DIGITAL IMAGING AND COMPUTER VISION Series

DEEP LEARNING IN COMPUTER VISION

Principles and Applications

Edited by
Mahmoud Hassaballah
Ali Ismail Awad

CRC Press
Taylor & Francis Group

Deep Learning in Computer Vision

Digital Imaging and Computer Vision Series

Series Editor

Rastislav Lukac

Foveon, Inc./Sigma Corporation San Jose, California, U.S.A.

Dermoscopy Image Analysis
by M. Emre Celebi, Teresa Mendonça, and Jorge S. Marques

Semantic Multimedia Analysis and Processing
by Evaggelos Spyrou, Dimitris Iakovidis, and Phivos Mylonas

Microarray Image and Data Analysis: Theory and Practice
by Luis Rueda

Perceptual Digital Imaging: Methods and Applications
by Rastislav Lukac

Image Restoration: Fundamentals and Advances
by Bahadir Kursat Gunturk and Xin Li

Image Processing and Analysis with Graphs: Theory and Practice
by Olivier Lézoray and Leo Grady

Visual Cryptography and Secret Image Sharing
by Stelvio Cimato and Ching-Nung Yang

Digital Imaging for Cultural Heritage Preservation: Analysis, Restoration, and Reconstruction of Ancient Artworks
by Filippo Stanco, Sebastiano Battiato, and Giovanni Gallo

Computational Photography: Methods and Applications
by Rastislav Lukac

Super-Resolution Imaging
by Peyman Milanfar

Deep Learning in Computer Vision

Principles and Applications

Edited by
Mahmoud Hassaballah and Ali Ismail Awad

CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business

CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

Printed on acid-free paper

International Standard Book Number-13: 978-1-138-54442-0 (Hardback)

Library of Congress Cataloging-in-Publication Data

Names: Hassaballah, Mahmoud, editor. | Awad, Ali Ismail, editor.
Title: Deep learning in computer vision : principles and applications / edited by M. Hassaballah and Ali Ismail Awad.
Description: First edition. | Boca Raton, FL : CRC Press/Taylor and Francis, 2020. | Series: Digital imaging and computer vision | Includes bibliographical references and index.
Identifiers: LCCN 2019057832 (print) | LCCN 2019057833 (ebook) | ISBN 9781138544420 (hardback ; acid-free paper) | ISBN 9781351003827 (ebook)
Subjects: LCSH: Computer vision. | Machine learning.
Classification: LCC TA1634 .D437 2020 (print) | LCC TA1634 (ebook) | DDC 006.3/7--dc23
LC record available at https://lccn.loc.gov/2019057832
LC ebook record available at https://lccn.loc.gov/2019057833

Visit the Taylor & Francis Web site at
http://www.taylorandfrancis.com

and the CRC Press Web site at
http://www.crcpress.com

Contents

Foreword

Deep learning, while it has multiple definitions in the literature, can be defined as "inference of model parameters for decision making in a process mimicking the understanding process in the human brain"; or, in short: "brain-like model identification". We can say that deep learning is a way of data inference in machine learning, and the two together are among the main tools of modern artificial intelligence. Novel technologies away from traditional academic research have fueled R&D in convolutional neural networks (CNNs); companies like Google, Microsoft, and Facebook ignited the "art" of data manipulation, and the term "deep learning" became almost synonymous with decision making.

Various CNN structures have been introduced and invoked in many computer vision-related applications, with greatest success in face recognition, autonomous driving, and text processing. The reality is: deep learning is an art, not a science. This state of affairs will remain until its developers develop the theory behind its functionality, which would lead to "cracking its code" and explaining why it works, and how it can be structured as a function of the information gained with data. In fact, with deep learning, there is good and bad news. The good news is that the industry—not necessarily academia—has adopted it and is pushing its envelope. The bad news is that the industry does not share its secrets. Indeed, industries are never interested in procedural and textbook-style descriptions of knowledge.

This book, *Deep Learning in Computer Vision: Principles and Applications*—as a journey in the progress made through deep learning by academia—confines itself to deep learning for computer vision, a domain that studies sensory information used by computers for decision making, and has had its impacts and drawbacks for nearly 60 years. Computer vision has been and continues to be a system: sensors, computer, analysis, decision making, and action. This system takes various forms and the flow of information within its components, not necessarily in tandem. The linkages between computer vision and machine learning, and between it and artificial intelligence, are very fuzzy, as is the linkage between computer vision and deep learning. Computer vision has moved forward, showing amazing progress in its short history. During the sixties and seventies, computer vision dealt mainly with capturing and interpreting optical data. In the eighties and nineties, geometric computer vision added science (geometry plus algorithms) to computer vision. During the first decade of the new millennium, modern computing contributed to the evolution of object modeling using multimodality and multiple imaging. By the end of that decade, a lot of data became available, and so the term "deep learning" crept into computer vision, as it did into machine learning, artificial intelligence, and other domains.

This book shows that traditional applications in computer vision can be solved through invoking deep learning. The applications addressed and described in the eleven different chapters have been selected in order to demonstrate the capabilities of deep learning algorithms to solve various issues in computer vision. The content of this book has been organized such that each chapter can be read independently

of the others. Chapters of the book cover the following topics: accelerating the CNN inference on field-programmable gate arrays, fire detection in surveillance applications, face recognition, action and activity recognition, semantic segmentation for autonomous driving, aerial imagery registration, robot vision, tumor detection, and skin lesion segmentation as well as skin melanoma classification.

From the assortment of approaches and applications in the eleven chapters, the common thread is that deep learning for identification of CNN provides accuracy over traditional approaches. This accuracy is attributed to the flexibility of CNN and the availability of large data to enable identification through the deep learning strategy. I would expect that the content of this book to be welcomed worldwide by graduate and postgraduate students and workers in computer vision, including practitioners in academia and industry. Additionally, professionals who want to explore the advances in concepts and implementation of deep learning algorithms applied to computer vision may find in this book an excellent guide for such purpose. Finally, I hope that readers would find the presented chapters in the book interesting and inspiring to future research, from both theoretical and practical viewpoints, to spur further advances in discovering the secrets of deep learning.

Prof Aly Farag, PhD, Life Fellow, IEEE, Fellow, IAPR
Professor of Electrical and Computer Engineering
University of Louisville, Kentucky

Preface

Simply put, computer vision is an interdisciplinary field of artificial intelligence that aims to guide computers and machines toward understanding the contents of digital data (i.e., images or video). According to computer vision achievements, the future generation of computers may understand human actions, behaviors, and languages similarly to humans, carry out some missions on their behalf, or even communicate with them in an intelligent manner. One aspect of computer vision that makes it such an interesting topic of study and active research field is the amazing diversity of daily-life applications such as pedestrian protection systems, autonomous driving, biometric systems, the movie industry, driver assistance systems, video surveillance, and robotics as well as medical diagnostics and other healthcare applications. For instance, in healthcare, computer vision algorithms may assist healthcare professionals to precisely classify illnesses and cases; this can potentially save patients' lives through excluding inaccurate medical diagnoses and avoiding erroneous treatment. With this wide variety of applications, there is a significant overlap between computer vision and other fields such as machine vision and image processing. Scarcely a month passes where we do not hear from the research and industry communities with an announcement of some new technological breakthrough in the areas of intelligent systems related to the computer vision field.

