UCLouvain

École polytechnique de Louvain

Accelerating Convolutional Neural Network for FPGA

using Depthwise Separable Convolution and Pruning

Author: **Guillaume Gheysen**
Supervisors: **Jean-Didier Legat, Christophe de Vleeschouwer**
Readers: **Victor Joos de Ter Beerst, Martin Lefèvre**

Academic year 2019–2020
Master [120] in Computer Science and Engineering

Abstract

FPGA are flexible, GPU consummes to much energy

Acknowledgment

I would like to thank my two supervisors Jean-Didier Legat and Christophe de Vleeschouwer for their support.

Contents

List of Figures

List of Abrevation

AI Artificial Intelligence.

ASIC Application-Specific Integrated Circuit.

CNN Convolutional Neural Network.

CPU Central Processing Unit.

DL Deep Learning.

DRAM Dynamic Random Access Memory.

DSC Depthwise Separable Convolution.

DSP Digital Signal Processing.

FCN Fully Connected Layer.

FFT Fast Fourier Transform.

FIFO First-in First-out.

FM Feature Map.

FPGA Field-Programmable Gate Array.

GEMM General Matrix Multiplication.

GPU Graphical Processing Unit.

ML Machine Learning.

MoC Model of Computation.

NAS Neural Architectural Search.

NN Neural Networks.

PE Processing Element.

SIMD Single Instruction Multiple Data.

Chapter 1

Introduction

Artificial Intelligence (AI) is one of the newest fields in science and engineering which started to be investigated after World War II [1]. AI is an 'umbrella term' that includes multiples approaches and that encompasses a huge variety of subfields such as computer vision, natural language processing, voice recognition, etc. Machine Learning (ML), one of AI subfield founded in the 1950s, has revolutionized technology realms in the last decades. The concept of ML relates to the question of how to construct computer programs that automatically improve with experience (that can learn) [2]. Statistical machine learning has a major issue in the selection of an appropriate feature space where input instances have desired properties for solving a particular problem [3]. A relevant solution would be automatic learning of the features using Neural Networks (NN). From this idea spawned Deep Learning (DL) architectures composed of layers of non-linearity processings [4]. From the last decade, DL is gaining in interest and the domains of applications have grown rapidly [5]. The accuracy of those models has also increased. For example, in 2015, the image classification model on ImageNet dataset achieved 94.6% on average [6].

A particular type of DL model, Convolutional Neural Network (CNN), has demonstrated its effectiveness in the image detection and recognition applications [7]. However, the performance comes at the price for a large computational cost (billion of operations and millions of parameters). To train and evaluate a large scale CNN, general Central Processing Unit (CPU) would not provide enough computational power. To improve the throughput and latency of the CNN, dedicated hardware such as Graphical Processing Unit (GPU), Application-Specific Integrated Circuit (ASIC) and Field-Programmable Gate Array (FPGA) can be used. CPU and GPU clusters are the dominant platforms to perform CNN inferences because they offer the best performance in terms of computational throughput but those are power-hungry [8]. FPGA and ASIC seem then to be a promising solution because they are more energy-efficient. As FPGA has reconfigurability and can be developed faster than ASIC, it is more suitable for the development of a CNN because of the CNN streaming workloads.

CNN acceleration is moving towards FPGA for two reasons: recent technology put FPGA performance within striking distance to GPUs and recent trends in CNN increase the sparsity of CNNs and use extreme compact data types (FPGA are designed to handle irregular parallelism and custom data types) [9]. However, FPGA is resource-constraint and the challenge is to find a mapping between the computational model and the execution model.

CNN consists of two phases: a training stage where the model learns (back-propagation algorithm) and the inference stage where the model predicts on new sample data (feed-forward algorithm). Usually, CNNs are trained once using GPU or FPGA and inference is executed every time the CNN has a new input to process. Most efforts have then been focused on accelerating the inference stage. The core of this work is therefore aimed at proposing a way to accelerate the inference of a model through weights pruning. It is not efficiently usable on a GPU because of the irregular data access and the custom data type of the non-pruned weights.

