REPUBLIC OF TURKEY
YILDIZ TECHNICAL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

DESIGN OF A MULTILAYER CELLULAR NEURAL NETWORK EMULATOR AND ITS IMPLEMENTATION ON AN FPGA DEVICE

MURATHAN ALPAY

Ph.D. THESIS

DEPARTMENT OF ELECTRONICS AND COMMUNICATIONS ENGINEERING
PROGRAM OF ELECTRONICS

ADVISER
PROF. DR. VEDAT TAVŞANOĞLU

İSTANBUL, 2013

REPUBLIC OF TURKEY

YILDIZ TECHNICAL UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

DESIGN OF A MULTILAYER CELLULAR NEURAL NETWORK EMULATOR AND ITS IMPLEMENTATION ON AN FPGA DEVICE

A thesis submitted by Murathan ALPAY in partial fulfillment of the requirements for the degree of **DOCTOR OF PHILOSOPHY** is approved by the committee on 28.05.2013 in Department of Electronics and Communications Engineering, Electronics Program.

Thesis Adviser
Prof. Dr. Vedat TAVŞANOĞLU
Yıldız Technical University

Approved by the Examining Committee
Prof. Dr. Vedat TAVŞANOĞLU
Yıldız Technical University _____

Prof. Dr. Tülay YILDIRIM
Yıldız Technical University _____

Assoc. Prof. Dr. Sıddıka Berna Örs YALÇIN
İstanbul Technical University _____

Prof. Dr. Oruç BİLGİÇ
İstanbul Kültür University _____

Assist. Prof. Dr. Umut Engin AYTEN
Yıldız Technical University _____

This study was supported by The Scientific and Technological Research Council of Turkey (TÜBİTAK) under project number 108E023.

ACKNOWLEDGMENTS

Although this is one of the most important steps of my life, I don't feel like a very special or a very important person, but I do feel very happy and relieved. I think achieving the title Dr. is important but it is not the titles, positions or jobs that make a person special and important, it is the actions and behaviours towards other people and the support and care given to them by that person.

I thought for a long time about how I could mention all the people who supported me during the preparation of this thesis but I guess it is neither possible nor practical to mention all of them name by name so I want to start with the most important and special ones and apologize anyone whom I may have forgotten. Firstly, I want to thank my professor and instructor Dr. Vedat Tavşanoğlu for his valuable contributions and guidance in my both academic and personal life. He is not only a teacher but also a father who has given me support during my hard times for me. Secondly, I want to thank my valuable colleague and friend since undergraduate years, Dr. Nerhun Yıldız for his academic help, moral support and for his help during the preparation of this thesis. Third, I want to thank my special friend and collegue, Dr. Evren Cesur, whom I met during my graduate years, for again his academic help, moral support and for his help during the preparation of this thesis. Fourth, I want to thank my special friend and colleague, Dr. Umut Engin Ayten, who was my laboratory instructor during the undergraduate years, for the academic help and moral support he has given. Also, I want to thank all my teachers from primary school to PhD for teaching and guiding me. I want to continue with Işıl Kalafat, Oğuzhan Yavuz, Dr. Nergis Tural–Polat and all my other collegues who gave me support. Then, I want to thank both my mother and father for raising me and giving me moral and health support when I needed the most. After that, I want to thank my family for their support. Then, I also want to thank one of my special friends, Hakan Saraçoğlu for his moral support. Last but not the least I want to thank all my friends and acquaintances.

May, 2013

Murathan ALPAY

CONTENTS

LIST OF SYMBOLS

q	q^{th} layer of a MLCNN (from layer variable)
p	p^{th} layer of a MLCNN (to layer variable)
Q	number of dimensions of a CNN or MLCNN
P	number of layers of a MLCNN
M	neighborhood of the spatial interconnections of a 2–D CNN or MLCNN
R	layer–neighbourhood of a specialized MLCNN
I	number of cells in the first spatial dimension of a 2–D CNN or MLCNN
J	number of cells in the second spatial dimension of a 2–D CNN or MLCNN
i	space variable of the first spatial dimension of a 2–D CNN or MLCNN
j	space variable of the second spatial dimension of a 2–D CNN or MLCNN
$C(i,j)$	CNN or MLCNN cell
t	time variable
$x_{ij}(t)$	CNN cell state of a $C(i,j)$ cell at time t
$x_{ij,p}(t)$	MLCNN cell state of a $C(i,j)$ cell at time t
$\dot{x}_{ij}(t)$	time derivative of $x_{ij}(t)$
$\dot{x}_{ij,p}(t)$	time derivative of $x_{ij,p}(t)$
$y_{ij}(t)$	output of a $C(i,j)$ CNN cell at time t (output pixel in case of an image)
$y_{ij,p}(t)$	output of a $C(i,j)$ MLCNN cell at time t (output pixel in case of an image)
u_{ij}	constant–valued input of a $C(i,j)$ CNN cell (input pixel in case of an image)
$u_{ij,p}$	constant–valued input of a $C(i,j)$ MLCNN cell (input pixel in case of an image)
k	space variable of the first spatial dimension of a CNN or MLCNN template
l	space variable of the second spatial dimension of a CNN or MLCNN template
a_{kl}	constant–valued feedback coefficients of a CNN template
$a_{kl,qp}$	constant–valued feedback coefficients of a MLCNN template
b_{kl}	constant–valued input coefficients of a CNN template
$b_{kl,qp}$	constant–valued input coefficients of a MLCNN template
z	constant bias value of a CNN cell
z_p	constant bias value of a MLCNN cell
$f(.)$	output function
$f_L(.)$	logical output function
$\boldsymbol{A}$	feedback template of a CNN
$\boldsymbol{A}_{qp}$	feedback template of a MLCNN
$\boldsymbol{B}$	input template of a CNN
$\boldsymbol{B}_{qp}$	input template of a MLCNN

$\circledast$	template–dot–product operator
$\boldsymbol{Y}_{ij}(t)$	translated masked output image of a CNN
$\boldsymbol{Y}_{ij,q}(t)$	translated masked output image of a MLCNN
$\boldsymbol{U}_{ij}$	translated masked input image of a CNN
$\boldsymbol{U}_{ij,q}$	translated masked input image of a MLCNN
n	discrete–time variable
$x_{ij}(n)$	discrete–time state of a $C(i,j)$ CNN cell
$x_{ij,p}(n)$	discrete–time state of a $C(i,j)$ MLCNN cell
$y_{ij}(n)$	discrete–time output of a $C(i,j)$ CNN cell
$y_{ij,p}(n)$	discrete–time output of a $C(i,j)$ MLCNN cell
$\boldsymbol{Y}_{ij}(n)$	translated masked output image of discrete–time CNN
$\boldsymbol{Y}_{ij,q}(n)$	translated masked output image of discrete–time MLCNN
T_s	sampling period
$\bar{\boldsymbol{A}}$	discrete–time $\boldsymbol{A}$–template of a discrete–time CNN
$\bar{\boldsymbol{A}}_{qp}$	discrete–time $\boldsymbol{A}$–template of a discrete–time MLCNN
$\bar{\boldsymbol{B}}$	discrete–time $\boldsymbol{B}$–template of a discrete–time CNN
$\bar{\boldsymbol{B}}_{qp}$	discrete–time $\boldsymbol{B}$–template of a discrete–time MLCNN
$\bar{a}_{kl}$	discrete–time constant–valued feedback coefficients of a discrete–time CNN template
$\bar{a}_{kl,qp}$	discrete–time constant–valued feedback coefficients of a discrete–time MLCNN template
$\bar{b}_{kl}$	discrete–time constant–valued input coefficients of a discrete–time CNN template
$\bar{b}_{kl,qp}$	discrete–time constant–valued input coefficients of a discrete–time MLCNN template
$\bar{z}$	discrete–time constant bias value of a discrete–time CNN cell
$\bar{z_p}$	discrete–time constant bias value of a discrete–time MLCNN cell
N	number of iterations
g_{ij}	intermediate constant of a $C(i,j)$ discrete–time CNN cell
$g_{ij,qp}$	intermediate constant of a $C(i,j)$ discrete–time MLCNN cell
$\boldsymbol{I}_{ij,p}$	intermediate constant of a $C(i,j)$ discrete–time MLCNN cell
$h_{ij,p}(n)$	intermediate variable of a $C(i,j)$ discrete–time MLCNN cell
$l_{ij,p}(n)$	intermediate variable of a $C(i,j)$ discrete–time MLCNN cell
$h_{ij,qp}(n)$	intermediate variable of a $C(i,j)$ discrete–time MLCNN cell
a	number of extra $\boldsymbol{A}$–templates
b	number of extra $\boldsymbol{B}$–templates
tbw	total bit width parameter of inputs
$tbwa$	total bit width parameter of an APU
$tbwc$	total bit width parameter of a constant input
$tbwd$	total bit width parameter of a data input
$tbwt$	total bit width parameter of an intermediate signal
$frac$	bit width parameter of the fractional part of a fixed point number
int	bit width parameter of the integer part of a fixed point number
$\lfloor . \rfloor$	floor function

LIST OF ABBREVIATIONS

1–D	One–Dimensional
2–D	Two–Dimensional
APU	A Processing Unit
APU2L	APU for Two–layer DT CNN
APUML	APU for Multi–layer DT CNN
APU2ML	Two–input APU for Multi–layer DT CNN
APU3ML	Three–input APU for Multi–layer DT CNN
ASIC	Application Specific Integrated Circuit
BPU	B Processing Unit
BPUML	BPU for Multi–layer DT CNN
BW	Black and White
CMOS	Complementary Metal–oxide–semiconductor
CNN	Cellular Neural Network
CNN–UM	CNN Universal Machine
CT CNN	Continuous–Time CNN
DSP	Digital Signal Processors
DT CNN	Discrete–Time CNN
DVI	Digital Visual Interface
Full–HD	1920×1080 resolution at 60 Hz frame rate
FPGA	Field–Programmable Gate Array
FSR	Full Signal Range
GPU	Graphical Processing Unit
GTF	Gabor–type Filter
I/O	Input/Output
LAM	Local Analog Memory of CACE1K CNN–UM Chip
LLM	Local Logic Memory of CACE1K CNN–UM Chip
LLU	Local Logic Unit of CACE1K CNN–UM Chip
MHz	Mega Hertz
MLCNN	Multi–layer CNN
MP/s	Megapixels per second
PC	Personal Computer
QVGA	Quarter Video Graphics Array
RAM	Random Access Memory
RD–CNN	Reaction–Diffusion CNN
RGB	Red Green Blue
RTCNNP	Real–Time CNN Processor
RTL	Register Transfer Level

TÜBİTAK	The Scientific and Technological Research Council of Turkey
VHDL	Very–high–speed integrated-circuit (VHSIC) Hardware Description Language
VLSI	Very–large–scale Integration
VPU2L	Video Processing Unit for Two–layer DT CNN
xPU	x Processing Unit

LIST OF FIGURES

LIST OF TABLES

Page

ABSTRACT

DESIGN OF A MULTILAYER CELLULAR NEURAL NETWORK EMULATOR AND ITS IMPLEMENTATION ON AN FPGA DEVICE

Murathan ALPAY

Department of Electronics and Communications Engineering

Ph.D. Thesis

Adviser: Prof. Dr. Vedat TAVŞANOĞLU

Electronics engineering, which was once a subbranch of electrical engineering, became a main branch and separated into many subbranches in the near past. Image processing is one of these subbranches which is also an interdisciplinary engineering topic. Types of image processing are also increased and developed due to the technological advances. Nowadays there are many analog and digital systems, which are capable of realizing various image processing applications. One of these systems is called a cellular neural netwok (CNN), which was put forward as a structure of cells where the output of each cell is computed by using the inputs and outputs of the neighbouring cells. The CNN structure is used in not only image processing tasks but also other types of applications, such as chaotic systems and solving partial differential equations which are beyond the scope of this thesis.

The CNN was firstly defined as a continous–time and discrete–space cell–grid. A CNN structure can be used in image processing by corresponding each input and output of an image processing system to the input and output of each cell. The 2–D grid of a CNN can either be implemented as an analog application specific integrated circuit (ASIC), or emulated on a digital hardware. The maximum resolution of the analog and digital implementations to date are 176×144 and 640×480, respectively. There are higher resolutions reported for the digital emulations, however, the frame rate of the emulation drops drastically. A recently introduced CNN emulator called a second generation Real–Time Cellular Neural Network Processor (RTCNNP–v2) is reported to be capable of processing full–HD 1080p@60 (1920×1080 resolution, 60 Hz frame rate) images in real–time.

A single–layer CNN structure is generally used in the early CNN applications which is capable of realizing many image processing applications. However, it is either difficult

or impossible to realize some applications with a single–layer CNN structure, e.g., algorithms that require more sensitivity, precision, carrying out computations with complex numbers, modelling systems with multi–order partial differential equations, etc. It is reported that these requirements can be met by the introduction of two– and multi–layer CNN structures.

In this thesis, the design and implementation stages of the architecture of a generalized two–layer DT CNN emulator and a specialized multi–layer DT CNN emulator are proposed based on the RTCNNP–v2 structure, which are capable of processing full–HD 1080p@60 images in real–time. While a two–layer DT CNN emulator design is capable of realizing all the templates in the general mathematical model, a new measure of layer neigbourhood is defined for the multi–layer DT CNN emulation. This measure limits the intra–layer connections to only neighbouring layers. Furthermore, the proposed measure is extended to support far–neigbourhood interconnections. A two–layer RTCNNP architecture is implemented on an Altera Stratix IV GX 230 FPGA device, with a maximum number of 24 Euler iterations. The visible pixel rate and the pixel clock frequency of the prototype is 124.4 MP/s and 148.5 MHz, respectively, while the throughput is 124.4 MP/s.