With the recent rapid progress on deep convolutional neural networks, deep learning has achieved remarkable performance in various fields. In particular, it has brought a revolution to the computer vision community, introducing non-traditional and efficient solutions to several problems that had long remained unsolved. Due to this promising performance, it is gaining more and more attention and is being applied widely in computer vision for several tasks such as object detection and recognition, object segmentation, pedestrian detection, aerial imagery registration, video processing, scene classification, autonomous driving, and robot localization as well as medical image-related applications. If the phrase "deep learning for computer vision" is searched in Google, millions of search results will be obtained. Under these circumstances, a book entitled *Deep Learning in Computer Vision* that covers recent progress and achievements in utilizing deep learning for computer vision tasks will be extremely useful.

The purpose of this contributed volume is to fill the existing gap in the literature for the applications of deep learning in computer vision and to provide a bird's eye view of recent state-of-the-art models designed for practical problems in computer vision. The book presents a collection of eleven high-quality chapters written by renowned experts in the field. Each chapter provides the principles and fundamentals of a specific topic, introduces reviews of up-to-date techniques, presents outcomes, and points out challenges and future directions. In each chapter, figures, tables, and examples are used to improve the presentation and analysis of covered topics. Furthermore, bibliographic references are included in each chapter, providing a good starting point for deeper research and further exploration of the topics considered in this book. Further, this book is structured such that each chapter can be read independently from the others as follows:

Chapter 1 presents a state-of-the-art of CNN inference accelerators over FPGAs. Computational workloads, parallelism opportunities, and the involved memory accesses are analyzed. At the level of neurons, optimizations of the convolutional and fully connected layers are explained and the performances of the different methods compared, while at the network level, approximate computing and data-path optimization methods are covered and state-of-the-art approaches compared. The methods and tools investigated in this chapter represent the recent trends in FPGA CNN inference accelerators and will fuel future advances in efficient hardware deep learning.

Chapter 2 concentrates on object detection problem using deep CNN (DCNN): the recent developments of several classical CNN-based object detectors are discussed. These detectors significantly improve detection performance either through employing new architectures or through solving practical issues like degradation, gradient vanishing, and class imbalance. Detailed background information is provided to show the progress and improvements of different models. Some evaluation results and comparisons are reported on three datasets with distinctive characteristics.

Chapter 3 proposes three methods for fire detection using CNNs. The first method focuses on early fire detection with an adaptive prioritization mechanism for surveillance cameras. The second CNN-assisted method improves fire detection accuracy with a main focus on reducing false alarms. The third method uses an efficient deep CNN for fire detection. For localization of fire regions, a feature map selection algorithm that intelligently selects appropriate feature maps sensitive to fire areas is proposed.

Chapter 4 presents an accurate and real-time multi-biometric system for identifying a person's identity using a combination of two discriminative deep learning approaches to address the problem of unconstrained face recognition: CNN and deep belief network (DBN). The proposed system is tested on four large-scale challenging datasets with high diversity in the facial expressions—SDUMLA-HMT, FRGC V 2.0, UFI, and LFW—and new state-of-the-art recognition rates on all the employed datasets are achieved.

Chapter 5 introduces a study of the concept of sequence learning using RNN, LSTM, and its variants such as multilayer LSTM and bidirectional LSTM for action and activity recognition problems. The chapter concludes with major issues of sequence learning for action and activity recognition and highlights recommendations for future research.

Chapter 6 discuses semantic segmentation in autonomous driving applications, where it focuses on constructing efficient and simple architectures to demonstrate the benefit of flow and depth augmentation to CNN-based semantic segmentation networks. The impact of both motion and depth information on semantic segmentation is experimentally studied using four simple network architectures. Results of experiments on two public datasets—Virtual-KITTI and CityScapes—show reasonable improvement in overall accuracy.

Chapter 7 presents a method based on deep learning for geolocalizing drones using only onboard cameras. A pipeline has been implemented that makes use of the availability of satellite imagery and traditional computer vision feature detectors and descriptors, along with renowned deep learning methods (semantic segmentation), to be able to locate the aerial image captured from the drone within the satellite imagery. The method enables the drone to be autonomously aware of its surroundings and navigate without using GPS.

Chapter 8 is intended to be a guide for the developers of robot vision systems, focusing on the practical aspects of the use of deep neural networks rather than on theoretical issues.

The last three chapters are devoted to deep learning in medical applications. Chapter 9 covers basic information about CNNs in medical applications. CNN developments are discussed from different perspectives, specifically, CNN design, activation function, loss function, regularization, optimization, normalization, and network depth. Also, a deep convolutional neural network (DCNN) is designed for brain tumor detection using MRI images. The proposed DCNN architecture is evaluated on the RIDER dataset, achieving accurate detection accuracy within a time of 0.24 seconds per MRI image.

Chapter 10 discusses automatic segmentation of skin lesion boundaries from surrounding tissue and presents a novel deep learning segmentation methodology via full-resolution convolutional network (FrCN). Experimental results show the great promise of the FrCN method compared to state-of-the-art deep learning segmentation approaches such as fully convolutional networks (FCN), U-Net, and SegNet with overall segmentation.

Chapter 11 is about the automatic classification of color skin images, where a highly accurate method is proposed for skin melanoma classification utilizing two modified deep convolutional neural networks and consisting of three main steps. The proposed method is tested using the well-known MED-NODE and DermIS & DermQuest datasets.

It is very necessary to mention here that the book is a small piece in the puzzle of computer vision and its applications. We hope that our readers find the presented chapters in the book interesting and that the chapters will inspire future research both from theoretical and practical viewpoints to spur further advances in the computer vision field.

The editors would like to take this opportunity to express their sincere gratitude to the contributors for extending their wholehearted support in sharing some of their latest results and findings. Without their significant contribution, this book could not have fulfilled its mission. The reviewers deserve our thanks for their constructive and timely input. Special profound thanks go to Prof Aly Farag, Professor of Electrical and Computer Engineering, University of Louisville, Kentucky for writing the Foreword for this book. Finally, the editors acknowledge the efforts of the CRC Press Taylor & Francis for giving us the opportunity to edit a book on deep learning for computer vision. In particular, we would like to thank Dr Rastislav Lukac, the editor of the Digital Imaging and Computer Vision book series, and Nora Konopka for initiating this project. Really, the editorial staff at CRC Press has done a meticulous job, and working with them was a pleasant experience.

Mahmoud Hassaballah
Qena, Egypt

Ali Ismail Awad
Luleå, Sweden

Taylor & Francis
Taylor & Francis Group
http://taylorandfrancis.com

Editors Bio

Mahmoud Hassaballah was born in 1974, Qena, Egypt. He received his BSc degree in Mathematics in 1997 and his MSc degree in Computer Science in 2003, both from South Valley University, Egypt, and his Doctor of Engineering (D Eng) in computer science from Ehime University, Japan in 2011. He was a visiting scholar with the department of computer & communication science, Wakayama University, Japan in 2013 and GREAH laboratory, Le Havre Normandie University, France in 2019. He is currently an associate professor of computer science at the faculty of computers and information, South Valley University, Egypt. He served as a reviewer for several journals such as *IEEE Transactions on Image Processing, IEEE Transactions on Fuzzy Systems, Pattern Recognition, Pattern Recognition Letters, IET Image Processing, IET Computer Vision, IET Biometrics, Journal of Real-Time Image Processing, and Journal of Electronic Imaging*. He has published over 50 research papers in refereed international journals and conferences. His research interests include feature extraction, object detection/recognition, artificial intelligence, biometrics, image processing, computer vision, machine learning, and data hiding.

