Parallel Computing: Technology Trends
I. Foster et al. (Eds.)

doi:10.3233/APC200099

Accelerating Binarized Convolutional Neural Networks with Dynamic Partial Reconfiguration on Disaggregated FPGAs

Panagiotis SKRIMPONIS [a], Emmanouil PISSADAKIS [b], Nikolaos ALACHIOTIS [b,c] and Dionisios PNEVMATIKATOS [b,c]

[a] *NYU Tandon School of Engineering, Brooklyn, NY, USA*
[b] *Technical University of Crete, Chania, Greece*
[c] *FORTH-ICS, Greece*

Abstract.
Convolutional Neural Networks (CNNs) currently dominate the fields of artificial intelligence and machine learning due to their high accuracy. However, their computational and memory needs intensify with the complexity of the problems they are deployed to address, frequently requiring highly parallel and/or accelerated solutions. Recent advances in machine learning showcased the potential of CNNs with reduced precision, by relying on binarized weights and activations, thereby leading to Binarized Neural Networks (BNNs). Due to the embarassingly parallel and discrete arithmetic nature of the required operations, BNNs fit well to FPGA technology, thus allowing to considerably scale up problem complexity. However, the fixed amount of resources per chip introduces an upper bound on the dimensions of the problems that FPGA-accelerated BNNs can solve. To this end, we explore the potential of remote FPGAs operating in tandem within a disaggregated computing environment to accelerate BNN computations, and exploit dynamic partial reconfiguration (DPR) to boost aggregate system performance. We find that DPR alone boosts throughput performance of a fixed set of BNN accelerators deployed on a remote FPGA by up to 3x in comparison with a static design that deploys the same accelerator cores on a software-programmable FPGA locally. In addition, performance increases linearly with the number of remote devices when inter-FPGA communication is reduced. To exploit DPR on remote FPGAs and reduce communication, we adopt a versatile remote-accelerator deployment framework for disaggregated datacenters, thereby boosting BNN performance with negligible development effort.

Keywords. Binarized Neural Network, FPGA accelerator, Dynamic Partial Reconfiguration

1. Introduction

Considerable improvements in the development of high-performance systems for neural networks using multi-core technology have been proposed in recent years [1]. However, various challenges in power, cost, and performance scaling remain, due to the ever increasing model sizes (e.g., 50MB for GoogLeNet [2], 200MB for ResNet-101 [3], 250MB for AlexNet [4], or 500MB for VGG-Net [5]) that inevitably introduce pro-

hibitively high computational costs, steadily raising the need for accelerated solutions. The need for models with low memory and compute requirements is imperative.

Several works have been introduced to address the aforementioned challenges, and reduce the resource utilization requirements of CNNs, e.g., by exploiting the sparsity of the network connections [6], or by narrowing the data width [7, 8]. Another promising method is binarization, which relies on a considerably more compact data representation for the network weights and the neuron values than the one employed by regular CNNs. The underlying idea is to constrain each value to be either +1 or -1. Consequently, this reduces storage and memory bandwidth requirements and allows to replace floating-point operations with binary operations, thereby paving the way for efficient deep learning using FPGA technology.

Binarized Convolutional Neural Networks (BNNs) were first presented by Courbariaux et al. [9], who introduced a method to train BNNs with the permutation invariant MNIST, CIFAR-10, and SVHN [10] datasets, achieving state-of-art accuracy. Rastegari et al. [11] successfully trained a BNN with ImageNet models, reportedly improving accuracy, boosting performance, and reducing the model size, when compared with a full-precision AlexNet [4] implementation. Existing implementations of CNNs on FPGAs face several challenges due to their prohibitively high requirements for storage, memory bandwidth, and compute capacity. This problem exacerbates with more complex state-of-art models, such as the VGG model [7] that has 16 layers and 138×10^6 weights.

In this work, we investigate the potential of Dynamic Partial Reconfiguration (DPR) on modern FPGA-based multiprocessor system-on-chip (MPSoC) devices when deployed within a disaggregated-computing environment to boost BNN performance. Resource disaggregation addresses the problem of fixed resource proportionality in datacenters by creating and managing pools of different resource types, e.g., compute, memory, and accelerators. The immense parallel nature of BNNs suggests the eminent need for a disaggregated accelerator solution.

