 http://dx.doi.org/10.1109/TVLSI.2023.3286122

An Energy-Efficient GeMM-based Convolution Accelerator with On-the-fly im2col

Jordi Fornt, Pau Fontova-Musté, Martí Caro, Jaume Abella, Francesc Moll, Josep Altet, and Christoph Studer

***Abstract*—Systolic array architectures have recently emerged as successful accelerators for deep convolutional neural network (CNN) inference. Such architectures can be used to efficiently execute general matrix-matrix multiplications (GeMM), but computing convolutions with this primitive involves transforming the 3D input tensor into an equivalent matrix, which can lead to an inflation of the input data, increasing the off-chip memory traffic which is critical for energy efficiency. In this work, we propose a GeMM-based systolic array accelerator that uses a novel data feeder architecture to perform on-chip, on-the-fly convolution lowering (also known as *im2col*), supporting arbitrary tensor and kernel sizes as well as strided and dilated (or atrous) convolutions. By using our data feeder, we reduce memory transactions and required bandwidth on state-of-the-art CNNs by a factor of two, while only adding an area and power overhead of 4% and 7% respectively. An ASIC implementation of our accelerator in 22 nm technology fits in less than 1.1 mm^2 and reaches an energy efficiency of 1.10 TFLOP/sW with 16-bit floating point arithmetic.**

***Index Terms*—Convolutional neural network accelerators, energy efficiency, systolic arrays, im2col, convolution lowering**

I. Introduction

Deep CNN models achieve high inference accuracy at the cost of computational complexity, so the use of hardware accelerators is key for their deployment in real-world applications. Powerful computing engines, such as GPUs, have been extensively used for CNN training and inference [1], but for many applications, these prove to be too power-hungry.

Systolic array-based accelerators, specialized towards deep learning, have been shown to provide superior energy efficiency than GPUs, and many different systolic architectures have been proposed in the literature [2]–[7]. These accelerators often focus on efficiently executing general matrix-matrix multiplications (GeMM) [2]–[4]. Computing convolution operations in this setting requires transforming the input 3D tensor into an equivalent matrix, using a technique known as convolution lowering [8] (a.k.a. the *im2col* algorithm).

Lowering the convolution helps make computations efficient, but the resulting matrix can be much larger than its tensor equivalent, greatly increasing the amount of data to be transferred to the array. For this reason, many accelerators choose to manage the lowering step internally by including some *im2col units*, e.g. the systolic array generator Gemmini [9] contains an optional *im2col* module to accelerate the lowering task. This relieves the CPU from the burden of managing the *im2col*, but it still suffers from the overhead in memory accesses, as the inflated matrices are fetched from the L2 cache or off-chip memory. SPOTS [10] implements a GeMM-based systolic accelerator with an *im2col* unit for on-the-fly convolution lowering and sparsity support. It avoids the data overhead introduced in the lowering step by first moving the tensors into its internal SRAM memories and then managing the *im2col* task internally. However, it requires including very large (13% of its total area) intermediate buffers that compromise the area efficiency of the system, and its *im2col unit* cannot support dilated convolutions.

To the best of our knowledge, USCA [11] is the only accelerator that currently provides hardware support for on-the-fly convolution lowering as well as dilated convolutions, but it focuses on accelerating convolutions on sparse tensors. While some classic networks like ResNet [12] can be sparsified to large degrees without much accuracy degradation, there is a trend in modern architectures like transformer networks [13] to struggle in achieving high sparsity [14]. Hence, basing the energy efficiency strategy of an accelerator in sparsity puts it at risk of obsolescence unless this trend is reverted. Furthermore, USCA only reports synthesis results, lacking a full physical design, which is critical for an accurate power estimation.

In this work, we develop a CNN accelerator that efficiently computes convolutions with dense tensors, based on a systolic array architecture and a novel data feeder that performs on-the-fly *im2col* transformation and supports dilated convolutions. Furthermore, we also perform the full synthesis and physical design process for an ASIC implementation of the accelerator, providing accurate area and power results.