Keywords: Multi–layer cellular neural networks, image processing, field–programmable gate–arrays, real–time systems

ÖZET

ÇOK KATMANLI BİR HÜCRESEL SİNİR AĞI EMÜLATÖRÜNÜN TASARLANMASI VE FPGA ÜZERİNDE GERÇEKLENMESİ

Murathan ALPAY

Elektronik ve Haberleşme Mühendisliği Anabilim Dalı

Doktora Tezi

Tez danışmanı: Prof. Dr. Vedat TAVŞANOĞLU

Elektronik mühendisliği eskiden elektrik mühendisliğinin bir alt dalı iken yakın geçmişte artık anadal haline gelmiş ve birçok alt dala ayrılmıştır. Bu alt dallardan birisi de birden fazla mühendislik alanı içine giren görüntü işleme olup görüntü işleme türleri de teknolojinin gelişimine bağlı olarak artmış ve gelişmiştir. Günümüzde çeşitli görüntü işleme uygulamaları gerçekleyebilen analog ve dijital birçok sistem bulunmaktadır. Bu sistemlerden bir tanesi de her bir hücrenin çıkışının komşuluğundaki hücrelerin giriş ve çıkışlarının kullanılarak hesaplandığı bir yapı olarak ortaya atılan hücresel sinir ağıdır (HSA). HSA yapısı sadece görüntü işleme değil örneğin kaotik sistemler ve kısmi türevli diferansiyel denklemlerin çözülmesi gibi uygulamalarda da kullanılmaktadır fakat görüntü işleme dışındaki uygulamalar bu tezin kapsamı dışındadır.

HSA, ilk olarak sürekli zamanlı ve uzayda ayrık bir hücreler topluluğu olarak tanımlanmıştır. HSA yapısının görüntü işlemede kullanılması amacıyla her bir piksele ait giriş ve çıkış her bir hücrenin giriş ve çıkışına karşı düşürülür. İki boyutlu HSA hücreleri topluluğu bir uygulamaya özel tümdevre (UÖTD) olarak gerçekleştirilebilir ya da dijital bir donanımla emülasyonu yapılabilir. Günümüze dek analog gerçekleştirmelerde azami çözünürlük 176×144 iken dijital gerçekleştirmelerde ise 640×480 olarak bildirilmiştir. Dijital emülasyonlar için daha yüksek çözünürlükler bildirilmiş olmasına rağmen bu emülasyonların çerçeve hızları önemli ölçüde düşmektedir. Son zamanlarda ortaya atılan ve ikinci nesil gerçek zamanlı hücresel sinir ağı işlemcisi (GZHSAİ–v2) adı verilen bir HSA emülatörü ise yüksek çözünürlüklü full–HD 1080p@60 (1920×1080 çözünürlük, 60 Hz çerçeve hızı) görüntüleri işleyebilmektedir.

HSA yapısı ilk uygulamalarda tek katmanlı olarak kullanılmıştır. Bu haliyle birçok görüntü işleme uygulamasını gerçekleyebiliyorken hassasiyet ve duyarlılık isteyen uygulamalar,

kompleks sayılarla yapılan işlemler ve yüksek mertebeden kısmi türevli diferansiyel denklemlere sahip sistemlerin modellenmesi gibi algoritmaların tek katmanlı HSA yapısı ile gerçeklenmesi zor ya da imkansız olmaktadır.

Bu tezde full–HD 1080p@60 video görüntülerini işleyebilen ayrık zamanlı iki ve çok katmanlı HSA mimarisi tasarımları ve gerçeklenmeleri önerilmiştir. Bu tasarımlar iki ve çok katmanlı GZHSAİ olarak adlandırılmış olup GZHSAİ–v2 yapısına dayanmaktadır. İki katmanlı yapı genel matematiksel modeldeki şablonların tümünü gerçekleyebiliyorken, çok katmanlı yapı için ise yeni bir katman komşuluğu kavramı tanımlanmıştır. Bu kavrama göre katmanlar arası bağlantılar sadece belirli bir komşuluk mesafesiyle sınırlanmıştır. İki katmanlı bir GZHSAİ mimarisi Altera Stratix IV GX 230 modeli bir alanda programlanabilir kapı dizisi (FPGA) üzerinde gerçeklenmiştir. Tasarlanan prototip bu FPGA üzerinde 24 Euler iterasyonu yapabilmektedir. Görünen piksel hızı 124.4 MP/s, piksel saat frekansı 148.5 MHz, veri çıkış hızı ise 124.4 MP/s'dir.

Anahtar Kelimeler: Çok katmanlı hücresel sinir ağları, görüntü işleme, alanda programlanabilir kapı dizileri, gerçek zamanlı sistemler

CHAPTER 1

INTRODUCTION

1.1 Literature Review

A Cellular Neural Network (CNN) is a cell–based paradigm [1] capable of modelling any system that can be divided into its smallest integral parts which are called cells. It has many applications such as image and video processing, artificial vision, biomedical systems, chaotic systems, circuit and network design, etc. A Q–dimensional P–layer CNN structure consist of a Q–dimensional (Q–D) spatial grid of neural cells and each cell contains P memory nodes. The templates of such a structure are obtained by defining local spatial interconnections between the neural cells. These templates are used for tuning the spatio-temporal dynamics of the system for specific tasks. Generally, a 2–D single–layer CNN structure with space invariant templates [2] is used in image processing applications. However the presentation of multi–layer CNN implementations including some multi–layer CNN emulators enhanced the quality of these applications and enabled the computation of complex image processing tasks in the last decade. In the multi–layer CNN case, the templates are responsible for not only intra–layer interconnections but also inter–layer interconnections between the neural cells.

A multi–layer continuous–time CNN (CT CNN) implementation [3, 4, 5] has the same advantages, which are also stated in [6], as the single–layer CT CNN: it is fully parallel by its nature, its convergence rate is considerably faster then that of a digital implementation, it is easier to merge the architecture with an imaging sensor and obtain a focal plane processor to directly process the captured data as a pre–processor or artificial retina, etc. However, the highest implemented number of cells in a multi–layer CT CNN application

specific integrated circuit (ASIC) is 32×64 [7], the highest implemented number of cells in a multi–layer CT CNN chip is 32×32 [3] and it is also reported that the Eye-RIS chip [8], which has 176×144 cells, is capable of conceptual multi–layer emulation, to date. Therefore, even a low–resolution input comparable to QVGA (320×240) may only be processed by tiling which divides the input into smaller parts called tiles. Tiling causes significant reduction in the processing speed of the system. Consequently, I/O bandwidth limit of a multi–layer CT CNN processor makes it impossible to process a video stream like full–HD 1080p@60 (1920×1080 resolution at 60 Hz frame rate) in real–time [6].

Difference equations of a multi–layer Discrete–Time CNN (DT CNN) are obtained by discretization of state equations of a multi–layer CT CNN. The discretization has sampling and approximation stages. Then, as also stated in [6], the difference equations may be solved on a software unit as a PC, DSP or GPU, or they may be solved on a hardware unit which can be implemented as an ASIC or on an FPGA device. Software solutions are easier to design and modify while hardware implementations provide higher performance.

Using an FPGA device for a multi–layer DT CNN implementation is also preferable in most cases as it is for a single–layer DT CNN implementation [6]: an FPGA device has very flexible parallel structures, faster then software implementations and cheaper than ASIC solutions. Consequently, the most notable multi–layer DT CNN implementation is the Falcon [9], based on the Castle architecture [10], which is implemented on an FPGA device. The implementation reported in [11] is classified as a multi–layer CNN structure, however it is in fact a group of independent CNNs connected end to end, which is beyond the scope of this thesis. In [12], a two–layer application specific DT CNN implementation for a real–time autowave generation is given. An application specific DT CNN implementation is reported in [13] as a specialized multi–layer structure, however, it is not capable of processing a video stream like full–HD 1080p@60 (1920×1080 resolution at 60 Hz frame rate) in real–time. Alternative FPGA architectures of DT CNN were proposed in [14, 15, 16], which are named as Real–Time CNN Processors (RTCNNPs) which have two generations: a first generation which is called RTCNNP–v1 [14] and a second–generation RTCNNP which is called RTCNNP–v2 [15, 16]. The architectures

proposed in this thesis are called the two– and multi–layer RTCNNP, as they are based on the RTCNNP–v2. It is crucial to point out that there are more application–specific two– and multi–layer CNN implementations than [7, 13, 17, 12], however these are beyond the scope of this thesis since they are designed to be application specific.

Finally, this research was supported by The Scientific and Technological Research Council of Turkey (TÜBİTAK) under project number 108E023, and a total number of four PhD theses are introduced at the end of the project. The first thesis [18] is the foundation of the others, including this one, in which first generation of a Real–Time CNN Processor (RTCNNP–v1) was introduced. In the second one [6], a second generation RTCNNP (RTCNNP–v2) is designed and implemented. In the third thesis [19], which is also based on the second one, the CNN-based Gabor–type filter (GTF) implementation is given. Finally, in this thesis, extending the architecture proposed in the second thesis to support two– or multi–layer CNN structures is proposed.

1.2 Objective of The Thesis

Aim of this work is to design a real–time generalized two–layer and a specialized multi–layer DT CNN implementation supporting not only higher frame–rates, but also higher resolutions, including full–HD 1080p@60. Consequently, it will be possible to use algorithms which can be realized with two– and multi–layer CNN structures can be used in image processing applications of modern systems.

1.3 The Original Contribution

As stated in [6], the most original contribution of the RTCNNP–v2 [15] is the introduction of a local control structure for the pipelined CNN emulator arrays, hence making the new design almost infinitely flexible, reconfigurable and reusable as opposed to the RTCNNP–v1 [14]. To this end, this thesis makes use of the flexibility, reconfigurability and reusability of the RTCNNP–v2 architecture which is the basis of the architecture proposed in this thesis.

The most original contribution of this thesis is the extension of the RTCNNP–v2 archi-

tecture to two– and multi–layer CNN by proposing a new topology for the processors of RTCNNP–v2. The topology is designed by dividing the computation not only in the time domain but also in the layer domain. Furthermore, the full–pipelining of RTCNNP–v2 is preserved with the proposed topology. Moreover, a measure of layer neigbourhood is defined, which limits the number of the inter–layer connections. The last but not the least, the number of layers in a multi–layer CNN implementation is designed to be reconfigurable which also makes the architecture flexible.

CHAPTER 2

THE SINGLE–LAYER AND MULTI–LAYER CELLULAR NEURAL NETWORK STRUCTURES

In this chapter, first the mathematical model of continuous–time space–invariant single–layer CNNs is revised as given in [6], then the structure of the two– and multi–layer structures are briefly given along with their mathematical models.

2.1 Continuous–Time Space–Invariant Single–Layer CNNs

In the most general case, a single–layer CNN structure is a discrete–space, continous–time Q–dimensional spatial grid of neural cells, each containing P memory nodes, and has space–variant local interconnections. However, mostly 2–dimensional (2–D), M–neighborhood and space–invariant CNN structures are used in image processing applications. A 2–D CNN grid and its local interconnections are given in Figure 2.1, where it is assumed that only the nearest neighbors are connected with each other, which is called a *one–neighborhood CNN*.

The Chua–Yang CNN model [1] of an M–neighborhood single–layer space–invariant continuous–time CNN with $I \times J$ rectangular array of $C(i, j)$ cells is completely described in [2] by the cell state and output equation pair

$$\dot{x}_{ij}(t) = -x_{ij}(t) + \sum_{k,l=-M}^{M} \left(a_{kl} y_{i+k\,j+l}(t) + b_{kl} u_{i+k\,j+l}\right) + z, \tag{2.1}$$

$$y_{ij}(t) = f(x_{ij}(t)) = 0.5\left(\left|x_{ij}(t)+1\right| - \left|x_{ij}(t)-1\right|\right), \tag{2.2}$$

where (i, j), $i \in \{1,2,...I\}$, $j \in \{1,2,...J\}$ are the spatial Cartesian coordinates, $x_{ij}(t)$ is the cell state at time t, u_{ij} is the constant–valued cell input, a_{kl} and b_{kl}, $k,l \in \{-M,...0,...M\}$,

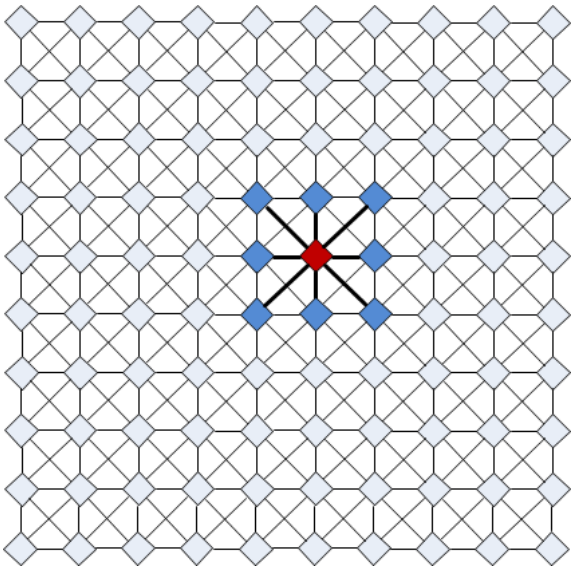


Figure 2.1 A 10×10 spatial grid of a CNN and its one–neighbourhood spatial interconnections

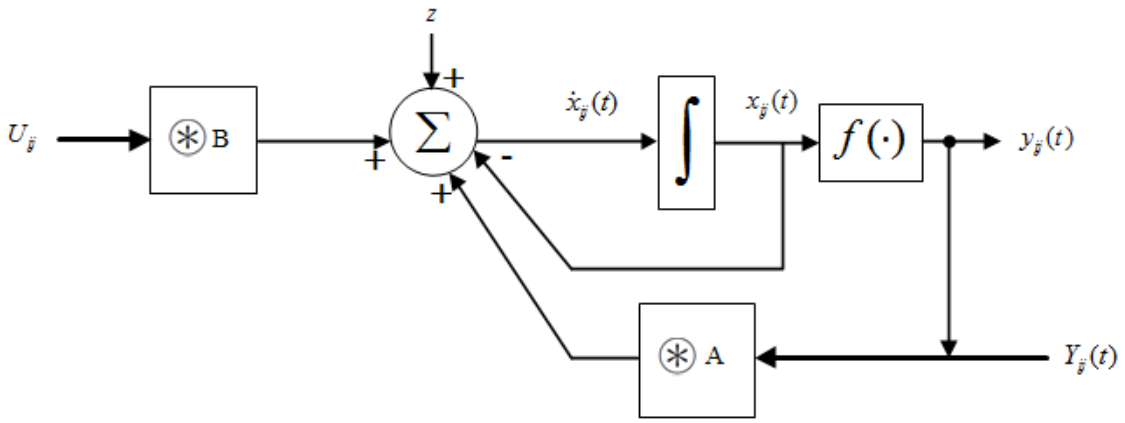


Figure 2.2 Convolutional block diagram of a single–layer CT CNN and its one–neighbourhood spatial interconnections