Devising a FPGA-based MPSoC disaggregated accelerator solution that exploits DPR beneficially to the performance of BNNs introduces additional challenges: 1. DPR brings flexibility in accelerator deployment, yet the high DPR overhead may diminish the expected performance gains. Thus, a beneficial computation-to-PR ratio is needed in order to justify the DPR overhead and improve performance, 2. The limited on-board memory resources set an upper bound on the maximum size of images that can be processed on a single node, and 3. The evident need for low-latency communication and synchronization in accelerator-rich environments becomes significantly more imperative in disaggregated computing platforms, where communication between remote nodes interconnected over a network is required [12, 13]. To address these challenges, we make the following contributions:

- We map BNN computations to ReFiRe [22], a remote-accelerator deployment framework for disaggregated computing. This allows to transparently exploit inter-FPGA parallelism and overcome the physical resource boundary per device for BNN computations by relying on the framework to stir computation to multiple disaggregated FPGA-based accelerator nodes. We find that throughput improves linearly with the number of accelerator devices, without requiring additional effort for communication or synchronization, neither on the host nor on the accelerator sides.

- We boost overall BNN throughput performance of a fixed set of three accelerator cores [14] per remote FPGA by transparently exploiting DPR and intra-layer parallelism through dedicated features of the employed accelerator deployment framework. We find that throughput improves up to 3x using DPR on a remote device than deploying the same set of accelerators locally through a software-programmable design flow [15]. Importantly, the proposed approach is highly generic and versatile, thus allowing to boost performance of existing CNN and/or BNN accelerators using DPR and parallelism, with negligible development effort.

2. Background

2.1. Convolutional Neural Networks

A typical CNN classifier consists of a parameterized pipelined multi-layer architecture. Layers require configuration of their parameters, often called weights, which must be determined by training the CNN offline on pre-classified data. Once the parameters are determined, the CNN can be deployed for the classification of new data points. The first layer takes as input a multi-channel input image and outputs a set of feature maps (fmaps). Each of the following layers read the fmaps, performs some computation on them, and produces a new set of fmaps to be fed into the next layer. Finally, a classifier produces the probability of that image belonging to each output class. The layer types are the following:

Convolutional layers realize a filter-like process, convolving the input fmaps with a $K \times K$ weight kernel. The results are summed, added with a bias, and passed through a non-linearity function to produce a single output fmap. This process is given in Eq. 1:

$$y_n = f(\sum_{m=1}^{M} x_m * w_{n,m} + b_n). \tag{1}$$

Pooling layers map each input fmap to an output fmap where every pixel is the max/mean of a $K \times K$ window of input pixels. They are inserted through a CNN to reduce the size of the intermediate fmaps.

Fully-Connected layers apply a linear transformation on the input 1-D vectors with a weight matrix. A bias is applied on the result, which is then passed through a non-linearity function to produce a single 1×1 output. This process is given in Eq. 2:

$$y_n = f(\sum_{m=1}^{M} x_m * w_{n,m} + b_n). \tag{2}$$

2.2. Binarized Convolutional Neural Networks

A BNN is essentially an extremely quantized, reduced-precision CNN model where weights and fmap pixels are binarized using the sign function. Positive weights are mapped to +1 and negative weights to -1, using a compact single-bit representation. Therefore, BNNs require significantly less storage than standard CNNs. The binarization of the neural networks can either be partial or full. In order to be considered full, it has to encompass the following aspects: binary input activations, binary synapse weights,

and binary output activations. Due to the quantization effect, there is no need for biasing since it does not compromise accuracy. However, in order to improve accuracy and scale down the error, a new layer type has to be introduced:

The **Batch normalization** [16] layer reduces the quantization error of the binarization by linearly shifting and scaling the input distribution to have zero mean and unit variance. The transformation is given in Eq. 3:

$$y = \frac{x - \mu}{\sqrt{\sigma^2 + \varepsilon}} \gamma + \beta. \tag{3}$$

2.3. CIFAR-10 BNN Model

The CIFAR-10 dataset [17] contains sixty thousand 32×32 3-channel images consisting of photos taken of real world vehicles and animals. For the experiments, out of the 60,000 images, 50,000 images were chosen for training and 10,000 images for testing. Training of the CIFAR-10 BNN model was done using open-source Python code provided by Courbariaux et al. [9], which uses the Theano and Lasagne deep learning frameworks.

