An Energy-efficient Convolution Unit for Depthwise Separable Convolutional Neural Networks

Yi Sheng Chong*, Wang Ling Goh†, Yew Soon Ong‡, Vishnu P. Nambiar§, Anh Tuan Do¶
*†School of Electrical and Electronic Engineering, Nanyang Technological University (NTU), Singapore
*Energy Research Institute, Interdisciplinary Graduate Programme, NTU, Singapore
‡School of Computer Science and Engineering, NTU, Singapore
§¶Institute of Microelectronics, A*STAR, Singapore
*yisheng002@e.ntu.edu.sg, †ewlgoh@ntu.edu.sg, ‡asysong@ntu.edu.sg,
§vishnu_paramasivam@ime.a-star.edu.sg, ¶doat@ime.a-star.edu.sg

***Abstract*—High performance but computationally expensive Convolutional Neural Networks (CNNs) require both algorithmic and custom hardware improvement to reduce model size and to improve energy efficiency for edge computing applications. Recent CNN architectures employ depthwise separable convolution to reduce the total number of weights and MAC operations. However, depthwise separable convolution workload does not run efficiently in existing CNN accelerators. This paper proposes an energy-efficient CONV unit for pointwise and depthwise operation. The CONV unit utilizes weight stationary to enable high efficiency. The row partial sum reduction is engaged to increase parallelism in pointwise convolution thereby lightening the memory requirements on output partial sums. Our design achieves a maximum efficiency of 3.17 TOPS/W at 0.85V/40nm CMOS which is well-suited for energy constrained edge computing applications.**

I. Introduction

Convolutional Neural Networks (CNNs) has seen promising performance in areas of computer vision, such as image classification and object recognition. With its high accuracy, CNNs are widely adopted in autonomous vehicles [1], Internet of Things (IoT) devices [2] and robot vision [3]. These applications typically require CNNs to work under resource-constrained environment. This creates huge challenges, as CNN models often require millions of parameters and computations [4]–[6], as shown in Figure 1(b). Thus, custom energy efficient hardware accelerators are demanded. Besides, there is a rising interest to develop small and compact CNN models, which further help alleviate the computational needs.

Recent CNN models such as Xception [4] and MobileNetv1 [5] exploit depthwise separable convolution to reduce computational workloads. For instance, MobileNetv1 reported 8-9 times computation reduction with an insignificant trade off of only 1% accuracy [5]. Depthwise separable convolution involves both depthwise and pointwise convolutions. Pointwise convolution becomes the prominent workload, as shown in Figure 1(c). Therefore, hardware accelerator designs that can efficiently support depthwise separable convolution are in demand, to take advantages of the recent compact CNN models.

Various hardware accelerator designs have been proposed, but many are not optimized for depthwise separable convolution. For instance, Eyeriss [7] proposed row stationary dataflow to minimize data movement and maximize data reuse in the hardware. When processing pointwise convolution, Eyeriss' dataflow has limited diagonal input feature map sharing and horizontal weight kernel sharing in its processing elements (PEs). Besides, another accelerator, KOP3 [8] introduced a CNN processor that is optimized for 3×3 weight kernels. KOP3 allows multiple weight kernels to remain stationary for them to be processed by the same input image. However, depthwise convolution does not need weight sharing, as it only requires a pair of input pixel and weight kernel. Besides, [9] proposed a matrix multiplication engine to process depthwise separable CNNs. [10] recommended an optimized execution order for tiled matrix multiplication to reduce energy consumption. But, while a matrix multiplication engine can easily represent operations in most CNN models, the flexible matrix multiplication hardware design is not well optimized for low power consumption and high throughput dedicated CNN hardwares.