However, as pruning has already been studied by various works for standard convolution, this work concentrates on building an architecture using pruning on depthwise separable convolution, an alternative way to perform convolution which uses fewer parameters and has a lesser computational complexity.

Structure of the thesis

This master thesis is divided into 8 chapters.

Chapter 2 details the theory behind CNN and goes deeper into its computational background. It explores also model compression optimizations allowing a reduction of the computational complexity of the model and the size of the parameters.

The concept of FPGA is explained in chapter 3. We detail there the workflow and FPGA designs. Solutions to the issue of mapping a CNN model on FPGA are described too.

Chapter 4 and chapter 5 discuss the state-of-art of the inference optimization techniques on 4 and the compression methods.

Chapter 6 is focused on how to build an efficient architecture on FPGA and uses this knowledge to design our architecture using pruning and depthwise separable convolution.

Chapter 7 explains the simulation on SystemVerilog of the architecture developed in chapter 6. Results will also be discussed on possible improvements.

Finally, a conclusion on results obtained and discussion on future works is done in chapter 8.

Chapter 2

Convolutional Neural Network

A Convolutional Neural Network (CNN) is a type of Neural Networks that is specialized in analyzing visual imagery. It has shown explemplary performance on serveral competitions related to Computer Vision and Image Processing [10].

The first section introduces the building blocks of a CNN. We start from the first and most simple element and then show the improvements.

The second section is concentrated on the training of a CNN.

The third section details various models and techniques to perform model compression.

2.1 Layers

A layer is a high-level build block in DL. It is a set of operations, weights and non-linear functions. It transforms an input into an output which will be used as input for another layer. The collections of layer is then used to build a Convolutional Neural Network. We begin our understanding of the layer theory by explaining the perceptron, the simplest element of a neural Network.

2.1.1 Perceptron

The perceptron was first introduced in 1958 by F. Rosenblatt [11]. It is a computational model based on on the brain. A biological neuron receives information through chemical mechanisms and once a threshold is passed, the cumulate charges are released (we say the neuro fires) and the information is transmitted to other neurons.

Mathematically, a perceptron has n inputs (x_1, ..., x_n) which can be expressed as vector $\bar{x}$, n wheights ($\bar{w}$) and a bias **b**. The perceptron, when receiving an input vector, performs a weighted sum. If the wheighted sum is above a threshold (controlled by the bias which can lower or raise it), the perceptron is 'activated'

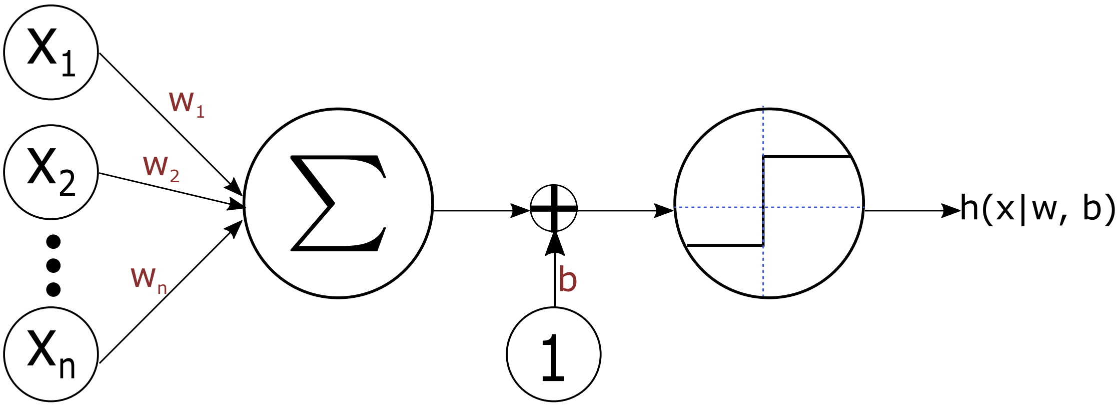

Figure 2.1: The Perceptron

and outputs a non-zero value (1). The figure 2.1 illustrates this. We can write the operation in a vector form:

$$h(x|w,b) = h(\sum_{i=1}^{n} x_i \cdot w_i + b) = h(\bar{w}^T \bar{x} + b)$$

where the function h is the activation function of the perceptron which was desribe as a boolean level function. The section 2.1.6 describes other activation function.