II. Architecture

The architecture of the proposed accelerator, depicted in Figure 1, is built around an Output Stationary (OS) systolic array. Three independent SRAM memories with double buffering hold the tensors involved in the convolution (input feature maps, weights, and outputs/partial sums). Our novel *Data Feeder* module is used to lower the input tensors stored in the ifmap SRAM and stream the resulting matrices to the array rows. Since we fetch the data in tensor shape from the DRAM, we avoid the inflation of memory transactions caused by the *im2col* step. During the convolution lowering process, the data feeder generates the data streams according to the dilation rate of the convolution. In parallel, the *Weight Fetcher* block streams the weight data, and the *Partial Sums Manager* extracts the partially accumulated results from the array and, if needed, inserts preload data used to initialize the partial sums.

J. Fornt is with the Barcelona Supercomputing Center (BSC), the Universitat Politècnica de Catalunya (UPC) and formerly with the Eidgenössische Technische Hochschule (ETH) Zürich. M. Caro and F. Moll are with the BSC and the UPC. P. Fontova-Musté and J. Abella are with the BSC. J. Altet is with the UPC. C. Studer is with the ETH Zürich. Author emails: {jordi.fornt, pau.fontova, mcaroroc, jaume.abella}@bsc.es, {francesc.moll, josep.altet}@upc.edu, studer@iis.ee.ethz.ch.

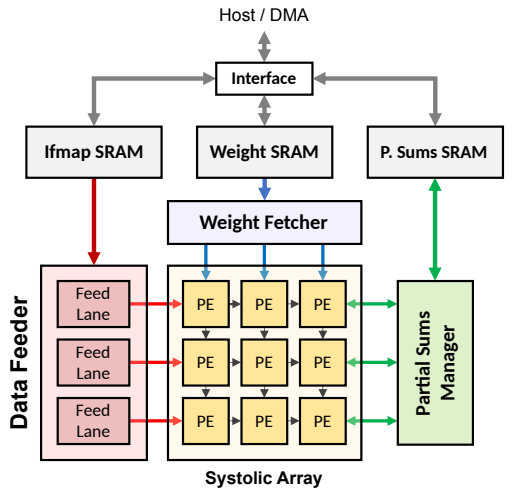

Fig. 1. Proposed convolution accelerator architecture using a 3x3 systolic array and our data feeder.

A. Systolic Array GeMM Engine

We implement an OS array as our GeMM engine, in which each Processing Element (PE) is assigned to a pixel of the output tensor. The horizontal spatial dimension of the tensor is mapped to the array rows, and the output channels dimension to the columns. The systolic array is composed of a mesh of PEs, the design of which is depicted in Figure 2. We kept the PEs as simple as possible to maximize area efficiency. A zero detection and gating circuit, inspired by works like [7], [15] is included before the multiplier in order to avoid unnecessary switching when any input value is zero. Even though our work focuses on dense CNNs rather than sparse networks, this technique is inexpensive and can save power even at low sparsity levels (see Section III).

The Multiply-Accumulate (MAC) operation results are accumulated in an internal register (the *Accumulator*). A secondary register, called *Reserve Register*, is used to hold partial sum data during the extraction of results or insertion of preload data. When the execution of a computation context finishes, the accumulator and reserve register values are swapped, and the first MAC of the next context is performed. This solution has two benefits: first, it eliminates the need to access all partial sums from outside of the array concurrently (which does not scale well) by connecting the reserve registers together and shifting in and out the values. Second, it allows to concatenate computation contexts without stalling the PEs, improving the overall performance and utilization, while having a small power and area overhead (see Section III).

B. Data Feeder for On-the-fly im2col

Our novel data feeder (see Figure 3) is composed of the *Index Counters*, a set of configuration registers, and an array of *Feed Lanes*, replicated for all array rows as shown in Figure 1. The main idea of our feeder architecture is to read all SRAM addresses of a context in a single pass and distribute the data values to the PEs that need them. The ifmap tensor stored in the corresponding SRAM has its dimensions flattened in the order [C,Y,X], with the x-axis contiguous in memory.