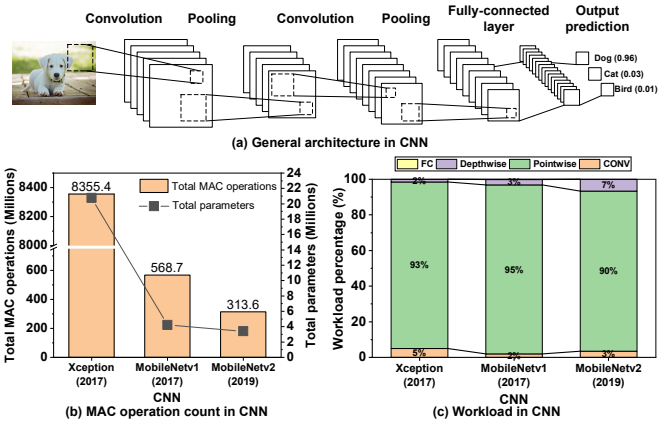

Fig. 1. (a) General CNN architecture. (b) Number of MAC operations and parameters. (c) Workload breakdown in recent CNN architectures.

This work proposes a convolution (CONV) unit design that can efficiently support pointwise and depthwise convolution by having a configurable $2^n \times 2^n$ computation unit and row partial sum reduction. Following this, section II analyzes different convolution workloads. The proposed design is elaborated in Section III. Section IV presents and discusses the experiment results. Lastly the conclusion is drawn in Section V.

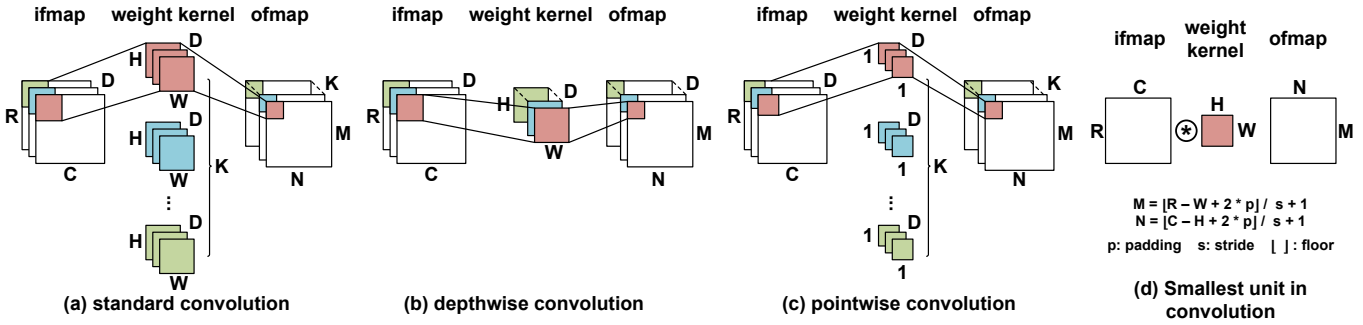

Fig. 2. Comparison of different convolution workloads. Depthwise and pointwise convolution are special cases of standard convolution. This creates unique dataflow requirement for efficient processing in custom hardware accelerators.

II. Depthwise Separable Convolution with Data Reuse

A. Standard versus depthwise separable convolution

For the standard convolution in Figure 2(a), a group of weight kernels is convolved with the input feature map (*ifmap*) to produce one output feature map (*ofmap*) channel. Multiple groups of weight kernels are convolved with the same *ifmap* to produce multiple *ofmap* channels. An *ifmap* with R rows, C columns and depth of D, when convolved with K kernel groups where each has height of H, width of W and depth of D, will require $(H \times W \times D \times M \times N \times K)$ MAC operations.

Most standard convolution however can be separated into depthwise and pointwise convolutions [4], [5], as shown in Figure 2(b) and (c). Depthwise convolution involves *ifmap* channels convolved with respective weights. It requires $(H \times W \times D \times M \times N)$ MAC operations. Pointwise convolution has similar operations with standard convolution, but considers weight kernels of size 1×1. It requires $(M \times N \times D \times K)$ MAC operations. Thus the depthwise separable convolution requires $(H \times W \times D \times M \times N + M \times N \times D \times K)$ MAC operations. Thus, depthwise separable convolution only requires $(\frac{1}{K} + \frac{1}{H \times W})$ times number of MAC operations, as compared to standard convolution. For instance, if $K = 128$ and $H = W = 3$, a depthwise separable convolution will require 8.4 times lesser MAC operations.