$M \in \mathbb{N}$ are the constant–valued feedback and input coefficients, respectively, z is the bias value and y_{ij} is the cell output (Fig. 2.2). Eq. (2.1) can be written as

$$\dot{x}_{ij}(t) = -x_{ij}(t) + \boldsymbol{A} \circledast \boldsymbol{Y}_{ij}(t) + \boldsymbol{B} \circledast \boldsymbol{U}_{ij} + z, \tag{2.3}$$

where $\circledast$ is a convolution–like operator called *template–dot–product*, $\boldsymbol{A}$ and $\boldsymbol{B}$ are the feedback and feed–forward templates, $\boldsymbol{Y}_{ij}(t)$ and $\boldsymbol{U}_{ij}$ are the translated masked output

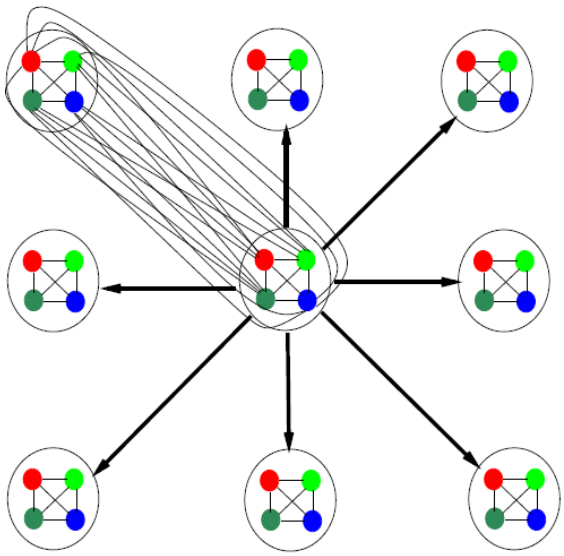


Figure 2.3 A fourth–order single–layer CNN with nine cells [20]

and input, respectively. For $M = 1$

$$\boldsymbol{A} = \begin{bmatrix} a_{-1\,-1} & a_{-1\,0} & a_{-1\,1} \\ a_{0\,-1} & a_{00} & a_{01} \\ a_{1\,-1} & a_{10} & a_{11} \end{bmatrix}, \qquad \boldsymbol{B} = \begin{bmatrix} b_{-1\,-1} & b_{-1\,0} & b_{-1\,1} \\ b_{0\,-1} & b_{00} & b_{01} \\ b_{1\,-1} & b_{10} & b_{11} \end{bmatrix},$$

$$\boldsymbol{Y}_{ij}(t) = \begin{bmatrix} y_{i-1\,j-1}(t) & y_{i-1\,j}(t) & y_{i-1\,j+1}(t) \\ y_{i\,j-1}(t) & y_{ij}(t) & y_{i\,j+1}(t) \\ y_{i+1\,j-1}(t) & y_{i+1\,j}(t) & y_{i+1\,j+1}(t) \end{bmatrix}, \qquad \boldsymbol{U}_{ij} = \begin{bmatrix} u_{i-1\,j-1} & u_{i-1\,j} & u_{i-1\,j+1} \\ u_{i\,j-1} & u_{ij} & u_{i\,j+1} \\ u_{i+1\,j-1} & u_{i+1\,j} & u_{i+1\,j+1} \end{bmatrix}.$$

2.2 Multi–Layer CNNs

MLCNNs are generally used in modelling linear or nonlinear multi–order partial differential equations or systems having more than one state equation, since a P^{th} order single–layer CNN design is much more complex than that of a first order P–layer CNN. For example, a fourth–order single–layer CNN structure with nine cells is given in Figure 2.3 to illustrate the connection irregularity [20]. In this figure, a small circle indicates a state variable, while a group of small circles (a big circle) is a CNN cell. Only connections between two cells are shown to avoid clutter, although all the neigbouring cells should be connected in the same manner. On the other hand, by using a first–order four–layer CNN, the structure in Figure 2.4 is obtained instead of the one given in Figure 2.3. This

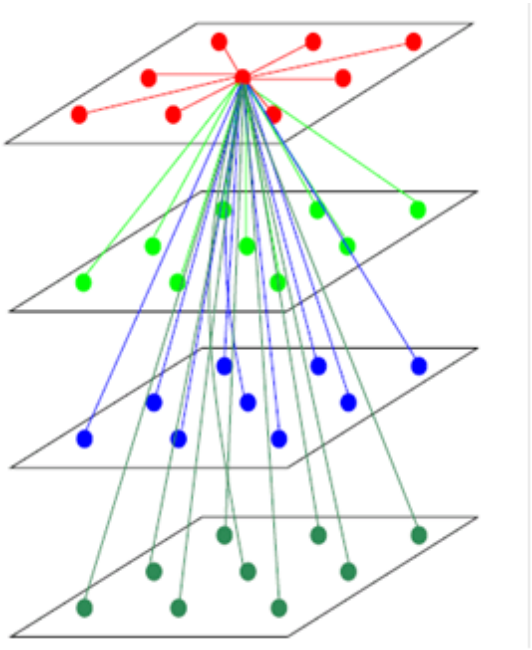


Figure 2.4 A first–order four–layer CNN with nine cells [20]

structure is more regular, where each state–variable is assigned to a different layer [20]. In Figure 2.4, only connections of the cell at the center in the first layer is shown to avoid clutter. Therefore a multi–layer CNN is an approximation of a multi–order single–layer CNN. Furthermore, MLCNNs are also used in spatio–temporal bandpass filtering [7, 21], or to implement improved alternatives of typical CNN applications such as edge or corner detection [20].

A two–layer CNN, which is a special case of a MLCNN, is mainly used for carrying out any kind of computations with complex numbers since a single–layer CNN with complex–valued templates can also be realized with a two–layer CNN with real–valued templates. This is achieved by dividing a computation with complex numbers into its real and imaginary parts and assigning them to the first and second layers, respectively. The CNN–based Gabor–type filter (GTF) [22] can be given as an example in this sense. Furthermore, two–layer CNNs are also used in modelling second–order partial differential equations or systems having two state equations such as RD-CNN [23] and autowave generation [24].

A question arises if the MLCNN structure has a target of reducing the number of iterations needed for a single–layer CNN structure, however, there is no such target and the number of iterations depend on the application type. For example, an autowave gener-

ation application such as [24] require hundreds or thousands of iterations while typical CNN applications such as edge or corner detection require only a few iterations. As a result, it is impractical to compare MLCNN structures with single–layer CNN structures since some algorithms are more suitable for MLCNNs while others are more suitable for single–layer CNNs. Further comparisons are beyond the scope of this thesis.

2.2.1 Continuous–Time Space–Invariant Multi–Layer CNNs

An M–neighborhood space–invariant continuous–time discrete–space single–layer CNN with $K \times L$ rectangular array of $C(i, j)$ cells given in Section 2.1 can be expanded to a MLCNN that is completely described by the cell state and output equation pair

$$\dot{x}_{ij,p}(t) = -x_{ij,p}(t) + \sum_{q=1}^{P} \sum_{k,l=-M}^{M} \left(a_{kl,qp} y_{i+k\,j+l,q}(t) + b_{kl,qp} u_{i+k\,j+l,q} \right) + z_p, \tag{2.4a}$$

$$y_{ij,p}(t) = f(x_{ij,p}(t)) = 0.5(|x_{ij,p}(t) + 1| - |x_{ij,p}(t) - 1|) \tag{2.4b}$$

where (i, j), $i \in \{1, 2, ...I\}$, $j \in \{1, 2, ...J\}$ are the spatial Cartesian coordinates, $x_{ij,p}(t)$ is the cell state of the p^{th} layer at time t, $y_{ij,p}(t)$ is the cell output of the p^{th} layer at time t, z_p is the bias of the p^{th} layer, $u_{ij,p}$ is the constant-valued cell input of the p^{th} layer, and $a_{kl,qp}$ and $b_{kl,qp}$, $k, l \in \{-M, ...0, ...M\}$, $p \in \{1, ...P\}$, $M \in \mathbb{N}$ are the constant–valued feedback and input coefficients, respectively. Eq. (2.4a) can be written as

$$\dot{x}_{ij,p}(t) = -x_{ij,p}(t) + \sum_{q=1}^{P} \boldsymbol{A}_{qp} \circledast \boldsymbol{Y}_{ij,q}(t) + \sum_{q=1}^{P} \boldsymbol{B}_{qp} \circledast \boldsymbol{U}_{ij,q} + z_p \tag{2.5}$$

where $\circledast$ is the *template dot product* operator, $\boldsymbol{A}_{qp}$ and $\boldsymbol{B}_{qp}$ are the feed–back and input templates, and $\boldsymbol{Y}_{ij,q}(t)$ and $\boldsymbol{U}_{ij,q}$ are the translated masked output and input, respectively. When $q = p$, the templates are intra–layer templates, inter–layer otherwise. For $M = 1$:

$$\boldsymbol{A}_{qp} = \begin{bmatrix} a_{-1\,-1,qp} & a_{-1\,0,qp} & a_{-1\,1,qp} \\ a_{0\,-1,qp} & a_{00,qp} & a_{01,qp} \\ a_{1\,-1,qp} & a_{10,qp} & a_{11,qp} \end{bmatrix}, \quad \boldsymbol{B}_{qp} = \begin{bmatrix} b_{-1\,-1,qp} & b_{-1\,0,qp} & b_{-1\,1,qp} \\ b_{0\,-1,qp} & b_{00,qp} & b_{01,qp} \\ b_{1\,-1,qp} & b_{10,qp} & b_{11,qp} \end{bmatrix},$$

$$\boldsymbol{Y}_{ij,q}(t) = \begin{bmatrix} y_{i-1\,j-1,q}(t) & y_{i-1\,j,q}(t) & y_{i-1\,j+1,q}(t) \\ y_{i\,j-1,q}(t) & y_{ij,q}(t) & y_{i\,j+1,q}(t) \\ y_{i+1\,j-1,q}(t) & y_{i+1\,j,q}(t) & y_{i+1\,j+1,q}(t) \end{bmatrix}, \quad \boldsymbol{U}_{ij,q} = \begin{bmatrix} u_{i-1\,j-1,q} & u_{i-1\,j,q} & u_{i-1\,j+1,q} \\ u_{i\,j-1,q} & u_{ij,q} & u_{i\,j+1,q} \\ u_{i+1\,j-1,q} & u_{i+1\,j,q} & u_{i+1\,j+1,q} \end{bmatrix}.$$

A block diagram of this mathematical model is given in Figure 2.5.

2.2.2 Discrete–Time Space–Invariant Multi–Layer CNNs

The mathematical model of a multi–layer DT CNN is obtained by sampling (2.5) in the time domain by

$$\begin{aligned} x_{ij,p}(t)\big]_{t=nT_s} &= x_{ij,p}(nT_s) \triangleq x_{ij,p}(n) \\ \dot{x}_{ij,p}(t)\big]_{t=nT_s} &= \dot{x}_{ij,p}(nT_s) \triangleq \dot{x}_{ij,p}(n) \\ y_{ij,p}(t)\big]_{t=nT_s} &= y_{ij,p}(nT_s) \triangleq y_{ij,p}(n) \\ \boldsymbol{Y}_{ij,q}(t)\big]_{t=nT_s} &= \boldsymbol{Y}_{ij,q}(nT_s) \triangleq \boldsymbol{Y}_{ij,q}(n) \end{aligned} \tag{2.6}$$

and applying Forward–Euler approximation

$$\frac{dx_{ij,p}(n)}{dt} \simeq \frac{x_{ij,p}(n+1) - x_{ij,p}(n)}{T_s}$$

to (2.5) which yields the discrete state equation as

$$x_{ij,p}(n+1) = x_{ij,p}(n) + T_s(-x_{ij,p}(n) + \sum_{q=1}^{P} \boldsymbol{A}_{qp} \circledast \boldsymbol{Y}_{ij,q}(n) + \sum_{q=1}^{P} \boldsymbol{B}_{qp} \circledast \boldsymbol{U}_{ij,q} + z_p) \tag{2.7}$$

The discrete output equation is similarly obtained by sampling (2.4b) in the time domain and is given as

$$y_{ij,p}(n) = f(x_{ij,p}(n)) = 0.5(|x_{ij,p}(n)+1| - |x_{ij,p}(n)-1|). \tag{2.8}$$

A block diagram of this mathematical model is given in Figure 2.6.