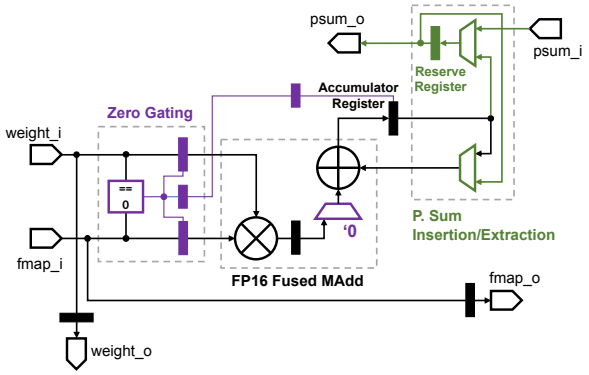

Fig. 2. Processing Element design. Zero gating highlighted in purple, partial sum management in green.

To perform the readout, we define the *Interest Region* of a context as the union of the convolution kernels centered on the output pixels currently being computed, as depicted in Figure 4 (left part). All memory locations corresponding to the interest region are read sequentially once, and each feed lane selects the elements it needs from every data word coming out of the SRAM. Once all positions of the current region of interest have been read, the region is relocated to a new context. The readout sequence is controlled by the index counters module, which comprises 5 independent counters with configurable step size and limit: 3 are used to traverse the interest region (through the x, y and channel axes), and the other 2 move the whole region to a new context (only for the x and y axes, since the input channel is fully reduced on each context). The sum of all the counter outputs generates a pointer that is used to go through the ifmap tensor.

The index pointer initially points to the first interest element in the current row of the region, as illustrated in Figure 4. Subsequent reads on the same tensor row are performed by incrementing the pointer by the bus width (in data elements). When the index reaches the end of the current interest region row, it moves to the first position of the next row that must be read. The pointer value is split to generate the SRAM read address (higher bits) and the *Global Word Offset* (lower bits), which informs about the location inside the SRAM data word of the first element of interest in the region.

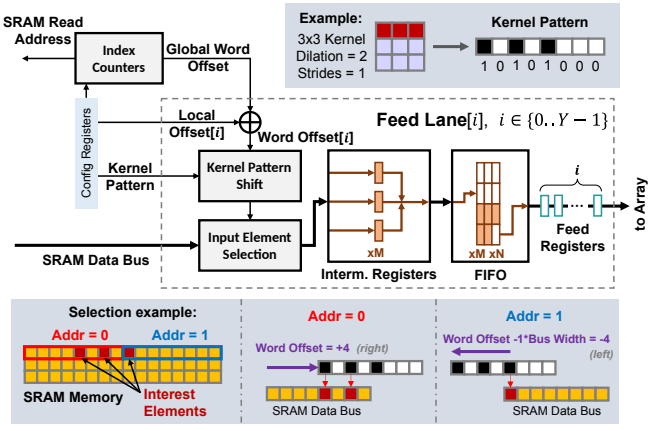

Fig. 3. Data feeder design and an example of feature map selection from an 8-element memory bus.

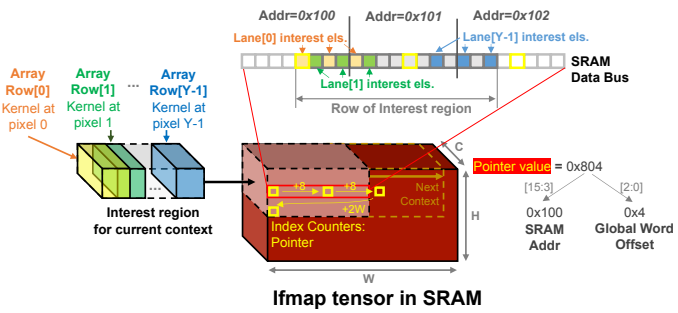

Fig. 4. Example and illustration of the index counters sequence, assuming an 8-element memory bus.

The information provided by the global word offset fully describes how the first feed lane (first row) can access its first element of interest inside the SRAM data bus. For all remaining lanes, we obtain this location by adding a *Local Offset* to the global word offset, representing the difference in kernel locations (i.e., output pixel locations) between rows of the array. Contiguous lanes represent contiguous output values, so a convolution is configured by setting the local offsets to $[s, 2s, ..., (Y-1)s]$, with s being the strides and Y being the number of array rows. The result of this addition is defined as the *Word Offset*, and it indicates to each feed lane the location of its first interest element.

