolution, standard convolution, or regular convolution) is one of the most common operations in convolutional neural networks (in both inference and training), and other variants of CNNs [7, 39, 45–49, 64, 65, 111, 112]. A direct convolution is performed by sliding the filter (W_{xy}) over the input (i_{xy}) with a specific stride (stride 2 in the example ❶ in Figure 1), generally starting at the top left corner, so as to move the filter to the boundary of the input (❶ in Figure 1).

2.1.2 Transposed Convolution

A transposed convolution operation forms the same connectivity as a direct convolution but in the backward direction, which requires upsampling the input into an output of larger dimensions. Transposed convolutions are commonly used in CNN training and in emerging CNN workloads [40, 41, 46, 47, 49, 64, 65, 67–72]. Figure 1 ❷ shows an example that calculates the input gradients (δi_{xy}) in the backward propagation pass of CNN training. A transposed convolution is computed by convolving the error matrix (O_{xy}) with the forward pass filter (W_{xy}) rotated 180°. Transposed convolutions introduce zero padding into the error matrix to produce an output of larger dimensions, up-sampling the backpropagated errors. The error matrix might require zero-padding in the borders, as in Figure 1 ❷. If the stride is greater than one (the example Figure 1 ❷ has stride 2), the error matrix also require internal zero padding as zero-valued rows and columns.

2.1.3 Dilated Convolution

Dilated convolutions are commonly used in CNN training and in emerging CNN workloads [40, 41, 46, 47, 49, 64, 65, 73–77]. Figure 1 ❸ shows a dilated convolution example that calculates the filter gradients (δW_{xy}) with dilation rate = 2 (i.e., stride 2) in the backward propagation pass of CNN training. A dilated convolution is computed by convolving the input (i_{xy}) with a padded filter (O_{xy}) to augment its dimensions. This convolution inserts zero padding as rows and columns in the filter when the dilatation rate (i.e., the stride of the convolution when training a CNN) is greater than one. A dilation rate of 1 does not introduce any padding in the filter.

2.2 Spatial Architectures for CNN Inference

A spatial compute array is the key component in many popular low-cost CNN accelerators [50, 58, 97, 113–123]. A spatial architecture consists of a matrix of simple processing elements (PEs), interconnected with one or several internal networks. Each PE is able to perform a MAC operation. By orchestrating data into and out of the PE network, spatial architectures can efficiently implement either matrix multiplications or convolutions. Examples of spatial architectures include Eyeriss V1/V2 [50, 113], Google's TPU [97, 117], NVIDIA's CUDA Tensor Cores [124], Nanofabrics [125], TRIPS [126], RAW [127], SmartMemories [128], FlexFlow [114–116], SCNN [129], and Morph [130].

Figure 2 illustrates the core elements of a common spatial architecture for CNN inference. At the core is an array of interconnected PEs. Data is cached on a global on-chip buffer, which utilizes various network-on-chips (NoCs) to exchange data with the PE array. In common designs [50, 114], this network enables data transfer between vertically adjacent PEs, simultaneous broadcast to all PEs, and multicasting values to individual sets of PEs. On the left of Figure 2 we can see the off-chip memory that stores temporary data that overflows the global buffer, and the complete set of ifmaps, filters, and final ofmaps. The internal architecture of the PEs (right side of Figure 2) can differ slightly, based on the chosen dataflow, accelerator function (e.g. sparse/non-sparse CNN acceleration), and other optimizations (e.g. reduced precision, clock-gating). PEs generally store small amounts of weight or partial sum data which is reused during dataflow [50, 131].

2.3 CNN Dataflows on Spatial Architectures

We describe the most widely-used dataflows for performing convolutions in spatial architectures, used to evaluate EcoFlow in Section 6. An in-depth discussion of each dataflow can be found in [50].

Convolution Dataflows. Row stationary (RS) [50, 132] is a state-of-the-art dataflow for performing convolutions in spatial architectures. The RS dataflow attempts to minimize the overall energy consumed by off-chip data accesses by re-using the convolutional filters and ifmaps. RS minimizes data movement across all data types by effectively assigning each PE a 1D convolution to perform. The results of these 1D convolutions (or partial sums) are accumulated with other partial sums from other PEs to produce the final ofmap. RS has been shown to be the most energy efficient dataflow on spatial architectures [50], compared to Weight Stationary (WS)[133–135] and Output Stationary (OS) [136–138] dataflows.

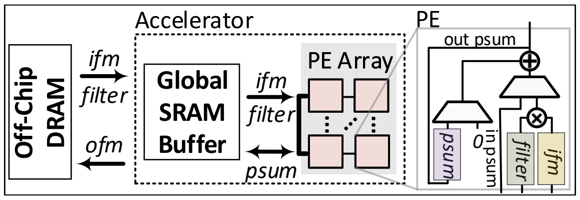


Fig. 2: Common Spatial architecture for CNN inference acceleration.

Although previous works claim that the choice of dataflow is not critical for direct convolutions [139], in this work we demonstrate that this choice *does* matter for transposed and dilated convolutions. Using direct convolution dataflows for transposed and dilated convolutions can result into low performance and poor energy efficiency.

Matrix Multiplication Dataflows. Lowering a convolution into a matrix multiplication is a well known technique that is used today in many CNN frameworks and accelerators, i.e., TPUs [97, 117]. For a detailed explanation of the lowering process, we refer the reader to [140]. After lowering, several dataflows can be used for the matrix multiplication [141]. A common approach uses an output stationary dataflow in which partial sums are accumulated locally, and inputs are forwarded to adjacent rows [50]. The matrices are fed into the PE array from the top and left edges of the array [113]. This is the approach used in our reference implementation in Section 6.

3 Motivation and Goal

We describe the main inefficiencies of transpose and dilated convolutions, and how related works require a specialized accelerators to solve this problem (Section 3.1). Our goal in this paper is to introduce minimal changes to an existing DNN inference accelerator to perform tranpose and dilated convolutions (Section 3.2).

3.1 Inefficiencies of Transposed and Dilated Convolutions

To understand the mechanics and bottlenecks of transpose and dilated convolution, we analyze the backward pass of CNN training on representative convolutional layers with different strides from two common CNNs, ResNet-50 [2] and AlexNet [101]. Figure 3 shows the percentage of multiplications by zero required to compute both transposed and dilated convolutions. We observe that for strides larger than 1, the zero multiplications dominate utilization by large margins. For example, more than 70% of multiplications for 2-stride convolutions are zero. The larger the stride, the larger the number of zero multiplications.

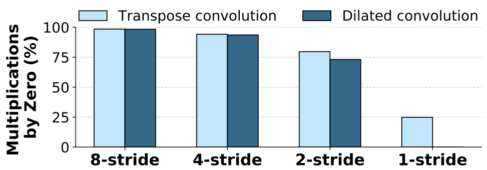


Fig. 3: Padding-induced zero multiplications in transpose and dilated convolutions during input and filter gradient calculation of representative CNN layers with different strides.

We make two observations. First, the PEs that execute zero operations *cannot* be used to perform useful operations, which causes resource under-utilization. Second, although the result of the multiplication is zero, inputs coming from other PEs might need to be accumulated and transmitted to the next node, which practically increases the latency of useful computations and reduce performance.

3.1.1 Analyzing Transpose Convolutions

Performing a transposed convolution in a spatial architecture designed for CNN inference requires significant padding to obtain the correct ofmap dimensions (i.e., up-sampling). Figure 4 shows two examples of the required padding in the input for obtaining the desired up-sampled ofmap[1]. In the example, layer Ⓐ requires 40 outer padding elements in the inputs (81% of the matrix), and layer Ⓑ requires 40 outer padding elements and 5 inner padding elements in the inputs (92% of the matrix).

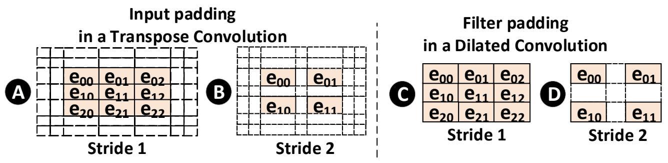

Input padding in a Transpose Convolution
Filter padding in a Dilated Convolution
A
B
C
D
e_{00} e_{01} e_{02}
e_{10} e_{11} e_{12}
e_{20} e_{21} e_{22}
e_{00} e_{01}
e_{10} e_{11}
Stride 1
Stride 2
Stride 1
Stride 2

Fig. 4: Example of the zero-padding required to calculate transpose and dilated convolutions.

We can formulate the amount of padding required by a particular transposed convolution by considering ifmap, stride, filter sizes. For a $N \times N$ ifmap, $K \times K$ filter, and stride S, the number of inner padding elements is given by $[S(N-1)+1]^2 - N^2$. The number of outer padding elements is given by $4(K-1)[S(N-1)+1] + 4(K-1)^2$. The total number of zero-padding elements increases *linearly with the ifmap size*, and *quadratically with the stride*.

Transposed convolutions are used for upsampling a input to produce a high-resolution output feature map or media. For example, semantic segmentation [142] and super-resolution [40] CNNs output images that are of the same or higher resolution than their input. Generative Adversarial Networks [10, 12, 143] use transposed convolutions for the same purpose.

1. The higher the stride, the higher the up-sampling.

Existing proposals. There are several works that propose to accelerate transposed convolutions with specialized GAN accelerators [80–82, 144, 145]. Although these works achieve significant performance and energy improvements, they do it at the cost of designing a specialized accelerator for GANs instead of maintaining a simple, efficient, and more general spatial architecture optimized for CNN inference.

3.1.2 Analyzing Dilated Convolutions

Figure 4 illustrates two examples (Ⓒ and Ⓓ) of the required filter zero padding in a dilated convolution. Unlike in transposed convolution, the error matrix is only padded internally. In Ⓒ, the stride is one, so the filter gradients can be calculated without padding. When the stride is larger than one, filter gradient calculation requires inner padding. Ⓓ shows an example of this, with stride 2. 56% of the padded error matrix is zero. The amount of inner padding follows the same trend as above, increasing linearly with the ifmap size and quadratically with the stride.

Dilated convolutions are used in the forward pass of a handful of emerging, state-of-art classification networks [49, 146, 147]. Dilated convolutions are also used for aiding visual interpretation of CNNs [148].

Existing proposals. DT-CNN [81] proposes an specialized hardware accelerator to perform both transposed and dilated convolutions using delay cells. Unlike EcoFlow, DT-CNN is a specialized architecture customized for optimizing image segmentation workloads.

3.2 Goal

Our proposal builds on two key observations: (1) the padding required to perform transposed and dilated convolutions on spatial architectures has a very negative effect on efficiency, and (2) the padding is strictly determined by the characteristics of the convolution and the dimensions of the feature maps and kernel, and thus the location of zero-values is static and deterministic. Our goal in this work is to exploit these two observations in order to (1) eliminate zero padding to avoid low resource occupation, (2) minimize energy and memory requirements, (3) maximize throughput, and (4) introduce minimal changes to the spatial architecture of common CNN inference accelerators. To this end, we develop EcoFlow.

4 EcoFlow

We introduce EcoFlow, a new set of dataflows and data mapping algorithms for calculating transpose convolutions (Section 4.1) and dilated convolutions (Section 4.2) in spatial architectures of CNN accelerators that are optimized for executing direct convolutions.

The core idea of EcoFlow is to meticulously orchestrate dataflow and map computation as to avoid zero padding and occupy PEs with only useful operations. EcoFlow efficiently reuses the spatial architecture used for direct convolutions to execute both transpose and dilated convolutions efficiently. One of the main characteristics of EcoFlow is that it can be mapped to existing CNN inference spatial architectures with minimal hardware changes, which allows efficient execution of transposed, dilated, and direct convolutions. EcoFlow requires small modifications in the network-on-chip to enable efficient data movements without wasting hardware resources.

The dataflow and mapping onto hardware is computed at *compile time*. EcoFlow's mapping is more complex than other state-of-the-art dataflows, but this added complexity is a one-time cost during the initial compilation step. The compiler calculates a Finite State Machine (FSM) that is loaded into the PEs to perform the convolutions at runtime. We explain the details of the hardware architecture in Section 4.4.

4.1 Transpose Convolutions

In this section, we explain the steps EcoFlow takes during compilation time (Section 4.1.1) and runtime (Section 4.1.2) to perform transposed convolutions.

Without loss of generality, we use an example of the transposed convolution that calculates the input gradients in the CNN training algorithm. In this context, the input of the convolution is the padded error (the amount of padding depends on stride in the forward pass), the filter corresponds to the rotated filter from the forward pass, and the output of the convolution are the calculated input gradients.