B. Dataflow and data reuse of convolution operation

The smallest unit in any convolution operation is a pair of *ifmap* channel and weight kernel, as shown in Figure 2(d). An *ofmap* channel is generated when the weight kernel convolved with the *ifmap* channel, and the *ofmap* pixels are generated at every stride step, s. The *ofmap* channel can be categorized as intermediate or final. The *ofmap* channel is intermediate in standard and pointwise convolution, since it needs to be accumulated with one or more *ofmaps*. The *ofmap* channel is final in depthwise operation, as the *ofmap* channel needs not to be accrued with other *ofmap* channel. Hence, the idea is to deduce the best option to run the smallest unit, to support all types of convolutions.

Data reuse in the smallest unit of the convolution is studied. The potential data reuse strategies are weight stationary and input stationary [11]. For weight stationary, the weight kernel is reused by MN times when striding through the *ifmap* channel. For input stationary, the *ifmap* pixels are only reused once. To have maximum data reuse, weight stationary is preferred.

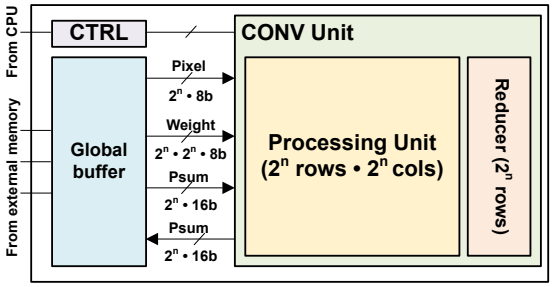

Fig. 3. The CONV unit receives data from and sends computed partial sums (*psums*) to the global buffer. A global controller (CTRL) is used to coordinate the data flow.

The different convolution operations are studied using weight stationary. Depthwise convolution always generates final partial sums (*psums*). However, pointwise convolution generates a lot of intermediate *psums*, due to its small kernel size. The intermediate *psums* may need to be stored temporarily, which increase memory requirement. It also has low *ifmap* reuse. On the contrary, the small kernels create an unique condition for higher parallelism. To produce a final *ofmap* pixel, it requires an $(1 \times 1 \times D)$ *ifmap* cuboid and an $(1 \times 1 \times D)$ weight kernel. The pairs of *ifmap* pixel and weight kernel can be parallelly multiplied to generate intermediate *psums*. These *psums* are subsequently reduced to generate the final *psum*. On the other hand, standard convolution can be seen as multiple depthwise convolutions, but the *ofmap* channels need to be accumulated. This analysis gives significant insights to our proposed design.

III. Proposed Design

Figure 3 shows how the convolution (CONV) unit interfaces with the global controller and on-chip buffer. The controller coordinates the operations between the accelerator and the external controller. The global buffer receives the required data from external memory and stores the computed results. The CONV unit is where the computation occurs. A given CNN model is processed layer-by-layer. Due to the limited on-

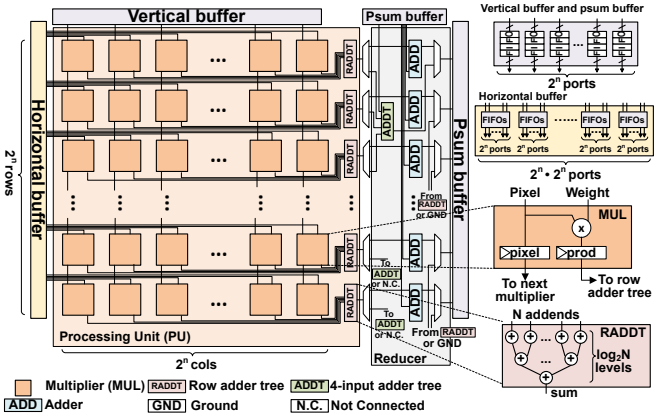

Fig. 4. Proposed CONV unit, consisting of $2^n \times 2^n$ multiplier (MUL) array, 2^n row adder tress, a reducer and 4 FIFO buffers.

chip buffer size, computation workload in each layer is further partitioned such that they can be processed sequentially by the CONV unit. A ping-pong buffer is set-up to load next batch of data, while the current data is being processed.

A. Convolution (CONV) unit design

Figure 4 shows the block diagram of the proposed CONV unit. It consists of a processing unit (PU) and a reducer. The PU consists of a multiplier (MUL) array and row adder trees (RADDTs). The MUL array is designed to be scalable at the factor of 2^n. This enables efficient workload partitioning when processing recent CNN models, as they usually have 2^n weight kernels per group in pointwise convolution, and 2^n channels in depthwise convolution [4], [5].