3. Related work

Multiple studies have explored the potential of FPGAs for implementing Artificial Neural Networks (ANNs). Zhang et al. [18] proposed an analytical CNN design scheme that is based on the roofline model [19] to explore various optimizations, such as loop tiling and transformation, to reduce resource underutilization and match the computation throughput to the available memory bandwidth on FPGAs. The study reports 61.62 GFLOPS peak performance at 100 MHz on a VC707 FPGA board.

Qiu et al. [7] presented a dynamic-precision data quantization method, as well as a convolver design for embedded FPGAs that performs well for all CNN layer types. The authors observed accuracy loss due to data quantization of as low as 0.4%, while the average performance of the convolutional layers and the full CNN is 187.8 GOP/s and 137.0 GOP/s at 150 MHz, respectively, when mapped to a Xilinx Zynq ZC706 board.

In 2015, Courbariaux et al. [9] introduced the idea of constraining weights to only two possible values, e.g., -1 and 1, in order to improve hardware performance of CNNs, since multiply-accumulate operations can be replaced by simple accumulations. Since then, multiple studies have presented FPGA-based accelerator architectures for BNNs. Umuroglu et al. [20] presented FINN, a framework to design efficient FPGA accelerators for BNNs by tailoring per-layer compute resources to user-provided throughput requirements. Employing a ZC706 board, the authors report up to 21,906 image classifications per second on the CIFAR-10 and SVHN datasets. Liang et al. [21] presented a BNN FPGA architecture that relies on bit-level XNOR and shifting operations, as well as data quantization and on-chip storage to achieve high performance. The authors report up to 9396.41 GOP/s for the CIFAR-10 dataset at 150MHz on a Stratix-V platform.

Zhao et al. [14] presented a novel design of a BNN accelerator for FPGAs, which is synthesized from a high-level language (C++) to Verilog using the Xilinx SDSoC [15] design flow. The overall accelerator, which operates at 143MHz and achieves 200 GOP/s for the CIFAR-10 dataset, consists of three computational cores, namely FP-Conv (first convolutional layer), Bin-Conv (binary convolutional layers), and Bin-FC (binary fully

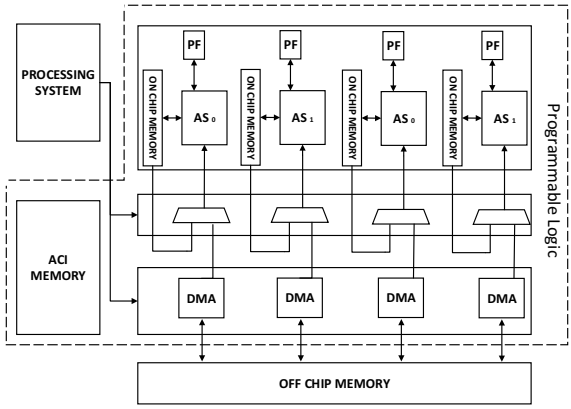

Figure 1. ReFiRe-based BNN accelerator architecture.

connected layers). Our work builds upon the work by Zhao et al. [14] and demonstrates how the proposed BNN accelerator can be mapped to the ReFiRe [22] remote-accelerator deployment framework, which allows to boost BNN performance without the need to redesign the aforementioned computational cores. ReFiRe [22] reduces communication and synchronization requirements between remote accelerator nodes (FPGA-based MPSoCs) in disaggregated datacenters by shifting control flow and partial reconfiguration decisions to the remote side through arbitrarily long instructions that encapsulate complex sequences of operations and their respective synchronization requirements. The framework abstracts away all the complexity of performing DPR on remote/disaggregated FPGAs, and allows to transparently exploit intra-FPGA parallelism per BNN layer, as well as inter-FPGA parallelism at image granularity. Note that, although DPR has been previously explored to boost CNN performance [23], this is the first work, to the best of the author's knowledge, that explores dynamic partial reconfiguration on disaggregated FPGAs to improve BNN performance.

