Tactics to Directly Map CNN graphs on Embedded FPGAs

Kamel Abdelouahab, Maxime Pelcat, Jocelyn Sérot, Cédric Bourrasset, François Berry

TACTICS TO DIRECTLY MAP CNN GRAPHS ON EMBEDDED FPGAS

Kamel Abdelouahab[1], Maxime Pelcat[1,2], Jocelyn Sérot[1], Cédric Bourrasset[3], and François Berry[1]

[1]*Institut Pascal, Univérsite Clermont Auvergne, France*
[2]*IETR, INSA Rennes, France*
[2]*Atos/Bull CEPP CINES, France*

ABSTRACT

Deep Convolutional Neural Networks (CNNs) are the state-of-the-art in image classification. Since CNN feed forward propagation involves highly regular parallel computation, it benefits from a significant speed-up when running on fine grain parallel programmable logic devices. As a consequence, several studies have proposed FPGA-based accelerators for CNNs. However, because of the large computational power required by CNNs, none of the previous studies has proposed a *direct* mapping of the CNN onto the physical resources of an FPGA, allocating each processing actor to its own hardware instance.

In this paper, we demonstrate the feasibility of the so called *direct hardware mapping (DHM)* and discuss several tactics we explore to make DHM usable in practice. As a proof of concept, we introduce the HADDOC2 open source tool, that automatically transforms a CNN description into a synthesizable hardware description with platform-independent direct hardware mapping[1].

1 INTRODUCTION

Convolutional Neural Networks (CNNs) [2] have become a *de-facto* standard for increasing the robustness and accuracy of machine vision systems. However, this accuracy comes at the price of a high computational cost. As a result, implementing CNNs on embedded devices with real-time constraints is a challenge. A solution to this challenge is to take advantage of the massive fine grain parallelism offered by embedded Field-Programmable Gate Arrays (FPGAs) and benefit from the extensive concurrency exhibited by CNN-based algorithms. By embedded FPGAs, we refer to devices with limited power consumption and cost, typically under 20W and 300$. When porting a CNN to an embedded FPGA, the problem boils down to finding an efficient mapping between the computational model of the CNN and the execution model supported by the FPGA. Based on works related to the implementation of real-time vision applications on FPGA-powered embedded platforms [3], we advocate the use of a *dataflow* model to solve this mapping problem. In this approach, a CNN algorithm is described as a graph of dataflow actors exchanging data through unidirectional channels and this dataflow graph is statically and physically mapped onto the target FPGA using a library of pre-defined computing elements implementing actors.

In the sequel, we demonstrate the feasibility of the *Direct Hardware Mapping (DHM)* approach for implementing CNN-based applications onto embedded FPGAs. DHM associates each CNN processing entity to private resources, maximizing parallelism. To support this demonstration, we introduce HADDOC2, a framework that provides a fully automated hardware generation for CNNs using DHM. The HADDOC2 tool is compatible with the Caffe deep learning framework [4] and generates platform-independent VHDL synthesizable code.

The paper is organized as follows. Section 2 reviews state-of-the-art implementations of CNNs on FPGAs. Section 3 recalls the main features of CNNs from a computational point of view, focusing on parallelism issues. Section 4 describes the DHM approach and how it is supported by

[1]This is a pre-print version. Please refer to the original paper in [1]

the HADDOC2 framework. Section 5 presents an assessment of the efficiency of the approach, reporting performance and resource utilization of DHMs-based implementations for three CNNs, and Section 6 concludes the paper.

2 RELATED WORK

Several studies leverage on FPGA computational power to implement the feed-forward propagation of CNNs. A complete review of these studies can be found in [5]. In most approaches, CNN-based applications are implemented by mapping a limited subset of processing elements onto the target device, multiplexing in time the processing elements and processing data in an SIMD fashion. This is the case for instance in [6] where authors describe a CNN accelerator implemented on a Zynq XC706 board.

The dataflow-based implementation of CNNs is investigated in [7] where authors describe Neuflow, an acceleration engine for CNNs relying on a dataflow execution model. The CNN graph is transformed into a set of dataflow instructions, where each instruction is described as a hardware configuration of 2D-processing elements called *Processing tiles (PTs)*. The execution of the graph is carried out by sequencing the instructions on an FPGA.