In the following section 2.1.2, we present the layer of perceptron (fully connected layer).

2.1.2 Fully Connected

The perceptron of section 2.1.1 can be considered as a linear classifier for which the decision boundary is the hyperplane

$$b + w_1 \cdot x_1 + ... + w_n \cdot x_n = 0$$

It is limited because it has only a linear boundary. To have a non-linear model, we can have multiple perceptron giving multiple outputs. This layer of perceptron is called a Fully Connected Layer (FCN) and is illustrated at figure 2.2.

2.1.3 2D Convolution

blabla

2.1.4 Depthwise Separable Convolution

blabla

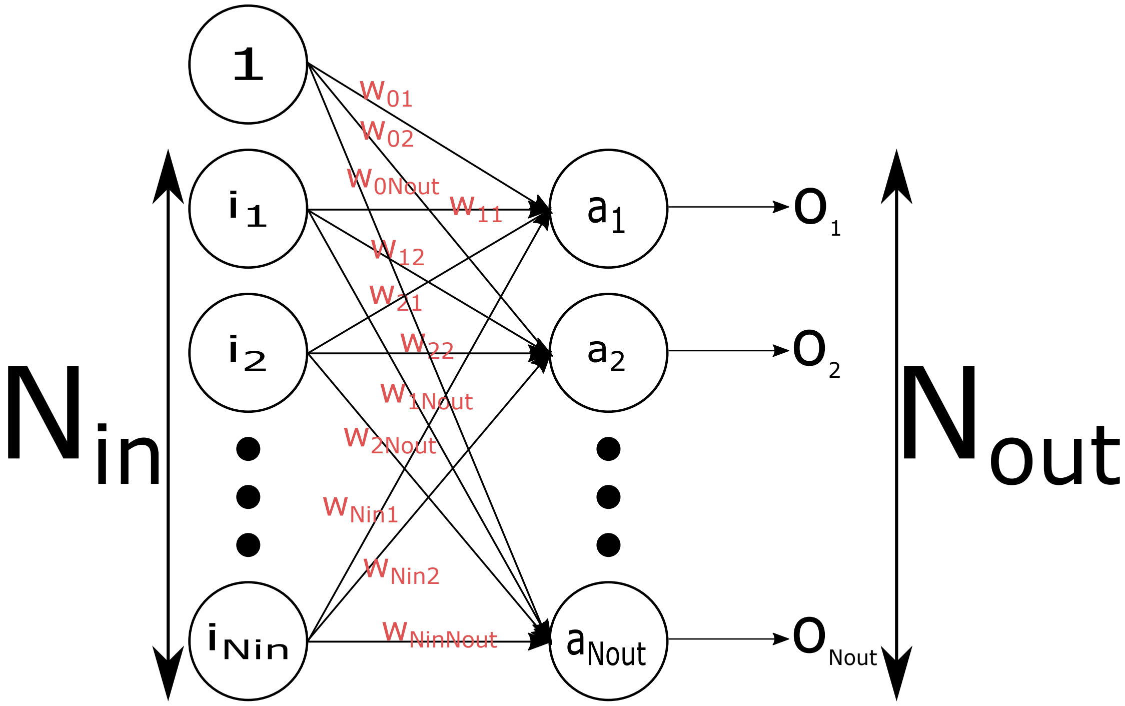

Figure 2.2: A fully connected layer

2.1.5 Bottleneck Convolution

blabla

2.1.6 Activation

blabla

2.1.7 Pooling

blabla

2.2 Training

blabla

2.3 Models

blabla

2.3.1 Pruning

blabla

2.3.2 Quantification

blabla

Chapter 3

FPGA

blabla

Chapter 4

Accelerating CNN inference on FPGA

As said at chapter 1, interest has been made on accelerating the inference phase on FPGA. This chapter reviews the acceleration approaches to perform an efficient inference.

