LETTER

An efficient ReRAM-based inference accelerator for convolutional neural networks via activation reuse

Yan Chen[1,2a), **Jing Zhang**[1], **Yuebing Xu**[1], **Yingjie Zhang**[3], **Renyuan Zhang**[2], **and Yasuhiko Nakashima**[2]

Abstract In this paper, a novel resistive random access memory (ReRAM) based accelerator is proposed for convolution neural network (CNN) inference accelerations. In ReRAM-based CNN computation, weight parameters can be pre-programmed in ReRAM crossbar arrays, and activations are generated by processing the multiplication-and-accumulation (MAC) operations in the ReRAM crossbar arrays. However, prior works cannot reuse activations in computation, in which the activation dominates the data movements and raises significant energy cost. To deal with this dilemma, a tiling-based dataflow is proposed to enable activation reuse among adjacent ReRAM crossbar arrays to reduce the activation movements. We then develop a ReRAM-based CNN accelerator that can well suit the dataflow to reduce the cost of ReRAM access. Evaluation results show that the proposed design achieves 1.8× energy saving and 2.8× bandwidth saving compared with a state-of-the-art PipeLayer accelerator.

Keywords: ReRAM-based accelerator, convolution neural networks, data reuse, processing-in-memory
Classification: Integrated circuits

1. Introduction

Convolutional neural networks (CNNs) [1] have been extensively adopted in various computer vision tasks, giving impressive accuracy breakthroughs in classification, recognition, and so forth [2, 3, 4, 5, 6]. These significant accuracy improvements mainly come from the successes of both scaling up neural networks to tens of millions of parameters [6, 7, 8, 9] and learning from the massive amounts of datasets [10, 11]. However, the very deep CNNs and large-scale datasets also lead to the high demand of computation capability. For example, the representive AlexNet model [2], which is composed of 60 MB weight parameters and 630 MB connections, are over 100× more than those of Lenet5 [12]. Consequently, it is critical to develop efficient hardware solutions for large-scale CNN deployments.

Various dedicated hardware solutions have been developed for efficient CNN accelerations. In particular, the emerging novel nonvolatile memories such as metal-oxide resistive random access memory (ReRAM) [13, 14, 15, 16, 17, 18] have the capability of performing arithmetic operations beyond data storage. For example, PRIME [13] dynamically configures ReRAM crossbar arrays as process elements or as normal memory for energy harvest. PipeLayer [18] utilizes weight replication to boost the throughput and performs multiplication-and-accumulation (MAC) operations in ReRAM crossbar arrays. In the ReRAM-based MAC computation, the weight parameters can be pre-programmed in ReRAM crossbar arrays before calculation, and the activations are generated by processing the MAC operations in the ReRAM crossbar arrays. This outperforms conventional FPGA- and ASIC-based hardware solutions [19, 20, 21, 22, 23, 24, 25, 26, 27, 28] in both memory access and computation for large-scale CNN accelerations, which struggle on the huge performance gap between computation and memory. However, existing ReRAM-based CNN accelerators have to consume considerable energy and bandwidth for the activation access [13, 18], because the activations are dynamically generated during the MAC operations. Especially, the input activations are frequently loaded as inputs, which dominate the memory access in CNN deployments.

Fortunately, the significant reusable input activations in CNNs can be utilized to reduce the activation movements. For example, statistical results from representative CNNs, such as AlexNet, VGG, and ResNet, show that over 80% of the input activations in convolutional layers can be reused, as shown in Table I. However, existing ReRAM-based accelerators [13, 18] cannot reuse input activations because they cannot tackle the overlapped reusable input activations in the convolutional layers, resulting in significant energy cost. Furthermore, exploiting activation reuse in ReRAM crossbar arrays directly as conventional accelerators [20, 23, 29, 30] do will incur severe performance and energy overheads, due to the required heavy weight movements in MAC operations. Consequently, it is difficult to reuse input activations in the ReRAM-based computation. Thus, an efficient ReRAM-based accelerator which enables to eliminate the redundant activation access is urgently required for large-scale CNN deployments.

Table I. Statistical reusable input activations in Conv layers of famous CNNs.