2.2.3 Full Signal Range Model of Multi–Layer DT CNNs

Although it is possible to implement (2.7) directly, Full Signal Range (FSR) model [25] of DT CNN is easier to implement, hence designers of most DT CNN implementations

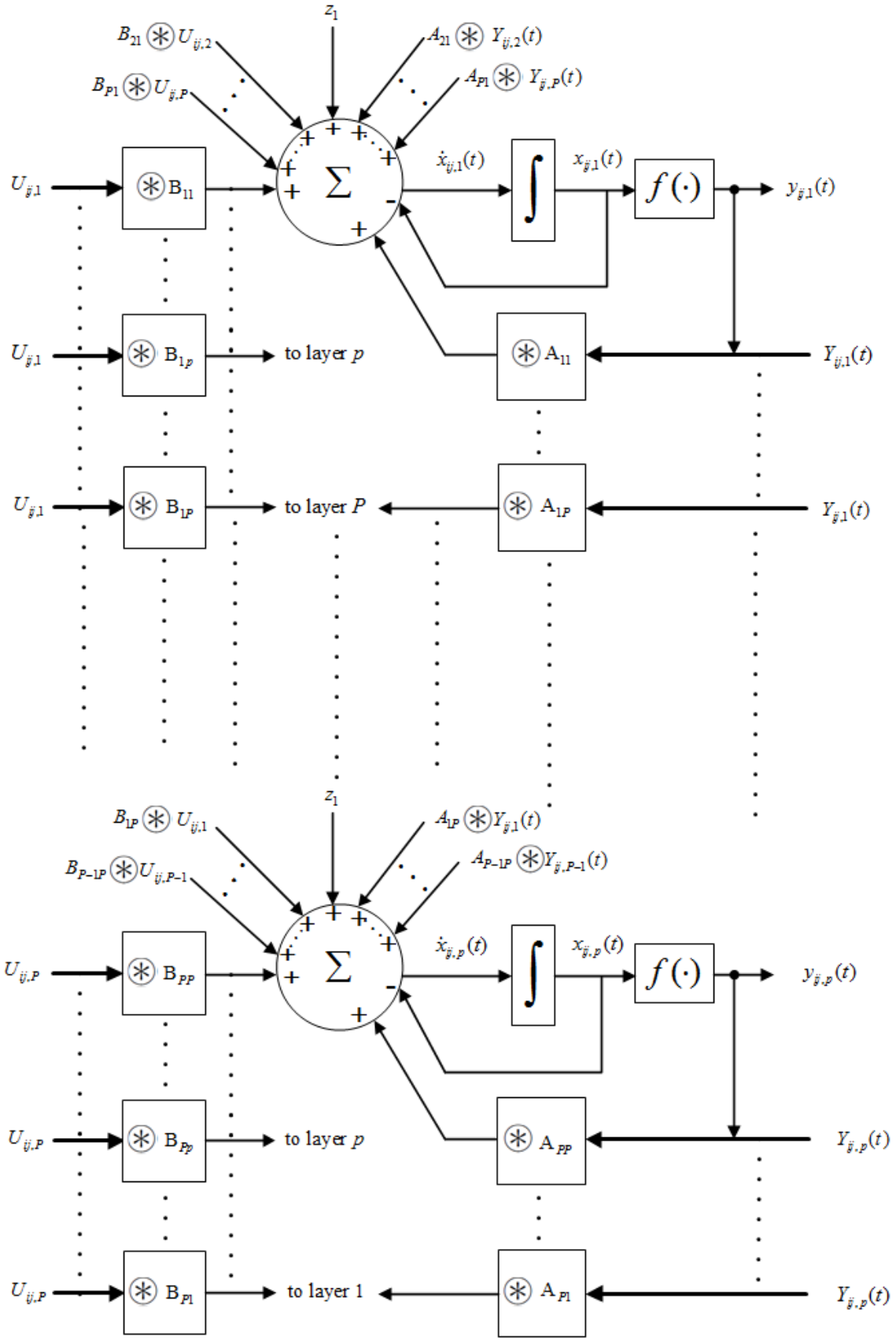


Figure 2.5 Block diagram of a multi–layer CT CNN and its one–neighbourhood spatial interconnections

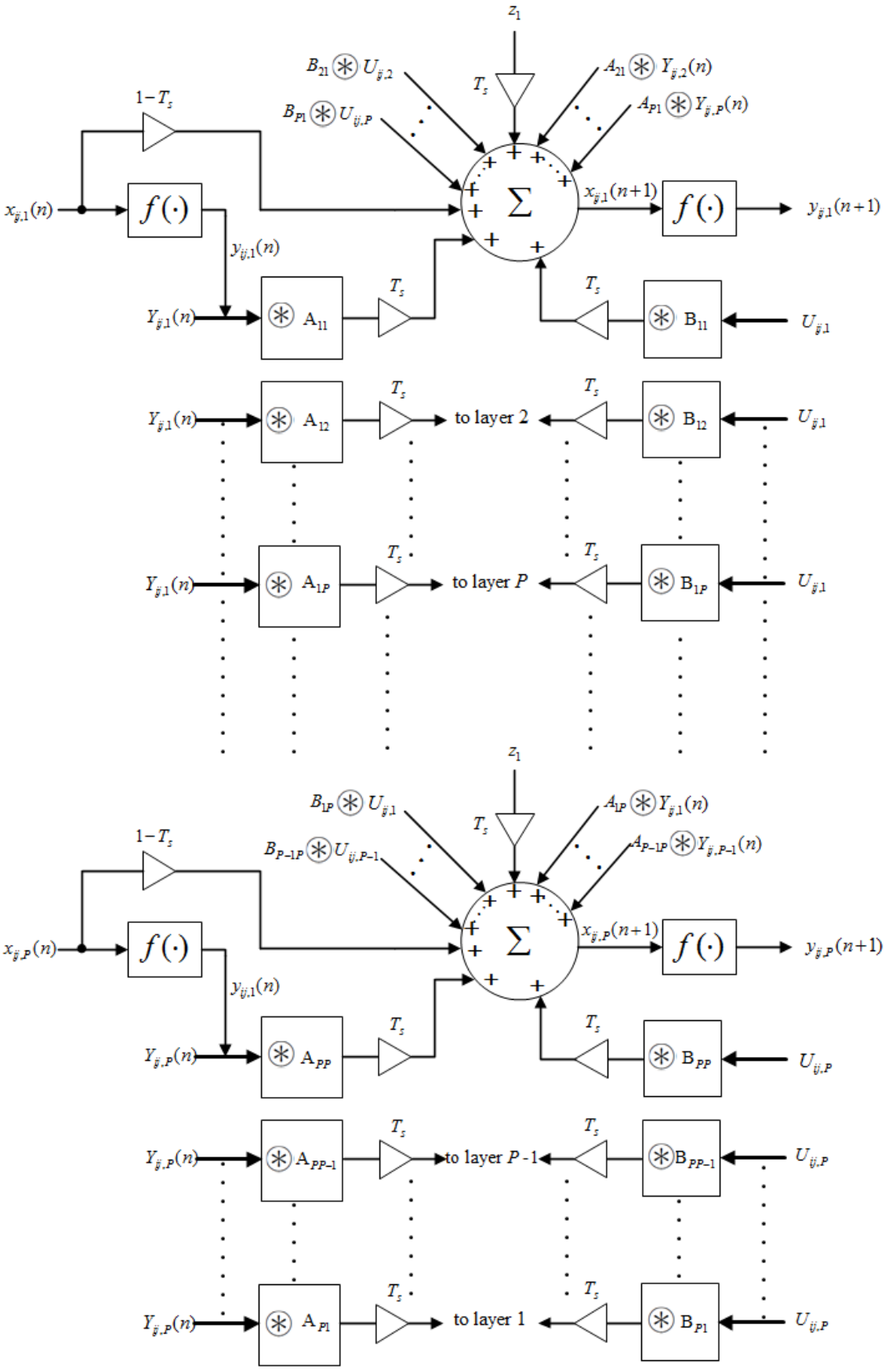


Figure 2.6 Block diagram of a multi–layer DT CNN and its one–neighbourhood spatial interconnections

are inspired by the idea and applied the FSR model to a DT CNN, however the method of obtaining the FSR model of a DT CNN is not clearly described in the literature until the description given in [6]. Therefore, The cell–state equation of the FSR model of a multi–layer DT CNN is then obtained by taking $y_{ij,p}(n) \triangleq x_{ij,p}(n)$ and $y_{ij,p}(n+1) \triangleq f(x_{ij,p}(n+1))$ in (2.7) and (2.8), respectively as

$$x_{ij,p}(n+1) = (1-T_s)y_{ij,p}(n) + T_s \sum_{q=1}^{P} \boldsymbol{A}_{qp} \circledast \boldsymbol{Y}_{ij,q}(n) + T_s \sum_{q=1}^{P} \boldsymbol{B}_{qp} \circledast \boldsymbol{U}_{ij,q} + T_s z_p \tag{2.9}$$

$$y_{ij,p}(n+1) = f(x_{ij,p}(n+1)) = 0.5(|x_{ij,p}(n+1)+1| - |x_{ij,p}(n+1)-1|). \tag{2.10}$$

which can be written as

$$x_{ij,p}(n+1) = \sum_{q=1}^{P} \bar{\boldsymbol{A}}_{qp} \circledast \boldsymbol{Y}_{ij,q}(n) + \sum_{q=1}^{P} \bar{\boldsymbol{B}}_{qp} \circledast \boldsymbol{U}_{ij,q} + \bar{z}_p \tag{2.11}$$

$$y_{ij,p}(n+1) = f(x_{ij,p}(n+1)) = 0.5(|x_{ij,p}(n+1)+1| - |x_{ij,p}(n+1)-1|). \tag{2.12}$$

where new template coefficients and thresholds are defined by:

$$\bar{a}_{kl,qp} = \begin{cases} (1-T_s) + T_s a_{kl,qp} & k,l = 0 \text{ and } q = p, \\ T_s a_{kl,qp} & \text{otherwise,} \end{cases}$$

$$\bar{b}_{kl,qp} = T_s b_{kl,qp}, \qquad \bar{z_p} = T_s z_p.$$

Using (2.11) and (2.12), output equation of the FSR model of a multi–layer DT CNN can be written as

$$y_{ij,p}(n+1) = f\big(\sum_{q=1}^{P} \bar{\boldsymbol{A}}_{qp} \circledast \boldsymbol{Y}_{ij,q}(n) + \sum_{q=1}^{P} \bar{\boldsymbol{B}}_{qp} \circledast \boldsymbol{U}_{ij,q} + \bar{z}_p \big). \tag{2.13}$$

For $P = 1$, output equation of the FSR model of a single–layer DT CNN can be obtained as

$$y_{ij}(n+1) = f\big(\bar{\boldsymbol{A}} \circledast \boldsymbol{Y}_{ij}(n) + \bar{\boldsymbol{B}} \circledast \boldsymbol{U}_{ij} + \bar{z}\big). \tag{2.14}$$

In order to show two examples of this equation in open form (2.15) and (2.16) are obtained

by taking $P = 2$ and $P = 6$ in (2.13), respectively.

$$y_{ij,1}(n+1) = f\left(\bar{\boldsymbol{A}}_{11} \circledast \boldsymbol{Y}_{ij,1}(n) + \bar{\boldsymbol{A}}_{21} \circledast \boldsymbol{Y}_{ij,2}(n) + \bar{\boldsymbol{B}}_{11} \circledast \boldsymbol{U}_{ij,1} + \bar{\boldsymbol{B}}_{21} \circledast \boldsymbol{U}_{ij,2} + \bar{z}_{ij,1}\right), \tag{2.15a}$$

$$y_{ij,2}(n+1) = f\left(\bar{\boldsymbol{A}}_{22} \circledast \boldsymbol{Y}_{ij,2}(n) + \bar{\boldsymbol{A}}_{12} \circledast \boldsymbol{Y}_{ij,1}(n) + \bar{\boldsymbol{B}}_{22} \circledast \boldsymbol{U}_{ij,2} + \bar{\boldsymbol{B}}_{12} \circledast \boldsymbol{U}_{ij,1} + \bar{z}_{ij,2}\right). \tag{2.15b}$$

$$\begin{aligned} y_{ij,1}(n+1) = f\big(&\bar{\boldsymbol{A}}_{11} \circledast \boldsymbol{Y}_{ij,1}(n) + \bar{\boldsymbol{A}}_{21} \circledast \boldsymbol{Y}_{ij,2}(n) + \bar{\boldsymbol{A}}_{31} \circledast \boldsymbol{Y}_{ij,3}(n) + \bar{\boldsymbol{A}}_{41} \circledast \boldsymbol{Y}_{ij,4}(n) \\ &+ \bar{\boldsymbol{A}}_{51} \circledast \boldsymbol{Y}_{ij,5}(n) + \bar{\boldsymbol{A}}_{61} \circledast \boldsymbol{Y}_{ij,6}(n) + \bar{\boldsymbol{B}}_{11} \circledast \boldsymbol{U}_{ij,1} + \bar{\boldsymbol{B}}_{21} \circledast \boldsymbol{U}_{ij,2} \\ &+ \bar{\boldsymbol{B}}_{31} \circledast \boldsymbol{U}_{ij,3} + \bar{\boldsymbol{B}}_{41} \circledast \boldsymbol{U}_{ij,4} + \bar{\boldsymbol{B}}_{51} \circledast \boldsymbol{U}_{ij,5} + \bar{\boldsymbol{B}}_{61} \circledast \boldsymbol{U}_{ij,6} + \bar{z}_{ij,1}\big), \end{aligned} \tag{2.16a}$$

$$\begin{aligned} y_{ij,2}(n+1) = f\big(&\bar{\boldsymbol{A}}_{12} \circledast \boldsymbol{Y}_{ij,1}(n) + \bar{\boldsymbol{A}}_{22} \circledast \boldsymbol{Y}_{ij,2}(n) + \bar{\boldsymbol{A}}_{32} \circledast \boldsymbol{Y}_{ij,3}(n) + \bar{\boldsymbol{A}}_{42} \circledast \boldsymbol{Y}_{ij,4}(n) \\ &+ \bar{\boldsymbol{A}}_{52} \circledast \boldsymbol{Y}_{ij,5}(n) + \bar{\boldsymbol{A}}_{62} \circledast \boldsymbol{Y}_{ij,6}(n) + \bar{\boldsymbol{B}}_{12} \circledast \boldsymbol{U}_{ij,1} + \bar{\boldsymbol{B}}_{22} \circledast \boldsymbol{U}_{ij,2} \\ &+ \bar{\boldsymbol{B}}_{32} \circledast \boldsymbol{U}_{ij,3} + \bar{\boldsymbol{B}}_{42} \circledast \boldsymbol{U}_{ij,4} + \bar{\boldsymbol{B}}_{52} \circledast \boldsymbol{U}_{ij,5} + \bar{\boldsymbol{B}}_{62} \circledast \boldsymbol{U}_{ij,6} + \bar{z}_{ij,2}\big), \end{aligned} \tag{2.16b}$$