Four buffers are placed next to the CONV unit: vertical buffer, horizontal buffer and two *psums* buffers. These buffers work in first-in-first-out (FIFO) mode. The CONV unit reads out every clock, while the on-chip controller feeds next data to the buffer. The CONV unit stops reading out when no new data is available. Thus, only a small buffer size is needed. The vertical buffer stores *ifmap* pixels and feeds the data to the first MUL row. The horizontal buffer stores weight kernels. Each MUL has a dedicated bus to receive weight data. One of the *psum* buffers receives old *psums* from the global buffer, to accumulate with current *psums* in the reducer. The other *psum* buffer stores the final *psum* from the reducer before leaving to the global buffer.

Each MUL allows its *ifmap* pixel moving downwards to next MUL, to maximize data reuse. Each MUL row has an adder tree to reduce the products immediately and produce a *psum*, thus reducing memory requirement. The reducer receives 2^n current *psums* from the PU and 2^n old *psums* from the buffer. All *psums* are handled uniquely for different convolution operations.

Figure 5 shows how the proposed CONV unit handles pointwise convolution. Using weight stationary, each group of weight kernels is pre-loaded onto one MUL row from the horizontal buffer. Since the same *ifmap* is used by multiple weight kernels, input pixels are pushed down from the vertical

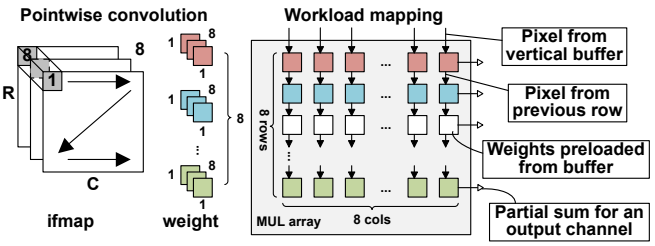

Fig. 5. Illustration of pointwise workload mapping on an 8×8 MUL array.

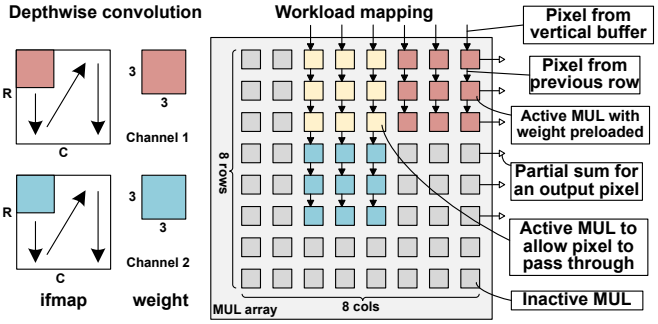

Fig. 6. Illustration of depthwise workload mapping on an 8×8 MUL array.

buffer to produce 2^n *ofmap* pixels concurrently. Cuboids of *ifmap* channel pixels are reused by 2^n times before they leave the MUL array. The final *psums* of respective *ofmap* channels are collected at respective rows.

Figure 6 shows how the CONV unit supports 3×3 depthwise convolution. 3×3 depthwise CONV is commonly found in recent CNN architectures [4], [5]. On an 8×8 MUL array, two weight kernels are processed concurrently. Since the CONV unit adopts row reduction and weight stationary with *ifmap* pixels flowing top to bottom, the weight kernels are preloaded on the MUL array in the diagonal direction without overlapping. Each MUL row produces a *psum*. Every 3 *psums* are added to produce one complete *psum*. For weight kernel larger than 3×3, some preprocessing are required to break the convolution into smaller pieces for the CONV unit to handle [8].

The CONV unit also supports standard convolution. The standard convolution is viewed as multiple depthwise convolutions. However, since our design is optimized for pointwise and depthwise operations, data reuse and speed performance are less efficient in this case when compared to other hardware accelerators.