The three main optimization can be categorized according to [9]:

- **Algorithmic Optimizations**: the computational cost of the convolution can be reduced by vectorizing them. More details can be found in section 4.1.
- **Datapath Optimization**: because of the limited resources on a FPGA, memory if often the bottleneck and optimizing the memory management can increase the throughput. More details can be found in section 4.2.
- **CNN model Optimization**: an important issue of CNN is their computational complexity and their hardware utilization. A solution is then to use approximate computing and trade accury for acceleration. This work focus on this kind of optimization. More details can be found in section 4.3.

4.1 Algorithmic Optimizations

4.1.1 General Matrix Multiplication (GEMM)

It is a common way to process CNN on CPU and GPU. We convert the convolution as a matrix-vector multiplication. The process of a convolution layer can be observed on Figure 4.1. However, this approach is not suggested for FPGA: [12, 13] point out that the FMs have to be copied multiple times when flattened to a vector. It leads to a huge memory footprint and either ineffiency in storage or complex memory management access patterns.

4.1.2 Winograd Transform

The Winograd minimal filter algorithm is first introduced by [14]. In Winograd filtering, data is processed by blocs referred as *tiles*, as following:

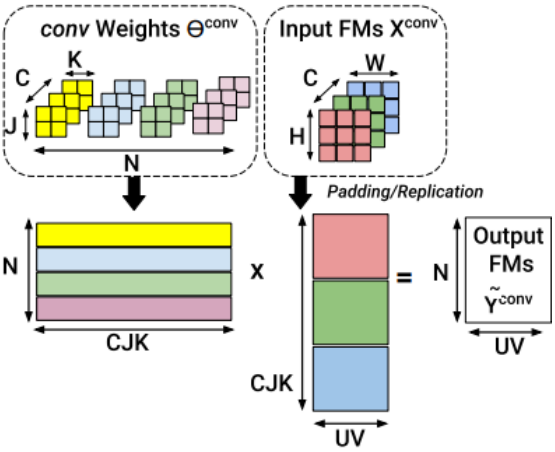

Figure 4.1: GEMM base processing on conv layer

- An input FM tile g of size $(N_{ix} \times N_{iy})$ is pre-processed: $\tilde{g} = \boldsymbol{G^T} g \boldsymbol{G}$, where $\boldsymbol{G}$ is a transformation matrix defined in the Winograd Algorithm [14].
- In a same way, d, the filter of size $(K_x \times K_y)$ is transformed into $\tilde{d}$: $\tilde{d} = \boldsymbol{B^T} d \boldsymbol{B}$, where $\boldsymbol{B}$ is a transformation matrix defined in the Winograd Algorithm [14].
- The output tile Y of the Winograd Filtering algorithm, denoted $F(N_{ix} \times N_{iy}, K_x \times K_y)$ is computed using equation 4.1, where $\boldsymbol{A}$ is a transformation matrix defined in the Winograd Algorithm [14] and $\odot$ indicates element-wise multiplication.

$$Y = \boldsymbol{A^T} [\ \tilde{g} \odot \tilde{d}\ ] \boldsymbol{A} \tag{4.1}$$

[15] demonstrated that Winograd convolution is efficient when the kernel is small $(K_* \leq 3)$ and the number of multiplication can be reduced by a factor of 2.25× (in returns the number of addition is increased). According to [16], 3×3 kernel is a standard for modern networks, which leads that Winograd Transform can be applied in modern network. However, we can only apply this computational transform to convolutions when the stride is equal to 1.