Ali Ismail Awad (SMIEEE, PhD, PhD, MSc, BSc) is currently an Associate Professor (Docent) with the Department of Computer Science, Electrical, and Space Engineering, Luleå University of Technology, Luleå, Sweden, where he also serves as a Coordinator of the Master Programme in Information Security. He is a Visiting Researcher with the University of Plymouth, United Kingdom. He is also an Associate Professor with the Electrical Engineering Department, Faculty of Engineering, Al-Azhar University at Qena, Qena, Egypt. His research interests include information security, Internet-of-Things security, image analysis with applications in biometrics and medical imaging, and network security. He has edited or co-edited five books and authored or co-authored several journal articles and conference papers in these areas. He is an Editorial Board Member of the following journals: *Future Generation Computer Systems*, *Computers & Security*, *Internet of Things: Engineering Cyber Physical Human Systems*, and *Health Information Science and Systems*. Dr Awad is currently an IEEE senior member.

Taylor & Francis
Taylor & Francis Group
http://taylorandfrancis.com

Contributors

Ahmad El Sallab
Valeo Company
Cairo, Egypt

Ahmed Nassar
IRISA Institute
Rennes, France

Alaa S. Al-Waisy
University of Bradford
Bradford, UK

Ali Ismail Awad
Luleå University of Technology
Luleå, Sweden
and
Al-Azhar University
Qena, Egypt

Amin Ullah
Sejong University
Seoul, South Korea

Ashraf A. M. Khalaf
Minia University
Minia, Egypt

François Berry
University Clermont Auvergne
Clermont-Ferrand, France

Guanghui Wang
University of Kansas
Kansas City, Kansas

Hazem Rashed
Valeo Company
Cairo, Egypt

Hesham F.A. Hamed
Egyptian Russian University
Cairo, Egypt
and
Minia University
Minia, Egypt

Javier Ruiz-Del-Solar
University of Chile
Santiago, Chile

Kaidong Li
University of Kansas
Kansas City, Kansas

Kamel Abdelouahab
Clermont Auvergne University
Clermont-Ferrand, France

Khalid M. Hosny
Zagazig University
Zagazig, Egypt

Khan Muhammad
Sejong University
Seoul, South Korea

Mahmoud Hassaballah
South Valley University
Qena, Egypt

Mahmoud Khaled Abd-Ellah
Al-Madina Higher Institute for
Engineering and Technology
Giza, Egypt

Maxime Pelcat
University of Rennes
Rennes, France

Miyoung Lee
Sejong University
Seoul, South Korea

Mohamed A. Kassem
Kafr El Sheikh University
Kafr El Sheikh, Egypt

Mohamed Elhelw
Nile University
Giza, Egypt

Mohamed M. Foaud
Zagazig University
Zagazig, Egypt

Mohammed A. Al-Masni
Kyung Hee University
Seoul, South Korea
and
Yonsei University
Seoul, South Korea

Mugahed A. Al-Antari
Kyung Hee University
Seoul, South Korea
and
Sana'a Community College
Sana'a, Republic of Yemen

Patricio Loncomilla
University of Chile
Santiago, Chile

Rami Qahwaji
University of Bradford
Bradford, UK

Salman Khan
Sejong University
Seoul, South Korea

Senthil Yogamani
Valeo Company
Galway, Ireland

Shumoos Al-Fahdawi
University of Bradford
Bradford, UK

Sung Wook Baik
Sejong University
Seoul, South Korea

Tae-Seong Kim
Kyung Hee University
Seoul, South Korea

Tanveer Hussain
Sejong University
Seoul, South Korea

Usman Sajid
University of Kansas
Kansas City, Kansas

Wenchi Ma
University of Kansas
Kansas City, Kansas

Yuanwei Wu
University of Kansas
Kansas City, Kansas

1 Accelerating the CNN Inference on FPGAs

Kamel Abdelouahab, Maxime Pelcat, and François Berry

CONTENTS

1.1 INTRODUCTION

The exponential growth of big data during the last decade motivates for innovative methods to extract high semantic information from raw sensor data such as videos, images, and speech sequences. Among the proposed methods, convolutional neural networks (CNNs) [1] have become the de facto standard by delivering near-human accuracy in many applications related to machine vision (e.g., classification [2], detection [3], segmentation [4]) and speech recognition [5].

This performance comes at the price of a large computational cost as CNNs require up to 38 GOPs to classify a single frame [6]. As a result, dedicated hardware is required to accelerate their execution. Graphics processing units GPUs are the most widely used platform to implement CNNs as they offer the best performance in terms of pure computational throughput, reaching up 11 TFLOPs [7]. Nevertheless, in terms of power consumption, field-programmable gate array (FPGA) solutions are known to be more energy efficient (vs. GPU). While GPU implementations have demonstrated state-of-the-art computational performance, CNN acceleration will soon be moving towards FPGAs for two reasons. First, recent improvements in FPGA technology put FPGA performance within striking distance of GPUs with a reported performance of 9.2 TFLOPs for the latter [8]. Second, recent trends in CNN development increase the sparsity of CNNs and use extremely compact data types. These trends favor FPGA devices, which are designed to handle irregular parallelism and custom data types. As a result, next-generation CNN accelerators are expected to deliver up to 5.4× better computational throughput than GPUs [7].

As an inflection point in the development of CNN accelerators might be near, we conduct a survey on FPGA-based CNN accelerators. While a similar survey can be found in [9], we focus in this chapter on the recent techniques that were not covered in the previous works. In addition to this chapter, we refer the reader to the works of Venieris et al. [10], which review the toolflows automating the CNN mapping process, and to the works of Sze et al., which focus on ASICs for deep learning acceleration.

The amount and diversity of research on the subject of CNN FPGA acceleration within the last 3 years demonstrate the tremendous industrial and academic interest. This chapter presents a state-of-the-art review of CNN inference accelerators over FPGAs. The computational workloads, their parallelism, and the involved memory accesses are analyzed. At the level of neurons, optimizations of the convolutional and fully connected (FC) layers are explained and the performances of the different methods compared. At the network level, approximate computing and data-path optimization methods are covered and state-of-the-art approaches compared. The methods and tools investigated in this survey represent the recent trends in FPGA CNN inference accelerators and will fuel the future advances on efficient hardware deep learning.

1.2 BACKGROUND ON CNNS AND THEIR COMPUTATIONAL WORKLOAD

In this first section, we overview the main features of CNNs, mainly focusing on the computations and parallelism patterns involved during their inference.

1.2.1 General Overview

Deep* CNNs are feed-forward†, sparsely connected‡ neural networks. A typical CNN structure consists of a pipeline of layers. Each layer inputs a set of data, known as a feature map (FM), and produces a new set of FMs with *higher-level semantics*.

1.2.2 Inference versus Training

As typical machine learning algorithms, CNNs are deployed in two phases. First, the *training* stage works on a known set of annotated data samples to create a model with a *modeling* power (which semantics extrapolates to natural data outside the training set). This phase implements the *back-propagation* algorithm [11], which iteratively updates CNN parameters such as convolution weights to improve the predictive power of the model. A special case of CNN training is *fine-tuning*. When *fine-tuning* a model, weights of a previously trained network are used to initialize the parameters of a new training. These weights are then adjusted for a new constraint, such as a different dataset or a reduced precision.

The second phase, known as *inference*, uses the learned model to classify new data samples (i.e., inputs that were not previously seen by the model). In a typical setup, CNNs are trained/fine-tuned only once, on large clusters of GPUs. By contrast, the inference is implemented each time a new data sample has to be classified. As a consequence, the literature mostly focuses on accelerating the inference phase. As a result, our discussion overviews the main methods employed to accelerate the inference. Moreover, since most of the CNN accelerators benchmark their performance on models trained for image classification, we focus our chapter on this application. Nonetheless, the methods detailed in this survey can be employed to accelerate CNNs for other applications such object detection, image segmentation, and speech recognition.