For kernels larger than 1x1, each feed lane generally needs to take several elements from the data bus on a single cycle. These elements need not be contiguous to the first interest element, since the accelerator supports dilated convolutions. To identify which positions must be selected after the first one (which is pointed to by the word offset), we define the *Kernel Pattern*, a configurable bit vector that defines the location of interest elements in contiguous memory positions, based on the convolution kernel. With this signal we fully describe the horizontal kernel size and dilation rate. Figure 3 shows an example for a 3x3 kernel with a dilation rate of 2.

By right-shifting the kernel pattern by the word offset, we align it with the correct location of the interest elements on the data bus, as depicted in the lower part of Figure 3. Once aligned, each bit of the shifted pattern points to the data elements that must be taken. If several contiguous reads are needed to cover all elements of interest, the bus width (in data elements) is subtracted to the shift amount after every read. A negative shift amount in this case denotes a left-shift. With this, the shifted kernel pattern is aligned with the interest elements in subsequent reads. When the readout of the current row of the interest region complete, the shift amount is reset and the sequence is repeated. After all positions of the region of interest have been covered, we move to a new context.

A simple multiplexer-based selection circuit takes the data elements pointed by the shifted kernel pattern and stores them in a set of intermediate registers. When all registers are full, their contents are pushed to a FIFO memory that holds the values until they are consumed by the array. The number of intermediate registers (M) determines how many values in parallel can be taken on each feed lane. If there are more interest elements in the bus than free registers, a stall signal is raised by the feed lane, indicating that the index counting and data readout pipeline must stop until all lanes have been able to gather the required data. Finally, the FIFO output passes through the *Feed Registers*, a stage of concatenated registers that enforce the staggered latency between array rows: e.g., if the first array row starts getting data at cycle 0, the second one will start at cycle 1, and so on.

TABLE I
CONVOLUTION ACCELERATOR SPECIFICATIONS (POST-LAYOUT)

Technology	GF22FDX (22 nm)
Chip Area	1 mm x 1.09 mm
Logic Gate Count	2657k (NAND2)
No. of PEs	256 (16x16)
Arithmetic Precision	16-bit floating point
SRAM Sizes / Total memory	32 kB (x2) / 192 kB
Supply Voltage	0.8 V
Clock Period / Max. Frequency	1.8 ns / 555 MHz
Peak Throughput	284 GFLOP/s
Avg. Power @ Peak Throughput	258 mW
Peak Enery Efficiency	1.10 TFLOP/sW

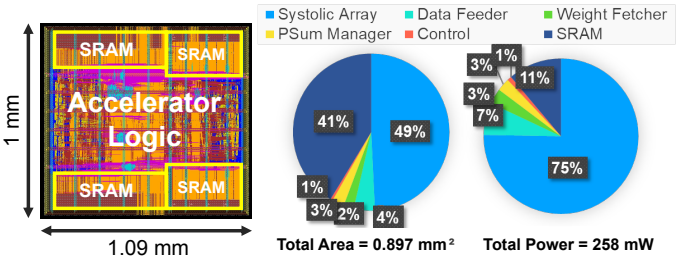

Fig. 5. Accelerator floorplan (left). Area and power breakdown of the accelerator (right). Power is estimated with dense random inputs.

III. IMPLEMENTATION AND RESULTS

We base our evaluation on an ASIC implementation of the accelerator described in Section II, using a 16x16 systolic array. Three double-buffered 32 kB single-port SRAMs are used to store the ifmaps, weights and partial sums, each with a 256-bit data bus. We implement the PE arithmetic units in 16-bit floating point in order to compute state-of-the-art CNNs with high accuracy and without retraining. The accumulation of partial sums is also performed using FP16 precision. Note that the accelerator architecture is agnostic to arithmetic representation, and could also be implemented using integer operators. To implement the FP16 MAC operator, we used a simplified version of the FPnew floating point unit from the open-source PULP platform [16].