4. Disaggregated Acceleration Framework

The ReFiRe [22] framework allows to efficiently deploy remote/disaggregated accelerators by improving the computation-to-communication ratio between a host processor and an arbitrary number of accelerator devices. This is achieved by relying on complex instructions of variable length, henceforth referred to as Advanced Coprocessor Instructions (ACIs), which describe partial reconfiguration events and the required flow of data among a set of remote partially reconfigurable accelerator cores.

4.1. Hardware architecture

The hardware architecture of the remote accelerator is illustrated in Figure 1. There are four accelerator slots (AS), with each AS being a partially reconfigurable region (PRR). Each AS has a private Parameter file (PF) to facilitate accelerator configuration. Furthermore, four Direct Memory Access (DMA) engines are responsible for transferring data between external memory and each AS. Depending on the BNN layer processed at each point in time, input data to each AS can arrive either from on-chip storage (output data

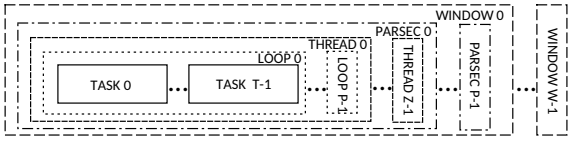

Figure 2. Hierarchy of ACI instruction classes (source: [22]).

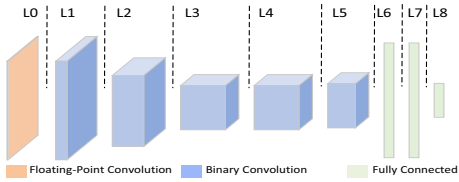

Figure 3. Binarized Neural Network architecture.

of a previous layer) or from external memory (input data of the very first layer). The ACI memory holds the active ACI at each juncture, and directs computation and PR events on each AS.

4.2. ACI architecture

The ACI memory consists of three main parts, namely SYNC, COMPUTE, and PARAM. The SYNC area facilitates host-accelerator synchronization. The PARAM area facilitates the parameter-based configuration of each accelerator per AS. The COMPUTE area holds a sequence of instructions that correspond to an application-specific execution scenario. There are five ACI instruction types, organized into a hierarchy of classes: WINDOW, PARSEC, THREAD, LOOP and TASK, as illustrated in Figure 2.

The TASK class contains information related to the input and output data per AS for a given operation. The multiplexer and the PF per AS are configured and initialized, respectively, based on data extracted from each TASK class. The LOOP class is a container class for TASK objects, which allows to reduce host-accelerator synchronization requirements by providing the required number of iterations per accelerator operation in an AS, as well as the desired stride for both the input and the output data. The THREAD class dictates an order of operations that are performed sequentially, whereas the PARSEC class indicates a parallel section with a number of THREAD classes that execute in parallel on different AS. Finally, the WINDOW class dictates PR requirements, as each WINDOW starts with one or more requests for partial reconfiguration.

4.3. Mapping the BNN to an ACI

The architecture of the BNN consists of nine layers, with the first six being convolutional layers while the next three are fully connected layers, as illustrated in Figure 3. The first layer (L0 in Fig. 3) receives fixed-point input data and binary weights, whereas the rest of the layers (L1 through L8 in Fig. 3) operate only on binary data. The convolutional layers rely on 3×3 filtering and edge padding, while the fully connected layers apply batch normalization prior to pooling, and binarization before writing data out to the buffers. The accelerator system presented by Zhao et al. [14] designed three accelerators, which

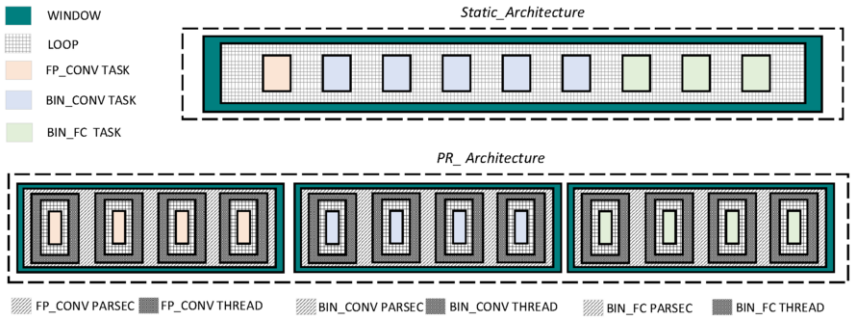

Figure 4. Illustration of the ACI format for the *Static_Architecture* and the *PR_Architecture* for FPGA-based BNN acceleration.

we employ as-is in our disaggregated accelerator systems. The *FP_CONV* core implements the L0 layer of the BNN. The *BIN_CONV* core is employed for the following five binary-only convolution layers (L1 through L5). Finally, the *BIN_FC* core accelerates the last three BNN layers (L6 through L8).