1.2.3 Inference, Layers, and CNN Models

CNN inference refers to the *feed-forward* propagation of B input images across L layers. This section details the computations involved in the major types of these layers. A common practice is to manipulate layers, parameters, and FMs as multidimensional arrays, as listed in Table 1.1. Note that when it will be relevant, the type of the layer will be denoted with superscript, and the position of the layer will be denoted with subscript.

---

* Includes a large number of layer, typically above three.

† The information flows from the neurons of a layer ℓ towards the neurons of a layer. $\ell + 1$

‡ CNNs implement the weight sharing technique, applying a small number of weights across all the input pixels (i.e., image convolution).

TABLE 1.1
Tensors Involved in the Inference of a Given Layer ℓ with Their Dimensions

$\boldsymbol{X}$	Input FMs	$B \times C \times H \times W$	B	Batch size (Number of input frames)
$\boldsymbol{Y}$	Output FMs	$B \times N \times V \times U$	$W/H/C$	Width/Height/Depth of Input FMs
Θ	Learned Filters	$N \times C \times J \times K$	$U/V/N$	Width/Height/Depth of Output FMs
β	Learned biases	N	K/J	Horizontal/Vertical Kernel size

A convolutional layer (*conv*) carries out the feature extraction process by applying – as illustrated in Figure 1.1 – a set of three-dimensional convolution filters Θ^{conv} to a set of B input volumes $\mathbf{X}^{\text{conv}}$. Each input volume has a depth C and can be a color image (in the case of the first *conv* layer), or an output generated by previous layers in the network. Applying a three-dimensional filter to three-dimensional input results in a 2D *(FM)*. Thus, applying N three-dimensional filters in a layer results in a three-dimensional output with a depth N.

In some CNN models, a learned offset β^{conv} – called a *bias* – is added to processed feature maps. However, this practice has been discarded in recent models [6]. The computations involved in feed-forward propagation of *conv* layers are detailed in Equation 1.1.

$$\forall \{b,n,u,v\} \in [1,B] \times [1,N] \times [1,V] \times [1,U]$$

$$\mathbf{Y}^{\text{conv}}[b,n,v,u] = \beta^{\text{conv}}[n] + \sum_{c=1}^{C} \sum_{j=1}^{J} \sum_{k=1}^{K} \mathbf{X}^{\text{conv}}[b,c,v+j,u+k] \cdot \Theta^{\text{conv}}[n,c,j,k] \tag{1.1}$$

One may note that applying a depth convolution to a 3D input boils down to applying a mainstream 2D convolution to each of the 2D channels of the input, then, at each point, summing the results across all the channels, as shown in Equation 1.2.

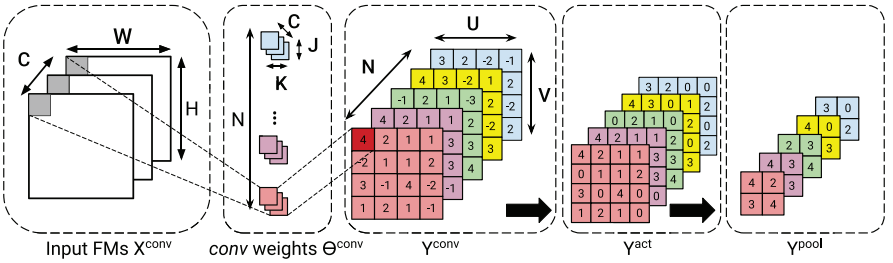

FIGURE 1.1 Feed-forward propagation in *conv*, *act*, and *pool* layers (batch size $B = 1$, bias β omitted).

$$\forall n \in [1, N]$$

$$\boldsymbol{Y}[\boldsymbol{n}]^{\text{conv}} = \beta^{\text{conv}}[n] + \sum_{c=1}^{C} \text{conv2D}\left(\mathbf{X}[\boldsymbol{c}]^{\text{conv}}, \Theta[\boldsymbol{c}]^{\text{conv}}\right) \tag{1.2}$$

Each *conv* layer of a CNN is usually followed by an activation layer that applies a *nonlinear* function to all the values of FMs. Early CNNs were trained with TanH or Sigmoid functions, but recent models employ the rectified linear unit (ReLU) function, which grants faster training times and less computational complexity, as highlighted in Krizhevsky et al. [12].

$$\forall \{b, n, u, v\} \in [1, B] \times [1, N] \times [1, V] \times [1, U]$$

$$\boldsymbol{Y}^{\text{act}}[b, n, h, w] = \text{act}(\boldsymbol{X}^{\text{act}}[b, n, h, w]) \mid \text{act:=TanH, Sigmoid, ReLU} \ldots \tag{1.3}$$

The convolutional and activation parts of a CNN are directly inspired by the cells of visual cortex in neuroscience [13]. This is also the case with *pooling* layers, which are periodically inserted in between successive *conv* layers. As shown in Equation 1.4, *pooling* sub-samples each channel of the input FM by selecting either the *average*, or, more commonly, the *maximum* of a given neighborhood $\boldsymbol{K}$. As a result, the dimensionality of an FM is reduced, as illustrated in Figure 1.1.

$$\forall \{b, n, u, v\} \in [1, B] \times [1, N] \times [1, V] \times [1, U]$$

$$\boldsymbol{Y}^{\text{pool}}[b, n, v, u] = \max_{p,q \in [1:K]} \left(\boldsymbol{X}^{\text{pool}}[b, n, v + p, u + q]\right) \tag{1.4}$$

When deployed for classification purposes, the CNN pipeline is often terminated by FC layers. In contrast with convolutional layers, FC layers do not implement weight sharing and involve as much weight as input data (i.e., $W = K$, $H = J$,$U = V = 1$). Moreover, in a similar way as *conv* layers, a nonlinear function is applied to the outputs of FC layers.

$$\forall \{b, n\} \in [1, B] \times [1, N]$$

$$\boldsymbol{Y}^{\text{fc}}[b, n] = \beta^{\text{fc}}[n] + \sum_{c=1}^{C} \sum_{h=1}^{H} \sum_{w=1}^{W} \boldsymbol{X}^{\text{fc}}[b, c, h, w] \cdot \Theta^{\text{fc}}[n, c, h, w] \tag{1.5}$$

The Softmax function is a generalization of the Sigmoid function, and "squashes" a N-dimensional vector $\boldsymbol{X}$ to $Sigmoid(\boldsymbol{X})$ where each output is in the range [0,1]. The Softmax function is used in various multi-class classification methods, especially in CNNs. In this case, the Softmax layer is placed at the end of the network and the dimension of vector it operates on (i.e., N) represents the number of classes in the considered dataset. Thus, the input of the Softmax is the data generated by the last fully connected layer, and the output is the probability predicted for each class.

$$\forall \{b,n\} \in [1,B] \times [1,N]$$
$$\text{Softmax}(\mathbf{X}[b,n]) = \frac{\exp(\mathbf{X}[b,n])}{\sum_{c=1}^{N} \exp(\mathbf{X}[b,c])} \tag{1.6}$$

Batch normalization was introduced [14] to speed up training by linearly shifting and scaling the distribution of a given batch of inputs B to have zero mean and unit variance. These layers find also their interest when implementing binary neural networks (BNNs) as they reduce the quantization error compared to an arbitrary input distribution, as highlighted in Hubara et al. [15]. Equation 1.7 details the processing of *batch norm* layers, where the mean μ and the variance σ are statistics collected during the training, α and γ are parameters learned during the training, and ϵ is a hyper-parameter set empirically for numerical stability purposes (i.e., avoiding division by zero).

$$\forall \{b,n,u,v\} \in [1,B] \times [1,N] \times [1,V] \times [1,U]$$
$$\mathbf{Y}^{\text{BN}}[b,n,v,u] = \frac{\mathbf{X}^{\text{BN}}[b,n,u,v] - \mu}{\sqrt{\sigma^2 + \epsilon}} \gamma + \alpha \tag{1.7}$$

1.2.4 Workloads and Computations

The accuracy of CNN models has been increasing since their breakthrough in 2012 [12]. However, this accuracy comes at a high computational cost. The main challenge that faces CNN developers is to improve classification accuracy while maintaining a tolerable computational workload. As shown in Table 1.2, this challenge was successfully addressed by Inception [16] and ResNet models [17], with their use of bottleneck 1×1 convolutions that reduce both model size and computations while increasing depth and accuracy.