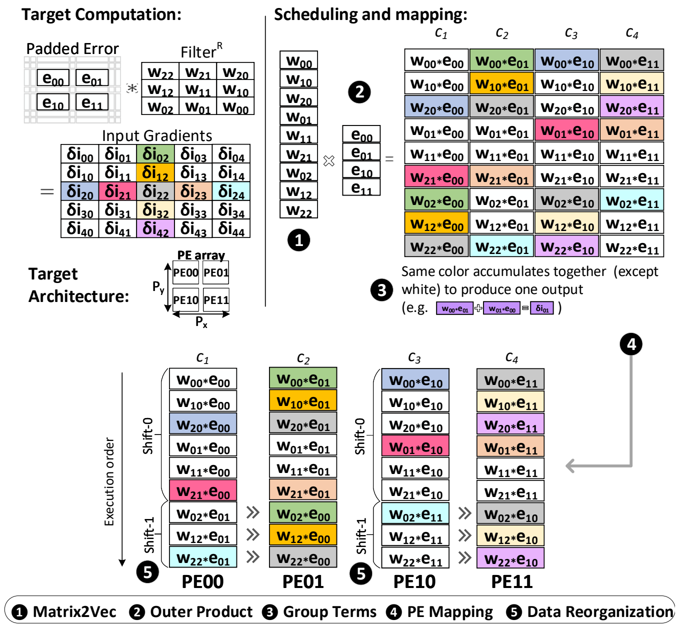


Fig. 5: Example of transposed convolution for calculating the input gradients on CNN training algorithm using EcoFlow. The symbol >> represents a column element `shift` by one.

4.1.1 Compilation Time.

The EcoFlow compiler determines (1) the computation scheduling required to compute the (transposed) convolution and (2) the mapping of computations onto the architecture's PEs array.

EcoFlow follows five steps to calculate the **computation scheduling** and **mapping**. To improve clarity, we walk through each step using the example in Figure 5: a transposed convolution with stride 2, 5×5 output (i.e., input gradients), 3×3 filter (i.e., rotated filter), and 7×7 input (i.e., padded error) reshaped using padding from the original 2×2 error):

❶ The EcoFlow compiler converts the rotated filter and the error matrix into symbolic vectors. In Figure 5, these vectors have dimensions 9×1 and 4×1, respectively.

❷ The compiler performs the symbolic outer product of both vectors by multiplying all elements of the filter by all elements of the error matrix. The resulting matrix contains all multiplications required to perform the transposed convolution for input gradient calculation. Each gradient is the sum of some subset of these products. Notably, this matrix does *not* contain any zero multiplication due to padding. In our example, this matrix has dimension 9×4.

❸ EcoFlow determines which matrix elements have to be accumulated together to produce a single input gradient, and marks them with the same *label*. The labels are determined by doing a transposed convolution with placeholder symbols. In the example, cells with the same color represent matrix elements with the same label. The exception to this are the white cells, which produce a single gradient by themselves; white cells do not need to be accumulated with other values.

❹ The compiler assigns each column of symbolic computations to a different PE. The mapping assigns consecutive columns to consecutive PEs, from top to bottom and from left to right in the PE array. The number of PEs used by EcoFlow is equal to the dimensions of the error matrix. In the example, the PE array is composed by 2×2 array, shown in the bottom left. This mapping can be reorganized to reduce the number of required PEs (see *Grouping*).

❺ The multiplications are reorganized with the goal of leveraging local point-to-point network to accumulate partial sums across connected, vertically-adjacent PEs. EcoFlow maps multiplications that must accumulate together either into the same PE, or across vertical PEs. The reorganization consists of *circular shifting* of these multiplication blocks across horizontal PEs. Each block shifts $\lfloor \frac{w\_idx}{W_x \times stride} \rfloor$ PEs over, where W_x is one dimension of the filter ($W_x = 3$ in the example) and w_idx is the index of the computation in the order of execution in each PE (e.g., $w_{00} * e_{00}$ has $w\_idx = 0$, $w_{10} * e_{00}$ has $w\_idx = 1$, etc.). Since the shifting is circular across horizontal PEs, computation blocks in the upper row of PEs shift from PE00 to PE01 and from PE01 to PE00 in the example. In the lower row, computation blocks shift between PE10 and PE11.

In Figure 5, the first six computation blocks are not shifted ($\lfloor \frac{w\_idx}{3 \times 2} \rfloor = 0$ for $0 \leq w\_idx < 6$), but the next three blocks are shifted over to the horizontally adjacent PE ($\lfloor \frac{w\_idx}{3 \times 2} \rfloor = 1$ for $6 \leq w\_idx \leq 9$). As a result of this reorganization, all the data that needs to be accumulated together is placed vertically. For example, the light blue multiply operations ($w_{22} * e_{01}$ and $w_{02} * e_{11}$) are shifted so they accumulate across on vertically adjacent PEs, PE00 and PE10.

The EcoFlow compiler also performs optimization techniques, called *grouping* and *expansion*, that allows to group

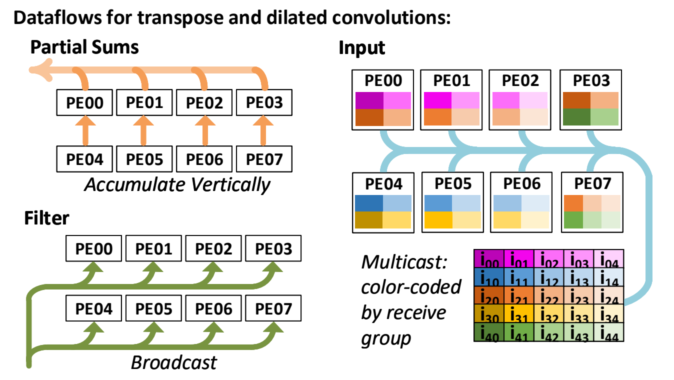


Fig. 6: Dataflow for each data type for transpose and dilated convolutions used in CNN training to calculate the input and filter gradients.

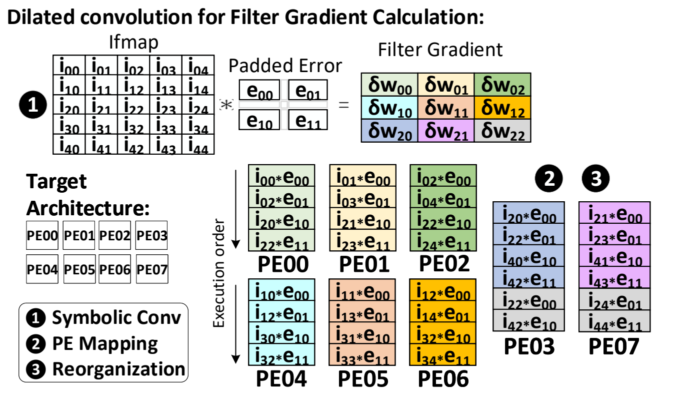


Fig. 7: Dilated convolution using EcoFlow to calculate the filter gradients in CNN training.

high-dimension convolutions into a small PE array, or to expand small-dimension convolutions into a large PE array.

4.1.2 Runtime.

The **dataflow** in EcoFlow leverages existing connections between vertically adjacent PEs and the on-chip multicast network present in spatial architectures. In this section, we describe data feeding and flow of the *partial sums*, *weights*, and *error maps* through the PE array. Figure 6 summarizes the dataflow of the three data types through the PE array.

Partial sums are accumulated locally and passed upward. Each filter-error product is added to a PE-local accumulation register. If a multiplication is the last one for a particular label (i.e., a color group) in one PE, the accumulated result is passed upward to the next PE in the same column. In Figure 5, the calculation of gradient element δi_{22} needs three steps. First, PE11 and PE01 compute $w_{00} \times e_{11}$ and $w_{20} \times e_{01}$, respectively. The results are stored in their internal accumulation registers. Next, the PE's compute $w_{02} \times e_{10}$ and $w_{22} \times e_{00}$, adding the result to the accumulation register. Third, PE11 passes the value in its accumulation register to PE01, and PE01 adds the received value to its accumulation register. The result is δi_{22}, which is then stored into the off-chip memory.

Filter weights are sequentially broadcast to all PEs and consumed every cycle. In Figure 5, the first set of multiplications use w_{00}, which is used by all PEs ($w_{00} \times e_{00}$, $w_{00} \times e_{01}$, $w_{00} \times e_{10}$, $w_{00} \times e_{11}$). The next broadcast weight is w_{10}, and so on.

Error matrix elements are sequentially multicast to the PE array. Each PE maintains a list of multicast groups to which it is subscribed, and receives the error elements required. For example, in Figure 5, PE00 receives the multicast groups $\{e_{00}, e_{01}\}$. Multicast groups are determined at compile time and loaded into each PE as part of an FSM.

4.2 Dilated Convolution

A dilated convolution is a direct convolution with a modified kernel (i.e., padded kernel) to match the desired output dimensions. Without loss of generality, we use an example of dilated convolution that calculates the filter gradients in the CNN training algorithm. In this context, the input of the convolution is the ifmap from the forward pass, the filter corresponds to the padded error (the amount of padding depends on stride in the forward pass), and the output of the convolution are the calculated filter gradients.

4.2.1 Compilation Time.

For a dilated convolution, EcoFlow performs computation scheduling and data mapping using a three-step process. For clarity, we walk through compilation using Figure 7 which illustrates filter gradient calculation with a 5×4 ifmap, 3×3 filter, and stride 2 convolutional layer.

❶ EcoFlow performs a symbolic convolution between the ifmap and the padded errors, determining the symbolic computations required to produce the filter gradients. During this step, the compiler forms groups that accumulate together to produce a single gradient element. In Figure 7, multiplications of the same color accumulate together.

❷ EcoFlow provisionally assigns the calculation of each filter gradient to one PE, eliminating inter-PE communications. Based on the necessity to parallelize channels in a filter, and to avoid potential slowdowns associated with large error maps, the compiler automatically reorganizes and re-distributes the compute schedule using assignment expansion, as explained in Section 4.2.2.

❸ Finally, the compiler determines multicast groups for the ifmap for use during execution. In the next section, we describe the dataflow for the partial sums, error matrix, and ifmap.

4.2.2 Runtime.

EcoFlow uses a straightforward dataflow for calculating the filter gradients. Similar to the calculation of the input gradients, the calculation of the filter gradients adapts well to the underlying on-chip network in state-of-art spatial architectures. Figure 6 describes the three main dataflows.

Error matrix elements are broadcast to each PE simultaneously. In Figure 6, e_{00} is used by all PEs in their first cycle, and is the first error to be broadcast.

The *input matrix* is distributed to the PEs using a multicast pattern determined by the compiler. Figure 6 illustrates the input matrix multicast used for the convolution described in Figure 7. In Figure 6, we see that each PE is part of at least four receive groups, corresponding to the number input matrix elements required in its computation schedule.

Finally, *partial sums* are accumulated within the PE. Each PE is responsible for multiplying and accumulating all the data it receives, and for storing the resulting filter gradient into off-chip memory. With expansion, each PE follows a similar procedure: it multiplies and accumulates all the data mapped to it, and sends the final value to its vertical neighbor. The top PE, after performing all the local operations, accumulates any passed-in results, and writes the final gradient to memory.

4.3 Memory Management

We describe EcoFlow's data reuse using two concepts from [50]. First, a *PE set* is the subset of PEs used to run a 2D convolution. If the physical array is large enough, several PE sets can be mapped concurrently in the array. Second, a *processing pass* is the contained, simultaneous execution of 2D convolutions in the PE array. In a single transpose convolution processing pass, each input element is read once from the global buffer, and the partial sums are stored back to the global buffer only once.

For a transpose or a dilated) convolution, EcoFlow has three types of reuse: 1) it reuses the input values by storing them in the global buffer using them with different filters, 2) it reuses the filters by broadcasting and using them across multiple PEs, and 3) it accumulates the partial sums within the PE and across vertical PEs. The filters are streamed from DRAM directly to the PE registers, and the inputs and partial sums are stored in the global buffer for reuse between processing passes.

To map PE sets into a processing pass, EcoFlow uses five parameters: n, r, t, q and p. EcoFlow fits $r \times t$ PE sets. Every t PE sets share the same inputs with t filters, and every r PE sets that run on r channels accumulate their partial sums within the PE array. Also, a processing pass can process n inputs, p filters and q channels at the same time. These parameters depend on the size of the internal PE registers. EcoFlow exhausts reuse opportunities of inputs and partial sum across different processing passes.

To optimize these parameters and allocate global buffer space for inputs and partial sums, our compiler pass runs an optimization procedure that finds parameters that minimize energy consumption for a given hardware configuration.

4.4 Hardware Architecture

EcoFlow targets spatial architectures similar to those described in Section 2.2. We use Eyeriss [50] as the baseline architecture, and we incorporate changes to the on-chip network and PE array to support EcoFlow.

On-Chip Network Requirements. The baseline architecture uses four on-chip networks: 1) a filter broadcast network to send filter weights to the PE, 2) an ifmap multicast network to send a unique ifmap element to each PE (i.e., one multicast group per PE) 3) an ofmap network that delivers partial sums to the global buffer, and 4) a network of local unidirectional point-to-point links that transmit partial sums through PEs in a column.