4.1.3 Fast Fourier Transform (FFT)

The FFT is an algorithm to transform the 2D convolution into element-wise multiplication in the frequency domain. The equation is observed at equation 4.2

$$conv2D(FM_I[ic], K[oc, ic]) = IFFT(FFT(FM_I[ic]) \odot FFT(K[oc, ic])) \tag{4.2}$$

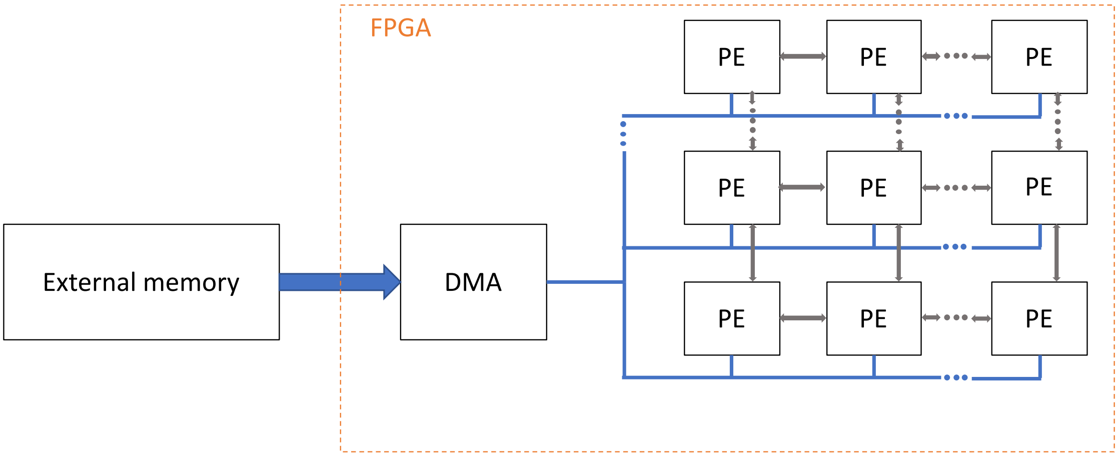

Figure 4.2: Static Systolic Arrays

The arithmetic complexity of the 2D convolution can be reduced to $O(N_{ix}^2 log_2(N_{ix}))$ [17] and the computational complexity of the FFT can be reduced to $O(N_{ix} log_2(K_x))$ using the Overlap-and-Add Method [18]. However, the FFT finds its interest when kernel are large [15] ($K_* \geq 5$), which is not a standard kernel size according to [16].

4.2 Datapath Optimization

4.2.1 Systolic Arrays

Systolic arrays were implemented on the first FPGA-based accelerators. A static systolic array is a static array of PE doing a part of the computation and communicating with its neighbors. An illustration of its principle can be found on figure 4.2.1. The configuration can only support convolution with a kernel size K_* lower than a bound, such that $K_* \leq K_m$. The systolic array allows spatial data reutilisation between rows and columns and temporal data reutilisation.

However, Systolic Arrays has also issues. First, when the kernel size is much lesser than the maximal kernel size ($K_* << K_m$), there is an underutilization of the resources. For example, [19] notes that for a 3×3 kernel, only 9% of the Digital Signal Processing (DSP) blocks are used. Second, data caching is not implemented. It means that it has always to fetch input from the external memory. Memory becomes the bottleneck and the performance is bounded by the device memory bandwidth.

4.2.2 Data-flow MoC for CNNs

Data-flow Model of Computation (MoC) can be used to accelerate CNNs on FPGA. This approach is motivated because the feed-forward aspect of the inference stage of the CNN is purely data driven. Data-flow Model of Computation (MoC) were firstly investigated by [20]. We describe the CNN as a network where:

- **nodes** are processing unit called an *actor*. Each actor follows a data-driven exectution where the execution is triggered by the availability of input, which is the case for a CNN.
- **edges** are communication FIFO channels. Actors exchange data called *tokens* through those FIFO channels.

A represention of such network can be found in figure 4.2.2.