1.2.4.1 Computational Workload

As shown in Equations 1.1 and 1.5, the processing of CNN involves an intensive use of Multiply Accumulate (MAC) operation. All these MAC operations take place at *conv* and FC layers, while the remaining parts of network are element-wise transformations that can be generally implemented with low-complexity computational requirements.

TABLE 1.2
Popular CNN Models with Their Computational Workload*

Model	AlexNet [12]	GoogleNet [16]	VGG16 [6]	VGG19 [6]	ResNet101 [17]	ResNet-152 [17]
Top1 err (%)	42.9%	31.3%	28.1%	27.3%	23.6% %	23.0%
Top5 err (%)	19.80%	10.07%	9.90%	9.00%	7.1%	6.7%
L_c	5	57	13	16	104	155
$\sum_{\ell=1}^{L_c} \mathcal{C}_\ell^{\text{conv}}$	666 M	1.58 G	15.3 G	19.5 G	7.57 G	11.3 G
$\sum_{\ell=1}^{L_c} \mathcal{W}_\ell^{\text{conv}}$	2.33 M	5.97 M	14.7 M	20 M	42.4 M	58 M
Act	ReLU					
Pool	3	14	5	5	2	2
L_f	3	1	3	3	1	1
$\sum_{\ell=1}^{L_f} \mathcal{C}_\ell^{\text{fc}}$	58.6 M	1.02 M	124 M	124 M	2.05 M	2.05 M
$\sum_{\ell=1}^{L_f} \mathcal{W}_\ell^{fc}$	58.6 M	1.02 M	124 M	124 M	2.05 M	2.05 M
$\mathcal{C}$	724 M	1.58 G	15.5 G	19.6 G	7.57 G	11.3 G
$\mathcal{W}$	61 M	6.99 M	138 M	144 M	44.4 M	60 M

* Accuracy Measured on Single-Crops of ImageNet Test-Set

In this chapter, the computational workload $\mathcal{C}$ of a given CNN corresponds to the number of MACs it involves during inference*. The number of these MACs mainly depends on the topology of the network, and more particularly on the number of *conv* and FC layers and their dimensions. Thus, the computational workload can be expressed as in Equation 1.8, where L_c is the number of *conv* (fully connected) layers, and $\mathcal{C}_\ell^{\text{conv}}$ ($\mathcal{C}_\ell^{\text{fc}}$) is the number of MACs occurring on a given convolution (fully connected) layer ℓ.

$$\mathcal{C} = \sum_{\ell=1}^{L_c} \mathcal{C}_\ell^{\text{conv}} + \sum_{\ell=1}^{L_f} \mathcal{C}_\ell^{\text{fc}} \tag{1.8}$$

$$\mathcal{C}_\ell^{\text{conv}} = N_\ell \times C_\ell \times J_\ell \times K_\ell \times U_\ell \times V_\ell \tag{1.9}$$

$$\mathcal{C}_\ell^{\text{fc}} = N_\ell \times C_\ell \times W_\ell \times H_\ell \tag{1.10}$$

* Batch size is set to 1 for clarity purposes.

In a similar way, the number of weights, and consequently the size of a given CNN model, can be expressed as follows:

$$\mathcal{W} = \sum_{\ell=1}^{L_c} \mathcal{W}_\ell^{\text{conv}} + \sum_{\ell=1}^{L_f} \mathcal{W}_\ell^{\text{fc}} \tag{1.11}$$

$$\mathcal{W}_\ell^{\text{conv}} = N_\ell \times C_\ell \times J_\ell \times K_\ell \tag{1.12}$$

$$\mathcal{W}_\ell^{\text{fc}} = N_\ell \times C_\ell \times W_\ell \times H_\ell \tag{1.13}$$

For state-of-the-art CNN models, L_c, N_ℓ, and C_ℓ can be quite large. This makes CNNs *computationally and memory intensive*, where for instance, the classification of a single frame using the VGG19 network requires 19.5 billion MAC operations.

It can be observed in the same table that most of the MACs occur on the convolution parts, and consequently, 90% of the execution time of a typical inference is spent on *conv* layers [18]. By contrast, FC layers marginalize most of the weights and thus the size of a given CNN model.

1.2.4.2 Parallelism in CNNs

The high computational workload of CNNs makes their inference a challenging task, especially on low-energy embedded devices. The key solution to this challenge is to leverage on the extensive concurrency they exhibit. These parallelism opportunities can be formalized as follows:

- **Batch Parallelism:** CNN implementations can simultaneously classify multiple frames grouped as a *batch B* in order to reuse the filters in each layer, minimizing the number the memory accesses. However, and as shown in [10], batch parallelism quickly reaches its limits. This is due to the fact that most of the memory transactions result from storing intermediate results and not loading CNN parameters. Consequently, reusing the filters only slightly impacts the overall processing time per image.
- **Inter-layer Pipeline Parallelism:** CNNs have a feed-forward hierarchical structure consisting of a succession of data-dependent layers. These layers can be executed in a pipelined fashion by launching layer (ℓ) before ending the execution of layer ($\ell - 1$). This pipelining costs latency but increases throughput.

Moreover, the execution of the most computationally intensive parts (i.e., *conv* layers), exhibits the four following types of concurrency:

- **Inter-FM Parallelism:** Each two-dimensional plane of an FM can be processed separately from the others, meaning that P_N elements of $\boldsymbol{Y}^{\text{conv}}$ can be computed in parallel ($0 < P_N < N$).

- **Intra-FM Parallelism:** In a similar way, pixels of a single output FM plane are data-independent and can thus be processed concurrently by evaluating $P_V \times P_U$ values of $\boldsymbol{Y}^{\text{conv}}[n]$ $(0 < P_V \times P_U < V \times U)$.
- **Inter-convolution Parallelism:** Depth convolutions occurring in *conv* layers can be expressed as a sum of 2D convolutions, as shown in Equation 1.2. These 2D convolutions can be evaluated simultaneously by computing concurrently P_c elements $(0 < P_c < C)$.
- **Intra-convolution Parallelism:** The 2D convolutions involved in the processing of *conv* layers can be implemented in a pipelined fashion such as in [76]. In this case $P_J \times P_K$ multiplications are implemented concurrently $(0 < P_J \times P_K < J \times K)$.

1.2.4.3 Memory Accesses

As a consequence of the previous discussion, the inference of a CNN shows large vectorization opportunities that can be exploited by allocating multiple computational resources to concurrently process multiple features. However, this parallelization can not accelerate the execution of a CNN if no datacaching strategy is implemented. In fact, memory bandwidth is often the bottleneck when processing CNNs.