The previously evoked approaches require an external memory to store intermediate results, which in turn, even with the help of a DMA, limits the final speedup. The study in [8] features a partitioning of the CNN graph with one bit-stream per subgraph in a way that only on-chip memory is needed to store intermediate results. This however requires the reconfiguration of the FPGA whenever data has to enter a different subgraph, which adds a substantial reconfiguration time overhead. By contrast, the DHM approach introduced in the present paper performs all processing *on the fly* and does not require any external memory to store intermediate results. Throughput is therefore not influenced by off-chip memory bandwidth.

3 CNN COMPUTATION

A typical CNN structure performs a succession of convolutions interspersed with sub-sampling layers. The last layers of a CNN are fully connected layers performing classification. Convolutional layers are the most computationally intensive layers and are commonly responsible for more than 90% of the CNN execution time [9]. As a consequence, we focus in this paper on the implementation of convolutional layers.

A convolutional layer (l) extracts N feature maps from C input channels by performing N convolutions of size $K \times K$ on each input. This filtering is followed by the application of a non-linear activation function act and a bias term b_n to each set of features. As shown in equation 1, $N \times C$ convolutions (resulting in $N \times C \times K \times K$ multiplications) are required to process a given layer.

$$
\begin{aligned}
&\forall l = 1 : L \textit{ (Number of layers)} \\
&\quad \forall n = 1 : N^{(l)} \textit{ (Number of output feature maps)} \\
&\qquad \boldsymbol{f_n^{(l)}} = \text{act}\left[\boldsymbol{b_n^{(l)}} + \sum_{c=1}^{C^{(l)}} conv(\boldsymbol{\phi_c^{(l)}}, \boldsymbol{w_{nc}^{(l)}})\right]
\end{aligned} \tag{1}
$$

where $\boldsymbol{f_n^{(l)}}$ is the n^{th} output feature map of layer (l), $\boldsymbol{\phi_c^{(l)}}$ is the c^{th} input feature map and $\boldsymbol{w_{nc}^{(l)}}$ is a pre-learned filter.

The computation described in Equation 1 exhibits four sources of concurrency. First, CNNs have a feed-forward hierarchical structure consisting of a succession of data-dependent layers. Layers can therefore be executed in a *pipelined* fashion by launching layer (l) before ending the execution of layer $(l-1)$. Second, each neuron of a layer can be executed independently from the others, meaning that each of the $N^{(l)}$ element of equation 1 can be computed in parallel. Third, all of the convolutions performed by a single neuron can also be evaluated simultaneously by computing concurrently the $C^{(l)}$ elements of equation 1. Finally, each 2D image convolution can be implemented in a pipelined fashion [10] computing the $K \times K$ multiplications concurrently.

4 Direct Hardware Mapping of CNNs

A CNN can be modeled by a dataflow process network (DPN) where nodes correspond to processing actors and edges correspond to communication channels. Each actor follows a purely data-driven execution model where execution (firing) is triggered by the availability of input operands [11]. The DHM approach consists of *physically* mapping the whole graph of actors onto the target device. Each actor then becomes a computing unit with its specific instance on the FPGA and each edge becomes a signal.

This approach fully exploits CNN concurrency. All neurons in a layer are mapped on the device to take advantage of inter-neuron parallelism (Fig. 1-a). In neurons, each convolution is mapped separately (Fig. 1-b) and finally, within a convolution engine, each multiplier is instantiated separately (Fig 1-c). As an example, Fig. 2 illustrates how a convolution layer C1 ($C = 3, N = 5, K = 3$) extracts 5 features from a 3-feature input pixel flow. In this example, 15 convolutions and 5 activation blocks are mapped onto the FPGA as a result of the layer graph transformation, which corresponds to 135 multiplications, 20 sums and 5 activations. DHM of pooling layers is also performed but lowest-level implementation elements are kept out of the scope of this paper.