The CNN can be therefore modeled as a topology matrix and we only have to explore those matrix components (instead of tiling and unrolling parameters of section 4.2.3) to minimize latency or energy consumption. Those parameters are then used to derive PE and buffer configuration. However, as pointed by [21], the direct hardware mapping of CNN network means that all the computations must be unrolled, we are then bounded by the hardware resources and the size of the CNN, preventing implementing this approach for deep models.

4.2.3 Loop Optimization

Single Instruction Multiple Data (SIMD) accelerators were proposed to solve the static systolic array inefficiency. The general computation flow can be described as follow:

1. Fetch FMs and weights from the external memory (DRAM) to on-chip buffer.
2. The FMs and weights are streamed into the PE.
3.

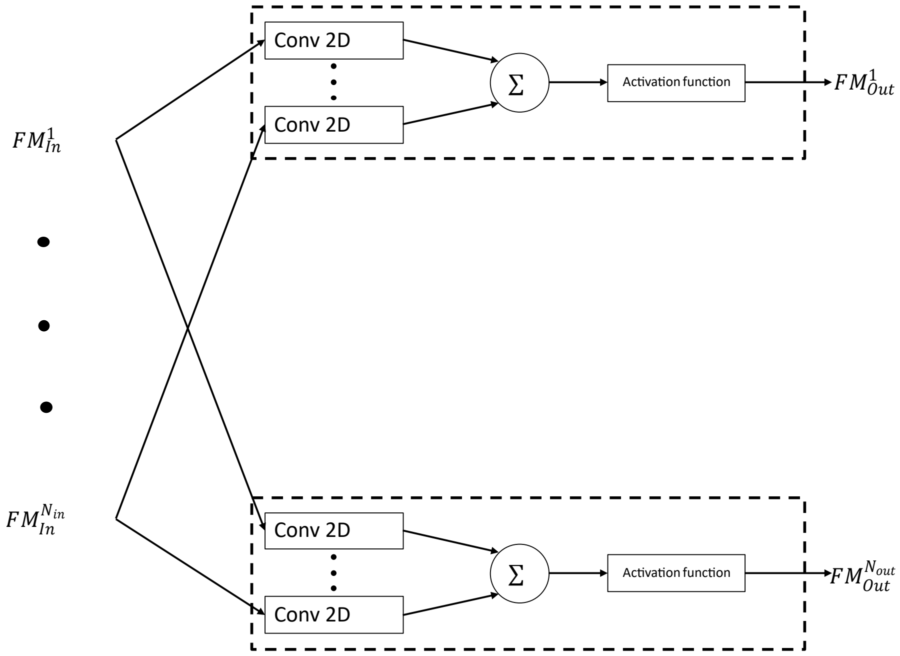

Figure 4.3: Graph represenation of a convolution layer

Loop Unrolling

Loop Tiling

Loop Interchange

4.2.4 Design space exploration

4.3 Model Optimizations

4.3.1 Efficient Model Design

The size of the model can be reduced by changing the architectures of the models. As many approaches have been proposed, we focus our interest on architectures that target the embedded space.

SqueezeNet [22] uses an architecture similar to AlexNet and replaces all layers (except the first and last one) by **Fire Module**. The **Fire Module** is a building blocks where the convolution filter is composed of two layers. The first one is composed only of 1×1 filters and the second one is composed of 1×1 and 3×3 convolutions. We can reduce with this method the number of parameters of AlexNet from 240MB to 4.8MB. The number of parameters can even be reduced to 0.47MB with no loss of accuracy from the baseline AlexNet method by applying Deep Compression [23].

NasNet [24] uses a search method, Neural Architectural Search (NAS), to find good convolutional architectures on a dataset of interest. A controller recurrent neural network saloes child networks with different architecture. The learned architecture is flexible as it may be scaled in terms of computational cost. However the resulting network ends up very complex [16].

MobileNet.