In *FC* parts, the execution can be memory-bounded because of the high number of weights that these layers contain, and consequently, the high number of memory reads required.

This is expressed in Equation 1.14, where $\mathcal{M}_\ell^{\text{fc}}$ refers to the number of memory accesses occurring in an FC layer ℓ. This number can be written as the sum of memory accesses reading the inputs $\mathbf{X}_\ell^{\text{fc}}$, the memory accesses reading the weights $(\theta_\ell^{\text{fc}})$, and the number of memory accesses writing the results $(\mathbf{Y}_\ell^{\text{fc}})$.

$$\mathcal{M}_\ell^{\text{fc}} = \text{MemRd}(\mathbf{X}_\ell^{\text{fc}}) + \text{MemRd}(\theta_\ell^{\text{fc}}) + \text{MemWr}(\mathbf{Y}_\ell^{\text{fc}}) \tag{1.14}$$

$$= C_\ell H_\ell W_\ell + N_\ell C_\ell H_\ell W_\ell + N_\ell \tag{1.15}$$

$$\sim N_\ell C_\ell H_\ell W_\ell \tag{1.16}$$

Note that the fully connected parts of state-of-the-art models involve large values of N_ℓ and C_ℓ, making the memory reading of weights the most impacting factor, as formulated in Equation 1.16. In this context, batch parallelism can significantly accelerate the execution of CNNs with a large number of FC layers.

In the *conv* parts, the high number of MAC operations results in a high number of memory accesses, as each MAC requires at least 2 memory reads and 1 memory write*. This number of memory accesses accumulates with the high dimensions of data manipulated by *conv* layers, as shown in Equation 1.18. If all these accesses are towards external memory (for instance, DRAM), throughput and energy consumption

* This is the best-case scenario of a fully pipelined MAC, where intermediate results do not need to be loaded.

will be highly impacted, because DRAM access engenders high latency and energy consumption, even more than the computation itself [21].

$$\mathcal{M}_\ell^{\text{conv}} = \text{MemRd}(\mathbf{X}_\ell^{\text{conv}}) + \text{MemRd}(\theta_\ell^{\text{conv}}) + \text{MemWr}(\mathbf{Y}_\ell^{\text{conv}}) \tag{1.17}$$

$$= C_\ell H_\ell W_\ell + N_\ell C_\ell J_\ell K_\ell + N_\ell U_\ell V_\ell \tag{1.18}$$

The number of these DRAM accesses, and thus latency and energy consumption, can be reduced by implementing a memory-caching hierarchy using on-chip memories. As discussed in the next sections, state-of-the-art CNN accelerators employ register files as well as several levels of caches. The former, being the fastest, is implemented at the nearest of the computational capabilities. The latency and energy consumption resulting from these caches is lower by several orders of magnitude than external memory accesses, as pointed out in Sze et al. [22].

1.2.4.4 Hardware, Libraries, and Frameworks

In order to catch the parallelism of CNNs, dedicated hardware accelerators are developed. Most of them are based on GPUs, which are known to perform well on regular parallelism patterns thanks to simd and simt execution models, a dense collection of floating-point computing elements that peak at 12 TFLOPs, and high capacity/bandwidth on/off-chip memories [23]. To support these hardware accelerators, specialized libraries for deep learning are developed to provide the necessary programming abstraction, such as CudNN on Nvidia GPU [24]. Built upon these libraries, dedicated frameworks for deep learning are proposed to improve productivity of conceiving, training, and deploying CNNs, such as Caffe [25] and TensorFlow [26].

Beside GPU implementations, numerous FPGA accelerators for CNNs have been proposed. FPGAs are fine-grained programmable devices that can catch the CNN parallelism patterns with no memory bottleneck, thanks to the following:

1. A high density of hard-wired digital signal processor (DSP) blocks that are able to achieve up to 20 (8 TFLOPs) TMACs [8].
2. A collection of in situ on-chip memories, located next to DSPs, that can be exploited to significantly reduce the number of external memory accesses.

As a consequence, CNNs can benefit from a significant acceleration when running on reconfigurable hardware. This has caused numerous research efforts to study FPGA-based CNN acceleration, targeting both high performance computing (HPC) applications [27] and embedded devices [28].

In the remaining parts of this chapter, we conduct a survey on methods and hardware architectures to accelerate the execution of CNN on FPGA. The next section lists the evaluation metrics used, then Sections 1.4 and 1.5 respectively study the computational transforms and the data-path optimization involved in recent CNN accelerators. Finally, the last section of this chapter details how approximate computing is a key in FPGA-based deep learning, and overviews the main contributions implementing these techniques.

1.3 FPGA-BASED DEEP LEARNING

Accelerating a CNN on an FPGA-powered platform can be seen as an optimization effort that focuses on one or several of the following criteria:

- *Computational Throughput* ($\mathcal{T}$): A large number of the works studied in this chapter focus on reducing the CNN execution times on the FPGA (i.e., the computation latency), by improving the computational throughput of the accelerator. This throughput is usually expressed as the number of MACs an accelerator performs per second. While this metric is relevant in the case of HPC workloads, we prefer to report the throughput as the number of frames an accelerator processes per second (fps), which better suits the embedded vision context. The two metrics can be directly related using Equation 1.19, where $\mathcal{C}$ is defined in Equation 1.8, and refers to the number of computations a CNN involve in order to process a single frame:

$$\mathcal{T}_{(\text{FPS})} = \frac{\mathcal{T}_{(\text{MACS})}}{\mathcal{C}_{(\text{MAC})}} \tag{1.19}$$

- *Classification/Detection Perf.* ($\mathcal{A}$): Another way to reduce CNN execution times is to trade some of their modeling performance in favor of faster execution timings. For this reason, the classification and detection metrics are reported, especially when dealing with *approximate computing* methods. Classification performance is usually reported as top-1 and top-5 accuracies, and detection performance is reported using the mAP50 and mAP75 metrics.
- *Energy and Power Consumption* ($\mathcal{P}$): Numerous FPGA-based acceleration methods can be categorized as either latency-driven or energy-driven. While the former focus on improving the computational throughput, the latter considers the power consumption of the accelerator, reported in watts. Alternatively, numerous latency-driven accelerators can be ported to low-power-range FPGAs and perform well under strict power consumption requirements.
- *Resource Utilization* ($\mathcal{R}$): When it comes to FPGA acceleration, the utilization of the available resources (lut, DSP blocks, sram blocks) is always considered. Note that the resource utilization can be correlated to the power consumption*, but improving the ratio between the two is a technological problem that clearly exceeds the scope of this chapter. For this reason, both power consumption and resources utilization metrics will be reported when available.

An FPGA implementation of a CNN has to satisfy to the former requirements. In this perspective, the literature provides three main approaches to address the problem

* At a similar number of memory accesses. These accesses typically play the most dominant role in the power consumption of an accelerator.