CNNs	ratio of reusable activations in Conv layers
AleNet	92.0%
VGG16	88.9%
ResNet18	83.2%

[1]College of Electrical and Information Engineering, Hunan University, China
[2]Graduate School of Information Science, Nara Institute of Science and Technology, Ikoma-shi 630-0192, Japan
[3]College of Computer Science and Electronic Engineering, Hunan University, China
a) chenyan1226@hnu.edu.cn

DOI: 10.1587/elex.16.20190396
Received June 20, 2019
Accepted July 24, 2019
Publicized August 26, 2019
Copyedited September 25, 2019

In this paper, a ReRAM-based accelerator architecture and dataflow is introduced for efficient CNN deployments. The dataflow tiles the convolutional layers based on the size of stride so that the activations can be reused without the impacts of the different stride sizes. We then develop a ReRAM-based accelerator to well suit the dataflow and to enable activation reuse by shifting the reusable input activations to adjacent ReRAM crossbar arrays, without reloading inputs, for efficient energy and bandwidth savings. Evaluation results show that the proposed design achieves 1.8× energy saving and 2.8× bandwidth saving than the state-of-the-art PipeLayer accelerator.

2. Preliminaries and motivations

Fig. 1 illustrates the convolving computation of a state-of-the-art accelerator, PipeLayer, for a convolutional (Conv) layer. To begin with, Fig. 1(a) depicts the basic computation of a Conv layer. The output activations (*out*) in each output feature map ($R \times C$) are generated by convolving input activations (*ia*) with the shared N channels $k_x \times k_y$ kernel weights (*w*) under a stride S. M groups of *w* generate M channels of *out*. Fig. 1(b) depicts the convolution computation of PipeLayer in ReRAM crossbar arrays. Weights are replicated in the ReRAM crossbar arrays, and input activations are reshaped into vectors ($k_x \times k_y \times N$) for the MAC operations without reuse.

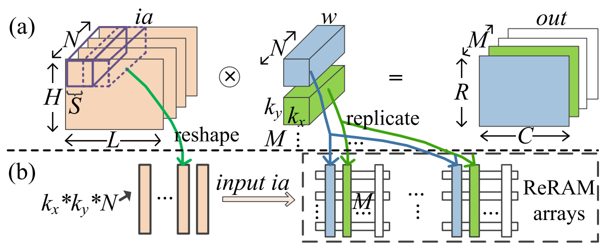

Fig. 1. Inference example of PipeLayer. (a) Conv layer. (b) Mapping the Conv layer to ReRAM crossbar arrays with the weight replication.

Both input activation loading and output activation storing dominate the memory access in ReRAM-based CNN inference, since activations are generated during the MAC operations and weights can be pre-programmed without update. In particular, the input activation loading operation dominates the activation access in Conv layer's deployments. The total capacity of the loaded input activations for a Conv layer (Fig. 1(a)) with a $k_x \times k_y \times N \times R \times C$ size is $k_x \times k_y \times N/M$ times larger than that for the output activation storage. Fortunately, there are large amounts of reusable input activations in Conv layers, as shown in Table I. Also, Conv layers occupy over 90% of the computation in most popular CNN models [19]. These two characteristics motivate the authors to reduce the redundant memory access by utilizing the reusable input activations. Nevertheless, the reusable input activations are sensitive to the size of stride S. To deal with the different sizes of S, we tile Conv layers into pieces based on S, so that the input activations can be reused without the impact of the stride.

3. Proposed dataflow and architecture

Proposed Tiling-based Dataflow Fig. 2 outlines the proposed dataflow, which enables input activation reuse in ReRAM crossbar arrays for Conv layers. The key objective is to shift prior loaded input activations to adjacent groups of ReRAM crossbar arrays for reuse.

The tiling-based dataflow includes two key steps. First, to deal with different sizes of S, the input activations and weights are partitioned into pieces based on S, as shown in Fig. 2① and ②. After partition, each piece of activations has N channels and $S \times k_y$ activations in each channel. In this case, the virtual new stride is resized to "1" so that the input activations can be efficiently reused by shifting them to adjacent group of ReRAM crossbar arrays. There are no overlapped activations among pieces of inputs so that they can be reused without the impact of different stride sizes. Second, the ReRAM crossbar arrays are organized into groups ($G_1, G_2, \ldots$) mainly based on the stride and kernel sizes, as shown in Fig. 2③. The ReRAM crossbar arrays within the same group share the same input activations, as shown in ④, and the weights are replicated to boost the throughput, as shown in ⑤. Based on the two above key operations, the dataflow enables to avoid the impacts of the stride in the process of reusing input activations.