EcoFlow requires an expansion of the multicast network, so that each PE in the array can belong to several multicast groups. For example, in Figure 7, PE02 belongs to these four multicast groups: $\{i_{02}, i_{04}, i_{22}, i_{24}\}$. The multi-cast group i_{02} is consists of PE00 and PE02, and likewise for other groups. To support this, we extend the original multicast network of Eyeriss [50]. To support an $R \times C$ array of PEs, Eyeriss has a vertical Y-bus consisting of R horizontal X-buses. Each X-bus has a row ID, and each PE has a column ID. These IDs are reconfigurable, allowing different layers to map onto the same array.

We extend this network to have several row IDs per X-bus, and several column IDs per PE. For a $N \times N$ filter with stride S, the total number of row IDs that each X-bus needs to store is given by $\lceil \frac{N}{S} \rceil$. The number of bits needed by each row ID is $\lceil (\log_2 2N - S) \rceil$. $2N - S$ quantifies the total number of groups in a row. The equations to calculate the column ID requirements are exactly the same. We size the ID registers to support the largest layers in the CNN. For example, AlexNet requires five 5-bit row IDs per bus, while ResNet-50 requires four 4-bit row IDs per bus.

We estimate the area overhead of our NoC modifications by accounting for the additional logic gates and storage elements required to support the worst case CNN evaluated in this work. The extra IDs and comparison logic affect all the PE multicast controllers within the PE array. Our results show that the additional changes in the NoC introduce a 2.9% area overhead in the PE array.

EcoFlow also uses larger bandwidth to keep all PEs continuously utilized. Table 1 shows the maximum bus width required by EcoFlow in the three networks to run at maximum throughput on all evaluated CNNs. First, EcoFlow requires a 64+16 bits wide multicast global input network (GIN) for filters+ifmaps (forward pass), for errors+filters (input gradient calculation), and for ifmaps+errors (filter gradient calculation). Second, EcoFlow requires a 64 bits wide global output network (GON) for ofmaps (forward pass), input gradients (input gradient calculation), and filter gradients (filter gradient calculation). Third, EcoFlow requires a 64 bits wide local network (Local) for transmitting psums between vertical PEs.

	GIN	GON	Local
Eyeriss	64 + 16 bits	64 bits	64 bits
EcoFlow	80 + 32 bits	64 bits	64 bits

TABLE 1: Bus bit width of the multicast global input (GIN), global output (GON), and local (Local) networks.

We observe that EcoFlow does not require additional bandwidth for GON and Local networks, and it requires 40% more bandwidth for the GIN network.

PE Requirements. Like a typical PE design, EcoFlow needs an FSM to orchestrate loads and stores to registers, accumulations, stores to the global buffer, and communication with its neighboring PE. The compiler generates these FSMs. EcoFlow accumulates in each PE a variable amount of partial sums before the PE sends the result to the above PE or to memory, and it needs to accumulate the corresponding partial sums together (e.g., the same colors needs to accumulated together in Figure 5). This requires a slightly more complex FSM in the PE, compared to row-stationary dataflow.

Memory Requirements. EcoFlow does not require a different memory hierarchy than other spatial architecture accelerators. We use commodity DRAM chips and a highly banked global buffer.

5 SASiML: The Spatial Architecture Simulator

To evaluate EcoFlow, we develop SASiML, a new cycle-accurate simulator that mimics the hardware of a spatial architecture. SASiML models all the components of the PEs, the network, and the memory hierarchy. Each component of SASiML can be fully microprogrammed, and all the latency and energy parameters are fully parametrizable. SASiML can estimate the latency and energy consumed by direct, transpose and dilated convolutions of a particular layer; many other metrics such as PE utilization and bandwidth can be measured. We also develop a new compiler that automatically generates the signals required by SASiML to execute a particular CNN layer. We *open-source* both the simulator and the compiler to help enable the development of new dataflows and high-accuracy simulation environments for new spatial architectures and dataflows. This can be freely found at https://github.com/CMU-SAFARI/sasiml.

5.1 The Simulator

SASiML models the on-chip hardware of a spatial architecture and off-chip DRAM memory. SASiML contains architecture models for Eyeriss [50] and TPU [97]. SASiML is extensible and fully programmable. The level of abstraction of SASiML is similar to RTL: we model a synchronous digital circuit in terms of the flow of digital signals (data) between hardware registers, and the logical operations performed on those signals. In addition to a timing simulator, SASiML is a functional simulator that propagates the input values through the PE array to get the output, which allows to validate that the implementation of the dataflow at microprogramming level is correctly implemented.

The simulator has three main components: (1) a PE array, each of which has a global buffer, local registers, pipelined multiply-and-accumulate unit, and input/output queues connected to neighbouring PEs (2) a network on chip that interconnects neighboring PEs and PEs to the global buffer, and (3) a highly banked global buffer (e.g., 27 banks in our evaluation in Section 6). All components update their state at every clock cycle.

The basic organization of the simulator is simple: 1) the components connect together according to the specific design of the PEs and networks, 2) all components are controlled through input and output signals that are microprogrammed, and 3) all components update their state cycle by cycle. All components of SASiML are configurable, including memory sizes, network bandwidth, energy parameters. We support two variants of PEs, one tailored for convolutions (e.g., Eyeriss) and one for tailored for matrix multiplications (e.g., TPUs).

5.2 The Compiler

For simplifying the generation of the microprogramming control signals for SASiML, we implement a compiler. The inputs to the compiler are all the characteristics of the hardware and the CNN layers (e.g., feature map and filter dimensions). SASiML can perform inference and training with row-stationary, TPU, or EcoFlow dataflows.

5.3 Validation

We validate SASiML by analyzing that the output values match the expected golden results, and that the timings and power consumption are similar to the results reported by a real chip Eyeriss accelerator [50]. We configure SASiML with the same row-stationary dataflow parameters and the same accelerator configuration as reported in [50]. Table 2 shows the execution time, power, total size of global buffer accesses, and total size of all DRAM accesses for both Eyeriss and SASiML while running inference on AlexNet [101]. We expect some variations because the Eyeriss paper [50] does not provide full detail about their exact procedure for measuring timing, and about their memory management mechanisms for convolutions with high filter/channel count that overflow the global buffer. We calculate the power based on the energy parameters for a 45nm technology node reported by Horowitz [149]. There are two challenges for validating the power. First, the technology node of the Eyeriss chip is 65nm, not 45nm. We address this by scaling the energy consumption up by a factor of 1.4, based on estimations obtained from previous studies [150]. Second, SASiML does not model the energy of many details that have a large influence in the energy consumption, such as the clock network, which consumes between 33-45% of the power [50]. We address this issue by using the Amdahl's law to estimate the total power consumption, so we are able to compare our results with the power consumed by the real chip [50].

		CONV5	**CONV4**	**CONV3**	**CONV2**	**CONV1**
SASiML	**Exec. Time**	12.5ms	18.8ms	25ms	39.5ms	15.2ms
	Power	207mW	*	*	*	273mW
	GB acc.	23.8MB	35.6MB	66MB	74MB	16.8MB
	DRAM acc.	1.5MB	2.1MB	2.6MB	4.11MB	3.6MB
Eyeriss	**Exec. Time**	11ms	16ms	21.8ms	39.2ms	16.5ms
	Power	236mW	235mW	266mW	288mW	332mW
	GB acc.	24.9MB	37.4MB	50.2MB	77.6MB	18.5MB
	DRAM acc.	1.3MB	2.1MB	3.0MB	4.0MB	5.0MB

* Eyeriss [113] does not report the detailed power breakdown of these layers, so it is not possible to cross-verify these particular results.

TABLE 2: Comparison of execution time, global buffer (GB) accesses and DRAM accesses of SASiML and Eyeriss [50].

We observe that the results of SASiML are similar to the real chip Eyeriss measurements, and follow the same trends across layers. We make three key observations. First, the reported SASiML execution time is within 0.07% to 10% of the real Eyeriss accelerator. Second, the amount of data accessed in memory (GB and DRAM) by SASiML has a deviation of 0% to 24% from real measurements. Third, the power consumption reported by SASiML shows a good approximation, and the results are relatively accurate, despite the fact we could not model many details that are missing in the real Eyeriss chip paper.

We conclude that SASiML is an cycle accurate simulator that allows to model different spatial arrays with different NoCs and PE configurations at a microprogramming level of detail, which enables to functionally verify the correctness of dataflows and its implementation.

6 Evaluation

We evaluate transpose and dilated convolutions using workloads that contain both types of convolutions: CNN training (Section 6.2) and GAN training (Section 6.3).

6.1 Experimental Setup

We use the SASiML simulator and the SASiML compiler (Section 5) to evaluate EcoFlow. We model the energy of the accelerator with values obtained from a 45nm process [149]. We model DRAM energy using DRAMPower [151]. We compare EcoFlow to the row-stationary (RS) dataflow [50] used in Eyeriss and to a lowering-based convolutional dataflow used in TPUs [97, 117]. Table 3 shows the configuration of the target architecture used in evaluation. We chose an array of 13×15 PE elements, matching prior work and tuned with RS and TPU dataflows to fit the dimensions of the evaluated layers.

PE Array	13 x 15 PEs
PE Array Clock	200 MHz
PE Register File (ifmap, filter, psum)	75, 224, 24
PE Register Latency	1 cycle
Global Buffer	108KB / 27 banks
DRAM	4GB DDR4 1866MHz
Clock Gating	Zero Operations
Multiplier/Accumulator	2-stage/1-stage
I/O Queues	8 entries
On-chip Network Latency	1 cycle

TABLE 3: Configuration of the base CNN accelerator.

We implement a clock-gating mechanism that activates when the PE receives a zero value [50]. This is included in all our baselines. The NoC of Eyeriss and EcoFlow are similar, implementing dedicated networks for each data type. We use the on-chip networks described in Table 1. The TPU uses a much simpler NoC with only two uni-directional connections between neighbour PEs (for propagating input and filter values), while the partial sums are accumulated locally. We evaluate CNN training in Section 6.2 and GANs in Section 6.3.

To estimate the execution time of the end-to-end CNN training algorithm (i.e., execution time of all layers), we first profile the evaluated models in GPU and CPU to get the average breakdown of the execution time per layer, and we apply the Amdahl's law to calculate the expected total performance gains.

6.1.1 Optimizing CNN Training for EcoFlow

To get the maximum benefit from EcoFlow on CNN training, we need to replace pooling layers with larger strides when possible. Prior work demonstrates that pooling can be replaced by a convolutional layer with increased stride *without loss in accuracy* [152]. The authors show that for the tested CNNs, when they replace pooling with a convolutional layer with 2-stride, there is no accuracy loss. We corroborate and extend these results with experiments of our own on six larger, more recent CNNs. We train two variants of each CNN topology: one with pooling layers and one with pooling layers replaced with larger stride. We use the CIFAR-10 [153] and ImageNet [104] training and test datasets, and retain the default learning hyper-parameters given in [154, 155].

Table 4 summarizes our results. We observe that using a larger stride (Stride) instead of pooling layers marginally reduces accuracy (<2%), and in some cases, improves accuracy. This can be an acceptable trade-off in some applications, given the performance advantages.

	CIFAR-10			**ImageNet**		
CNN	**Original**	**Stride**	**Diff.**	**Original**	**Stride**	**Diff.**
ResNet-18 [2]	94.6%	94.2%	-0.4%	69.6%	69.5%	-0.1%
ResNet-101 [2]	94.6%	93.7%	-0.9%	77.6%	76.9%	-0.7%
DenseNet-201 [156]	94.0%	93.7%	-0.3%	78.6%	76.8%	-1.8%
VGG-19 [102]	92.5%	92.1%	-0.4	74.5%	74.6%	+0.1%
MobileNet-v2 [157]	90.7%	90.7%	+0.0%	74.7%	73.14%	-1.56%

TABLE 4: Accuracy comparison of CNNs that downsample using pooling layers (original) versus a larger stride (Stride).

6.2 CNN Training Evaluation

Table 5 details characteristics of 8 sample layers that we evaluate from six representative and widely-used CNNs, namely AlexNet [101], ResNet-50 [2], Shufflenet [158], Inception [103], Xception [159], and MobileNet [157].

Our complete evaluation tested 72 layers in total. These layer topologies and networks encompass a most of the layers used in popular networks, and include recent winning topologies of the ILSVRC competitions [104]. We use a batch size of four in our evaluations. We also evaluate the variant of each layer that includes the larger stride optimization described in Section 6.1.1. We denote these layers with a suffix of `opt`.