To map the required BNN computations to an ACI, we place each accelerator call in a dedicated TASK class, which also contains the respective core's configuration parameters and input/output address and sizes. The number of images that are processed in-between PR events is defined as the number of iterations of a LOOP class, with the stride being the image size. The THREAD and PARSEC classes allow to expose parallelism per layer by partitioning processing over multiple AS that host the same accelerator core. Finally, the WINDOW class performs one PR event per AS to deploy a different accelerator core to serve the needs of the next BNN layer. Due to the fact that there are three accelerator cores, the final ACI that implements the BNN consists of three WINDOW classes, one per accelerator core. Figure 4 illustrates alternative execution scenarios based on different ACI structures for the BNN. The *Static_Architecture* is identical to the reference execution scenario that is implemented on a software-programmable FPGA by Zhao et al. [14]. Due to the fact that ReFiRe is a native partially reconfigurable architecture, the *Static_Architecture* involves the initial deployment of the three accelerators in three AS. This is achieved by placing all nine TASK classes (one per BNN layer) in the same WINDOW class. The *PR_Architecture* exploits PR at run time and exposes intra-layer parallelism through the PARSEC/THREAD classes. Therefore, three WINDOW classes are required, one per accelerator core, and multiple ACI-based iterations are performed.

5. Implementation and Evaluation

To evaluate performance on disaggregated FPGAs, we employ two ZCU102 boards, each containing a Zynq UltraScale+ MPSoC, interconnected over a Small Form-factor Pluggable (SFP) 10-Gbps link. Each MPSoC hosts an ARM Cortex-A53 64-bit quad-core processor running at 1.2 GHz. One board serves as the host processor, whereas the second is the accelerator platform. The host board runs Ubuntu 16.04 on its Application Processing Unit (APU), while all communication is established through the SFP link. All

three accelerator cores deployed in ReFiRe are retrieved from *https://github.com/cornell-zhang/bnn-fpga*. Table 1 provides resource utilization per accelerator on the ZCU102 evaluation platform, hosting a Zynq Ultrascale+ MPSoC. We evaluate two alternative execution scenarios, the *Static_Architecture* and *PR_Architecture* (illustrated in Fig. 4) using the CIFAR-10 dataset.

Table 1. Resource utilization for the three BNN accelerator cores on the Zynq Ultrascale+ MPSoC

ACCEL.	LUTs	FFs	BRAMs	DSPs	Power (W)
FP_CONV	11609	13802	16	0	0.112
BIN_CONV	13208	5849	86	2	0.050
BIN_FC	4432	6148	20	2	0.086

5.1. Static_Architecture

We initially reproduce, using ReFiRe, the same static execution scenario that was evaluated by Zhao et al. [14] using SDSoC. Thus, we first deploy the accelerator cores *FP_CONV*, *BIN_CONV*, and *BIN_FC* through an initial configuration WINDOW. In this scenario, all 10,000 images we used for evaluation are processed sequentially, directing the layers output to the next, as dictated by the BNN architecture. This approach required 128 sec to complete. As a reference, we note that the SDSoC-based approach [14] using the exact same accelerators and number of images required 103.1 sec. The observed delay is due to data exchanges between remote FPGAs for ACI transfers and synchronization.