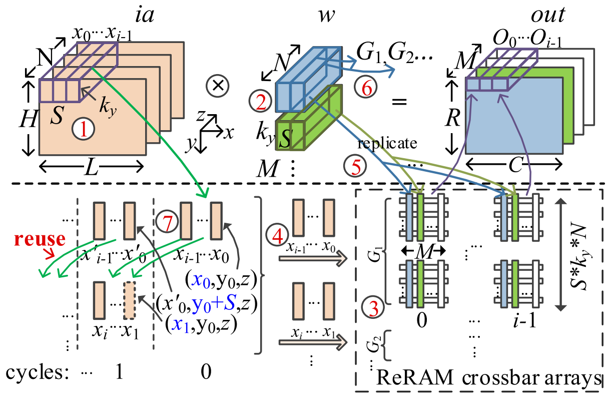

Fig. 2. Proposed dataflow with input activation reuse.

Fig. 2⑥ and ⑦ illustrate an example of reusing input activations. The different pieces of weights are mapped to different groups of ReRAM crossbar arrays, as shown in ⑥, which provides the opportunity to reuse input activations in adjacent group of ReRAM crossbar arrays. The loaded input activation vectors, for example, $\langle x_1, \ldots, x_{i-1} \rangle$ (without x_0, $i < C$), for G_1 at cycle #0, as shown in ⑦, can be reused by shifting them to adjacent group of ReRAM crossbar arrays (G_2) at the followed cycle #1. This is because they are shared by the adjacent output activation vector (i.e., $\langle O_1, \ldots, O_{i-1} \rangle$). Consequently, the input activations can efficiently be reused by shifting them to adjacent groups of ReRAM crossbar arrays.

The proposed dataflow outperforms prior accelerators, such as PRIME and PipeLayer, which do not support activation reuse. The reduced redundant memory access for a Conv layer reaches $1 - S/k_x$ of the total loaded input activations, which contributes to huge energy and bandwidth saving. In addition, the dataflow can also be applied to the fully-connected (FC) layers of CNNs. This is because the FC layers can be regarded as the typical convolution computation of Conv layers with a "1×1" edge size of output feature maps.

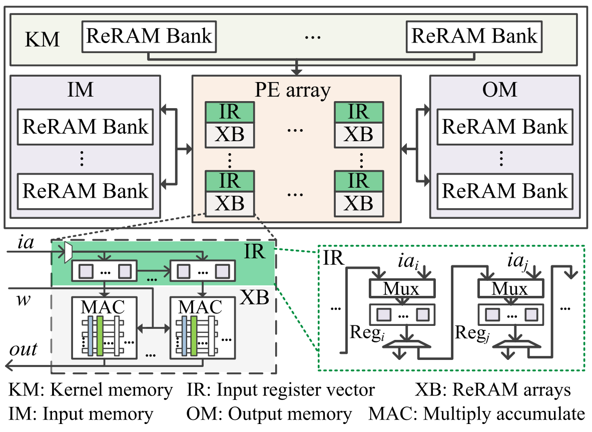

Fig. 3. Proposed architecture with activation reuse.

Proposed ReRAM-based Architecture Fig. 3 depicts the proposed ReRAM-based architecture for CNN inference, which enables activation reuse and weight replication. Two activation memory components, IM and OM, alternately accommodate input activations and output activations, and the kernel memory component (KM) stores weights. Process element (PE) arrays are composed of input register components (IRs) and ReRAM crossbar array components (XBs), mainly performing the MAC operations. The memory access of activations includes loading input activations (*ia*) from IM/OM to IR and storing the generated output activations (i.e., *out*) back to OM/IM. In contrast, the weight parameters (*w*) can be pre-programmed and replicated in XBs before the MAC operations, which is similar to PRIME and PipeLayer.

IR plays a key role to enable input activation reuse. It temporally accommodates input activations from IM/OM before transferring them into XB for computation. IR is organized as a chain topology for accommodating input activations, so that they can be reused in adjacent registers for ReRAM crossbar arrays. For example, ia_i and ia_j denote the input activations of Fig. 2 for different ReRAM groups (i.e., G_1 and G_2). ia_i can be reused in computation by shifting them to the adjacent register vector (Reg_j) in the followed cycle. Consequently, the activations can be efficiently reused in the process of MAC operations.