$$\begin{aligned} y_{ij,3}(n+1) = f\big(&\bar{\boldsymbol{A}}_{13} \circledast \boldsymbol{Y}_{ij,1}(n) + \bar{\boldsymbol{A}}_{23} \circledast \boldsymbol{Y}_{ij,2}(n) + \bar{\boldsymbol{A}}_{33} \circledast \boldsymbol{Y}_{ij,3}(n) + \bar{\boldsymbol{A}}_{43} \circledast \boldsymbol{Y}_{ij,4}(n) \\ &+ \bar{\boldsymbol{A}}_{53} \circledast \boldsymbol{Y}_{ij,5}(n) + \bar{\boldsymbol{A}}_{63} \circledast \boldsymbol{Y}_{ij,6}(n) + \bar{\boldsymbol{B}}_{13} \circledast \boldsymbol{U}_{ij,1} + \bar{\boldsymbol{B}}_{23} \circledast \boldsymbol{U}_{ij,2} \\ &+ \bar{\boldsymbol{B}}_{33} \circledast \boldsymbol{U}_{ij,3} + \bar{\boldsymbol{B}}_{43} \circledast \boldsymbol{U}_{ij,4} + \bar{\boldsymbol{B}}_{53} \circledast \boldsymbol{U}_{ij,5} + \bar{\boldsymbol{B}}_{63} \circledast \boldsymbol{U}_{ij,6} + \bar{z}_{ij,3}\big), \end{aligned} \tag{2.16c}$$

$$\begin{aligned} y_{ij,4}(n+1) = f\big(&\bar{\boldsymbol{A}}_{14} \circledast \boldsymbol{Y}_{ij,1}(n) + \bar{\boldsymbol{A}}_{24} \circledast \boldsymbol{Y}_{ij,2}(n) + \bar{\boldsymbol{A}}_{34} \circledast \boldsymbol{Y}_{ij,3}(n) + \bar{\boldsymbol{A}}_{44} \circledast \boldsymbol{Y}_{ij,4}(n) \\ &+ \bar{\boldsymbol{A}}_{54} \circledast \boldsymbol{Y}_{ij,5}(n) + \bar{\boldsymbol{A}}_{64} \circledast \boldsymbol{Y}_{ij,6}(n) + \bar{\boldsymbol{B}}_{14} \circledast \boldsymbol{U}_{ij,1} + \bar{\boldsymbol{B}}_{24} \circledast \boldsymbol{U}_{ij,2} \\ &+ \bar{\boldsymbol{B}}_{34} \circledast \boldsymbol{U}_{ij,3} + \bar{\boldsymbol{B}}_{44} \circledast \boldsymbol{U}_{ij,4} + \bar{\boldsymbol{B}}_{54} \circledast \boldsymbol{U}_{ij,5} + \bar{\boldsymbol{B}}_{64} \circledast \boldsymbol{U}_{ij,6} + \bar{z}_{ij,4}\big), \end{aligned} \tag{2.16d}$$

$$\begin{aligned} y_{ij,5}(n+1) = f\big(&\bar{\boldsymbol{A}}_{15} \circledast \boldsymbol{Y}_{ij,1}(n) + \bar{\boldsymbol{A}}_{25} \circledast \boldsymbol{Y}_{ij,2}(n) + \bar{\boldsymbol{A}}_{35} \circledast \boldsymbol{Y}_{ij,3}(n) + \bar{\boldsymbol{A}}_{45} \circledast \boldsymbol{Y}_{ij,4}(n) \\ &+ \bar{\boldsymbol{A}}_{55} \circledast \boldsymbol{Y}_{ij,5}(n) + \bar{\boldsymbol{A}}_{65} \circledast \boldsymbol{Y}_{ij,6}(n) + \bar{\boldsymbol{B}}_{15} \circledast \boldsymbol{U}_{ij,1} + \bar{\boldsymbol{B}}_{25} \circledast \boldsymbol{U}_{ij,2} \\ &+ \bar{\boldsymbol{B}}_{35} \circledast \boldsymbol{U}_{ij,3} + \bar{\boldsymbol{B}}_{45} \circledast \boldsymbol{U}_{ij,4} + \bar{\boldsymbol{B}}_{55} \circledast \boldsymbol{U}_{ij,5} + \bar{\boldsymbol{B}}_{65} \circledast \boldsymbol{U}_{ij,6} + \bar{z}_{ij,5}\big), \end{aligned} \tag{2.16e}$$

$$\begin{aligned} y_{ij,6}(n+1) = f\big(&\bar{\boldsymbol{A}}_{16} \circledast \boldsymbol{Y}_{ij,1}(n) + \bar{\boldsymbol{A}}_{26} \circledast \boldsymbol{Y}_{ij,2}(n) + \bar{\boldsymbol{A}}_{36} \circledast \boldsymbol{Y}_{ij,3}(n) + \bar{\boldsymbol{A}}_{46} \circledast \boldsymbol{Y}_{ij,4}(n) \\ &+ \bar{\boldsymbol{A}}_{56} \circledast \boldsymbol{Y}_{ij,5}(n) + \bar{\boldsymbol{A}}_{66} \circledast \boldsymbol{Y}_{ij,6}(n) + \bar{\boldsymbol{B}}_{16} \circledast \boldsymbol{U}_{ij,1} + \bar{\boldsymbol{B}}_{26} \circledast \boldsymbol{U}_{ij,2} \\ &+ \bar{\boldsymbol{B}}_{36} \circledast \boldsymbol{U}_{ij,3} + \bar{\boldsymbol{B}}_{46} \circledast \boldsymbol{U}_{ij,4} + \bar{\boldsymbol{B}}_{56} \circledast \boldsymbol{U}_{ij,5} + \bar{\boldsymbol{B}}_{66} \circledast \boldsymbol{U}_{ij,6} + \bar{z}_{ij,6}\big). \end{aligned} \tag{2.16f}$$

The 'FSR model of a two–layer DT CNN' and 'FSR model of a multi–layer DT CNN' hereafter shortly referred as 'two–layer DT CNN' and 'multi–layer DT CNN', respec-

tively, since other mathematical models are not used in this thesis.

CHAPTER 3

THE DIFFERENCES BETWEEN MULTI–LAYER CNNs, MULTIPLE–LAYER CNNs AND MULTI–CNN SYSTEMS

Although MLCNNs are initially defined by Chua and Yang with the introduction of CNN theory [1] in 1988, MLCNN applications did not emerge immediately. Then Harrer defined a multiple–layer CNN structure [26] which was different from the original model given in [1]. Multiple–layer CNN structure has time–variant templates without any inter–layer connections. Therefore MLCNN and multiple–layer CNN structures are different from each other with different topologies. The most important difference between these structures is that a multiple–layer CNN may have some topological differences like connecting the inputs, outputs or initial states of two different layers at two distinct iteration steps, has no inter–layer templates, etc. On the other hand, a MLCNN has inter–layer templates and it does not have any interconnection between two different layers at two distinct iteration steps but it has interconnections between two different layers at the same iteration step. Yang gave an explicit definition of MLCNNs [20] where the original model [1] is explained more thoroughly with only notation differences.

Output equation of a generalized mathematical model of MLCNN is given in (2.13). On the other hand, output equation of a generalized mathematical model of multiple–layer CNN structures is given in (3.1) [26]. Some elementary building blocks for interconnecting multiple layers are given in Figure 3.1. In the parallel mode, outputs of two distinct layers are given as an input to a logical function block called f_L block to obtain the output. In the cascade mode, the output of the previous layer is connected to the input of the subsequent layer which gives the output of the network. In the feedback loop mode, the outputs of successive layers are connected to the inputs of each other closing the loop.

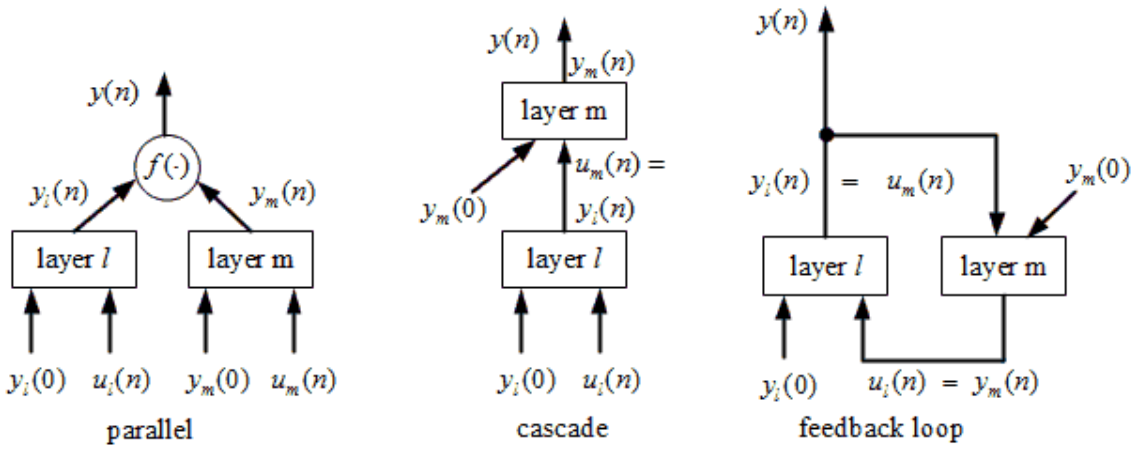


Figure 3.1 Some interconnection types of multiple–layer CNN structures [26]

Any other multiple–layer CNN structure can be obtained by combining these interconnection types hence the mathematical model changes due to the interconnection type of the structure.

$$y_{ij,p}(n+1) = f\left(\bar{\boldsymbol{A}}_p(n) \circledast \boldsymbol{Y}_{ij,p}(n) + \bar{\boldsymbol{B}}_p(n) \circledast \boldsymbol{U}_{ij,p}(n) + \bar{z}_p(n)\right). \tag{3.1}$$

The implementations of these structures and even other kinds of structures like multi–CNN systems began to emerge with the development of the first multiple–layer DT CNN chip [27]. Note that, there are some instances of misclassifications of the MLCNN and multiple–layer CNN structures like in [11, 28], hence care should be taken while conducting research in the area.

3.1 The Multi–CNN Architecture of Martinez et al.

The multi–CNN architecture [11, 28] of Martinez et al. is implemented as a DT CNN emulator on an FPGA device. It is reported as a multi–layer DT CNN however it does not realize the multi–layer model given in multi–layer CNN theory [1, 20]. It is actually a group of different CNNs forming a cascade and carrying out the computation in the time domain hence it may be called as a kind of multiple–layer DT CNNs but not a multi–layer DT CNN. It is also previously designed by the same team as a single–layer DT CNN emulator [29], implemented on an FPGA device. The single–layer DT CNN model and the simulated realization of Martinez et al. is given in their earlier work [30].

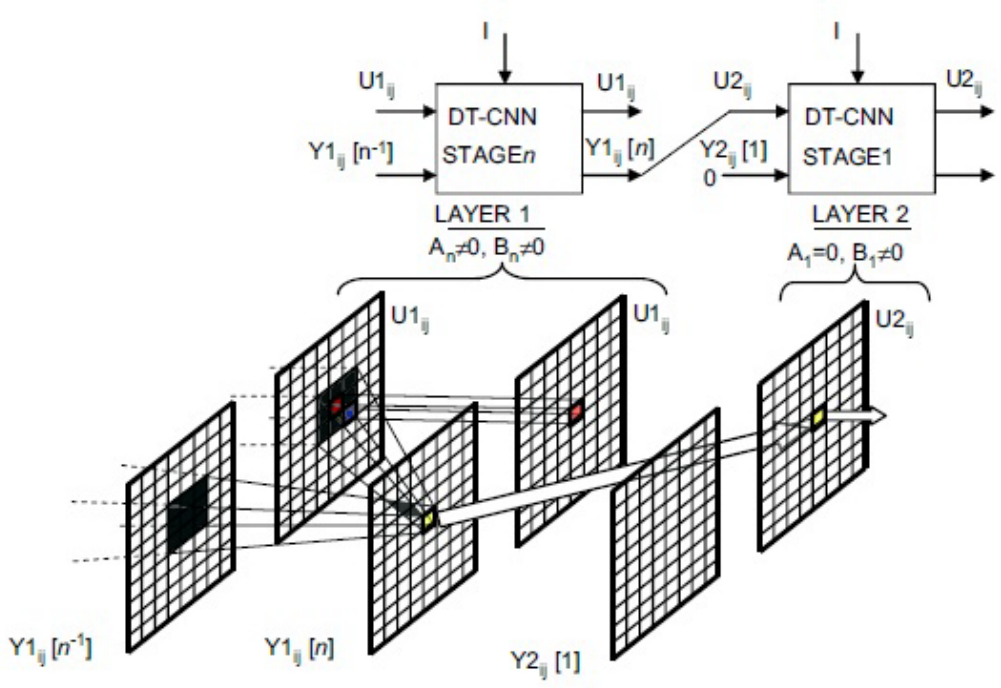


Figure 3.2 The multi–CNN system of Martinez et al. [11]

The processor core of the multi–CNN architecture of Martinez et al. does not have a difference with the single–layer processor core. The processor array of the multi–CNN architecture is also similar to the single–layer processor array (Figure 3.2). It is realized by connecting the output data port of the last stage of one CNN to the input data port of the first stage of the next CNN. The system is reported to work in real–time.

3.2 Summary

This thesis deals with the implementation of a two–layer DT–CNN and a specialized case of multi–layer DT CNNs with time–invariant templates but it is not about the multiple–layer CNN given in [26]. In [31], it is stated that using one layer with time–variant templates in a multiple–layer CNN architecture instead of two time–invariant layers reduces the exploited hardware and eliminates the overhead of transferring data between the layers. However, this is only true for DT CNN architectures, which are not fully pipelined and not designed to process higher resolutions as full–HD data in real–time. The architectures proposed in this thesis use the RTCNNP–v2 core as basis for building the two– and multi–layer DT CNNs and are fully pipelined, so it also eliminates the overhead of transferring data between the layers, although the templates are time–invariant. On the

other hand, thanks to the reconfigurability of the RTCNNP–v2 backbone, multiple–layer CNNs or even multiple–layer MLCNNs can be realized via different topologies.