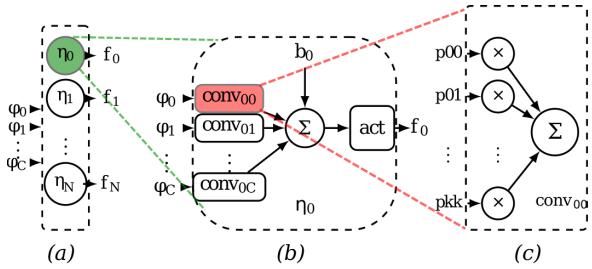

Figure 1: The 3 levels of DHM use on CNN entities: (a) in the convolution layers, (b) in the neurons, (c) in the convolution engines.

The *direct hardware mapping* approach exemplified above makes external memory accesses unnecessary, while classical FPGA implementations store intermediate results or parameters on external memory. The processing is then performed *on-the-fly* on *streams* of feature maps. Moreover, due to the fully pipelined execution model, the global throughput is only limited by the maximum clock frequency.

These advantages come at the cost of a high resource consumption since the whole graph has to be mapped onto the physical resources of the FPGA. This resource consumption could make DHM impractical. It is therefore crucial for DHM to explore tactics that efficiently translate CNN actors into hardware. The most important issues to solve are those related to the representation of numbers and the implementation of multiplications.

4.1 Approximate fixed-point data representations

Several studies have demonstrated that CNNs, and more generally deep learning applications, usually tolerate approximate computing with short fixed-point arithmetic. Frameworks such as Ristretto [12] fine-tune a CNN data representation to support fixed-point numerical representations with variable data lengths. The DHM approach advocated in this paper takes advantage of data and parameter quantization to reduce the amount of hardware resources by first inferring the minimal required precision and then *deriving* the hardware resources that exactly match this precision.

4.2 Implementing Multiplications with Logic Elements

Convolutions require many multiplications. If these multiplications are implemented using hard-wired Digital Signal Processing (DSP) blocks within the target FPGA, they become the bottleneck limiting the size of the implemented CNN. For instance, the second layer of the LeNet5 network [2] ($C = 6, N = 16, K = 5$) requires 2400 multipliers, exceeding the number of multipliers provided by the DSP blocks of most FPGAs, and especially of embedded FPGAs. We overcome this problem by systematically forcing the synthesis tool to implement multiplications with logical elements

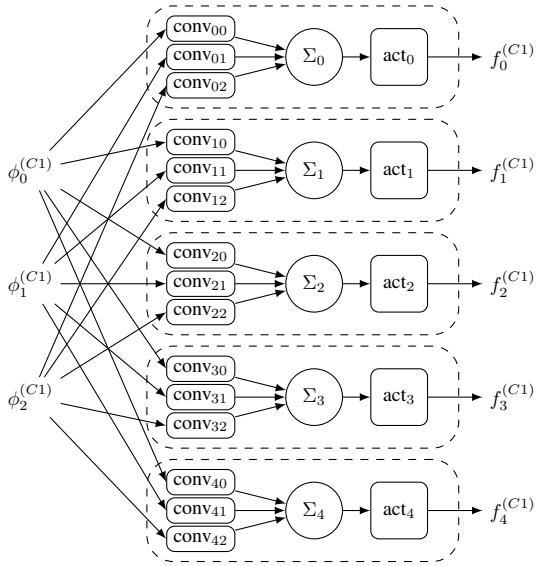

Figure 2: Applying the 3 levels of DHM (Fig. 1) to a convolutional layer C1 (N=5, C=3, K=3): 15 separate convolution engines (135 Multipliers and 15 adders) plus 5 adders and 5 activation blocks are required to process the fully parallel layer (bias omitted).

instead of DSP blocks, leading the resulting implementations to rely on AND gates and trees of half-adders [13].

In addition, we take advantage of the fact that the convolution kernels – and hence one operand of each multiplication – are constants derived from an offline training stage. Multipliers can thus be specialized to their constants. While this approach limits the flexibility of the system because it requires to re-synthesize the VHDL design whenever parameters values are changed, it delegates to the synthesis tool the task to perform low-level area and performance optimization. More particularly, multiplications by 0 (*resp* 1) are removed (*resp.* replaced by a simple signal connection) and multiplications by a power of 2 are transformed into shift registers.