The implemented data feeders use 3 intermediate registers per lane (M=3) to optimize the performance of 3x3 kernels, which are very common. The kernel pattern width is limited to 64 bits, so the data feeders can support any kernel shape with a horizontal dimension that fits in this vector. For convolutions without dilation (d=1) this means that the maximum kernel shape is [64,K_y], with the vertical dimension K_y limited only by the total capacity of the SRAMs. For dilated 3x3 convolutions, the maximum dilation coefficient is d=31.

The architecture was synthesized in 22 nm using Cadence Genus, and the physical design was performed with Cadence Innovus. Table I summarizes the most important specifications of the implementation. The throughput and power metrics are

TABLE II
MEMORY TRANSACTIONS AND BANDWIDTH SUMMARY, USING EXPLICIT LOWERING (*expl.*) AND OUR DATA FEEDER (*feed.*)

	Total MB		Avg. GB/s		Max. GB/s	
Network	**expl.**	**feed.**	**expl.**	**feed.**	**expl.**	**feed.**
ARPN	288	59	7.93	1.57	12.0	5.46
Resnet-50	263	173	6.85	4.12	12.0	10.7
VGG-16	1231	572	9.07	4.07	29.0	16.5
YOLOv3	3005	1040	7.90	2.72	12.0	8.72

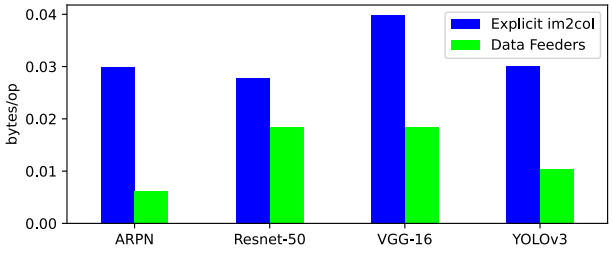

Fig. 6. Required bytes per operation during the execution of the benchmarks.

obtained from post-layout simulations of the design using this setup. We benchmark the performance of our design with four CNNs of different sizes: the Atrous Region Proposal Network (ARPN) used in [17], ResNet-50 [12], VGG-16 [18], and YOLOv3 [19] using input resolutions of 64x64, 256x256, 224x224, and 512x512, respectively.

Figure 5 shows the floorplan of the physical design of the accelerator, as well as a breakdown of the area and power consumption. The total area of the system is 0.897 mm^2, which we fit in a 1.09 mm^2 floorplan. The systolic array PEs make up for about 83% of the logic area excluding the SRAM macros, and the area overhead of the data feeder is less than 7% of the logic area (4% of the total).

The reserve registers we include in the PEs to extract the partial sums have a total area and power overhead of 0.76% and 0.90%, respectively. The zero gating strategy we implement in the array PEs allows to save power when the input values are zero, decreasing the total power by about 6% when both weights and feature maps present 10% of zeros. Taking the worst-case scenario as a baseline, (all values are random and the tensors are completely dense) the system consumes 258 mW during computation, with the PEs accounting for 75% of the power and the data feeder for about 7% (see Figure 5).

By converting the tensors to data streams on-the-fly, the data feeder decreases the overall memory traffic and bandwidth requirements of the system with respect to the software-based explicit *im2col*, which would require fetching the lowered matrices from the off-chip memory. We find that the number of bytes per operation decreases by more than 50% for most of our benchmarks (see Figure 6). This results in a significant reduction of the total data exchanged with the DRAM memory, as well as the average bandwidth required to feed the accelerator without stalls (see Table II).

Fewer memory transfers also enable energy savings on the DRAM, which typically accounts for a large portion of the system-level power consumption. Using the DRAMPower tool [20] we estimate that off-chip memory transactions using an LPDDR3 memory have an energy cost of about 120 pJ/byte.