CNN	**Layer#**	**IFM**	**OFM**	**Filter**	**# Filts**	**Str.**	**Opt.**
AlexNet	CONV1	3x224x224	55x55	11x11	64	4	Yes
AlexNet	CONV2	64x31x31	27x27	5x5	192	1	Yes
ResNet-50	CONV3	128x57x57	28x28	3x3	128	2	No
ShuffleNet	CONV2	58x57x57	28x28	3x3	58	2	No
ShuffleNet	CONV5	232x7x7	7x7	1x1	232	1	No
Inception	CONV3	192x17x17	8x8	3x3	320	2	No
Xception	CONV3	728x29x29	14x14	3x3	1	2	No
MobileNet	CONV5	512x15x15	7x7	3x3	1	2	No

TABLE 5: Eight of the 72 evaluated layers from three CNNs.

We train using 16 bits instead of the 32 bits used in typical training algorithms. A previous work [160] demonstrates, training with BFLOAT16 can achieve the same accuracy as training with FP32.

6.2.1 Performance results.

Figure 8 shows the speedup of input gradient calculation through each layer in TPU, RS and EcoFlow dataflows, normalized to TPU. Similarly, Figure 9 shows the speedup of the filter gradient calculation for the three dataflows. The numbers on top of the TPU bars indicate the absolute execution time of TPU in *milliseconds*. The layers starting with the letter "o" (e.g., Alexnet o-CONV1) are the optimized versions of the layers (Section 6.1.1).

We make two main observations. First, the speedup of EcoFlow for calculating the input gradients compared to TPU and RS is very high for strides larger than 1. As shown in the figure, the speedup is close to 4x for stride 2 (e.g, resnet50 CONV3), 11x for stride 4 (Alexnet CONV1), and 52x for stride 8 (Alexnet opt CONV1). For stride 1, the speedup is from 0% (e.g., resnet50 CONV2) to 10% (Alexnet CONV3). Second, the speedup of EcoFlow for calculating the filter gradients compared to TPU and RS is also very large for stride larger than 1. The speedup is more than 3x for stride 2 (e.g., resnet50 CONV3), 15.6x for stride 4 (Alexnet CONV1), and 60.1x for stride 8 (Alexnet o-CONV1). We conclude that EcoFlow performs the backward pass much more efficiently than RS and TPUs, especially for strides larger than 1.

Table 6 shows the speedup of the evaluated end-to-end CNN networks.

	Speedup			Energy savings		
	TPU	Eyeriss	EcoFlow	TPU	Eyeriss	EcoFlow
Alexnet	1	0.94	1.83	1	0.97	1.38
ResNet-50	1	0.99	1.07	1	1.02	1.06
ShuffleNet	1	0.98	1.08	1	1	1.07
Inception	1	1.01	1.08	1	0.99	1.08
Xception	1	1.01	1.11	1	1.00	1.10
Mobilenet	1	1.01	1.09	1	1.00	1.08

TABLE 6: Speedup and energy savings of end-to-end CNN training of convolutional layers in different architectures, normalized to TPU (larger is better).

We make two observations. First, Alexnet greatly benefits from EcoFlow, because more than 80% of the execution time is dedicated to execute convolution layers following by pooling layers, or convolutional layer with stride larger than one. Second, ResNet-50, ShuffleNet, Inception, Xception and Mobilenet have smaller benefits because many of their convolutional layers have stride 1. We conclude that EcoFlow has very significant end-to-end benefits in networks that use strides in convolutional or pooling layers. Notice that other modern networks with larger strides, like EfficientNet [161], would also greatly benefit from EcoFlow.

6.2.2 *Energy Results*

In this section, we evaluate the energy consumption of EcoFlow. PEs are clock gated when idle, and all other parameters are defined in Table 3.

Figure 10 shows the energy comparison of TPU, RS and EcoFlow for the input gradient and filter gradient calculation. The breakdown of the energy includes DRAM (DRAM), global buffer (GBUFF), internal scratchpad memories (SPAD), the multipliers and the adders (ALU), and all on-chip networks (NoC). We make four main observations. First, the energy consumption of EcoFlow is much lower than TPU and RS for strides larger than 1. For example, the maximum energy savings of EcoFlow is 26x for Alexnet-opt-CONV1 compared to TPU. For the filter gradients, EcoFlow saves up to 8.3x energy. Second, the energy savings of EcoFlow are coming mainly from SPAD and NoC, whereas the energy consumed by DRAM is maintained. Third, for some layers with stride 1, EcoFlow consumes more energy than TPU and RS, caused by an increased DRAM energy consumption. Four, the energy of the filter gradient calculation is dominated by DRAM in some layers, e.g., resnet50-CONV4, resnet50-CONV2, since the errors in these layers have little reuse and are memory bound. This happens when the kernel size is small. EcoFlow is most energy efficient for layers that have stride and kernel larger than one.

6.3 GAN Evaluation

In this section, we evaluate GAN convolutional layers executed in the spatial architecture described in Table 3. We compare EcoFlow to GANAX [144], a hardware GAN accelerator that optimizes the execution of GANs by avoiding unnecessary zero computations. The key idea introduced by GANAX is to identify repeated patterns in the GAN computation and create different microprograms to execute each of this patterns. GANAX requires significant changes over an Eyeriss architecture, a new SIMD-MIMD execution model, a new ISA, a new global buffer to store instructions, and decoupling of the PEs into execution units and access units.

Table 7 shows the properties of the evaluated GAN layers. The layers are used by two representative GANs, namely CycleGAN [11], and pix2pix [9]. The layers of the discriminator (Disc) are regular convolutional layers, and the layers of the generator (Gen) are transposed convolutions. EcoFlow accelerates the backward pass of the discriminator and the forward pass of the generator.

CNN	Layer#	IFM	OFM	Filter	# Filts	Str.
CycleGAN	Disc-CONV3	64x114x114	56x56	4x4	128	2
CycleGAN	Gen-TCONV1	256x56x56	113x113	3x3	128	2
pix2pix	Disc-CONV6	128x130x130	64x64	4x4	256	2
pix2pix	Gen-TCONV41	512x64x64	130x130	4x4	128	2

TABLE 7: Evaluated layers from two widely-used GANs.

6.3.1 *Performance Results*

Figure 11 shows the speedup of the backward (Input, Filter) and the forward passes of selected GAN layers, for RS, TPU, GANAX, and EcoFlow dataflows, normalized to RS. We make two observations. First, EcoFlow performs on the order of 4x better than RS and TPU. Because GANs use strides larger than 1 instead of pooling layers, EcoFlow accelerates most convolutional layers. Second, EcoFlow performs 3-4x times better than GANAX in the filter gradient calculations, because GANAX does not provide a dataflow to accelerate gradient calculation. However, GANAX performs very similar to EcoFlow in the forward pass of the generative layers, and in the calculation of the input gradients.

Table 8 shows the speedup of the evaluated end-to-end GAN networks.

We make the key observation that EcoFlow has large benefits in end-to-end training of GAN networks. The training performance of EcoFlow outperforms even specialized GAN architectures like GANAX, because EcoFlow can accelerate filter gradient calculations.

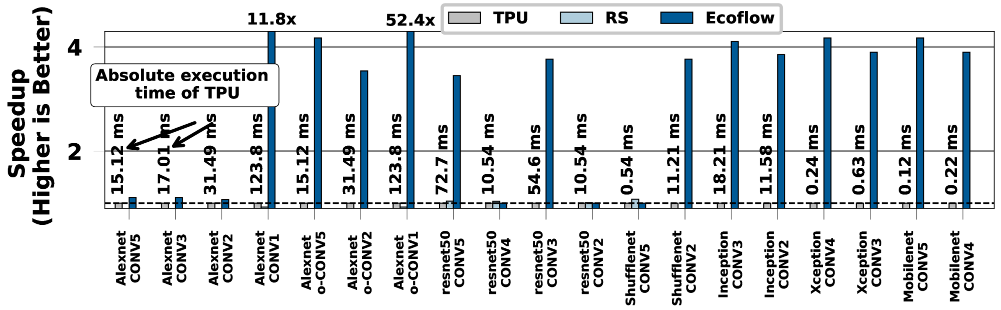


Fig. 8: Speedup of input gradient calculation, normalized to the TPU dataflow, and absolute TPU execution time.

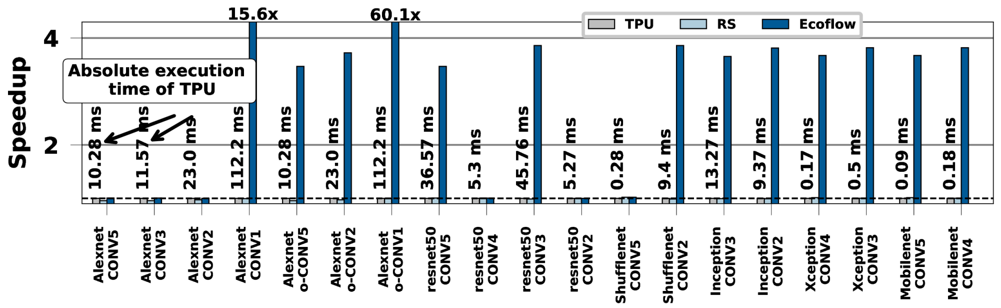


Fig. 9: Speedup of the filter gradient calculation, normalized to the TPU dataflow.

	Speedup				Energy savings			
	TPU	Eye.	GANAX	EcoFlow	TPU	Eye.	GANAX	EcoFlow
pix2pix	1	0.95	1.34	1.39	1	0.93	1.11	1.29
CGAN	1	0.94	1.37	1.42	1	1.04	1.32	1.37

TABLE 8: Speedup and energy savings (higher is better) of end-to-end training of two GANs, normalized to TPU.

6.3.2 Energy Results

Figure 12 shows the energy breakdown of the backward (Input,Filter) and the forward passes of selected GAN layers, for TPU, RS and EcoFlow dataflows, in absolute values. We could not compare to GANAX because some implementation details are missing in the paper (e.g., data reuse in each memory).

We make two main observations. First, the energy consumption of EcoFlow is much lower than the energy consumption of TPU and RS. For example, for the cyclegan-disc-CONV3 layer the energy savings of EcoFlow are in the order of 4x compared to TPU and RS. Second, similar to the results in CNN training (Section 6.2), the energy savings of EcoFlow are coming from reducing the energy in the SPADs, NOC and ALUs, whereas the DRAM energy consumption is very similar in all dataflows.

A key property of GANs is that they use larger strides instead of pooling layers, so most of the layers of state-of-the-art GANs benefit from EcoFlow.

7 Related Work

To our knowledge, this is the first work to design efficient dataflows to perform transpose and dilated convolutions on low-power CNN inference accelerators. We have already extensively compared EcoFlow to the Google TPU [97, 117], Eyeriss [50] and GANAX [144]. In this section, we describe other related works.

Specialized Inference Accelerators. Most existing specialized CNN accelerators are optimized for direct convolutions commonly used on CNN inference (e.g. Eyeriss [50], DaDiannao [162], Tetris [131], and Minerva [163]). WaveCore [164] and Google's TPUv2 [97] support CNN training, but suffer from challenges highlighted in Section 3. EcoFlow solves these issues, while introducing minimal changes to the CNN inference accelerator architecture.

Specialized Training Accelerators. Cambricon-Q [165] proposes a hybrid architecture consisting of an ASIC acceleration core and a near-data-processing (NDP) engine with the goal of improving the efficiency of statistic-based quantization. Equinox [166] proposes a custom inference accelerator that has the main goal of interleaving training during idle inference cycles. FPRaker [167] proposes a processing element that can perform MAC operations concurrently to accelerate DNN training. Unlike these works, EcoFlow targets a different problem, which is the inefficiency of convolutional dataflows used in DNN training and other DNN workloads.

Sparse Accelerators. Sparse accelerators [84, 129, 168–176] address the inefficiencies caused by zeros contained in sparse matrices, which is a fundamentally different problem than padding introduced by transpose and dilated convolutions. EcoFlow can be incorporated to these accelerators to obtain aggregated benefits.