4.3 Automated Hardware Generation with Haddoc2

The Haddoc2 framework is a set of tools built upon the DHM principle and upon the optimization tactics described in previous section. It generates a platform-independent hardware description of a CNN from a Caffe model [4]. CNN layers in Haddoc2 are described using a small number of basic predefined actors written in structural VHDL. These actors follow a dataflow execution semantics. The output can be synthesized for any FPGA device with tools supporting VHDL 93. The Haddoc2 framework and the library of CNN IP-cores supporting the DHM approach are open-source and available[2].

5 Experimental Results with Haddoc2

As proofs of concept, FPGA-based accelerators for three benchmark CNNs are implemented with Haddoc2: LeNet5 [2], SVHN [14] and CIFAR10 [15]. Table 1 details the topology of these CNNs where *mpool* refers to the pooling layer that reduces the dimensionality of each feature map and *tanh* is the hyperbolic tangent activation function. The Cifar10 and SVHN CNNs share the same topology with different kernel values, which is useful to study the impact of kernel proportions on a DHM-based implementation. For each network, the fixed-point representation is chosen to respect the classification accuracy, as a result of an exploration shown in Fig. 3. The study of quantization effects on CNNs is beyond the scope of this paper and can be found, for instance, in [16, 12]. In our case, a 3-bit representation is chosen for the LeNet5 network and a 6-bit representation for SVHN

[2]https://github.com/KamelAbdelouahab/haddoc2.

and CIFAR10[2]. The shares of its zero-valued parameters, one-valued parameters and power-of-two-valued parameters are evaluated and reported in table 1. They represent, by far, more than 90% of the parameters in all cases.

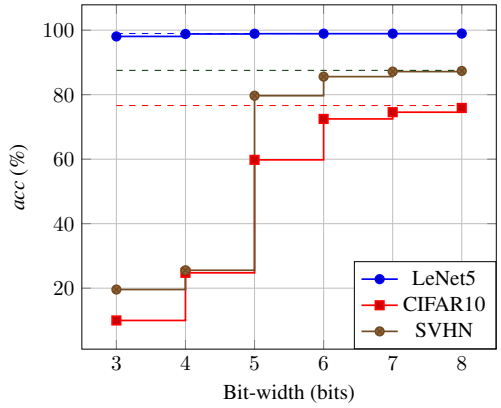

Figure 3: Evolution of classification accuracy vs bit-width for the studied CNNs. The dashed lines refers to accuracy of the baseline 32-bits floating point model.

Table 1: Topology of the convolutional layers of the studied CNNs.

	LeNet5 [2]			Cifar10 [15]			SVHN [14]		
Input Patches	28 x 28			32 x 32 x 3			32 x 32 x3		
Layer parameters	N	C	K	N	C	K	N	C	K
conv1+mpool+tanh	20	1	5	32	3	5	32	3	5
conv2+mpool+tanh	50	20	5	32	32	5	32	32	5
conv3+mpool+tanh	–	–	–	64	32	5	64	32	5
accuracy float (%)	98.96			76.63			87.54		
selected bit-width	3			6			6		
acc. bit-width (%)	98.32			73.05			86.03		
zero parameters(%)	88.59			33.78			37.14		
one parameters(%)	6.31			45.32			46.50		
pow2 parameters(%)	0.05			16.40			13.62		
other (%)	5.05			4.50			2.74		

In order to illustrate the impact of the developed tactics, Table 2 reports post-fitting results of a LeNet5 accelerator with a 5-bit precision on an embedded Intel Cyclone V 5CGXFC9E7 device, using 3 implementation strategies. In the first result, only DSP blocks are used to map all CNN multiplications. The resulting hardware requires 72× the available resources of the device. The second case features an implementation of multiplication based on logic elements and requires 3.8× the available logic. Using tailored multipliers reduces resources by a factor of 8.6×, fitting the CNN accelerator onto an Intel Cyclone V embedded FPGA.

Tables 3-a and 3-b respectively detail post-fitting results on two embedded FPGA platforms: the Intel Cyclone V 5CGXFC9E7 and the Xilinx Kintex7 XC7Z045FBG (using respectively Intel Quartus 16.1 and Xilinx Vivaldo 2016.4 synthetizers). To the best of our knowledge, these numbers are the first to demonstrate the applicability of a DHM-based approach for the implementation of CNNs on embedded FPGAs. The three hardware accelerators fit onto the embedded devices with no off-chip memory requirement. The reported memory footprint corresponds to line buffers used by dataflow-based convolution engines [10] and both synthesis tools instantiate LUT-based memory blocks to implement these buffers. As expected when using DHM, the logic utilization in the FPGA grows with the size of the CNN. In addition, the proportion of null kernels affects the amount of logic needed to map a CNN graph.