4. Simulation results

Simulation Setup We conduct the evaluation of the proposed design (denoted as "Prop") by comparing with the state-of-the-art ReRAM-based accelerator baseline, PipeLayer [18]. Specifically, PipeLayer enables to replicate weights in ReRAM crossbar arrays for high throughput and to process the MAC operations in ReRAM memory. Three representative CNN benchmarks, AlexNet [2], VGG [4], and ResNet [6], are adopted with images from ImageNet as inputs. Details of the benchmarks are shown in Table II. The latency and energy overheads of ReRAM crossbar arrays are profiled by the NvSim [31] tool. The size of a ReRAM crossbar array is 128×128. The latency and energy for the read/write operations of ReRAM crossbar arrays are respectively 9.668 ns/110.53 ns per spike and 126.6 pJ/628.1 pJ per spike. The energy consumption of weight access is devoid because they can be pre-programmed in ReRAM crossbar arrays before the MAC operations. Each data is 16 bits, and the resolution of each ReRAM cell is 4 bits.

Table II. CNN workload specifications.

CNNs	total layers	Conv layers	FC layers
AleNet	8	5	3
VGG16	16	13	3
ResNet18	18	17	1

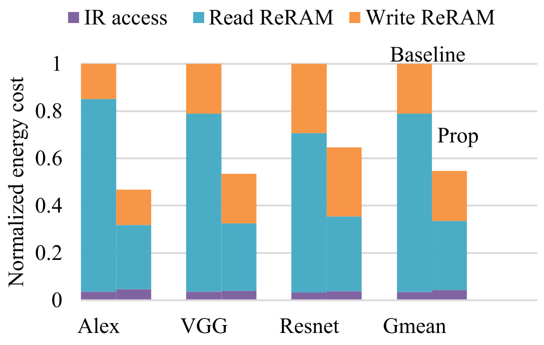

Fig. 4. Energy saving compared with the baseline.

Energy Saving Fig. 4 shows the normalized energy comparison between the baseline and the proposed architecture. The energy cost includes three folds: IR access, read data from ReRAM memory, and write data to ReRAM memory for output activation storage. On average, the proposed design achieves 1.8× energy saving than the baseline with 2.1×, 1.8×, and 1.5× energy saving on AlexNet, VGG, and ResNet, respectively. The energy saving comes from the reduced ReRAM memory access by input activation reuse.

Performance Fig. 5 shows the normalized execution time comparison of the proposed design against the baseline. On average, the proposed design takes a bit more execution time than the baseline, reaching 1.17×. This is because the tiling operation of the proposed dataflow incurs a bit more severe fragmentation issue, where the ReRAM crossbar arrays are idle in computation without sufficient inputs. However, the proposed design enables input activation reuse for significant energy saving. Furthermore, the Conv layers occupy most of the computation time (98.8% in the proposed design) for the CNN deployments. This further demonstrates that exploiting activation reuse on the Conv layers can achieve efficiency for the CNN deployments.

Bandwidth Fig. 6 shows the normalized bandwidth comparison between the proposed design and the baseline. The bandwidth is evaluated based on the data transmission of both input and output activations between the PEs and IM/OM components. On geometric average, the proposed design achieves 2.8× bandwidth saving than the baseline by reusing input activations. Specifically, the proposed design gains 3.2×, 2.8×, and 2.6× bandwidth saving than the baseline on AlexNet, VGG, and ResNet, respectively.

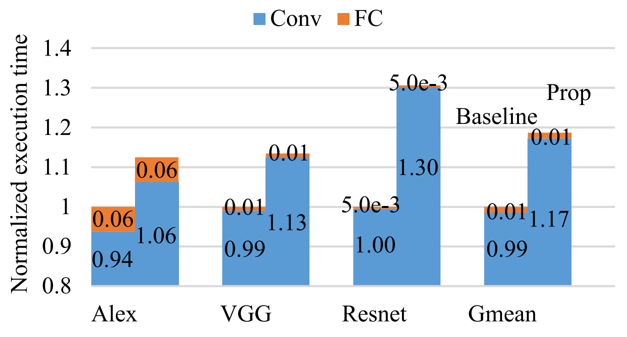

Fig. 5. Normalized execution time of the proposed design against the baseline for both Conv and FC layers.

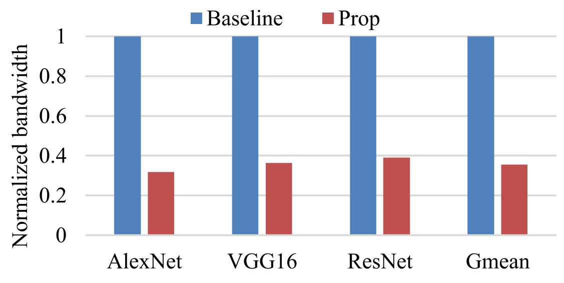

Fig. 6. Normalized bandwidth consumption compared with PipeLayer.