CHAPTER 4

OVERVIEW OF THE EXISTING TWO–LAYER AND MULTI–LAYER CNN IMPLEMENTATIONS

There are several previously designed two– and multi–layer CNN implementations found in the literature: some of them are general purpose CNN implementations [3, 9] which are capable of realizing many different applications while others are application specific [7, 13, 17, 12]. In the following, an overview of the most notable MLCNN architectures is given.

4.1 The Multi–Layer CT CNN Architecture of CACE1K Chip

The multi-layer architecture of CACE1K CNN–UM chip [5], which is realized using 0.5μ CMOS technology, is implemented as a two–layer CT CNN chip [4] containing $2 \times (32 \times 32)$ cells. The chip inherently implements a third layer, however, it is only an output layer and it is not a part of the actual multi–layer CNN structure. The chip is used in early applications of artificial vision [3]. The model of the CNN-UM chip is given in Figure 4.1. In this model, there are two physical layers, which have all of the $\boldsymbol{A}$– and intra–layer $\boldsymbol{B}$–templates of the original Chua-Yang model [1]. In other words, the inter–layer $\boldsymbol{B}$–templates of the original Chua-Yang model [1] are excluded hence this model has two $\boldsymbol{B}$–templates and four $\boldsymbol{A}$–templates in total, however, both of the $\boldsymbol{B}$–templates and the inter–layer $\boldsymbol{A}$–templates have only one non–zero entry. In other words, the mathematical model of this architecture only consist of two intra–layer $\boldsymbol{A}$–templates, two inter–layer feedback coefficients ($a_{00,12}$ and $a_{00,21}$) and two intra–layer input coefficients ($b_{00,11}$ and $b_{00,22}$). The FSR model [25] of CT CNN is used in the design of this architecture. This can be considered as the only similarity with the two–layer RTCNNP architecture proposed

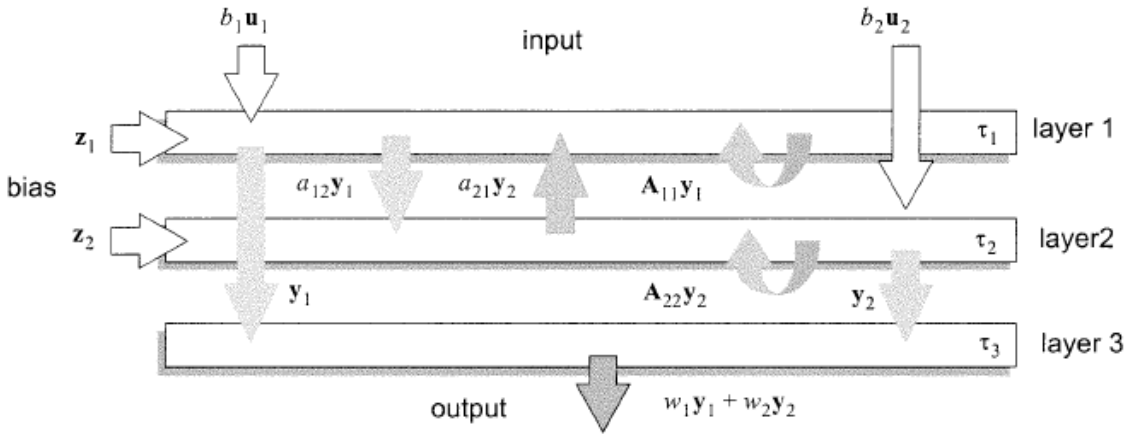


Figure 4.1 Model of three–layer CACE architecture [3]

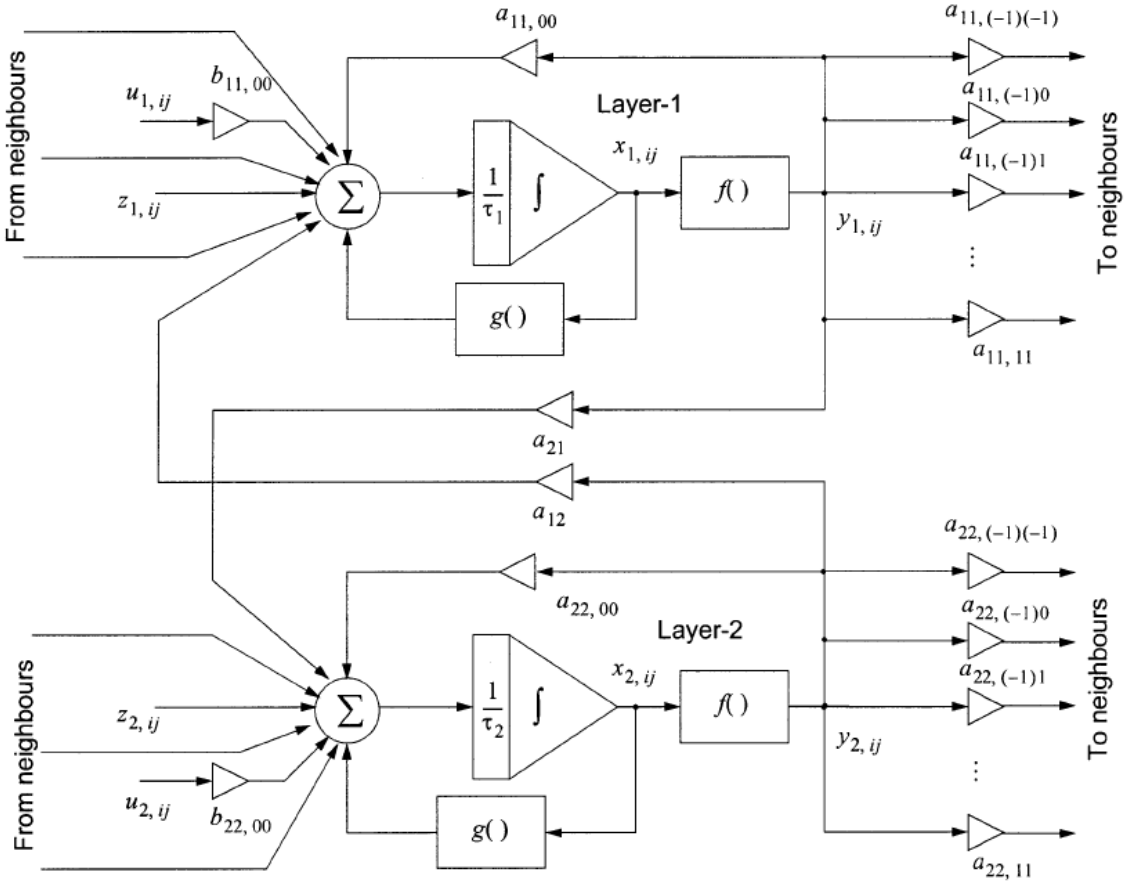
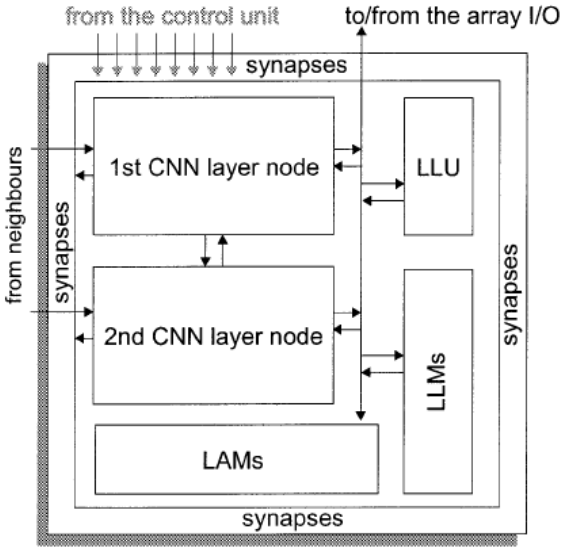


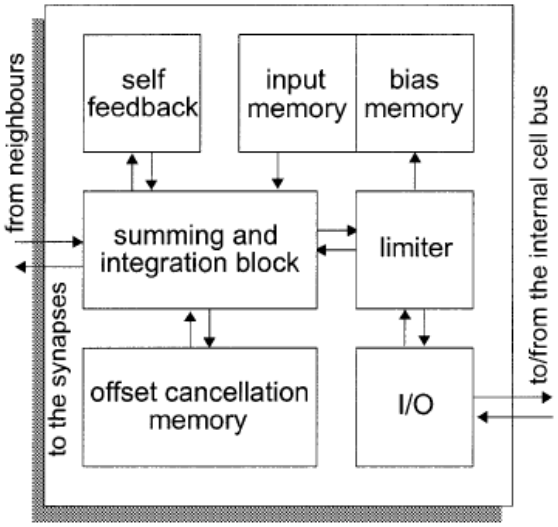
Figure 4.2 Functional block diagram of the two–layer architecture of CACE1K chip [4]

in Section 6.2 where the FSR model [25] of DT CNN is discussed. The functional block diagram of a cell of the two–layer architecture of CACE1K chip is given in Figure 4.2, the block diagram of the two–layer processor core of the CACE1K chip is given in Figure 4.3a and Figure 4.3b.

In Figure 4.3a, the basic cell structure has two nodes for two layers along with local ana-



(a) Top block diagram of the two–layer processor core of the CACE1K chip



(b) Sub block diagram of an analog CNN layer node

Figure 4.3 Block diagrams of the two–layer processor core of the CACE1K chip [4]

log memories (LAMs), local logic memories (LLMs) and a local logic unit (LLU). There are four LAMs and four LLMs, which are responsible for the storage of the intermediate results, and one LLU that carry out pixel–level logic operations. The internal structure of an analog CNN layer node (Figure 4.3b) consists of summming and integration, limiter, self feedback, input memory, bias memory, offset cancellation memory and I/O blocks. The summming and integration block receives contributions from the neighbours of the processing nodes and sums and integrates them like it is shown in Figure 4.2. The in-

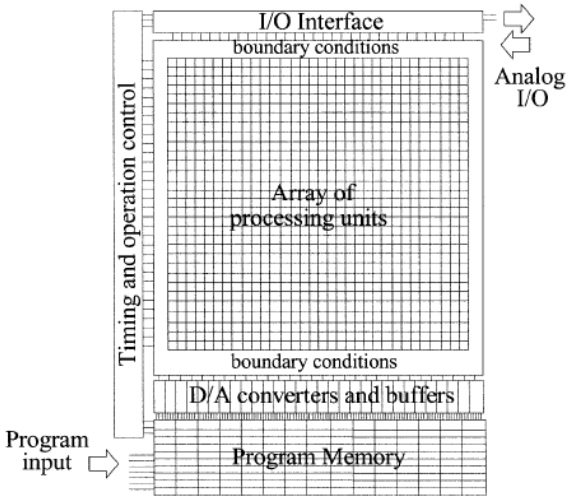


Figure 4.4 Processor Array of CACE1K chip [3]

put and bias memories are responsible for storing the input and bias of the current cell. The offset cancellation memory is responsible for cancellation of the offset of the synaptic blocks, therefore it communicates bidirectionally with the summing and integration block. The self feedback and limiter blocks are also shown in Figure 4.2 with $g()$ and $f()$ symbols respectively. The limiter block is responsible for implementing the FSR model of the CT CNN. Finally, the I/O block handles the communication with the intracell data bus. A two–layer processor core computes the result of a pixel, hence a 2–D input image having 32×32 pixels is processed by the 2–D array of processor cores (Figure 4.4) and an output image is obtained.

4.2 The Multi–Layer DT CNN Falcon Architecture

The single–layer Falcon architecture [9, 32], a DT CNN emulator implemented on an FPGA device, is based on the Castle architecture [10] which is a digital VLSI design implemented on a chip. The single–layer Falcon architecture uses the FSR model [25] of DT CNN like the Castle architecture but they make use of the FSR model in a different way such that the center element of the $\bar{\boldsymbol{A}}$–template differs in these architectures. Furthermore, as opposed to Castle, the architecture of Falcon can be configured to support MLCNN.

Processor array of the single–layer Falcon architecture is given in Figure 4.5. It has a

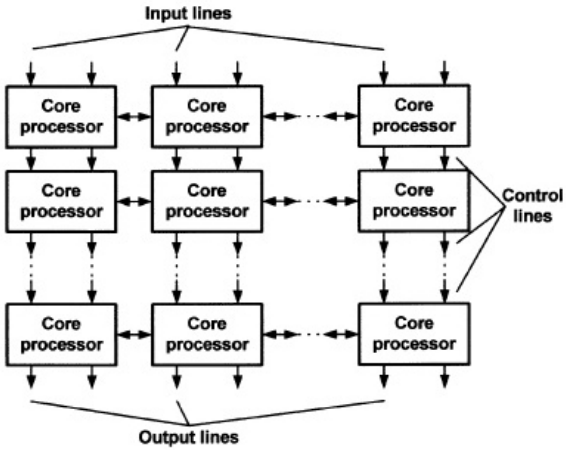


Figure 4.5 Processor Array of Falcon Architecture [9]

processor core forming a 2–D processor array which has input, output and control lines. The processed 2–D input is partitioned among the physical processors. Each processor column works on a long and narrow vertical stripe of the 2–D input image. In one cycle, a row of processors gets the result of the previous iteration from the row of processors above, calculates one iteration and sends the results to the row of processors below. In other words, a row of processors is responsible for the calculation of one iteration while a column of processors is responsible for the calculation of a long and narrow vertical stripe of a 2–D input image.