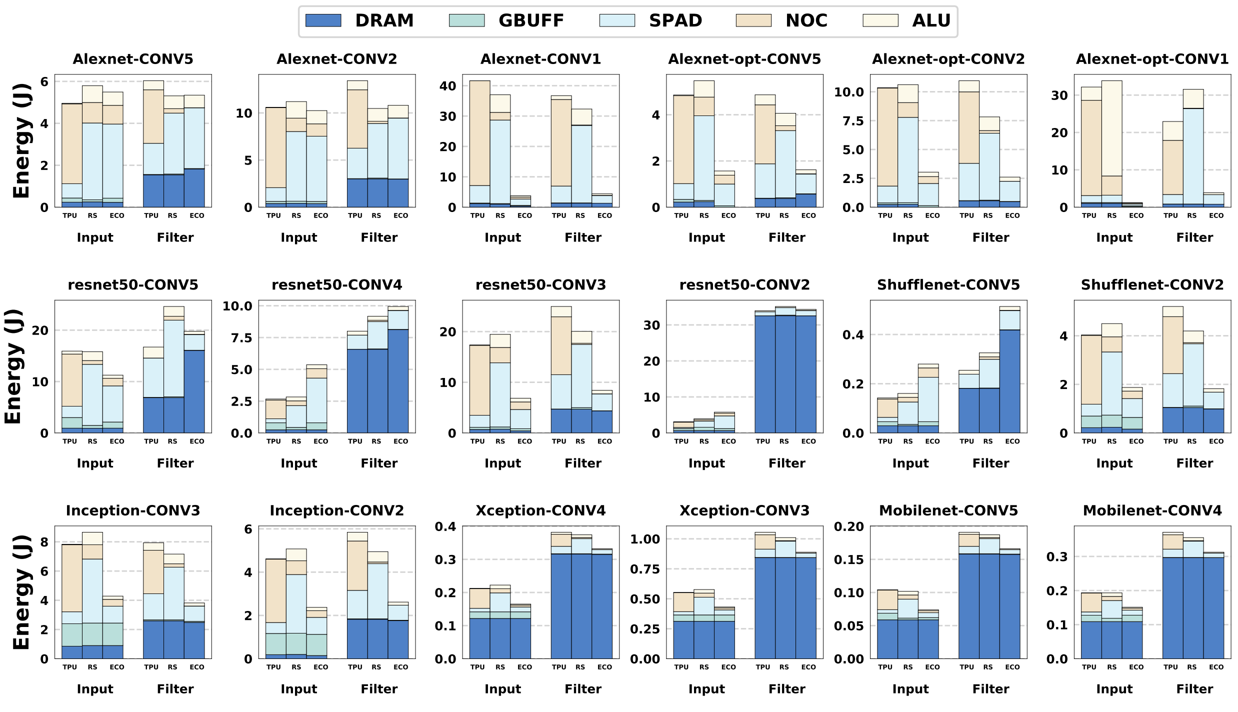


Fig. 10: Energy consumption of the evaluated layers.

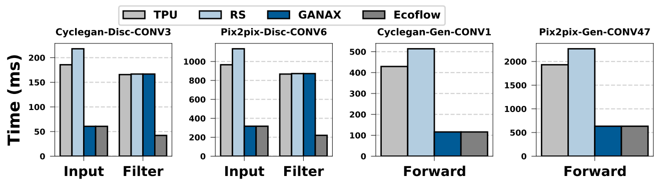


Fig. 11: Execution time of the evaluated GAN layers.

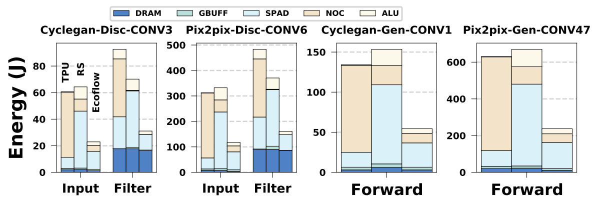


Fig. 12: Energy breakdown of the evaluated GAN layers.

GAN Accelerators. Prior works focus on accelerating GANs by performing transposed convolutions on new memory technologies [78, 79], FPGAs [177], and significantly modified spatial architectures [80, 81, 144]. Our work is unique in that 1) it focuses on both transposed and dilated convolutions, 2) it requires fewer hardware changes, 3) it proposes a multicast network that is able to effectively distribute the input data into the corresponding PEs, and 4) it evaluates GAN training and CNN training.

Winograd and Frequency-Domain Algorithms. Winograd is an alternative algorithm to perform matrix multiplications [178–180] that reduces the number of computations in CNNs via a series of data transformations. GradFlow, however, targets the orthogonal problem of the zero padding introduced by the training algorithm to upscale and backpropagate the errors through the network. Frequency domain backpropagation [181] replaces convolutions with simple point-wise multiplications, which avoids the inefficiencies of transposed and dilated convolutions. However, this approach requires computationally-intensive Fast Fourier Transforms (FFTs) and Inverse FFTs (IFFTs) at the boundary of every layer, and it requires a larger memory footprint.

Other Algorithms. Direct convolutions [182, 183] can avoid zero padding in the backward pass of some layers that meet some specific and restricted parameters. In contrast, EcoFlow is a general dataflow that can apply to the backward pass of any convolutional layer.

Other Techniques to Improve the Efficiency of DNN Workloads. There are other techniques to improve performance and reduce energy consumption in DNN workloads [131, 184–226]. For example, EDEN [187] reduces energy consumption by reducing the timing parameters and the voltage of DRAM, while [188] improves energy efficiency by reducing the voltage of SRAM in a DNN accelerator. Mensa [189, 190] tackle the problem of heterogeneity in ML workloads by considering several aspects (e.g., off-chip memory, on-chip buffers, compute-centric vs. data-centric acceleration, dataflow, etc.) to propose a family of accelerators where each accelerator tackles a different ML workload or layer. FloatPIM [203] is a Processing-in-Memory (PIM) [227, 228] approach that natively supports floating-point representation in resistive memories for CNN training workloads. Unlike EcoFlow, these approaches do not fundamentally re-design the dataflow of dilated and transposed convolutions to avoid inefficiencies in low-cost accelerators with limited hardware resources.

8 Conclusion

In this work, we aim to accelerate transpose and dilated convolutions in energy-efficient spatial architectures designed for CNN inference. We observe that a main source of inefficiencies of state-of-the-art CNN inference accelerators when executing transpose and dilated convolutions is the large amount of required zero padding, which diminishes the overall energy efficiency and performance.

To address this issue, we propose EcoFlow, a new set of mapping and dataflows for transpose and dilated convolutions. EcoFlow eliminates zero-padding by meticulously orchestrating the scheduling, dataflow, and data mapping to fit the characteristics of the target CNN inference accelerator. We show that, by introducing minimal changes to the CNN inference hardware, EcoFlow can significantly improve the energy efficiency and performance of common CNN training workloads. We conclude that EcoFlow enables commonly-used low-power CNN inference accelerators to efficiently perform CNN training, GAN training and other workloads that use transpose and dilated convolutions, with minimal hardware changes. We hope EcoFlow inspires future works on ML acceleration that take into account such important training workloads.

Acknowledgments

We thank the anonymous reviewers of ISCA'21/20/19, ASPLOS'20, and MICRO'20/19 for feedback, and the SAFARI Research Group members for valuable feedback and the stimulating intellectual environment they provide. We thank Taha Shahroodi for his feedback on earlier versions of this paper. We acknowledge the generous gifts provided by our industrial partners: Google, Huawei, Intel, Microsoft, VMware, and the Semiconductor Research Corporation.

References

[1] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, and T. N. Sainath, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal processing magazine*, 2012.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.

[3] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, "A convolutional neural network for modelling sentences," *arXiv preprint arXiv:1404.2188*, 2014.

[4] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, 2015.

[5] N. Ramachandran, S. C. Hong, M. J. Sime, and G. A. Wilson, "Diabetic retinopathy screening using deep neural network," *Clinical & experimental ophthalmology*, 2018.

[6] J. Redmon and A. Farhadi, "Yolo9000: better, faster, stronger," *arXiv preprint*, 2017.

[7] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *NIPS*, 2014.

[8] C. Han, K. Murao, S. SATOH, and H. Nakayama, "Learning more with less: GAN-based medical image augmentation," *Medical Imaging Technology*, 2019.

[9] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *CVPR*, 2017.

[10] C. Donahue, J. McAuley, and M. Puckette, "Adversarial audio synthesis," *arXiv preprint arXiv:1802.04208*, 2018.

[11] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *ICCV*, 2017.

[12] C. Vondrick, H. Pirsiavash, and A. Torralba, "Generating videos with scene dynamics," in *NIPS*, 2016.

[13] A. I. Khan, J. L. Shah, and M. M. Bhat, "CoroNet: A deep neural network for detection and diagnosis of COVID-19 from chest x-ray images," *Computer Methods and Programs in Biomedicine*, 2020.

[14] W. Shen, M. Zhou, F. Yang, C. Yang, and J. Tian, "Multi-scale convolutional neural networks for lung nodule classification," in *IPMI*, 2015.

[15] J.-Z. Cheng, D. Ni, Y.-H. Chou, J. Qin, C.-M. Tiu, Y.-C. Chang, C.-S. Huang, D. Shen, and C.-M. Chen, "Computer-aided diagnosis with deep learning architecture: applications to breast lesions in us images and pulmonary nodules in CT scans," *Scientific reports*, 2016.

[16] J. Kim, V. D. Calhoun, E. Shim, and J.-H. Lee, "Deep neural network with weight sparsity control and pre-training extracts hierarchical features and enhances classification performance: Evidence from whole-brain resting-state functional connectivity patterns of schizophrenia," *Neuroimage*, 2016.

[17] R. Li, W. Zhang, H.-I. Suk, L. Wang, J. Li, D. Shen, and S. Ji, "Deep learning based imaging data completion for improved brain disease diagnosis," in *MICCAI*, 2014.

[18] S. Feng, H. Zhou, and H. Dong, "Using deep neural network with small dataset to predict material defects," *Materials & Design*, 2019.

[19] Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: Automated testing of deep-neural-network-driven autonomous cars," in *ICSE*, 2018.

[20] R. Lindsey, A. Daluiski, S. Chopra, A. Lachapelle, M. Mozer, S. Sicular, D. Hanel, M. Gardner, A. Gupta, R. Hotchkiss *et al.*, "Deep neural network improves fracture detection by clinicians," *PNAS*, 2018.

[21] A. Y. Hannun, P. Rajpurkar, M. Haghpanahi, G. H. Tison, C. Bourn, M. P. Turakhia, and A. Y. Ng, "Cardiologist-level arrhythmia detection and classification in ambulatory electrocardiograms using a deep neural network," *Nature medicine*, 2019.

[22] K. Sharma, A. Aggarwal, T. Singhania, D. Gupta, and A. Khanna, "Hiding data in images using cryptography and deep neural network," *arXiv preprint arXiv:1912.10413*, 2019.

[23] J. Hermann, Z. Schätzle, and F. Noé, "Deep-neural-network solution of the electronic schrödinger equation," *Nature Chemistry*, 2020.

[24] C. Tian, J. Ma, C. Zhang, and P. Zhan, "A deep neural network model for short-term load forecast based on long short-term memory network and convolutional neural network," *Energies*, 2018.

[25] K. T. Schütt, M. Gastegger, A. Tkatchenko, K.-R. Müller, and R. J. Maurer, "Unifying machine learning and quantum chemistry with a deep neural network for molecular wavefunctions," *Nature communications*, 2019.

[26] W. Lu, G. Wan, Y. Zhou, X. Fu, P. Yuan, and S. Song, "DeepVCP: An end-to-end deep neural network for point cloud registration," in *ICCV*, 2019.

[27] L. Gao, X. Li, D. Liu, L. Wang, and Z. Yu, "A bidirectional deep neural network for accurate silicon color design," *Advanced Materials*, 2019.

[28] L. Li, L. G. Wang, F. L. Teixeira, C. Liu, A. Nehorai, and T. J. Cui, "DeepNIS: Deep neural network for nonlinear electromagnetic inverse scattering," *IEEE Transactions on Antennas and Propagation*, 2018.

[29] H. Su, W. Qi, C. Yang, J. Sandoval, G. Ferrigno, and E. De Momi, "Deep neural network approach in robot tool dynamics identification for bilateral teleoperation," *IEEE Robotics and Automation Letters*, 2020.

[30] N. Lubbers, J. S. Smith, and K. Barros, "Hierarchical modeling of molecular energies using a deep neural network," *The Journal of chemical physics*, 2018.

[31] W. Zhu and G. C. Beroza, "PhaseNet: a deep-neural-network-based seismic arrival-time picking method," *Geophysical Journal International*, 2019.

[32] A. Kaya, A. S. Keceli, C. Catal, H. Y. Yalic, H. Temucin, and B. Tekinerdogan, "Analysis of transfer learning for deep neural network based plant classification models," *Computers and electronics in agriculture*, 2019.

[33] D. K. Jain, P. Shamsolmoali, and P. Sehdev, "Extended deep neural network for facial emotion recognition," *Pattern Recognition Letters*, 2019.

[34] Z. Zhang, D. Robinson, and J. Tepper, "Detecting hate speech on twitter using a convolution-GRU based deep neural network," in *European semantic web conference*, 2018.

[35] C. Zhang, D. Song, Y. Chen, X. Feng, C. Lumezanu, W. Cheng, J. Ni, B. Zong, H. Chen, and N. V. Chawla, "A deep neural network for unsupervised anomaly detection and diagnosis in multivariate time series data," in *AAAI*, 2019.

[36] K. D. Julian, M. J. Kochenderfer, and M. P. Owen, "Deep neural network compression for aircraft collision avoidance systems," *Journal of Guidance, Control, and Dynamics*, 2019.

[37] 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 Design & Test*, 2020.

[38] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," in *Neural networks: Tricks of the trade*. Springer, 2012.