Finally, table 4 compares Haddoc2 performance to implementations on FPGA, GPU and ASIC. For the Cifar10 CNN, we find that a direct hardware mapping approach grants ×2.63 higher throughput on the same device

[2]Similarly to [12], a fine tuning of the CNN parameters has been performed after selecting the bit-width, which increases the classification accuracy of the quantized CNN.

when compared to fpgaConvNet, the state-of-the-art framework for mapping CNNs on FPGAs. For LeNet5, a $\times 1.28$ acceleration is reported which corresponds to a classification rate of 64.42 HD images/sec with a 3-scale pyramid. The GPU platform delivers the best performance in terms of computational throughput but the price is a high power consumption while ASIC technology gives the best throughput per Watt trade-off at the price of lower reconfigurability and higher production costs. For deeper CNN implementations, such as in [6], DHM is infeasible on current embedded FPGAs because the Logic Elements required to derive the accelerators exceed the available hardware resources.

However, and given the recent improvements of Binary Neural Networks (BNNs) –reported for instance in FINN [17]–, the implementation of deeper CNNs can be addressed by leveraging on BNNs. BNNs involve a rescheduling of the CNN graph as well as a retraining the network to perform operations using a single bit.

Table 2: Resource utilization by a DHM LeNet5 CNN with different implementations strategies for multipliers.

	DSP-based	LE-based	LE-based + const.
Logic Usage (ALM)	NA	433500 (381%)	50452 (44%)
DSP Block usage	24480 (7159 %)	0 (0%)	0 (0%)

Table 3: Resource Utilization of the three hardware accelerators: a- an Intel Cyclone V FPGA, b- a Xilinx Kintex 7 FPGA.

		LeNet5 [2]	Cifar10 [15]	SVHN [14]
a	Logic Elements (ALMs)	8067 (7%)	51276 (45%)	39513 (35%)
	DSP Blocks	0 (0 %)	0 (0%)	0 (0%)
	Block Memory Bits	176 (1%)	15808 (1%)	10878 (1%)
	Frequency	65.71 MHz	63.89 MHz	63.96 MHz
b	Slices	25031 (11%)	172219 (79%)	136675 (63%)
	DSP Blocks	0 (0%)	0 (0%)	0 (0%)
	LUTs as Memory	252 (1%)	1458 (2%)	1552 (1%)
	Frequency	59.37 MHz	54.17 MHz	54.49 MHz

Table 4: Comparison to state-of-the-art implementations

	Publication	Workload	Throughput	Platform
FPGA	Haddoc2	3.8 Mop	318.48 Gop/s[1]	Cyclone V
		24 Mop	515.78 Gop/s[1]	Cyclone V
		24.8 Mop	437.30 Gop/s[1]	Zynq XC706
	fpgaConvNet [8]	3.8 Mop	185.81 Gop/s[1]	Zynq XC706
		24.8 Mop	166.16 Gop/s[1]	Zynq XC706
	Qiu *et al.* [6]	30.76 Gop	187.80 Gop/s[1]	Zynq ZC706
	FINN [17]	112.5 Mop	2500 Gop/s[1]	Zynq ZC706
GPU	CudNN R3	1333 Mop	6343 Gop/s	Titan X
ASIC	Yoda NN [18]	24.8 Mop	525.4 Gop/s	UMC 65 nm
		23.4 Mop	454.4 Gop/s	UMC 65 nm
	NeuFlow [7]	350 Mop	1280 Gop/s	IBM 45nm SOI

6 Conclusion and Future work

This paper has investigated the feasibility of *direct hardware mapping (DHM)* for the implementation of FPGA-based CNN accelerators. We have demonstrated that current embedded FPGAs provide enough hardware resources to support this approach. To demonstrate DHM, the HADDOC2 tool has been introduced and used to automatically generate platform-independent CNN hardware accelerators from high level CNN descriptions. Tactics are presented for optimizing the area and resource utilization of arithmetic blocks. DHM opens new opportunities in terms of hardware implementations of CNNs and can be extended to ASIC technologies as well as Binary Neural Networks.