Processor core of a single–layer Falcon architecture is given in Figure 4.6. The Falcon processor core has a global control structure. The main parts of a Falcon processor core are the memory, mixer, arithmetic and template memory units. The memory unit stores the values of the previous iteration for the calculation of the current iteration. The mixer unit is also a kind of memory that stores values of the neigbouring stripes of a 2–D input image for the calculation of a stripe. The template memory stores the template values. The arithmetic unit, which is given in Figure 4.7, is responsible for carrying out the computation of the stored values in order to obtain the results of the current iteration of the stripe being processed. A simplified aritmetic unit (Figure 4.8) structure is used for the nearest neighborhood sized templates on the Falcon architecture. This simplified arithmetic unit of a Falcon processor core has a pipelined adder tree whereas the arithmetic unit of a Castle processor core has an adder tree without pipeline.

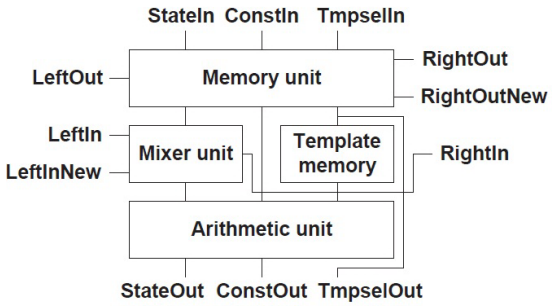


Figure 4.6 Processor core of Falcon Architecture [9]

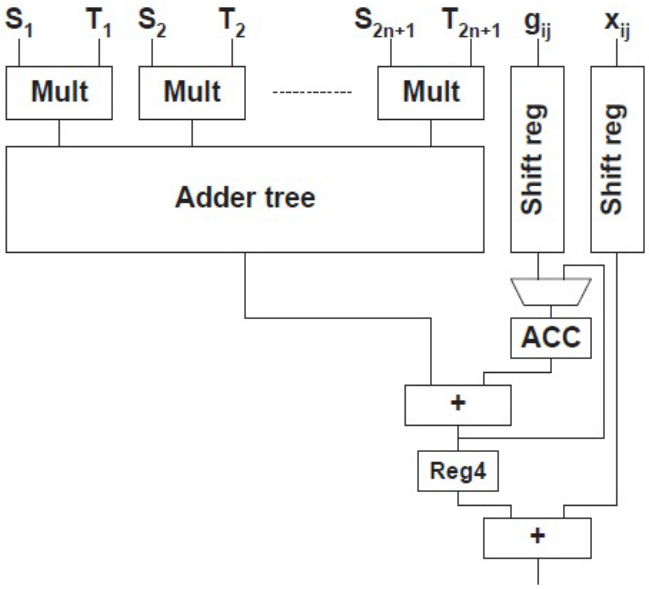


Figure 4.7 Arithmetic unit of Falcon processor core [9]

The processor array of the multi–layer Falcon architecture remains the same as that of a single–layer structure. On the other hand, the processor core of the multi–layer architecture (Figure 4.9) has significant changes except containing similar units, as core of the memory, mixer and template memory units remain the same. Arithmetic unit core needs P times more multipliers therefore the computation load increases. Architecture of the generalized multi–layer arithmetic unit is given in Figure 4.10. In this design, the number of multipliers, adders and registers in the pipelined adder tree is increased. The multi–layer aritmetic unit can be optimized according to a specific application to reduce the number of multipliers, adders and registers. The multi–layer processor core also needs a template select memory. Furthermore, the multi–layer processor core contains P memory, P mixer and P arithmetic units. Moreover, the arithmetic unit of each layer is connected to every

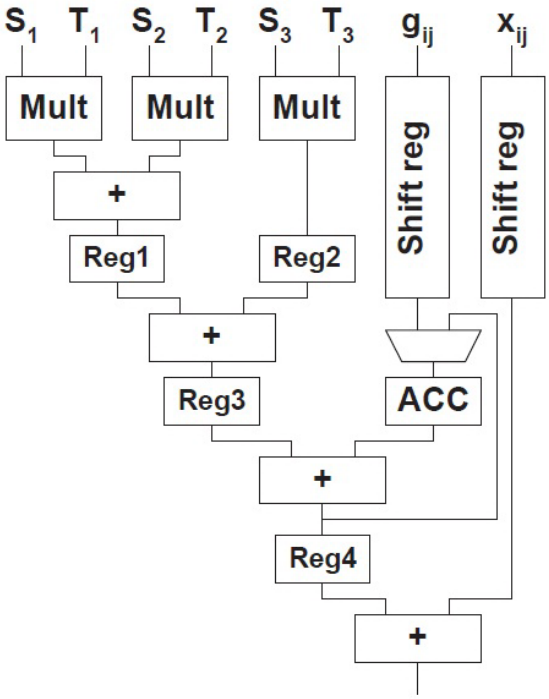


Figure 4.8 Simplified arithmetic unit of Falcon processor core [9]

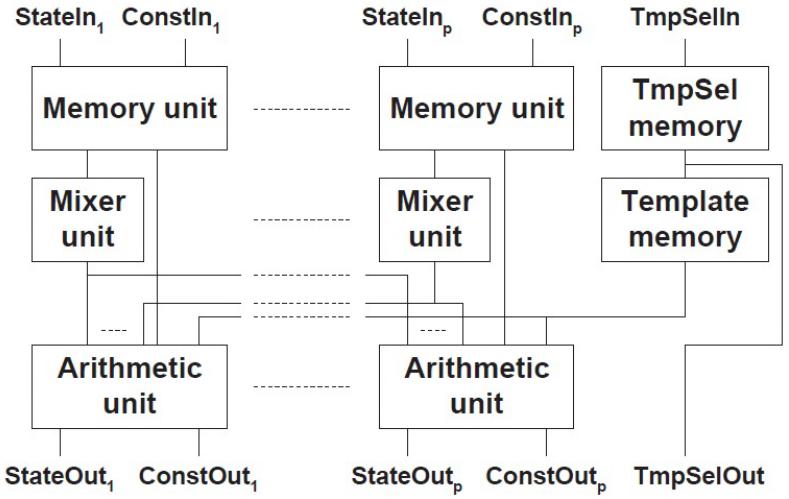


Figure 4.9 Processor core of multi–layer Falcon Architecture [9]

mixer; and the template memory is connected to every aritmetic unit. Finally, it is obvious that while a single–layer processor core contains 6 units, a multi–layer processor core contains $3P+2$ units with increased number of connections.

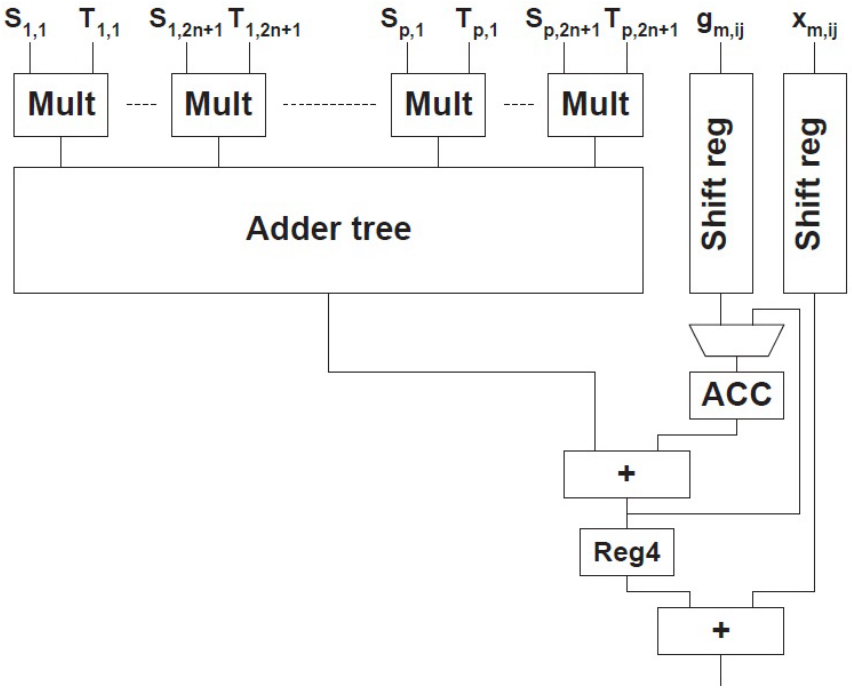


Figure 4.10 Arithmetic unit of multi–layer Falcon processor core [9]

CHAPTER 5

RTCNNP–v2 ARCHITECTURE

The RTCNNP–v2 architecture [15, 16, 6] is the fastest DT CNN emulator implemented on an FPGA device, to date. It is designed to add flexibility, modularity and reconfigurability to the RTCNNP–v1 architecture [14, 18]. Also, a new control structure for the pipelined CNN processor arrays is proposed over the RTCNNP–v1 architecture.

MLCNN architectures proposed in this thesis are built without changing the core and control structure of the RTCNNP–v2 architecture hence the RTCNNP–v2 core is one of the basic building blocks of the proposed architectures. Therefore, in this section, the mathematical design of the RTCNNP–v2 architecture is revised, and a summary of the RTCNNP–v2 architecture core and its processing array structure is given.

(2.14) can be rewritten indicating the functions of an A processing unit (APU) and a B processing unit (BPU).

$$y_{ij}(n+1) = \underbrace{f\left(\bar{\boldsymbol{A}} \circledast \boldsymbol{Y}_{ij}(n) + \overbrace{\bar{\boldsymbol{B}} \circledast \boldsymbol{U}_{ij} + \bar{z}}^{\text{BPU}}\right)}_{\text{APU}} \tag{5.1}$$

In (5.1), the BPU computes the constant term

$$g_{ij} = \bar{\boldsymbol{B}} \circledast \boldsymbol{U}_{ij} + \bar{z}, \tag{5.2}$$

which does not depend on the discrete–time variable n. By exploiting this property, it is possible to calculate g_{ij} only once for each input pixel, and use the same result as a constant through all Euler iterations. Now (5.1) can be rewritten as

$$y_{ij}(n+1) = f\left(\bar{\boldsymbol{A}} \circledast \boldsymbol{Y}_{ij}(n) + g_{ij}\right), \tag{5.3}$$

which is similar to (5.3), except for an $f(.)$ function.

Considering (5.2) and (5.3), a computation flow of a single–layer DT CNN can be written as

$$\begin{aligned}
&\text{constant calculation} && : && g_{ij} = \bar{\boldsymbol{B}} \circledast \boldsymbol{U}_{ij} + \bar{z} && \text{(5.4a)}\\
&\text{1st iteration} && : && y_{ij}(1) = f\left(\bar{\boldsymbol{A}} \circledast \boldsymbol{Y}_{ij}(0) + g_{ij}\right) && \text{(5.4b)}\\
&\text{2nd iteration} && : && y_{ij}(2) = f\left(\bar{\boldsymbol{A}} \circledast \boldsymbol{Y}_{ij}(1) + g_{ij}\right) && \text{(5.4c)}\\
&\vdots \quad \vdots && : && \vdots \qquad\qquad \vdots \qquad \vdots &&\\
&n\text{th iteration} && : && y_{ij}(n) = f\left(\bar{\boldsymbol{A}} \circledast \boldsymbol{Y}_{ij}(n-1) + g_{ij}\right) && \text{(5.4d)}\\
&\vdots \quad \vdots && : && \vdots \qquad\qquad \vdots \qquad \vdots &&\\
&N\text{th iteration} && : && y_{ij}(N) = f\left(\bar{\boldsymbol{A}} \circledast \boldsymbol{Y}_{ij}(N-1) + g_{ij}\right) && \text{(5.4e)}
\end{aligned}$$

where N is the total number of Euler iterations desired. Constant calculation (g_{ij}) is an independent process, while any iteration ($y_{ij}(n+1)$) depends on the results of the g_{ij} and the previous iteration ($y_{ij}(n)$). The computation flow is suitable for a fully pipelined architecture, as it is a feed–forward process chain. As opposed to the computation flow in the time domain, it is possible to compute an iteration result of a previous pixel while the iteration result of the current pixel is being computed in the spatial domains. The local control structure of the RTCNNP–v2 architecture completely eliminates the need of calculating the exact values of spatial shifts, caused by pipelining [6].

The x processing unit (xPU) is the core of the RTCNNP–v2 architecture (Figure 5.1). An xPU block can either be configured as an APU or a BPU hence the data and constant inputs of an xPU can be defined as

$$\text{data\_out} = \begin{cases} \text{data\_in} & \text{for BPU,} \\ \bar{\boldsymbol{A}} \circledast \text{data\_in} + \text{const\_in} & \text{for APU,} \end{cases} \tag{5.5}$$

$$\text{const\_out} = \begin{cases} \bar{\boldsymbol{B}} \circledast \text{data\_in} + \text{const\_in} & \text{for BPU,} \\ \text{const\_in} & \text{for APU,} \end{cases} \tag{5.6}$$

in terms of data input, constant input, $\bar{\boldsymbol{A}}$– and $\bar{\boldsymbol{B}}$– templates. The control outputs are the

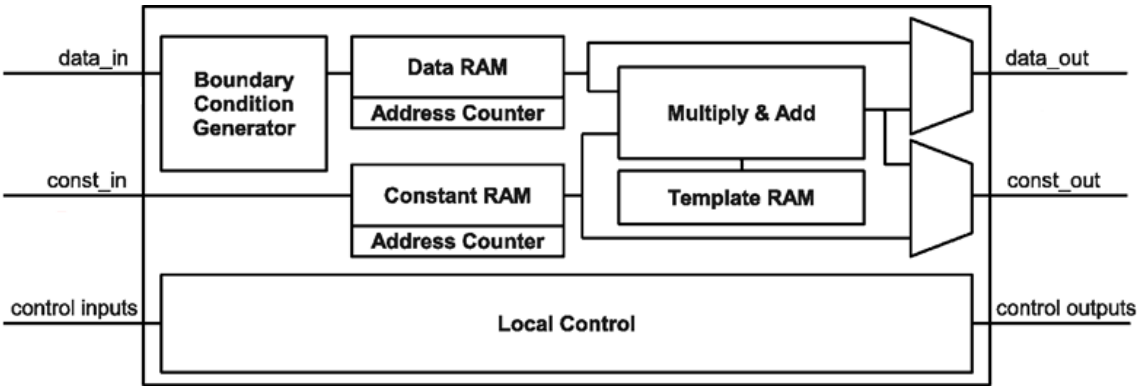


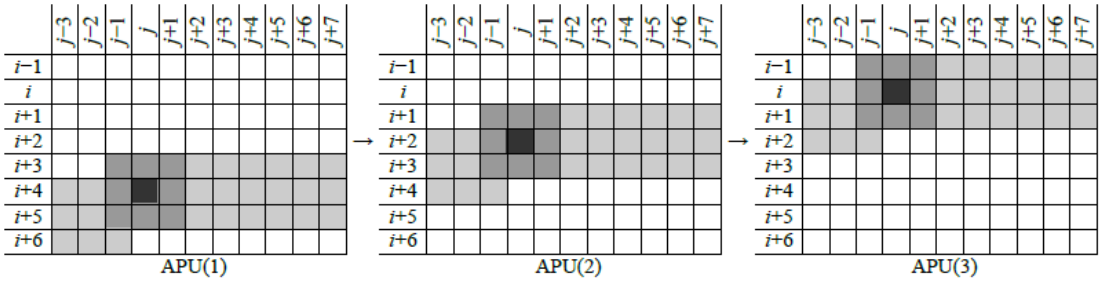
Figure 5.1 Processor core of RTCNNP–v2 Architecture [16]

same as the control inputs, only delayed in the local control block.