[39] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[40] C. Dong, C. C. Loy, K. He, and X. Tang, "Learning a deep convolutional network for image super-resolution," in *ECCV*, 2014.

[41] W. Shi, F. Jiang, and D. Zhao, "Single image super-resolution with dilated convolution based multi-scale information learning inception module," in *ICIP*, 2017.

[42] Z. Zhang, X. Wang, and C. Jung, "DCSR: Dilated convolutions for single image super-resolution," *IEEE Transactions on Image Processing*, 2018.

[43] R. Girshick, "Fast R-CNN," in *ICCV*, 2015.

[44] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *NIPS*, 2015.

[45] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," *arXiv preprint arXiv:1609.03499*, 2016.

[46] K. J. Han, R. Prieto, and T. Ma, "State-of-the-art speech recognition using multi-stream self-attention with dilated 1d convolutions," in *ASRU*, 2019.

[47] J. Pons, S. Pascual, G. Cengarle, and J. Serrà, "Upsampling artifacts in neural audio synthesis," *arXiv preprint arXiv:2010.14356*, 2020.

[48] S. Srivastava, P. Agarwal, G. Shroff, and L. Vig, "Hierarchical capsule based neural network architecture for sequence labeling," in *IJCNN*, 2019.

[49] F. Yu, V. Koltun, and T. Funkhouser, "Dilated residual networks," in *CVPR*, 2017.

[50] Y. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deeooli convolutional neural networks," *JSSC*, 2017.

[51] Y. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *ISCA*, 2016.

[52] X. Xu, Y. Ding, S. X. Hu, M. Niemier, J. Cong, Y. Hu, and Y. Shi, "Scaling for edge inference of deep neural networks," *Nature Electronics*, 2018.

[53] L. Liu, H. Li, and M. Gruteser, "Edge assisted real-time object detection for mobile augmented reality," in *MobiCom*, 2019.

[54] N. D. Lane and P. Warden, "The deep (learning) transformation of mobile and embedded computing," *Computer*, 2018.

[55] R. Hadidi, J. Cao, Y. Xie, B. Asgari, T. Krishna, and H. Kim, "Characterizing the deployment of deep neural networks on commercial edge devices," in *IISWC*, 2019.

[56] A. Mathur, N. D. Lane, S. Bhattacharya, A. Boran, C. Forlivesi, and F. Kawsar, "DeepEye: Resource efficient local execution of multiple deep vision models using wearable commodity hardware," in *MobiSys*, 2017.

[57] N. D. Lane, S. Bhattacharya, A. Mathur, P. Georgiev, C. Forlivesi, and F. Kawsar, "Squeezing deep learning into mobile and embedded devices," *IEEE Pervasive Computing*, 2017.

[58] R. Andri, L. Cavigelli, D. Rossi, and L. Benini, "YodaNN: An architecture for ultralow power binary-weight CNN acceleration," *TCAD*, 2017.

[59] S. Choi, J. Sim, M. Kang, and L.-S. Kim, "TrainWare: A memory optimized weight update architecture for on-device convolutional neural network training," in *ISLPED*, 2018.

[60] J. J. Stubbs, G. C. Birch, B. L. Woo, and C. G. Kouhestani, "Physical security assessment with convolutional neural network transfer learning," in *ICCST*, 2017.

[61] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in *ICDCS*, 2017.

[62] R. Ranjan and W.-S. Gan, "Natural listening over headphones in augmented reality using adaptive filtering techniques," *TASLP*, 2015.

[63] J. Donahue and K. Simonyan, "Large scale adversarial representation learning," in *NIPS*, 2019.

[64] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *CVPR*, 2015.

[65] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation," *arXiv preprint arXiv:1706.05587*, 2017.

[66] S. Ö. Arık, H. Jun, and G. Diamos, "Fast spectrogram inversion using multi-head convolutional neural networks," *IEEE Signal Processing Letters*, vol. 26, no. 1, pp. 94–98, 2018.

[67] Y. Li, J. Chang, Z. Wang, and C. Kong, "Inversion and reconstruction of supersonic cascade passage flow field based on a model comprising transposed network and residual network," *Physics of Fluids*, 2019.

[68] J. Linmans, J. Winkens, B. S. Veeling, T. S. Cohen, and M. Welling, "Sample efficient semantic segmentation using rotation equivariant convolutional networks," *arXiv preprint arXiv:1807.00583*, 2018.

[69] J. Gu and H. C. Kang, "Facial landmark detection by stacked hourglass network with transposed convolutional layer," *Journal of Korea Multimedia Society*, 2021.

[70] S. Yang, Z. Quan, M. Nie, and W. Yang, "Transpose: Keypoint localization via transformer," in *ICCV*, 2021.

[71] C. Zhang, Y. Zheng, B. Guo, C. Li, and N. Liao, "SCN: A novel shape classification algorithm based on convolutional neural network," *Symmetry*, 2021.

[72] L. A. Lim and H. Y. Keles, "Foreground segmentation using convolutional neural networks for multiscale feature encoding," *Pattern Recognition Letters*, 2018.

[73] L. Zhou, C. Zhang, and M. Wu, "D-LinkNet: LinkNet with pretrained encoder and dilated convolution for high resolution satellite imagery road extraction," in *CVPR Workshops*, 2018.

[74] B. Wang, Y. Lei, S. Tian, T. Wang, Y. Liu, P. Patel, A. B. Jani, H. Mao, W. J. Curran, T. Liu *et al.*, "Deeply supervised 3D fully convolutional networks with group dilated convolution for automatic MRI prostate segmentation," *Medical physics*, 2019.

[75] J.-Y. Liu and Y.-H. Yang, "Dilated convolution with dilated GRU for music source separation," *arXiv preprint arXiv:1906.01203*, 2019.

[76] D. Deb and J. Ventura, "An aggregated multicolumn dilated convolution network for perspective-free counting," in *CVPR Workshops*, 2018.

[77] S.-Y. Chang, B. Li, G. Simko, T. N. Sainath, A. Tripathi, A. van den Oord, and O. Vinyals, "Temporal modeling using dilated convolution and gating for voice-activity-detection," in *ICASSP*, 2018.

[78] F. Chen, L. Song, H. H. Li, and Y. Chen, "ZARA: A novel zero-free dataflow accelerator for generative adversarial networks in 3D ReRAM," in *DAC*, 2019.

[79] H. Mao, M. Song, T. Li, Y. Dai, and J. Shu, "LerGAN: A zero-free, low data movement and PIM-based GAN architecture," in *MICRO*, 2018.

[80] D. Im, D. Han, S. Choi, S. Kang, and H.-J. Yoo, "DT-CNN: An energy-efficient dilated and transposed convolutional neural network processor for region of interest based image segmentation," *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2020.

[81] D. Im, D. Han, S. Choi, S. Kang, and H.-J. Yoo, "DT-CNN: Dilated and transposed convolution neural network accelerator for real-time image segmentation on mobile devices," in *ISCAS*, 2019.

[82] M. Song, J. Zhang, H. Chen, and T. Li, "Towards efficient microarchitectural design for accelerating unsupervised GAN-based deep learning," in *HPCA*, 2018.

[83] C. Zhu, K. Huang, S. Yang, Z. Zhu, H. Zhang, and H. Shen, "An efficient hardware accelerator for structured sparse convolutional neural networks on FPGAs," *arXiv preprint arXiv:2001.01955*, 2020.

[84] S. Zhang, Z. Du, L. Zhang, H. Lan, S. Liu, L. Li, Q. Guo, T. Chen, and Y. Chen, "Cambricon-X: An accelerator for sparse neural networks," in *MICRO*, 2016.

[85] K. Kanellopoulos, N. Vijaykumar, C. Giannoula, R. Azizi, S. Koppula, N. M. Ghiasi, T. Shahroodi, J. G. Luna, and O. Mutlu, "Smash: Co-designing software compression and hardware-accelerated indexing for efficient sparse matrix operations," in *MICRO*, 2019.

[86] A. K. Mishra, E. Nurvitadhi, G. Venkatesh, J. Pearce, and D. Marr, "Fine-grained accelerators for sparse machine learning workloads," in *ASP-DAC*, 2017.

[87] S. Pal, J. Beaumont, D.-H. Park, A. Amarnath, S. Feng, C. Chakrabarti, H.-S. Kim, D. Blaauw, T. Mudge, and R. Dreslinski, "OuterSPACE: An outer product based sparse matrix multiplication accelerator," in *HPCA*, 2018.

[88] Y. Umuroglu and M. Jahre, "An energy efficient column-major backend for FPGA SpMV accelerators," in *ICCD*, 2014.

[89] J. Cui and Q. Qiu, "Towards memristor based accelerator for sparse matrix vector multiplication," in *ISCAS*, 2016.

[90] D. S. Cali, G. S. Kalsi, Z. Bingöl, C. Firtina, L. Subramanian, J. S. Kim, R. Ausavarungnirun, M. Alser, J. Gomez-Luna, A. Boroumand *et al.*, "GenASM: A high-performance, low-power approximate string matching acceleration framework for genome sequence analysis," in *MICRO*, 2020.

[91] M. Alser, Z. Bingöl, D. S. Cali, J. Kim, S. Ghose, C. Alkan, and O. Mutlu, "Accelerating genome analysis: A primer on an ongoing journey," *IEEE Micro*, 2020.

[92] D. Senol Cali, J. S. Kim, S. Ghose, C. Alkan, and O. Mutlu, "Nanopore sequencing technology and tools for genome assembly: computational analysis of the current state, bottlenecks and future directions," *Briefings in bioinformatics*, 2019.

[93] G. Singh, M. Alser, D. S. Cali, D. Diamantopoulos, J. Gómez-Luna, H. Corporaal, and O. Mutlu, "FPGA-based near-memory acceleration of modern data-intensive applications," *IEEE Micro*, 2021.

[94] H. Xin, D. Lee, F. Hormozdiari, S. Yedkar, O. Mutlu, and C. Alkan, "Accelerating read mapping with FastHASH," in *BMC genomics*, 2013.

[95] M. Alser, H. Hassan, H. Xin, O. Ergin, O. Mutlu, and C. Alkan, "GateKeeper: a new hardware architecture for accelerating prealignment in DNA short read mapping," *Bioinformatics*, 2017.

[96] J. S. Kim, D. S. Cali, H. Xin, D. Lee, S. Ghose, M. Alser, H. Hassan, O. Ergin, C. Alkan, and O. Mutlu, "GRIM-Filter: Fast seed location filtering in DNA read mapping using processing-in-memory technologies," *BMC genomics*, 2018.

[97] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, and A. Borchers, "In-datacenter performance analysis of a tensor processing unit," in *ISCA*, 2017.

[98] Y. You, Z. Zhang, C.-J. Hsieh, J. Demmel, and K. Keutzer, "ImageNet training in minutes," in *ICPP*, 2018.

[99] H. Zhu, B. Zheng, B. Schroeder, G. Pekhimenko, and A. Phanishayee, "DNN-Train: Benchmarking and analyzing DNN training," *Training*, 2018.

[100] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel, "Handwritten digit recognition with a back-propagation network," in *NIPS*, 1990.

[101] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *NIPS*, 2012.

[102] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[103] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *CVPR*, 2015.

[104] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, and M. Bernstein, "ImageNet large scale visual recognition challenge," *IJCV*, 2015.

[105] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, 2015.

[106] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *ECCV*, 2014.

[107] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *NIPS*, 2014.

[108] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016.

[109] C. M. Bishop, *Pattern recognition and machine learning*. Springer, 2006.

[110] J. Bouvrie, "Notes on convolutional neural networks," 2006.

[111] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[112] J. Mao, W. Xu, Y. Yang, J. Wang, Z. Huang, and A. Yuille, "Deep captioning with multimodal recurrent neural networks (m-RNN)," *arXiv preprint arXiv:1412.6632*, 2014.

[113] Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *JETCAS*, 2019.

[114] W. Lu, G. Yan, J. Li, S. Gong, Y. Han, and X. Li, "Flexflow: A flexible dataflow accelerator architecture for convolutional neural networks," in *HPCA*, 2017.

[115] H. Kwon, A. Samajdar, and T. Krishna, "MAERI: Enabling flexible dataflow mapping over DNN accelerators via reconfigurable interconnects," in *ASPLOS*, 2018.

[116] H. Kwon, M. Pellauer, and T. Krishna, "MAESTRO: an open-source infrastructure for modeling dataflows within deep learning accelerators," *arXiv preprint arXiv:1805.02566*, 2018.

[117] N. P. Jouppi, D. H. Yoon, M. Ashcraft, M. Gottscho, T. B. Jablin, G. Kurian, J. Laudon, S. Li, P. Ma, X. Ma *et al.*, "Ten lessons from three generations shaped Google's TPUv4i," in *ISCA*, 2021.