IV. Implementation and Simulation Results

A. Implementation

The proposed design is implemented with a configurable MUL array of size $2^n \times 2^n$. The vertical buffer has a capacity of $4 \times 2^n \times 8$ bits to store *ifmap* pixels. The horizontal and *psum* buffers have the capacity of $4 \times 2^n \times 2^n \times 8$ bits and $8 \times 2^n \times 16$ bits, respectively. The design is synthesized at 100 MHz using 40nm CMOS technology. Area is extracted using Cadence Genus, while energy and power data are extracted using Cadence Joules.

TABLE I
PERFORMANCE AND REQUIRED SRAM SIZE GIVEN MUL ARRAY SIZE AND WORKLOAD, FOR $n = 4$ AND 5

Type	Input workload	Kernel workload	Output size	PE Size	Bit width (In, W, Out)	Required SRAM (KB)	Clock cycles	Performance (GOPS)	Efficiency (TOPS/W)
PW	32 x 32 x 16	1 x 1 x 16 x 16	32 x 32 x 16	16 x 16	(8, 8, 16)	80.3	1044	50.2	3.10
PW	32 x 32 x 32	1 x 1 x 32 x 32	32 x 32 x 32	32 x 32	(8, 8, 16)	161	1060	197.8	3.49
DW	32 x 32 x 5	3 x 3 x 5	30 x 30 x 5	16 x 16	(8, 8, 16)	22.6	1470	5.5	1.78
DW	32 x 32 x 10	3 x 3 x 10	30 x 30 x 10	32 x 32	(8, 8, 16)	45.2	1920	8.4	1.26

TABLE II
COMPARISON WITH RECENT ACCELERATORS

	Technology (nm)	Area (mm^2)	Voltage (V)	Frequency (MHz)	Power (mW)	Efficiency (TOPS/W)	Normalized efficiency (TOPS/W)
Eyeriss [7]	65	12.25	1.0	200	278	0.24	0.48
Origami [12]	65	3.09	1.2	500	449	0.437	0.874
KOP3 [8]	65	3.98	1.0	200	72	1.25	2.5
Leuven [13]	40	2.4	0.85	200	76	0.94	0.94
Proposed design	40	1.03 (CONV unit only)	0.85	100	25.3	3.13	3.13

TABLE III
PERFORMANCE AND EFFICIENCY ON VARIOUS CNN MODELS FOR $n = 5$

	Year	Performance (GOPS)	Efficiency (TOPS/W)
Xception [4]	2017	96.4	3.17
MobileNetv1 [5]	2017	79.3	3.13
MobileNetv2 [6]	2019	52.8	2.89

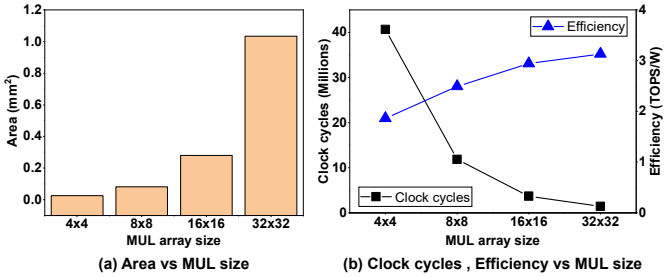

Fig. 7. Area, clock cycles and efficiency when processing MobileNetv1 under various MUL array size.

B. Results and performance analysis

Figure 7 shows how the performance scales when the CONV unit varies its MUL array size. When factor n varies from 2 to 5, the area of the CONV unit increases by about 3.4× in average at every step. Besides, the number of clock cycles reduces by 3.1× in average at every step when processing MobileNetv1 [5]. The CONV unit's overall efficiency increases when the MUL array scales up.

Figure 8 details the performance of the CONV unit when processing MobileNetv1. Increasing the MUL array size allows more than 3.2 times speed up on all pointwise workload and at least 1.2 times on all depthwise workload. On a 32×32 MUL array, the processing time on the heaviest pointwise workload requires only 0.59ms. As a result, processing MobileNetv1 inference requires only 14.3 ms/frame.