TABLE III
COMPUTE TIME AND PROPORTION OF DRAM WAITING TIME ASSUMING 6.4 GB/S OF DRAM BANDWIDTH

	Comp. time [ms]		Average GFLOP/s		DRAM stalls [% of time]	
Network	**expl.**	**feed.**	**expl.**	**feed.**	**expl.**	**feed.**
ARPN	45	37	215	258	19	0
Resnet-50	45	43	212	220	14	1.5
VGG-16	193	164	160	189	31	14
YOLOv3	472	384	211	260	20	0.5

Taking YOLOv3 as an example (see Table II), using our data feeder decreases the total energy consumed by the DRAM during inference by 236 mJ. In comparison, the energy overhead of the data feeder is 6.9 mJ, 34x smaller than the system-level energy reduction it enables.

To assess the accelerator performance for our benchmark CNNs, we assume that a commercial 32-bit-wide LPDDR3 DRAM memory operating at 800 MHz is used, with a maximum bandwidth of 6.4 GB/s. As summarized in Table III, the use of our data feeder also improves the throughput of the accelerator under these conditions. The performance difference comes mainly from the twofold decrease in the amount of data transactions, which helps avoid accelerator stalls. In YOLOv3, computation time is reduced almost by 20% when using our data feeder, even if we neglect the overhead of the CPU lowering the convolution in the *im2col* case.

Table IV compares our design with SPOTS [10] and USCA [11], the GeMM-based systolic accelerators in the literature most similar to this work, as well as Eyeriss [6], a successful systolic accelerator used commonly as a benchmark. Since these designs are implemented in different technology nodes, we also report the efficiency values scaled to 22 nm for a better comparison, using the equations defined in [21].

In the case of SPOTS, a comparison in terms of energy efficiency cannot be established since its power consumption is unreported. Similarly, USCA seems to present a higher energy efficiency, but the arithmetic used in its PEs is unreported, so a fair comparison can not be established since this choice greatly impacts the overall energy efficiency, as well as the accelerator accuracy. It should also be noted that the authors of USCA do not perform the full physical design of the accelerator, so its reported power may be an optimistic estimate. Lastly, compared to Eyeriss v2, our accelerator has a slightly lower energy efficiency in absolute terms, when considering dense tensor convolutions. However, it is important to note that Eyeriss uses 8-bit fixed point arithmetic, which is expected

TABLE IV
COMPARISON WITH SIMILAR STATE-OF-THE-ART DESIGNS

Design	**Tech. [nm]**	**Arith.**	**Efficiency [GOP/sW]**	**GOP/sW @22 nm**	**Dilated Conv.?**
SPOTS [10]	45	int16	N/R[a]	N/R	×
USCA [11]	28	N/R[b]	1863	3521	✓
Eyeriss v2 [6]	65	int8	253[c]	1799	×
This work	22	FP16	**1100**	**1100**	✓

[a] [10] does not report energy efficiency nor power consumption.
[b] [11] does not report the PE arithmetic.
[c] Top efficiency with dense tensors. With sparse AlexNet: 963 GOP/sW (6.84 TOP/sW when scaled to 22 nm).

to consume much less power than the 16-bit floating-point arithmetic we support in order to execute any state-of-the-art CNN with high accuracy. From our experiments with MAC-based processing elements in 22 nm technology we have seen that the difference between FP16 and int8 is more than enough to cover the energy efficiency gap between our accelerator and Eyeriss v2. Hence, when accounting for this difference, our proposed accelerator surpasses Eyeriss v2 in terms of energy efficiency when dealing with dense tensors, while also supporting dilated convolutions.

IV. Conclusions

We have presented an energy-efficient convolution accelerator for CNNs built upon a GeMM-based systolic array, coupled with our novel data feeder architecture, which enables on-chip, on-the-fly convolution lowering and supports dilated (or atrous) convolutions. Our accelerator reaches a peak throughput of 284 GFLOP/s and an energy efficiency of 1.10 TFLOP/sW using FP16 arithmetic and a 16x16 array. By formulating lowering as a selection task and leveraging the regularity of the 2D convolution, our data feeder design achieves superior area and energy efficiency than any other proposed *im2col* solution, with area and power overheads of 4% and 7%, respectively. We have demonstrated how an ASIC implementation of our design can achieve reductions of more than a factor of two in memory transactions when accelerating state-of-the-art CNNs, compared to software-based *im2col*, significantly improving the energy efficiency of the overall system. Additionally, the reduction in memory transactions relaxes the off-chip memory bandwidth requirements of the system, boosting the overall performance and energy efficiency of the accelerator.