Table III. The representative layers of workloads.

CNNs	AlexNet				ResNet		
Layers	Cnv1	Cnv2	Cnv5	FC8	Cnv1	res2a_branch1	res2a_branch2a
Note	AC1	AC2	AC5	AF8	RC1	R2aB1	R2aB2a
Type	Conv	Conv	Conv	FC	Conv	Conv	Conv
N	3	96	384	4096	3	64	64
M	96	256	256	1000	64	64	64
k_x, k_y	11	5	3	1	7	1	3
S	4	1	1	1	2	1	1

Impact of Kernel, Stride, and Channel: To look insight into the energy impact of the different layers of CNNs, Fig. 7 evaluates the energy saving over the baseline based on the representative layers of the benchmarks, as shown in Table III. The VGG benchmark are devoid because their layers can be represented by the layers in Table III, for example, AC5 and R2aB2a have the same kernel size ($k_x = 3$) and stride size ($S = 1$) as the Conv layers of VGG.

Fig. 7 shows the normalized energy saving against the baseline based on the layers in Table III. It can be observed that (a) the proposed design can achieve efficient energy saving when the layer has a large kernel size and a small stride size. For example, the proposed design achieves up to 2.99× energy saving than the baseline in the AC2 layer with $k_x = 5$ and $S = 1$; (b) we gain efficient energy saving even the size of stride S is larger than 1. For example, the proposed design achieves 1.18× energy saving at AC1, where $S = 4$; and (c) the proposed design takes a bit more

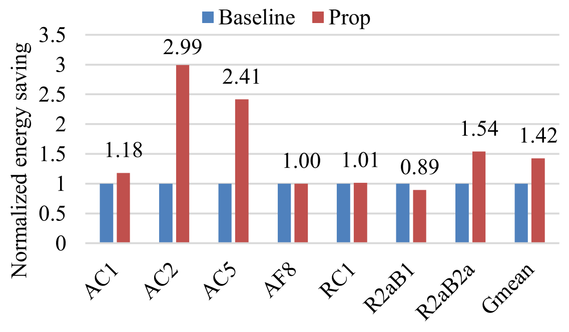

Fig. 7. Energy cost on layers of CNNs over the baseline.

energy consumption than the baseline when the kernel size of the convolution layer is 1, as shown in the R2aB1 layer, because of the fragmentation issue caused by the tiling operation. To sum it up, the proposed design can achieve efficient energy saving for most convolutional layers in CNN deployments.

5. Conclusion

A ReRAM-based dataflow and architecture have been introduced for better energy and bandwidth saving by exploiting input activation reuse to reduce redundant memory access. The input activations are reused by shifting them to adjacent ReRAM crossbar arrays to reduce the data movement between memory and process elements. Evaluation results show that the proposed design can achieve 1.8× energy saving and 2.8× bandwidth saving than a state-of-the-art PipeLayer accelerator.

Acknowledgments

This work is partially supported by the China Scholarship Council (CSC) and the National Natural Science Foundation of China (61573299).

References

[1] Y. LeCun and Y. Bengio: "Convolutional networks for images, speech, and time series," The Handbook of Brain Theory and Neural Networks **3361** (1995).

[2] A. Krizhevsky, *et al.*: "Imagenet classification with deep convolutional neural networks," NIPS (2012) 1097.

[3] M. D. Zeiler and R. Fergus: "Visualizing and understanding convolutional networks," ECCV (2014) 818 (DOI: 10.1007/978-3-319-10590-1_53).

[4] K. Simonyan and A. Zisserman: "Very deep convolutional networks for large-scale image recognition," ICLR (2015) 1150.

[5] C. Szegedy, *et al.*: "Going deeper with convolutions," CVPR (2015) 1 (DOI: 10.1109/CVPR.2015.7298594).

[6] K. He, *et al.*: "Deep residual learning for image recognition," CVPR (2016) 770 (DOI: 10.1109/CVPR.2016.90).

[7] G. Huang, *et al.*: "Deep networks with stochastic depth," ECCV (2016) 646 (DOI: 10.1007/978-3-319-46493-0_39).

[8] Y. Chen, *et al.*: "Dual path networks," NIPS (2017) 4467.

[9] H. Shin, *et al.*: "Deep convolutional neural networks for computer-aided detection: CNN architectures, dataset characteristics and transfer learning," IEEE Trans. Med. Imag. **35** (2016) 1285 (DOI: 10.1109/TMI.2016.2528162).