Table I shows the performance and required SRAM size of the CONV unit, given the MUL array size and target workload. All required data and the computed results are assumed to be stored in the global SRAM buffer. The throughput performance increases when MUL array size increases. The design

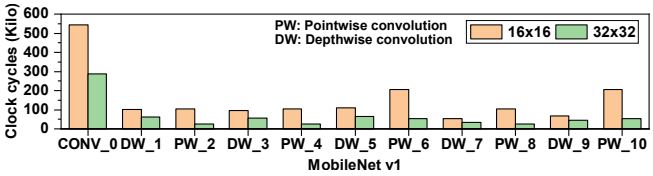

Fig. 8. Number of clock cycles to process first 11 layers in MobileNetv1, when $n = 4$ and 5.

achieves higher efficiency for pointwise convolution, but lower for depthwise convolution, when MUL array size scaled up. This is possibly because, more MUL units are activated to allow passing *ifmap* pixels, thus drawing energy. However, since depthwise convolution only takes up few percents of total workload, this drawback may not bring significant impact on the overall CNN processing efficiency.

Table III shows the performance and efficiency of the proposed design when running different CNN models. The efficiency is close to the peak efficiency when processing pointwise convolution. The efficiency drop is possibly because of the hindered processing of other CNN workloads.

Table II compares the proposed design with other hardware accelerators. Normalized efficiency is estimated based on the technology scaling. The power is scaled by 2× when normalized from 65nm to 40nm. Our design achieves an efficiency of 3.13 TOPS/W, which is 25% better than the state-of-the-art.

V. CONCLUSION

This paper proposes an energy-efficient CONV unit, which explores weight stationary and row-based adder-tree architecture to optimize pointwise and 3×3 depthwise convolution workload for edge-computing applications. The CONV unit also features a configurable multiplier (MUL) array with the factor of 2^n. When running MobileNetV1, our prototype of 32×32 CONV unit consumes 25.3 mW when operates at 100 MHz in 0.85V/40nm CMOS process, achieving an overall efficiency of 3.13 TOPS/W.

ACKNOWLEDGMENT

We thank the Programmatic grant no. A1687b0033, Singapore RIE 2020, AME domain.

REFERENCES

[1] M. Szarvas, A. Yoshizawa, M. Yamamoto, and J. Ogata, "Pedestrian detection with convolutional neural networks," in *IEEE Intelligent Vehicles Symposium, Proceedings*, vol. 2005, 2005, pp. 224–229.

[2] J. Zheng, Y. Wang, and W. Zeng, "CNN Based Vehicle Counting with Virtual Coil in Traffic Surveillance Video," in *Proceedings - 2015 IEEE International Conference on Multimedia Big Data, BigMM 2015*, jul 2015, pp. 280–281.

[3] J. Wang, Y. Ma, L. Zhang, R. X. Gao, and D. Wu, "Deep learning for smart manufacturing: Methods and applications," *Journal of Manufacturing Systems*, 2018.

[4] F. Chollet, "Xception: Deep Learning with Depthwise Separable Convolutions," *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, vol. 2017-Janua, pp. 1800–1807, oct 2016.

[5] A. G. Howard *et al.*, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," *arXiv preprint arXiv:1704.04861*, 2017.

[6] M. Sandler *et al.*, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," *arXiv preprint arXiv:1801.04381v4*, 2019.

[7] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," *IEEE Journal of Solid-State Circuits*, vol. 1, pp. 127–138, jan 2016.

[8] J. Yue *et al.*, "A 3.77TOPS/W Convolutional Neural Network Processor with Priority-Driven Kernel Optimization," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 66, pp. 277–281, feb 2019.

[9] L. Bai, Y. Zhao, and X. Huang, "A CNN Accelerator on FPGA Using Depthwise Separable Convolution," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 65, no. 10, pp. 1415–1419, oct 2018.

[10] H. N. Wu and C. T. Huang, "Data Locality Optimization of Depthwise Separable Convolutions for CNN Inference Accelerators," in *Proceedings of the 2019 Design, Automation and Test in Europe Conference and Exhibition, DATE 2019*, may 2019, pp. 120–125.

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

[12] L. Cavigelli and L. Benini, "Origami: A 803-GOp/s/W Convolutional Network Accelerator," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 27, pp. 2461–2475, nov 2017.

[13] B. Moons and M. Verhelst, "An Energy-Efficient Precision-Scalable ConvNet Processor in 40-nm CMOS," *IEEE Journal of Solid-State Circuits*, vol. 52, pp. 903–914, apr 2017.