5.2. PR_Architecture

Next, we evaluate the DPR-based execution scenario by populating all AS with the same accelerator core and rely on the PARSEC and THREAD classes to invoke them in parallel per layer. The DPR overhead per AS (using ICAP [26]) is 7 ms (2.5 MB bitstream sizes). Note that, Zhao et al. [14] report 5.7 ms per image without using DPR. Thus, to yield a beneficial computation-to-PR ratio to exploit DPR using ReFiRe, we organize processing in batches. Figure 5 illustrates how performance improves with the batch size. As can be observed in the figure, DPR allows to outperform the fully static architecture when the batch size exceeds 25 images/batch. Evidently, processing a single image in-between DPR events yields the worst-case performance, requiring 917 seconds in total for the 10,000 images, when the static design with 1 instance per accelerator finishes in 128 seconds. When the batch size exceeds 300 images, DPR allows up to 3.1x faster execution, due to the four accelerator instances per layer. Note that, aggregate system performance increases almost linearly with the number of disaggregated FPGAs used, due to the beneficial computation-to-synchronization ratio that the ACI offers. The overhead to create and transfer an ACI to a remote FPGA is as low as 1.33 sec.

5.3. Comparison with other works

A comparison with previous FPGA accelerator designs for CNN and BNN models is provided in Table 2. Suda et al.[24] and Qui et al.[7] reported 117 GOPS/s and 136 GOPS/s, respectively, significantly lower than the performance attained through ReFiRe. Li et al.[25] achieved 594 GOPS/s, with 22.5 GOP/s/W efficiency, due to the increased power consumption of the design. Our work outperforms the reference approach pro-

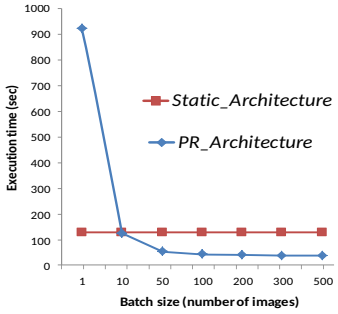

Figure 5. Execution time to process 10,000 images using the *Static_Architecture* and the *PR_Archtitecture* when the batch size (number of images in-between PR events) grows up to 500.

posed by Zhao et al. [14], achieving about 3.1 times higher performance and efficiency for the exact same set of accelerators. Umuroglou et al. [20] and Liang et al. [21] report considerably higher performance than all other approaches. Therefore, we intend to employ ReFiRe to further improve the performance of these accelerators by transparently introducing DPR and deploying disaggregated FPGAs.

Table 2. Performance comparison with other FPGA-based CNN/BNN accelerators. The presented accelerator system employs the same set of accelerator cores as Zhao et al. [14].

	Zhao et al.[14]	**This work**	Suda et al.[24]	Qiu et al.[7]	Li et al.[25]	Umuroglu et al.[20]	Liang et al.[21]
Platform	Zynq	**ZynqMP**	Stratix-V	Zynq	Virtex-7	Zynq	Stratix-V
	XC7Z020	**XCZU9EG**	5SGSD8	XC7Z045	VX690T	XC7Z045	5SGSD8
Clock(MHz)	143	**150**	120	150	156	200	150
Precision(bit)	Input: 8	**Input: 8**	8-16	16	16	Input: 8	Input: 8
	Weight: 1	**Weight: 1**				Weight: 1	Weight: 1
Model size (OPs)	1.24 G	**1.24 G**	30.9 G	30.76 G	1.45 G	112.5 M	1.23 G
Performance (GOP/s)	207.8	**667**	117	136	565.94	2465.5	9396.41
Power(W)	4.7	**5.97**	25.8	9.63	30.2	11.7	26.2
Efficiency (GOP/s/W)	44.2	**111.73**	4.57	14.22	22.15	210.72	358.64

6. Conclusions

Binarized Convolutional Neural Networks (BNNs) offer significant accuracy, performance and model compression over standard full-precision Convolutional Neural Networks (CNNs). This paper proposed a hardware accelerator architecture for BNNs on modern FPGA-based MPSoC devices deployed within a disaggregated-computing environment. We explored the trade-offs in exploiting Dynamic Partial Reconfiguration (DPR) to meet the performance, communication, and latency requirements. We find that throughput performance improves linearly with the number of accelerator devices, without requiring additional effort for communication or synchronization, neither on the host nor on the accelerator sides. We explored, a generic acceleration framework that improves performance of remote, fine-grained accelerators by encapsulating complex sequences of operations in arbitrarily long instructions called ACIs. We compared these accelerator instances against other FPGA-based BNN implementations in the literature. Our evaluation results show that disaggregation offers an attractive solution, which allows to expose near-peak accelerator performance at the application level, despite performing computations on remote nodes. Future work will focus on architectural improvements, exploring a low-precision network for a much larger and more complicated dataset like ImageNet and AlexNet.