[10] J. Deng, *et al.*: "ImageNet: A large-scale hierarchical image database," CVPR (2009) 248 (DOI: 10.1109/CVPR.2009.5206848).

[11] A. Karpathy, *et al.*: "Large-scale video classification with convolutional neural networks," CVPR (2014) 1725 (DOI: 10.1109/CVPR.2014.223).
[12] Y. LeCun, *et al.*: "Comparison of learning algorithms for handwritten digit recognition," ICANN **60** (1995) 53.
[13] P. Chi, *et al.*: "PRIME: A novel processing-in-memory architecture for neural network computation in reram-based main memory," ISCA (2016) 27 (DOI: 10.1109/ISCA.2016.13).
[14] A. Shafiee, *et al.*: "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," ISCA (2016) 14 (DOI: 10.1109/ISCA.2016.12).
[15] Y. Chen, *et al.*: "A novel memristor-based restricted Boltzmann machine for contrastive divergence," IEICE Electron. Express **15** (2018) 20171062 (DOI: 10.1587/elex.15.20171062).
[16] S.-Y. Sun, *et al.*: "Quaternary synapses network for memristor-based spiking convolutional neural networks," IEICE Electron. Express **16** (2019) 20190004 (DOI: 10.1587/elex.16.20190004).
[17] S. Yamamori, *et al.*: "Efficient mini-batch training on memristor neural network integrating gradient calculation and weight update," IEICE Trans. Fundamentals **E101-A** (2018) 1092 (DOI: 10.1587/transfun.E101.A.1092).
[18] L. Song, *et al.*: "Pipelayer: A pipelined reram-based accelerator for deep learning," HPCA (2017) 541 (DOI: 10.1109/HPCA.2017.55).
[19] J. Qiu, *et al.*: "Going deeper with embedded fpga platform for convolutional neural network," FPGA (2016) 26 (DOI: 10.1145/2847263.2847265).
[20] Y.-H. Chen, *et al.*: "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," IEEE J. Solid-State Circuits **52** (2017) 127 (DOI: 10.1109/JSSC.2016.2616357).
[21] Z. Du, *et al.*: "ShiDianNao: Shifting vision processing closer to the sensor," ISCA (2015) 92 (DOI: 10.1145/2749469.2750389).
[22] B. Liu, *et al.*: "Addressing the issue of processing element under-utilization in general-purpose systolic deep learning accelerators," ASP-DAC (2019) 733 (DOI: 10.1145/3287624.3287638).
[23] J. Jo, *et al.*: "Energy-efficient convolution architecture based on rescheduled dataflow," IEEE Trans. Circuits Syst. I, Reg. Papers **65** (2018) 4196 (DOI: 10.1109/TCSI.2018.2840092).
[24] C. Zhang, *et al.*: "Energy-efficient CNN implementation on a deeply pipelined FPGA cluster," ISLPED (2016) 326 (DOI: 10.1145/2934583.2934644).
[25] K. Guo, *et al.*: "Angel-eye: A complete design flow for mapping CNN onto embedded FPGA," IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. **37** (2017) 35 (DOI: 10.1109/TCAD.2017.2705069).
[26] X. Lian, *et al.*: "High-performance FPGA-based CNN accelerator with block-floating-point arithmetic," IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **27** (2019) 1874 (DOI: 10.1109/TVLSI.2019.2913958).
[27] J. IJzerman, *et al.*: "AivoTTA: An energy efficient programmable accelerator for CNN-based object recognition," SAMOS (2018) 28 (DOI: 10.1145/3229631.3229637).
[28] R. Andri, *et al.*: "YodaNN: An architecture for ultralow power binary-weight CNN acceleration," IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. **37** (2017) 48 (DOI: 10.1109/TCAD.2017.2682138).
[29] A. Aimar, *et al.*: "Nullhop: A flexible convolutional neural network accelerator based on sparse representations of feature maps," IEEE Trans. Neural Netw. Learn. Syst. **30** (2019) 644 (DOI: 10.1109/TNNLS.2018.2852335).
[30] M. Gao, *et al.*: "Tangram: Optimized coarse-grained dataflow for scalable NN accelerators," ASPLOS (2019) 807 (DOI: 10.1145/3297858.3304014).
[31] X. Dong, *et al.*: "Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. **31** (2012) 994 (DOI: 10.1109/TCAD.2012.2185930).