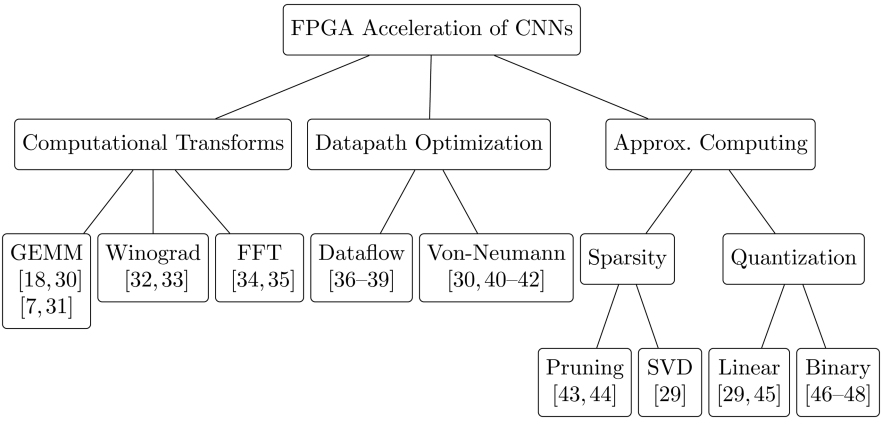

FIGURE 1.2 Main approaches to accelerate CNN inference on FPGAs.

of FPGA-based deep learning. These approaches mainly consists of computational transforms, data-path optimization, and approximate computing techniques, as illustrated in Figure 1.2.

1.4 COMPUTATIONAL TRANSFORMS

In order to accelerate the execution of *conv* and FC layers, numerous implementations rely on computational transforms. These transforms, which operate on the FM and weight arrays, aim at vectorizing the implementations and reducing the number of operations occurring during inference.

Three main transforms can be distinguished. The *im2col* method reshapes the feature and weight arrays in a way to transform depth convolutions into matrix multiplications. The *FFT* method operates on the frequency domain, transforming convolutions into multiplications. Finally, in *Winograd* filtering, convolutions boil down to element-wise matrix multiplications thanks to a tiling and a linear transformation of data.

These computational transforms mainly appear in temporal architectures and are implemented by means of variety of *linear algebra* libraries such OpenBLAS for CPUs* or cuBLAS for GPUs†. Besides this, various implementations make use of these transforms to efficiently map CNNs on FPGAs.

This section discusses the three former methods, highlighting their use-cases and computational improvements. For a better understanding, we recall that for each layer ℓ:

- The input feature map is represented as four-dimensional array $\mathbf{X}$, in which the dimensions $B \times C \times H \times W$ respectively refer to the batch size, the number of input channels, the height, and the width.

* https://www.openblas.net/
† https://developer.nvidia.com/cublas

- The weights are represented as four-dimensional array Θ, in which the dimensions $N \times C \times J \times K$ respectively refer to the depth of the output feature map, the depth of the input feature map, the vertical, and the horizontal kernel size.

1.4.1 The im2col Transformation

In CPUs and GPUs, a common way to process CNNs is to map *conv* and FC layers as general matrix multiplications (GEMMs). A number of studies generalize this approach to FPGA-based implementations.

For FC layers, in which the processing boils down to a matrix-vector multiplication problem, the GEMM-based implementations find their interest when processing a *batch* of FMs. As mentioned in Section 1.2.4.1, most of the weights of CNNs are employed in the FC parts. Instead of loading these weights multiple times to classify multiple inputs, features extracted from a batch of inputs are concatenated onto a $CHW \times B$ matrix. In this case, the weights are loaded only one time per batch, as depicted in Figure 1.3a. As a consequence, the former Equation 1.16 – which expressed the number of memory accesses occurring on FC layers – becomes the following:

$$\mathcal{M}_\ell^{\text{fc}} = \text{MemRd}(\theta_\ell^{\text{fc}}) + \text{MemRd}(\mathbf{X}_\ell^{\text{fc}}) + \text{MemWr}(\mathbf{Y}_\ell^{\text{fc}}) \tag{1.20}$$

$$= N_\ell C_\ell W_\ell H_\ell + B C_\ell H_\ell W_\ell + B N_\ell \tag{1.21}$$

$$\sim N_\ell C_\ell H_\ell W_\ell \tag{1.22}$$

As detailed in Section 1.2.4.2, the vectorization of FC layers is often employed in GPU implementations to increase the computational throughput while maintaining a constant memory bandwidth utilization. The same concept holds true for FPGA implementations [31, 48, 49], which batch the FC layers to map them as GEMMs.

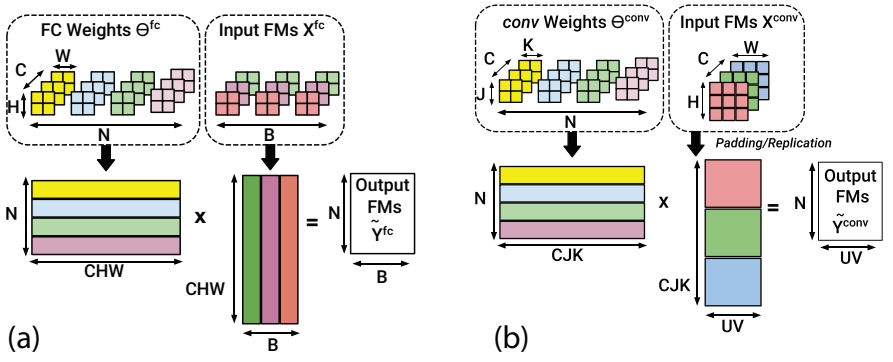

FIGURE 1.3 GEMM-based processing of FC layers (a) and conv layers (b).

3D convolutions can also be mapped as GEMMs using the so-called *im2col* method introduced in [30]. First, this method flattens all the weights of a given *conv* layer onto an $N \times CKJ$ matrix $\tilde{\Theta}$. Second, it rearranges the input feature maps onto a $CKJ \times UV$ matrix $\tilde{X}$, squashing each feature map to a column*. With these reshaped data, the output feature maps $\tilde{Y}$ are computed by multiplying of two former matrices, as illustrated in Figure 1.3b.

$$\tilde{\mathbf{Y}}^{\text{conv}} = \tilde{\Theta}^{\text{conv}} \times \tilde{\mathbf{X}}^{\text{conv}} \tag{1.23}$$

Suda et al. [29] and more recently, Zhang et al. [50] and Guan et al. [51] leverage on *im2col* to derive OpenCL-based FPGA accelerators for CNN. However, this method introduces redundant data in the input FM matrix, which can lead to either inefficiency in storage or complex memory access patterns. As a result, and as pointed out in [22], other strategies to map convolutions have to be considered.

1.4.2 Winograd Transform

Winograd minimal filtering algorithm, introduced in [52], is a computational transform that can be applied to process convolutions with a stride of 1, which is very common in CNN topologies.

This algorithm is particularly efficient when processing small convolutions (where $K \leq 3$), as advocated in [53]. In this work, authors outperformed the throughput of the conventional *im2col* method by a factor of ×7.2 when executing VGG16 on a TitanX GPU.

In Winograd filtering (Figure 1.4), data is processed by blocks, referred to as *tiles*, as follows:

1. An input FM tile x of size $(u \times u)$ is pre-processed: $\tilde{x} = \mathbf{A}^T x \mathbf{A}$
2. In a similar way, θ, the filter tile of size $(k \times k)$, is transformed into $\tilde{\theta}$: $\tilde{\theta} = \mathbf{B}^T x \mathbf{B}$

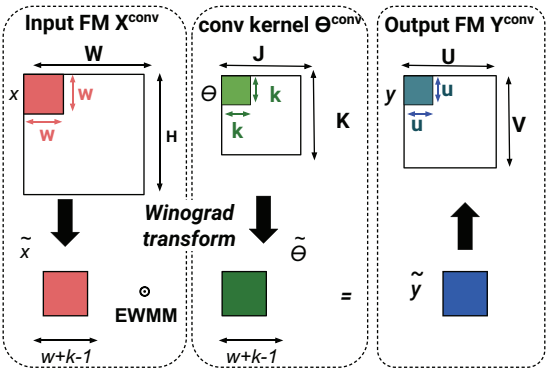

FIGURE 1.4 Winograd filtering $F(u \times u, k \times k)$.