Acknowledgments

This work is part of the project PCI2020-134984-2, funded by MCIN/AEI/10.13039/501100011033 and the European Union NextGenerationEU/PRTR, and the European Union's Horizon Europe Programme under project KDT Joint Undertaking (JU) under grant agreement No 101097224. This work has also been partially supported by the Spanish Ministry of Science and Innovation under grant PID2019-107255GB-C21 funded by MCIN/AEI/10.13039/501100011033.

References

[1] M. Capra *et al.*, "Hardware and Software Optimizations for Accelerating Deep Neural Networks: Survey of Current Trends, Challenges, and the Road Ahead," *IEEE Access*, vol. 8, p. 225134–225180, 2020.

[2] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings - International Symposium on Computer Architecture*, vol. Part F1286, pp. 1–12, Institute of Electrical and Electronics Engineers Inc., 6 2017.

[3] Z. Du *et al.*, "ShiDianNao: Shifting vision processing closer to the sensor," in *Proceedings - International Symposium on Computer Architecture*, vol. 13-17-June, pp. 92–104, Institute of Electrical and Electronics Engineers Inc., 6 2015.

[4] R. Xu *et al.*, "HeSA: Heterogeneous Systolic Array Architecture for Compact CNNs Hardware Accelerators," in *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 657–662, 2021.

[5] Y. Chen *et al.*, "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks," in *Proceedings - 2016 43rd International Symposium on Computer Architecture, ISCA 2016*, pp. 367–379, Institute of Electrical and Electronics Engineers Inc., 8 2016.

[6] Y. Chen *et al.*, "Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, pp. 292–308, 6 2019.

[7] S. Venkataramani *et al.*, "RaPiD : AI Accelerator for Ultra-low Precision Training and Inference," in *Proceedings - 2021 48th International Symposium on Computer Architecture, ISCA 2021*, pp. 153–166, 2021.

[8] S. Chetlur *et al.*, "cuDNN: Efficient Primitives for Deep Learning," 2014.

[9] H. Genc *et al.*, "Gemmini: Enabling Systematic Deep-Learning Architecture Evaluation via Full-Stack Integration," 2021.

[10] M. Soltaniyeh *et al.*, "SPOTS: An Accelerator for Sparse CNNs Leveraging General Matrix-Matrix Multiplication," 2021.

[11] W. Liu *et al.*, "USCA: A Unified Systolic Convolution Array Architecture for Accelerating Sparse Neural Network," in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, 2019.

[12] K. He *et al.*, "Deep residual learning for image recognition," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2016-Decem, pp. 770–778, IEEE Computer Society, 12 2016.

[13] A. Vaswani *et al.*, "Attention Is All You Need," *CoRR*, vol. abs/1706.03762, 2017.

[14] T. Hoefler *et al.*, "Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks," *CoRR*, vol. abs/2102.00554, 2021.

[15] L. Ye *et al.*, "Power-Efficient Deep Convolutional Neural Network Design through Zero-Gating PEs and Partial-Sum Reuse Centric Dataflow," *IEEE Access*, 2021.

[16] S. Mach *et al.*, "FPnew: An Open-Source Multiformat Floating-Point Unit Architecture for Energy-Proportional Transprecision Computing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 4, pp. 774–787, 2021.

[17] T. Guan and H. Zhu, "Atrous Faster R-CNN for Small Scale Object Detection," in *2017 2nd International Conference on Multimedia and Image Processing (ICMIP)*, pp. 16–21, 2017.

[18] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, International Conference on Learning Representations, ICLR, 9 2015.

[19] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," 2018.

[20] K. Chandrasekar *et al.*, "DRAMPower: Open-source DRAM Power & Energy Estimation Tool." http://www.drampower.info.

[21] A. Stillmaker and B. Baas, "Scaling equations for the accurate prediction of cmos device performance from 180nm to 7nm," *Integration*, vol. 58, pp. 74–81, 2017.