References

[1] E. Nurvitadhi, D. Sheffield, , A. Mishra, G. Venkatesh, and D. Marr, "Accelerating Binarized Neural Networks: Comparison of FPGA, CPU, GPU, and ASIC," in *ICFPT*, 2016, pp. 77–84.

[2] C. Szegedy *et al.*, "Going Deeper with Convolutions," *CoRR*, vol. abs/1409.4842, 2014.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, 2016, pp. 770–778.

[4] A. Krizhevsky *et al.*, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2012, pp. 1097–1105.

[5] K. Simonyan *et al.*, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *CoRR*, vol. abs/1409.1556, 2014.

[6] X. Xie *et al.*, "Exploiting Sparsity to Accelerate Fully Connected Layers of CNN-Based Applications on Mobile SoCs," *ACM Trans. Embedded Comput. Syst.*, vol. 17, no. 2, pp. 37:1–37:25, 2018.

[7] J. Qiu *et al.*, "Going Deeper with Embedded FPGA Platform for Convolutional Neural Network," *FPGA*, 2016.

[8] F. N. Iandola *et al.*, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <1MB model size," *CoRR*, vol. abs/1602.07360, 2016.

[9] M. Courbariaux, Y. Bengio, and J. David, "BinaryConnect: Training Deep Neural Networks with binary weights during propagations," *CoRR*, vol. abs/1511.00363, 2015.

[10] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading Digits in Natural Images with Unsupervised Feature Learning," in *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

[11] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks," *CoRR*, vol. abs/1603.05279, 2016.

[12] K. Katrinis *et al.*, "Rack-scale disaggregated cloud data centers: The dReDBox project vision," in *DATE 2016*. IEEE, 2016, pp. 690–695.

[13] A. M. Caulfield *et al.*, "A cloud-scale acceleration architecture," in *MICRO 2016*. IEEE, 2016, pp. 1–13.

[14] R. Zhao, W. Song, W. Zhang, T. Xing, J.-H. Lin, M. Srivastava, R. Gupta, and Z. Zhang, "Accelerating Binarized Convolutional Neural Networks with Software-Programmable FPGAs," *FPGA*, 2017.

[15] V. Kathail *et al.*, "SDSoC: A Higher-level Programming Environment for Zynq SoC and Ultrascale+ MPSoC," *FPGA*, 2016.

[16] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015.

[17] A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images," 2009.

[18] C. Zhang *et al.*, "Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks," *FPGA*, 2015.

[19] S. Williams, A. Waterman, and D. Patterson, "Roofline: An Insightful Visual Performance Model for Multicore Architectures," *Commun. ACM*, vol. 52, no. 4, pp. 65–76, 2009.

[20] Y. Umuroglu *et al.*, "FINN: A Framework for Fast, Scalable Binarized Neural Network Inference," *FPGA*, 2017.

[21] S. Liang *et al.*, "FP-BNN: Binarized neural network on FPGA," *Neurocomputing*, 2017.

[22] E. Pissadakis, N. Alachiotis, P. Skrimponis, D. Theodoropoulos, T. Korakis, and D. Pnevmatikatos, "ReFiRe: efficient deployment of Remote Fine-grained Reconfigurable accelerators," *ICFPT*, 2018.

[23] F. Kstner *et al.*, "Hardware/Software Codesign for Convolutional Neural Networks exploiting Dynamic Partial Reconfiguration on PYNQ," *IEEE International Parallel and Distributed Processing Symposium Workshops*, 2018.

[24] N. Suda *et al.*, "Throughput-Optimized OpenCL-based FPGA Accelerator for Large-Scale Convolutional Neural Networks," *FPGA*, pp. 16–25, 2016.

[25] H. Li *et al.*, "A High Performance FPGA-based Accelerator for Large-Scale Convolutional Neural Networks," *International Conference on Field Programmable Logic and Applications (FPL)*, pp. 1–9, 2016.

[26] Xilinx, "UltraScale Architecture Configuration," https://www.xilinx.com/support/documentation/user_guides/ug570-ultrascale-configuration.pdf, [Online; accessed 05-Jul-2018].