* That's what the im2col name refers to: flattening an image to a column.

3. Winograd filtering algorithm, denoted $F(u\times u, k\times k)$, outputs a tile y of size $(u\times u)$ that is computed according to Equation 1.24

$$y = \boldsymbol{C}^T\left[\tilde{\theta}\odot\tilde{x}\right]\boldsymbol{C} \tag{1.24}$$

where $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$ are transformation matrices defined in the Winograd algorithm [52] and $\odot$ denotes the Hadamard product also known as EWMM.

While a standard filtering requires $u^2 \times k^2$ multiplications, Winograd algorithm, denoted $F(u\times u, k\times k)$, requires $(u+k-1)^2$ multiplications [52]. In the case of tiles of a size $u=2$ and kernels of size $k=3$, this corresponds to an arithmetic complexity reduction of $\times 2.25$ [53], and in this case, transform matrices can be written as follows:

$$\mathbf{A}^T = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & -1 \end{bmatrix};\quad \mathbf{B}^T = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix}$$

$$\boldsymbol{C} = \begin{bmatrix} 1 & 0 & 0 \\ 1/2 & 1/2 & 1/2 \\ 1/2 & -1/2 & 1/2 \\ 0 & 0 & 1 \end{bmatrix} \tag{1.25}$$

Beside this complexity reduction, implementing Winograd filtering in FPGA-based CNN accelerators has two advantages. First, transformation matrices $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$ can be evaluated offline once u and k are determined. As a result, these transforms become multiplications with the constants that can be implemented by means of lut and shift registers, as proposed in [54].

Second, Winograd filtering can employ the loop optimization techniques discussed in Section 1.5.2 to vectorize the implementation. On one hand, the computational throughput is increased when *unrolling* the computation of the ewmm parts over multiple DSP blocks. On the other hand, memory bandwidth is optimized using loop *tiling* to determine the size of the FM tiles and filter buffers.

First, utilization of Winograd filtering in FPGA-based CNN accelerators is investigated in [32] and delivers a computational throughput of 46 GOPs when executing AlexNet convolutional layers. This performance is significantly improved by a factor of $\times 42$ in [31] when optimizing the data path to support Winograd convolutions (by employing loop unrolling and tiling strategies), and storing the intermediate FM in on-chip buffers (cf Section 1.4).

The same method is employed in [54] to derive a CNN accelerator on a Xilinx ZCU102 device that delivers a throughput of 2.94 TOPs on VGG convolutional layers. The reported throughput corresponds to half of the performance of a TitanX device, with $5.7\times$ less power consumption [23]*.

* Implementation in the TitanX GPU employs Winograd algorithm and 32-bit floating point arithmetic.

1.4.3 Fast Fourier Transform

Fast Fourier Transform (FFT) is a well known algorithm to transform the 2D convolutions into ewmm in the frequency domain, as shown in Equation 1.26:

$$\textbf{conv2D}(X[c], \Theta[n,c]) = \text{IFFT}(\text{FFT}(X[c]) \odot \text{FFT}(\Theta[n,c])) \tag{1.26}$$

Using FFT to process 2D convolutions reduces the complexity from $O(W^2 \times K^2)$ to $O(W^2 log_2(W))$, which is exploited to derive FPGA-based accelerators and to infer CNN [34]. When compared to standard filtering and Winograd algorithm, FFT finds its interest in convolutions with large kernel size ($K > 5$), as demonstrated in [53, 55]. The computational complexity of FFT convolutions can be further reduced to $O(Wlog_2(K))$ using the overlap-and-add method [56], which can be applied when the signal size is much larger than the filter size, which is typically the case in *conv* layers ($W >> K$). Works in [33, 57] leverage on the overlap-and-add to implement frequency domain acceleration for *conv* layers on FPGA, which results in a computational throughput of 83 GOPs for AlexNet (Table 1.3).

1.5 DATA-PATH OPTIMIZATIONS

As highlighted in Section 2.4.2, the execution of CNN exhibits numerous sources of parallelism. However, due to the resource limitations of FPGA devices, it might be impossible to fully exploit all the concurrency patterns, especially with the sheer volume of operations involved in deep topologies. In other words, the execution of recent CNN models cannot fully be unrolled sometimes, not even for a single *conv* layer.

To address this problem, the general approach, advocated in state-of-the-art implementations, is to map a limited number of processing elements (PEs) on the FPGA. These PEs are then reused by temporally iterating data through them.

1.5.1 Systolic Arrays

Early FPGA-based accelerators for CNN implemented systolic arrays to accelerate the 2D filtering in convolutions layers [58—61]. As illustrated in Figure 1.5a, systolic arrays employ *a static collection* of PE, typically arranged in a 2-dimensional grid. These PE operate as a co-processor under the control of a central processing unit. The configuration of systolic arrays is *agnostic* to the CNN model, making them inefficient to process large-scale networks for the following three reasons:

First, the static collection of PE can support convolutions only up to a given filter size K_m, where typical values of K_m range from 7 in [59] to 10 in [61]. Therefore, in convolutional layer (ℓ), $K_\ell > K_m$ is not supported by the accelerator. Second, systolic arrays suffer from under-utilization when processing layers in which the kernel size K_ℓ is much smaller then K_m. This is for instance the case in [61], where the processing of 3×3 convolutions uses only 9% of DSP blocks, while the processing of these layers can be further parallelized and thus accelerated. Third and finally, PE in systolic arrays do not usually include memory caches and have to fetch their inputs from

TABLE 1.3
Accelerators Employing Computational Transforms

Method	Entry	Network	Comp (GOP)	Params (M)	Bit-width	Desc.	Device	Freq (MHz)	Through (GOPs)	Power (W)	LUT (K)	DSP	Memory (MB)
Winograd	[33]	AlexNet-C	1.3	2.3	Float 32	OpenCL	Virtex7 VX690T	200	46	–	505	3683	56.3
	[32]	AlexNet-C	1.3	2.3	Float16	OpenCL	Arria10 GX1150	303	1382	44.3	246	1576	49.7
	[55]	VGG16-C	30.7	14.7	Fixed 16	HLS	Zynq ZU9EG	200	3045	23.6	600	2520	32.8
	[55]	AlexNet-C	1.3	2.3	Fixed 16	HLS	Zynq ZU9EG	200	855	23.6	600	2520	32.8
FFT	[34]	AlexNet-C	1.3	2.3	Float 32	–	Stratix5 QPI	200	83	13.2	201	224	4.0
	[34]	VGG19-C	30.6	14.7	Float 32	–	Stratix5 QPI	200	123	13.2	201	224	4.0
GEMM	[30]	AlexNet-C	1.3	2.3	Fixed 16	OpenCL	Stratix5 GXA7	194	66	33.9	228	256	37.9
	[50]	VGG16-F	31.1	138.0	Fixed 16	HLS	Kintex KU060	200	365	25.0	150	1058	14.1
	[50]	VGG16-F	31.1	138.0	Fixed 16	HLS	Virtex7 VX960T	150	354	26.0	351	2833	22.5
	[51]	VGG16-F	31.1	138.0	Fixed 16	OpenCL	Arria10 GX1150	370	866	41.7	437	1320	25.0
	[51]	VGG16-F	31.1	138.0	Float 32	OpenCL	Arria10 GX1150	385	1790	37.5	–	2756	29.0