[118] B. Khabbazan and S. Mirzakuchaki, "Design and implementation of a low-power, embedded cnn accelerator on a low-end FPGA," in *DSD*, 2019.

[119] X. Wei, Y. Liang, X. Li, C. H. Yu, P. Zhang, and J. Cong, "TGPA: tile-grained pipeline architecture for low latency CNN inference," in *ICCAD*, 2018, pp. 1–8.

[120] W. Choi, K. Choi, and J. Park, "Low cost convolutional neural network accelerator based on bi-directional filtering and bit-width reduction," *IEEE Access*, 2018.

[121] M. Wang and A. P. Chandrakasan, "Flexible low power CNN accelerator for edge computing with weight tuning," in *A-SSCC*, 2019.

[122] X. Hu, Y. Zeng, Z. Li, X. Zheng, S. Cai, and X. Xiong, "A resources-efficient configurable accelerator for deep convolutional neural networks," *IEEE Access*, 2019.

[123] S. Venkataramani, V. Srinivasan, W. Wang, S. Sen, J. Zhang, A. Agrawal, M. Kar, S. Jain, A. Mannari, H. Tran *et al.*, "RaPiD: AI accelerator for ultra-low precision training and inference," in *ISCA*, 2021.

[124] S. Markidis, S. W. Der Chien, E. Laure, I. B. Peng, and J. S. Vetter, "NVIDIA tensor core programmability, performance & precision," in *IPDPSW*, 2018.

[125] S. Copen Goldstein and M. Budiu, "NanoFabrics: spatial computing using molecular electronics," in *ISCA*, 2001.

[126] K. Sankaralingam, R. Nagarajan, H. Liu, C. Kim, J. Huh, D. Burger, S. W. Keckler, and C. R. Moore, "Exploiting ILP, TLP, and DLP with the polymorphous TRIPS architecture," in *ISCA*, 2003.

[127] W. Lee, R. Barua, M. Frank, D. Srikrishna, J. Babb, V. Sarkar, and S. Amarasinghe, "Space-time scheduling of instruction-level parallelism on a raw machine," in *ASPLOS*, 1998.

[128] K. Mai, T. Paaske, N. Jayasena, R. Ho, W. J. Dally, and M. Horowitz, "Smart memories: A modular reconfigurable architecture," *ISCA*, 2000.

[129] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally, "SCNN: An accelerator for compressed-sparse convolutional neural networks," in *ISCA*, 2017.

[130] K. Hegde, R. Agrawal, Y. Yao, and C. W. Fletcher, "Morph: Flexible acceleration for 3D CNN-based video understanding," in *MICRO*, 2018.

[131] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis, "Tetris: Scalable and efficient neural network acceleration with 3D memory," in *ASPLOS*, 2017.

[132] B. Zhang, H. Gu, K. Wang, and Y. Yang, "A novel conv acceleration strategy based on logical pe set segmentation for row stationary dataflow," *IEEE Transactions on Computers*, 2021.

[133] M. Sankaradas, V. Jakkula, S. Cadambi, S. Chakradhar, I. Durdanovic, E. Cosatto, and H. P. Graf, "A massively parallel coprocessor for convolutional neural networks," in *ASAP*, 2009.

[134] H.-J. Yoo, S. Park, K. Bong, D. Shin, J. Lee, and S. Choi, "A 1.93 tops/w scalable deep learning/inference processor with tetra-parallel mimd architecture for big data applications," in *ISSCC*, 2015.

[135] L. Cavigelli, D. Gschwend, C. Mayer, S. Willi, B. Muheim, and L. Benini, "Origami: A convolutional network accelerator," in *GLSVLSI*, 2015.

[136] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "ShiDianNao: Shifting vision processing closer to the sensor," in *ISCA*, 2015.

[137] M. Peemen, A. A. Setio, B. Mesman, and H. Corporaal, "Memory-centric accelerator design for convolutional neural networks," in *ICCD*, 2013.

[138] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *ICML*, 2015.

[139] X. Yang, M. Gao, J. Pu, A. Nayak, Q. Liu, S. E. Bell, J. O. Setter, K. Cao, H. Ha, and C. Kozyrakis, "DNN dataflow choice is overrated," *arXiv preprint arXiv:1809.04070*, 2018.

[140] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "CuDNN: Efficient primitives for deep learning," *arXiv preprint arXiv:1410.0759*, 2014.

[141] H. Kung and C. E. Leiserson, "Systolic arrays (for VLSI)," in *Sparse Matrix Proceedings*, 1979.

[142] H. Noh, S. Hong, and B. Han, "Learning deconvolution network for semantic segmentation," in *ICCV*, 2015.

[143] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.

[144] A. Yazdanbakhsh, K. Samadi, N. S. Kim, and H. Esmaeilzadeh, "GANAX: A unified MIMD-SIMD acceleration for generative adversarial networks," in *ISCA*, 2018.

[145] F. Shi, Z. Xu, T. Yuan, and S.-C. Zhu, "HUGE2: a highly untangled generative-model engine for edge-computing," *arXiv preprint arXiv:1907.11210*, 2019.

[146] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei, "Deformable convolutional networks," in *ICCV*, 2017.

[147] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," *arXiv preprint arXiv:1511.07122*, 2015.

[148] M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus, "Deconvolutional networks." in *CVPR*, 2010.

[149] M. Horowitz, "1.1 computing's energy problem (and what we can do about it)," in *ISSCC*, 2014.

[150] S. Rodriguez and B. Jacob, "Energy/power breakdown of pipelined nanometer caches (90nm/65nm/45nm/32nm)," in *ISLPED*, 2006.

[151] K. Chandrasekar, C. Weis, Y. Li, S. Goossens, M. Jung, O. Naji, B. Akesson, N. Wehn, and K. Goossens, "DRAMPower: Open-source DRAM power & energy estimation tool," *URL: http://www. drampower. info*, 2012.

[152] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for simplicity: The all convolutional net," *arXiv preprint arXiv:1412.6806*, 2014.

[153] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Tech. Rep., 2009.

[154] K. Liu, "pytorch-cifar," https://github.com/kuangliu/pytorch-cifar, 2019.

[155] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in PyTorch," 2017.

[156] F. Iandola, M. Moskewicz, S. Karayev, R. Girshick, T. Darrell, and K. Keutzer, "Densenet: Implementing efficient convnet descriptor pyramids," *arXiv preprint arXiv:1404.1869*, 2014.

[157] 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," *arXiv preprint arXiv:1704.04861*, 2017.

[158] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNET: An extremely efficient convolutional neural network for mobile devices," in *CVPR*, 2018.

[159] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *CVPR*, 2017.

[160] D. Kalamkar, D. Mudigere, N. Mellempudi, D. Das, K. Banerjee, S. Avancha, D. T. Vooturi, N. Jammalamadaka, J. Huang, H. Yuen *et al.*, "A study of bfloat16 for deep learning training," *arXiv preprint arXiv:1905.12322*, 2019.

[161] M. Tan and Q. V. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," *arXiv preprint arXiv:1905.11946*, 2019.

[162] 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 *MICRO*, 2014.

[163] B. Reagen, P. Whatmough, R. Adolf, S. Rama, H. Lee, S. K. Lee, J. M. Hernández-Lobato, G.-Y. Wei, and D. Brooks, "Minerva: Enabling low-power, highly-accurate deep neural network accelerators," in *ISCA*, 2016.

[164] S. Lym, A. Behroozi, W. Wen, G. Li, Y. Kwon, and M. Erez, "Mini-batch serialization: CNN training with inter-layer data reuse," *arXiv preprint arXiv:1810.00307*, 2018.

[165] Y. Zhao, C. Liu, Z. Du, Q. Guo, X. Hu, Y. Zhuang, Z. Zhang, X. Song, W. Li, X. Zhang *et al.*, "Cambricon-Q: a hybrid architecture for efficient training," in *ISCA*, 2021.

[166] M. P. Drumond Lages De Oliveira, L. Coulon, A. Pourhabibi Zarandi, A. C. Yüzügüler, B. Falsafi, and M. Jaggi, "Equinox: Training (for free) on a custom inference accelerator," in *MICRO*, 2021.

[167] O. M. Awad, M. Mahmoud, I. Edo, A. H. Zadeh, C. Bannon, A. Jayarajan, G. Pekhimenko, and A. Moshovos, "FPRaker: A processing element for accelerating neural network training," in *MICRO*, 2021.

[168] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[169] 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 *ISCA*, 2016.

[170] A. Aimar, H. Mostafa, E. Calabrese, A. Rios-Navarro, R. Tapiador-Morales, I.-A. Lungu, M. B. Milde, F. Corradi, A. Linares-Barranco, and S.-C. Liu, "Nullhop: A flexible convolutional neural network accelerator based on sparse representations of feature maps," *TNNLS*, 2018.

[171] 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," *arXiv preprint arXiv:1705.08922*, 2017.

[172] A. Page, A. Jafari, C. Shea, and T. Mohsenin, "SPARCNet: A hardware accelerator for efficient deployment of sparse convolutional networks," *JETC*, 2017.

[173] H. Nakahara, Y. Sada, M. Shimoda, K. Sayama, A. Jinguji, and S. Sato, "FPGA-based training accelerator utilizing sparseness of convolutional neural network," in *FPL*, 2019.

[174] J.-W. Jang, S. Lee, D. Kim, H. Park, A. S. Ardestani, Y. Choi, C. Kim, Y. Kim, H. Yu, H. Abdel-Aziz *et al.*, "Sparsity-aware and re-configurable NPU architecture for samsung flagship mobile SoC," in *ISCA*, 2021.

[175] H. Lu, L. Chang, C. Li, Z. Zhu, S. Lu, Y. Liu, and M. Zhang, "Distilling bit-level sparsity parallelism for general purpose deep learning acceleration," in *MICRO*, 2021.

[176] A. Gondimalla, N. Chesnut, M. Thottethodi, and T. Vijaykumar, "SparTen: A sparse tensor accelerator for convolutional neural networks," in *MICRO*, 2019.

[177] A. Yazdanbakhsh, M. Brzozowski, B. Khaleghi, S. Ghodrati, K. Samadi, N. S. Kim, and H. Esmaeilzadeh, "FlexiGAN: An end-to-end solution for FPGA acceleration of generative adversarial networks," in *FCCM*, 2018.

[178] M. Kim, C. Park, S. Kim, T. Hong, and W. W. Ro, "Efficient dilated-winograd convolutional neural networks," in *ICIP*, 2019.

[179] F. Shi, H. Li, Y. Gao, B. Kuschner, and S. Zhu, "Sparse winograd convolutional neural networks on small-scale systolic arrays," in *FPGA*, 2019.

[180] X. Liu, J. Pool, S. Han, and W. J. Dally, "Efficient sparse-winograd convolutional neural networks," *arXiv preprint arXiv:1802.06367*, 2018.

[181] J. H. Ko, B. Mudassar, T. Na, and S. Mukhopadhyay, "Design of an energy-efficient accelerator for training of convolutional neural networks using frequency-domain computation," in *DAC*, 2017.

[182] J. Zhang, F. Franchetti, and T. M. Low, "High performance zero-memory overhead direct convolutions," *arXiv preprint arXiv:1809.10170*, 2018.

[183] E. Georganas, S. Avancha, K. Banerjee, D. Kalamkar, G. Henry, H. Pabst, and A. Heinecke, "Anatomy of high-performance deep learning convolutions on SIMD architectures," in *SC*, 2018.

[184] D.-T. Nguyen, N.-M. Ho, and I.-J. Chang, "St-DRC: Stretchable DRAM refresh controller with no parity-overhead error correction scheme for energy-efficient DNNs," in *DAC*, 2019.

[185] D. T. Nguyen, H. Kim, H.-J. Lee, and I.-J. Chang, "An approximate memory architecture for a reduction of refresh power consumption in deep learning applications," in *ISCAS*, 2018.

[186] F. Tu, W. Wu, S. Yin, L. Liu, and S. Wei, "RANA: towards efficient neural acceleration with refresh-optimized embedded DRAM," in *ISCA*, 2018.

[187] S. Koppula, L. Orosa, A. G. Yağlıkçı, R. Azizi, T. Shahroodi, K. Kanellopoulos, and O. Mutlu, "EDEN: Enabling energy-efficient, high-performance deep neural network inference using approximate DRAM," in *MICRO*, 2019.

[188] N. Chandramoorthy, K. Swaminathan, M. Cochet, A. Paidimarri, S. Eldridge, R. V. Joshi, M. M. Ziegler, A. Buyuktosunoglu, and P. Bose, "Resilient low voltage accelerators for high energy efficiency," in *HPCA*, 2019.

[189] A. Boroumand, S. Ghose, B. Akin, R. Narayanaswami, G. F. Oliveira, X. Ma, E. Shiu, and O. Mutlu, "Google neural network models for edge devices: Analyzing and mitigating machine learning inference bottlenecks," in *PACT*, 2021.