ShuffleNet [25] is a computation-efficient architecture designed for mobile devices with very limited computing power. It reduces computation cost while maintaining accuracy by using **pointwise group convolution** which reduces computation complexity of 1 × 1 convolution. It uses also **channel shuffle** on the channels such that **group convolutions** obtain information from different groups. Then more powerful structures can be build with multiple group convolutional layer.

MobileNetV2 [16] is an improvement of MobileNet.

4.3.2 Pruning

4.3.3 Quantization

The approach

4.4 Conclusion

We can observe on table that pruning seem to be a preferable solutions because ...

On next chapter we are going to explore how to handle pruning on FPGA.

Chapter 5

Model Compression

babla

Chapter 6

Architecture Design

In this chapter we will discuss the approach to model an architecture to handle the pruning on dephtwise separable convolution.

Chapter 7

Measure and Results

blabla

Chapter 8

Conclusion

Bibliography

[1] Peter Norvig Stuart Russell. *Artificial Intelligence: A Modern Approach.* English. 3rd ed. Prentice Hall, Dec. 2009.

[2] Tom M. Mitchell. *Machine Learning.* en. McGraw-Hill series in computer science. New York: McGraw-Hill, 1997. ISBN: 978-0-07-042807-2.

[3] Ludovic Arnold et al. "An Introduction to Deep Learning". en. In: *Computational Intelligence* (2011), p. 12.

[4] Md Zahangir Alom et al. "The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches". In: *arXiv:1803.01164 [cs]* (Sept. 2018). arXiv: 1803.01164. URL: `http://arxiv.org/abs/1803.01164` (visited on 05/13/2020).

[5] Ritika Wason. "Deep learning: Evolution and expansion". en. In: *Cognitive Systems Research* 52 (Dec. 2018), pp. 701–708. ISSN: 13890417. DOI: `10.1016/j.cogsys.2018.08.023`. URL: `https://linkinghub.elsevier.com/retrieve/pii/S1389041717303546` (visited on 05/13/2020).

[6] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". en. In: *International Journal of Computer Vision* 115.3 (Dec. 2015), pp. 211–252. ISSN: 0920-5691, 1573-1405. DOI: `10.1007/s11263-015-0816-y`. URL: `http://link.springer.com/10.1007/s11263-015-0816-y` (visited on 05/13/2020).

[7] Ahmad Shawahna, Sadiq M. Sait, and Aiman El-Maleh. "FPGA-Based Accelerators of Deep Learning Networks for Learning and Classification: A Review". In: *IEEE Access* 7 (2019). Conference Name: IEEE Access, pp. 7823–7859. ISSN: 2169-3536. DOI: `10.1109/ACCESS.2018.2890150`.

[8] Zhiqiang Liu et al. "A Uniform Architecture Design for Accelerating 2D and 3D CNNs on FPGAs". en. In: *Electronics* 8.1 (Jan. 2019). Number: 1 Publisher: Multidisciplinary Digital Publishing Institute, p. 65. DOI: `10.3390/electronics8010065`. URL: `https://www.mdpi.com/2079-9292/8/1/65` (visited on 05/13/2020).

[9] Kamel Abdelouahab et al. "Accelerating CNN inference on FPGAs: A Survey". In: *arXiv:1806.01683 [cs]* (May 2018). arXiv: 1806.01683. URL: `http://arxiv.org/abs/1806.01683` (visited on 06/12/2020).

[10] Asifullah Khan et al. "A Survey of the Recent Architectures of Deep Convolutional Neural Networks". en. In: *Artificial Intelligence Review* (Apr. 2020). arXiv: 1901.06032. ISSN: 0269-2821, 1573-7462. DOI: 10.1007/s10462-020-09825-6. URL: http://arxiv.org/abs/1901.06032 (visited on 06/06/2020).

[11] In The Brain and F. Rosenblatt. *The Perceptron: A Probabilistic Model for Information Storage and Organization.*

[12] Vivienne Sze et al. "Efficient Processing of Deep Neural Networks: A Tutorial and Survey". In: *Proceedings of the IEEE* 105.12 (Dec. 2017). Conference Name: Proceedings of the IEEE, pp. 2295–2329. ISSN: 1558-2256. DOI: 10.1109/JPROC.2017.2761740.