[1] Performance of the feature extractor

REFERENCES

[1] Kamel Abdelouahab, Maxime Pelcat, Jocelyn. Serot, Cedric Bourrasset, and Franois Berry. Tactics to Directly Map CNN graphs on Embedded FPGAs. *IEEE Embedded Systems Letters*, pages 1–4, 2017.

[2] Y LeCun, L Bottou, Y Bengio, and P Haffner. Gradient Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[3] Jocelyn Sérot, Franois Berry, and Cdric Bourrasset. High-level dataflow programming for real-time image processing on smart cameras. *Journal of Real-Time Image Processing*, 12(4):635–647, 12 2016.

[4] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional Architecture for Fast Feature Embedding. In *Proceedings of the ACM International Conference on Multimedia*, 2014.

[5] Tadayoshi Horita, Itsuo Takanami, Masakazu Akiba, Mina Terauchi, and Tsuneo Kanno. An FPGA-based multiple-weight-and-neuron-fault tolerant digital multilayer perceptron (Full version). *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9030:148–171, 2015.

[6] Jiantao Qiu, Jie Wang, Song Yao, Kaiyuan Guo, Boxun Li, Erjin Zhou, Jincheng Yu, Tianqi Tang, Ningyi Xu, Sen Song, Yu Wang, and Huazhong Yang. Going Deeper with Embedded FPGA Platform for Convolutional Neural Network. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays - FPGA '16*, pages 26–35, New York, NY, USA, 2016. ACM.

[7] C Farabet, B Martini, B Corda, P Akselrod, E Culurciello, and Y LeCun. NeuFlow: A runtime reconfigurable dataflow processor for vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops - CVPRW '11*, pages 109–116, 6 2011.

[8] Stylianos I. Venieris and Christos Savvas Bouganis. FpgaConvNet: A Framework for Mapping Convolutional Neural Networks on FPGAs. In *Proceedings of the 24th IEEE International Symposium on Field-Programmable Custom Computing Machines, FCCM '16*, pages 40–47, 2016.

[9] Jason Cong and Bingjun Xiao. Minimizing computation in convolutional neural networks. In *Proceedings of the 24th International Conference on Artificial Neural Networks - ICANN '14*, pages 281–290. Springer, 2014.

[10] Richard G Shoup. Parameterized convolution filtering in a field programmable gate array. In *Selected papers from the Oxford 1993 international workshop on field programmable logic and applications on More FPGAs. Oxford, United Kingdom: Abingdon EE&CS Books*, pages 274–280, 1994.

[11] Edward A. Lee and Thomas M. Parks. Dataflow Process Networks. *Proceedings of the IEEE*, 83(5):773–801, 1995.

[12] Philipp Gysel, Mohammad Motamedi, and Soheil Ghiasi. Hardware-oriented Approximation of Convolutional Neural Networks. In *arXiv preprint*, page 8, 2016.

[13] Altera. Implementing Multipliers in FPGA Devices. Technical report, Altera, 2004.

[14] Yuval Netzer and Tao Wang. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, page 5, 2011.

[15] Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images. Technical report, 2009.

[16] Suyog Gupta, Ankur Agrawal, Pritish Narayanan, Kailash Gopalakrishnan, and Pritish Narayanan. Deep Learning with Limited Numerical Precision. In *Proceedings of the 32nd International Conference on Machine Learning - ICML '15*, pages 1737–1746. JMLR Workshop and Conference Proceedings, 2015.

[17] Yaman Umuroglu, Nicholas J Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. FINN: A Framework for Fast, Scalable Binarized Neural Network Inference. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays - FPGA '17*, pages 65–74, 2017.

[18] Renzo Andri, Lukas Cavigelli, Davide Rossi, and Luca Benini. YodaNN: An ultra-low power convolutional neural network accelerator based on binary weights. *Proceedings of IEEE Computer Society Annual Symposium on VLSI - ISVLSI '16*, 2016-Septe:236–241, 2016.