[190] A. Boroumand, S. Ghose, B. Akin, R. Narayanaswami, G. F. Oliveira, X. Ma, E. Shiu, and O. Mutlu, "Mitigating edge machine learning inference bottlenecks: An empirical study on accelerating Google edge models," *arXiv preprint arXiv:2103.00768*, 2021.

[191] H. Kwon, L. Lai, M. Pellauer, T. Krishna, Y.-H. Chen, and V. Chandra, "Heterogeneous dataflow accelerators for multi-DNN workloads," in *HPCA*, 2021.

[192] G. Jeong, E. Qin, A. Samajdar, C. J. Hughes, S. Subramoney, H. Kim, and T. Krishna, "RASA: Efficient register-aware systolic array matrix engine for CPU," in *DAC*, 2021.

[193] F. Muñoz-Martínez, J. L. Abellán, M. E. Acacio, and T. Krishna, "A novel network fabric for efficient spatio-temporal reduction in flexible DNN accelerators," in *NOCS*, 2021.

[194] S. Rashidi, W. Won, S. Srinivasan, S. Sridharan, and T. Krishna, "Themis: A network bandwidth-aware collective scheduling policy for distributed training of DL models," *arXiv preprint arXiv:2110.04478*, 2021.

[195] G. Jeong, G. Kestor, P. Chatarasi, A. Parashar, P.-A. Tsai, S. Rajamanickam, R. Gioiosa, and T. Krishna, "Union: A unified hw-sw co-design ecosystem in MLIR for evaluating tensor operations on spatial accelerators," in *PACT*, 2021.

[196] Y. Choi and M. Rhu, "PREMA: A predictive multi-task scheduling algorithm for preemptible neural processing units," in *HPCA*, 2020.

[197] P. Chatarasi, H. Kwon, A. Parashar, M. Pellauer, T. Krishna, and V. Sarkar, "Marvel: A data-centric approach for mapping deep learning operators on spatial accelerators," *TACO*, 2021.

[198] M. Alwani, H. Chen, M. Ferdman, and P. Milder, "Fused-layer CNN accelerators," in *MICRO*, 2016.

[199] M. Gao, X. Yang, J. Pu, M. Horowitz, and C. Kozyrakis, "Tangram: Optimized coarse-grained dataflow for scalable NN accelerators," in *ASPLOS*, 2019.

[200] Y. Shen, M. Ferdman, and P. Milder, "Maximizing CNN accelerator efficiency through resource partitioning," in *ISCA*, 2017.

[201] S. M. Jafri, H. Hassan, A. Hemani, and O. Mutlu, "Refresh triggered computation: Improving the energy efficiency of convolutional neural network accelerators," *TACO*, 2020.

[202] B. Salami, E. B. Onural, I. E. Yuksel, F. Koc, O. Ergin, A. C. Kestelman, O. Unsal, H. Sarbazi-Azad, and O. Mutlu, "An experimental study of reduced-voltage operation in modern FPGAs for neural network acceleration," in *DSN*, 2020.

[203] M. Imani, S. Gupta, Y. Kim, and T. Rosing, "FloatPIM: In-memory acceleration of deep neural network training with high precision," in *ISCA*, 2019.

[204] H. Jiang, X. Peng, S. Huang, and S. Yu, "CIMAT: A compute-in-memory architecture for on-chip training based on transpose SRAM arrays," *IEEE Transactions on Computers*, 2020.

[205] S. Zhang, K. Huang, and H. Shen, "A robust 8-bit non-volatile computing-in-memory core for low-power parallel MAC operations," *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2020.

[206] B. Y. Cho, J. Jung, and M. Erez, "Accelerating bandwidth-bound deep learning inference with main-memory accelerators," in *SC*, 2021.

[207] Y. Long, E. Lee, D. Kim, and S. Mukhopadhyay, "Q-PIM: A genetic algorithm based flexible dnn quantization method and application to processing-in-memory platform," in *DAC*, 2020.

[208] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *ISCA*, 2016.

[209] 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," *ISCA*, 2016.

[210] L. Song, X. Qian, H. Li, and Y. Chen, "PipeLayer: A pipelined ReRAM-based accelerator for deep learning," in *HPCA*, 2017.

[211] J. Fowers, K. Ovtcharov, M. Papamichael, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, L. Adams, and M. Ghandi, "A configurable cloud-scale DNN processor for real-time AI," in *ISCA*, 2018.

[212] H. Sharma, J. Park, N. Suda, L. Lai, B. Chau, V. Chandra, and H. Esmaeilzadeh, "Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network," in *ISCA*, 2018.

[213] 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 *et al.*, "PUMA: A programmable ultra-efficient memristor-based accelerator for machine learning inference," in *ASPLOS*, 2019.

[214] T. Gokmen, M. Onen, and W. Haensch, "Training deep convolutional neural networks with resistive cross-point devices," *Frontiers in neuroscience*, 2017.

[215] M. Cheng, L. Xia, Z. Zhu, Y. Cai, Y. Xie, Y. Wang, and H. Yang, "TIME: A training-in-memory architecture for memristor-based deep neural networks," in *DAC*, 2017.

[216] E. Azarkhish, D. Rossi, I. Loi, and L. Benini, "Neurostream: Scalable and energy efficient deep learning with smart memory cubes," *IEEE TPDS*, 2017.

[217] Y. Kwon, Y. Lee, and M. Rhu, "TensorDIMM: A practical near-memory processing architecture for embeddings and tensor operations in deep learning," in *MICRO*, 2019.

[218] J. Liu, H. Zhao, M. A. Ogleari, D. Li, and J. Zhao, "Processing-in-memory for energy-efficient neural network training: A heterogeneous approach," in *MICRO*, 2018.

[219] T.-H. Yang, H.-Y. Cheng, C.-L. Yang, I.-C. Tseng, H.-W. Hu, H.-S. Chang, and H.-P. Li, "Sparse ReRAM engine: Joint exploration of activation and weight sparsity in compressed neural networks," in *ISCA*, 2019.

[220] S. Angizi, Z. He, A. S. Rakin, and D. Fan, "CMP-PIM: an energy-efficient comparator-based processing-in-memory neural network accelerator," in *DAC*, 2018.

[221] 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 *DAC*, 2018.

[222] F. Chen, L. Song, and Y. Chen, "ReGAN: A pipelined ReRAM-based accelerator for generative adversarial networks," in *ASP-DAC*, 2018.

[223] D. Fan and S. Angizi, "Energy efficient in-memory binary deep neural network accelerator with dual-mode SOT-MRAM," in *ICCD*, 2017.

[224] H. Ji, L. Song, L. Jiang, H. Li, and Y. Chen, "ReCom: An efficient resistive accelerator for compressed deep neural networks," in *DATE*, 2018.

[225] C. Deng, F. Sun, X. Qian, J. Lin, Z. Wang, and B. Yuan, "TIE: Energy-efficient tensor train-based inference engine for deep neural network," in *ISCA*, 2019.

[226] X. Chen, X. Yin, M. Niemier, and X. S. Hu, "Design and optimization of FeFET-based crossbars for binary convolution neural networks," in *DATE*, 2018.

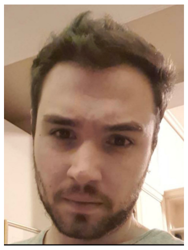
[227] O. Mutlu, S. Ghose, J. Gómez-Luna, and R. Ausavarungnirun, "A modern primer on processing in memory," *Emerging Computing: From Devices to Systems - Looking Beyond Moore and Von Neumann*, 2021.

[228] S. Ghose, A. Boroumand, J. S. Kim, J. Gómez-Luna, and O. Mutlu, "Processing-in-memory: A workload-driven perspective," *IBM Journal of Research and Development*, 2019.

Lois Orosa is a senior researcher at SAFARI Research group @ ETH Zürich, Switzerland. He received his BS and MS degrees in Telecommunication Engineering from the University of Vigo, Spain, his PhD degree from the University of Santiago de Compostela, Spain, and he held a postDoc position in the University of Campinas, Brazil. He was a visiting researcher at multiple companies (IBM, Recore Systems, Xilinx and Huawei) and universities (UIUC and Universidade Nova de Lisboa). His current research interests are in computer architecture, hardware security, reliability, memory systems, and machine learning (ML) accelerators. For more information, please see his webpage at https://loisorosa.github.io/.

Skanda Koppula is currently a research engineer at DeepMind. Previous to this, he worked at ETH Zürich in the SAFARI research group on memory systems, machine learning acceleration, and computer architecture. He completed his MEng and BSc from MIT in 2018.

Yaman Umuroglu received the PhD degree from the Norwegian University of Science and Technology (NTNU), Norway and a joint European MSc on Embedded Systems from the Erasmus Mundus EMECS programme. He is a research scientist at Xilinx Research Labs, Ireland. His research takes a full-stack view of machine learning with neural networks with a focus on highefficiency and high-performance implementations and spans hardware-network codesign, techniques for efficient arithmetic, sparsity, and quantization.



Konstantinos Kanellopoulos is currently pursuing his PhD at ETH Zürich in the SAFARI research group. He completed his MEng and BSc at NTUA. His research interests lie at the intersection of software and hardware.

Juan Gómez-Luna is a senior researcher and lecturer at SAFARI Research Group @ ETH Zürich. He received the BS and MS degrees in Telecommunication Engineering from the University of Sevilla, Spain, in 2001, and the PhD degree in Computer Science from the University of Córdoba, Spain, in 2012. Between 2005 and 2017, he was a faculty member of the University of Córdoba. His research interests focus on processing-in-memory, memory systems, heterogeneous computing, and hardware and software acceleration of medical imaging and bioinformatics. He is the lead author of PrIM (https://github.com/CMU-SAFARI/prim-benchmarks), the first publicly-available benchmark suite for a real-world processing-in-memory architecture, and Chai (https://github.com/chai-benchmarks/chai), a benchmark suite for heterogeneous systems with CPU/GPU/FPGA.

Michaela Blott received the master's degree from the University of Kaiserslautern in Germany and brings more than 25 years of computer architecture, FPGA and board design, in research institutions (ETH Zurich and Bell Labs) and development organizations. She is a distinguished engineer at Xilinx Research, Dublin, Ireland, where she heads a team of international scientists driving exciting research to define new application domains for Xilinx devices, such as machine learning. She is heavily involved with the international research community serving as the technical co-chair of FPL'2018, workshop organizer (H2RC, ITEM'2020), and member of numerous technical program committees (FPL, ISFPGA, DATE, etc.).

Kees Vissers graduated from Delft University in the Netherlands. He worked at Philips Research in Eindhoven, The Netherlands, for many years. The work included Digital Video system design, HW–SW co-design, VLIW processor design and dedicated video processors. He was a visiting industrial fellow at Carnegie Mellon University, where he worked on early High Level Synthesis tools. He was a visiting industrial fellow at UC Berkeley where he worked on several models of computation and dataflow computing. He was a director of architecture at Trimedia, and CTO at Chameleon Systems. For more than a decade he is heading a team of international researchers at Xilinx in the CTO office. The research topics include machine learning applications and architectures, wireless applications, image processing applications and new datacenter applications. These applications drive next generation programming environments and architectures. He is a Fellow at Xilinx.

Onur Mutlu is a Professor of Computer Science at ETH Zürich. He is also a faculty member at Carnegie Mellon University, where he previously held the Strecker Early Career Professorship. His current broader research interests are in computer architecture, systems, hardware security, and bioinformatics. A variety of techniques he, along with his group and collaborators, has invented over the years have influenced industry and have been employed in commercial microprocessors and memory/storage systems. He obtained his PhD and MS in ECE from the University of Texas at Austin and BS degrees in Computer Engineering and Psychology from the University of Michigan, Ann Arbor. He started the Computer Architecture Group at Microsoft Research (2006-2009), and held various product and research positions at Intel Corporation, Advanced Micro Devices, VMware, and Google. He received the IEEE High Performance Computer Architecture Test of Time Award, the IEEE Computer Society Edward J. McCluskey Technical Achievement Award, ACM SIGARCH Maurice Wilkes Award, the inaugural IEEE Computer Society Young Computer Architect Award, the inaugural Intel Early Career Faculty Award, US National Science Foundation CAREER Award, Carnegie Mellon University Ladd Research Award, faculty partnership awards from various companies, and a healthy number of best paper or ”Top Pick” paper recognitions at various computer systems, architecture, and security venues. He is an ACM Fellow ”for contributions to computer architecture research, especially in memory systems”, IEEE Fellow for ”contributions to computer architecture research and practice”, and an elected member of the Academy of Europe (Academia Europaea). His computer architecture and digital logic design course lectures and materials are freely available on YouTube (https://www.youtube.com/OnurMutluLectures), and his research group makes a wide variety of software and hardware artifacts freely available online (https://safari.ethz.ch/). For more information, please see his webpage at https://people.inf.ethz.ch/omutlu/.