[13] Chaoyang Zhu et al. "An Efficient Hardware Accelerator for Structured Sparse Convolutional Neural Networks on FPGAs". In: *arXiv:2001.01955 [cs, eess]* (Jan. 2020). arXiv: 2001.01955. URL: http://arxiv.org/abs/2001.01955 (visited on 06/14/2020).

[14] S. Winograd. *Arithmetic Complexity of Computations*. CBMS-NSF Regional Conference Series in Applied Mathematics. Society for Industrial and Applied Mathematics, 1980. ISBN: 978-0-89871-163-9. URL: https://books.google.be/books?id=eWb9EF7BmCYC.

[15] Andrew Lavin and Scott Gray. "Fast Algorithms for Convolutional Neural Networks". In: *arXiv:1509.09308 [cs]* (Nov. 2015). arXiv: 1509.09308. URL: http://arxiv.org/abs/1509.09308 (visited on 06/14/2020).

[16] Mark Sandler et al. "MobileNetV2: Inverted Residuals and Linear Bottlenecks". In: *arXiv:1801.04381 [cs]* (Mar. 2019). arXiv: 1801.04381. URL: http://arxiv.org/abs/1801.04381 (visited on 06/14/2020).

[17] Jong Hwan Ko et al. "Design of an energy-efficient accelerator for training of convolutional neural networks using frequency-domain computation". In: *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*. June 2017, pp. 1–6. DOI: 10.1145/3061639.3062228.

[18] Steven W. Smith. *The Scientist and Engineer's Guide to Digital Signal Processing's Table of Content*. 1st ed. California Technical Pub, 1997. ISBN: 0-9660176-3-3. URL: https://www.dspguide.com/pdfbook.htm (visited on 06/14/2020).

[19] Vinayak Gokhale et al. "A 240 G-ops/s Mobile Coprocessor for Deep Neural Networks". In: *2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops*. ISSN: 2160-7516. June 2014, pp. 696–701. DOI: 10.1109/CVPRW.2014.106.

[20] Lin Li et al. “Low power design methodology for signal processing systems using lightweight dataflow techniques”. In: *2016 Conference on Design and Architectures for Signal and Image Processing (DASIP)*. Oct. 2016, pp. 82–89. DOI: 10.1109/DASIP.2016.7853801.

[21] Kamel ABDELOUAHAB et al. “Tactics to Directly Map CNN graphs on Embedded FPGAs”. In: *IEEE Embedded Systems Letters* 9.4 (2017). Publisher: Institute of Electrical and Electronics Engineers, pp. 113–116. DOI: 10.1109/LES.2017.2743247. URL: https://hal.archives-ouvertes.fr/hal-01626462 (visited on 06/15/2020).

[22] Forrest N. Iandola et al. “SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size”. In: *arXiv:1602.07360 [cs]* (Nov. 2016). arXiv: 1602.07360. URL: http://arxiv.org/abs/1602.07360 (visited on 06/17/2020).

[23] Song Han, Huizi Mao, and William J. Dally. “Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding”. In: *arXiv:1510.00149 [cs]* (Feb. 2016). arXiv: 1510.00149. URL: http://arxiv.org/abs/1510.00149 (visited on 06/17/2020).

[24] Barret Zoph et al. “Learning Transferable Architectures for Scalable Image Recognition”. In: *arXiv:1707.07012 [cs, stat]* (Apr. 2018). arXiv: 1707.07012. URL: http://arxiv.org/abs/1707.07012 (visited on 06/17/2020).

[25] Xiangyu Zhang et al. “ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices”. en. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. Salt Lake City, UT: IEEE, June 2018, pp. 6848–6856. ISBN: 978-1-5386-6420-9. DOI: 10.1109/CVPR.2018.00716. URL: https://ieeexplore.ieee.org/document/8578814/ (visited on 06/17/2020).

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**