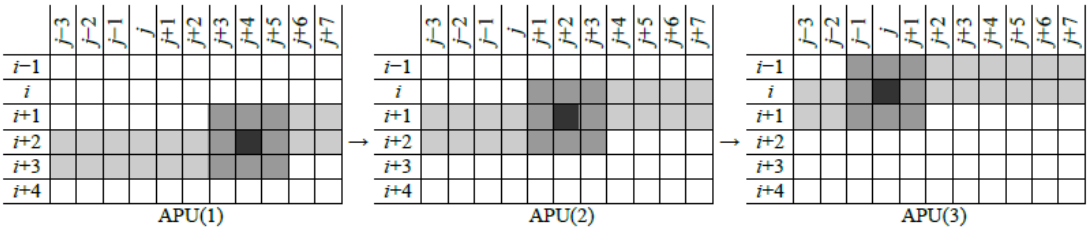
A brief overview of the simplified block diagram in Figure 5.1 can be given as follows. The boundary condition generator generates either zero–flux (Neumann) or fixed boundary conditions and it is programmable. The row–wise packed 2–D input image data and constant inputs are unpacked in the data and constant RAM units, respectively, where the address counters provide easy and local control. Template RAM is used to store the template values. Local control block is used for the synchronization of video signals. Multiply and add block is responsible for the template dot product and bias value addition operations. Finally, the output multiplexers route the results to the outputs depending on the xPU type.

It is also crucial to point out the memory organization of the RTCNNP–v2 core since it is the optimized version of the RTCNNP–v1 core. The memory usage of RTCNNP–v1 and RTCNNP–v2 cores are given in Figure 5.2a and Figure 5.2b, respectively. There is no need to store the data of a whole line after the optimization, hence the memory usage is reduced.

The simplified processor array of xPUs is given in Figure 5.3. For N iterations, $N+1$ xPUs containing 1 BPU and N APUs are connected end to end. BPU is always at the beginning of a processor chain, computing (5.4a). The data output of a BPU gives the original value of the data input (u_{ij}), delayed by the length of the pipeline. The first APU takes the data output (g_{ij}) of the BPU as its constant input and a data value which can be either any initial value ($y_{ij}(0)$) or the data output of the BPU (u_{ij}). The constant output



(a) Non-optimized memory structure



(b) Optimized memory structure

Figure 5.2 Memory organization of the RTCNNP–v2 core [15, 16]

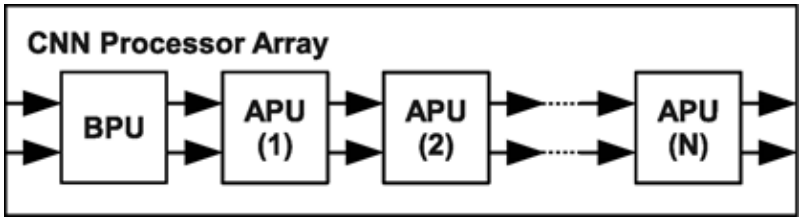


Figure 5.3 Processor array of the RTCNNP–v2 Architecture [16]

of the first APU (g_{ij}) is the same value as its data input; and its data output ($y_{ij}(1)$) is the result of the first iteration. All the remaining APUs are connected as a cascade structure for passing the constant value (g_{ij}) through the processor chain and for creating the output ($y_{ij}(n+1)$) of the current iteration. In this process, the output ($y_{ij}(n)$) of the previous iteration is used as the input of the current iteration. The reason why $N+1$ processors are used for N iterations is that the maximum speed requirement for the system to process real–time full–HD images. More detailed explanation can be found in [15, 16, 6].

CHAPTER 6

THE PROPOSED ARCHITECTURES FOR TWO–LAYER AND MULTI–LAYER CELLULAR NEURAL NETWORKS

In this chapter, the proposed two– and multi–layer CNN architectures are discussed in detail. In the first and second sections, two different approaches in the implementation of a two–layer CNN structure are proposed [33, 34]. In the third section, second approach of the proposed two–layer architecture is generalized for multi–layer CNN emulation. Fourth, the overview of the whole structure is given. Finally, the proposed architecture is compared to the multi–layer Falcon architecture.

6.1 First Approach to Design a Two–Layer Processor

Cell–output equation of an M–neighborhood space–invariant two–layer DT CNN for an iteration is previously given in (2.15), where it is seen that there are four $\boldsymbol{A}$– and four $\boldsymbol{B}$–template dot product operations in an Euler iteration. This equation pair can be realized via only one processor, two processors or any number of processors. Since a two–layer processor of a two–layer RTCNNP architecture uses single–layer processors (xPUs) of a single–layer RTCNNP–v2 architecture as basic building blocks, four processors are required for the calculation of an Euler iteration and four others are required for the calculation of the constants of two layers. Therefore, four xPUs are used as BPUs and four xPUs are used as APUs.

In the first approach, (2.15a) and (2.15b) are both divided into four parts, each of which is implemented by an xPU. Therefore (2.15) is rewritten indicating the functions of each

APU and BPU as given in (6.1a) and (6.1b).

$$y_{ij,1}(n+1) = \underbrace{f\Bigg(\bar{\boldsymbol{A}}_{21} \circledast \boldsymbol{Y}_{ij,2}(n) + \underbrace{\bar{\boldsymbol{A}}_{11} \circledast \boldsymbol{Y}_{ij,1}(n) + \overbrace{\bar{\boldsymbol{B}}_{21} \circledast \boldsymbol{U}_{ij,2}}^{\mathbf{BPU_{21}}} + \overbrace{\bar{\boldsymbol{B}}_{11} \circledast \boldsymbol{U}_{ij,1} + \bar{z}_{ij,1}}^{\mathbf{BPU_{11}}}}_{\mathbf{APU_{11}}}\Bigg)}_{\mathbf{APU_{21}}} \tag{6.1a}$$

$$y_{ij,2}(n+1) = \underbrace{f\Bigg(\bar{\boldsymbol{A}}_{22} \circledast \boldsymbol{Y}_{ij,2}(n) + \underbrace{\bar{\boldsymbol{A}}_{12} \circledast \boldsymbol{Y}_{ij,1}(n) + \overbrace{\bar{\boldsymbol{B}}_{12} \circledast \boldsymbol{U}_{ij,1}}^{\mathbf{BPU_{12}}} + \overbrace{\bar{\boldsymbol{B}}_{22} \circledast \boldsymbol{U}_{ij,2} + \bar{z}_{ij,2}}^{\mathbf{BPU_{22}}}}_{\mathbf{APU_{12}}}\Bigg)}_{\mathbf{APU_{22}}} \tag{6.1b}$$

An APU and a BPU is responsible for one $\boldsymbol{A}$– and one $\boldsymbol{B}$–template dot product operations, respectively [15, 16]. Since two $\boldsymbol{A}$– and two $\boldsymbol{B}$–template dot product operations are computed in both layers, computation of each process in the first layer of an iteration is given in (6.2a) and that of the second layer is given in (6.2b).

$$\begin{aligned}
\mathrm{BPU}_{11} \quad &: \quad g_{ij,11} = \bar{\boldsymbol{B}}_{11} \circledast \boldsymbol{U}_{ij,1} + \bar{z}_{ij,1} \\
\mathrm{BPU}_{21} \quad &: \quad g_{ij,21} = \bar{\boldsymbol{B}}_{21} \circledast \boldsymbol{U}_{ij,2} \\
\mathrm{APU}_{11}(n+1) \quad &: \quad h_{ij,1}(n+1) = \boldsymbol{A}_{11} \circledast \boldsymbol{Y}_{ij,1}(n) + g_{ij,11} + g_{ij,21} \\
\mathrm{APU}_{21}(n+1) \quad &: \quad y_{ij,1}(n+1) = f\big(x_{ij,1}(n+1)\big) = f\big(\bar{\boldsymbol{A}}_{21} \circledast \boldsymbol{Y}_{ij,2}(n) + h_{ij,1}(n+1)\big)
\end{aligned} \tag{6.2a}$$

$$\begin{aligned}
\mathrm{BPU}_{22} \quad &: \quad g_{ij,22} = \bar{\boldsymbol{B}}_{22} \circledast \boldsymbol{U}_{ij,2} + \bar{z}_{ij,2} \\
\mathrm{BPU}_{12} \quad &: \quad g_{ij,12} = \bar{\boldsymbol{B}}_{12} \circledast \boldsymbol{U}_{ij,1} \\
\mathrm{APU}_{12}(n+1) \quad &: \quad h_{ij,2}(n+1) = \boldsymbol{A}_{12} \circledast \boldsymbol{Y}_{ij,1}(n) + g_{ij,22} + g_{ij,12} \\
\mathrm{APU}_{22}(n+1) \quad &: \quad y_{ij,2}(n+1) = f\left(x_{ij,2}(n+1)\right) = f\left(\bar{\boldsymbol{A}}_{22} \circledast \boldsymbol{Y}_{ij,2}(n) + h_{ij,2}(n+1)\right)
\end{aligned} \tag{6.2b}$$

In order to show the computation flow of each iteration by means of the processing units

(processors), (6.2) can be written in a sequence as given in (6.3)

$$
\begin{aligned}
\text{BPU}_{11} \quad &: \quad g_{ij,11} = \bar{\boldsymbol{B}}_{11} \circledast \boldsymbol{U}_{ij,1} + \bar{z}_{ij,1} \\
\text{BPU}_{21} \quad &: \quad g_{ij,21} = \bar{\boldsymbol{B}}_{21} \circledast \boldsymbol{U}_{ij,2} \\
\text{BPU}_{22} \quad &: \quad g_{ij,22} = \bar{\boldsymbol{B}}_{22} \circledast \boldsymbol{U}_{ij,2} + \bar{z}_{ij,2} \\
\text{BPU}_{12} \quad &: \quad g_{ij,12} = \bar{\boldsymbol{B}}_{12} \circledast \boldsymbol{U}_{ij,1} \\
\text{APU}_{11}(1) \quad &: \quad h_{ij,1}(1) = \boldsymbol{A}_{11} \circledast \boldsymbol{Y}_{ij,1}(0) + g_{ij,11} + g_{ij,21} \\
\text{APU}_{21}(1) \quad &: \quad y_{ij,1}(1) = f\left(\bar{\boldsymbol{A}}_{21} \circledast \boldsymbol{Y}_{ij,2}(0) + h_{ij,1}(1)\right) \\
\text{APU}_{12}(1) \quad &: \quad h_{ij,2}(1) = \boldsymbol{A}_{12} \circledast \boldsymbol{Y}_{ij,1}(0) + g_{ij,22} + g_{ij,12} \\
\text{APU}_{22}(1) \quad &: \quad y_{ij,2}(1) = f\left(\bar{\boldsymbol{A}}_{22} \circledast \boldsymbol{Y}_{ij,2}(0) + h_{ij,2}(1)\right) \\
\vdots \qquad & \qquad\qquad\qquad \vdots \\
\text{APU}_{11}(N) \quad &: \quad h_{ij,1}(N) = \boldsymbol{A}_{11} \circledast \boldsymbol{Y}_{ij,1}(N-1) + g_{ij,11} + g_{ij,21} \\
\text{APU}_{21}(N) \quad &: \quad y_{ij,1}(N) = f\left(\bar{\boldsymbol{A}}_{21} \circledast \boldsymbol{Y}_{ij,2}(N-1) + h_{ij,1}\right) \\
\text{APU}_{12}(N) \quad &: \quad h_{ij,2}(N) = \boldsymbol{A}_{12} \circledast \boldsymbol{Y}_{ij,1}(N-1) + g_{ij,22} + g_{ij,12} \\
\text{APU}_{22}(N) \quad &: \quad y_{ij,2}(N) = f\left(\bar{\boldsymbol{A}}_{22} \circledast \boldsymbol{Y}_{ij,2}(N-1) + h_{ij,2}.\right)
\end{aligned}
\tag{6.3}
$$

The activation function ($f(.)$) is applied to the states of both layers ($x_{ij,1}(n+1)$ and $x_{ij,2}(n+1)$), after which the outputs of both layers ($y_{ij,1}(n+1)$ and $y_{ij,2}(n+1)$) are obtained. In addition to this computation, the bit widths of the outputs are also reduced to be compatible with the bit widths of the inputs of the next APU. Therefore an xPU has a saturator block, applying the activation function and performing the rounding operation, which can be enabled or disabled as required [6].

Since there are intermediate values in the designed processor core, the saturator blocks in APU_{11} and APU_{12} are disabled because the intermediate values must be computed without saturation. Therefore APU_{21} and APU_{22} are used as the same in the previous RTCNNP–v2 system because they give the outputs of the new core. The simplified block diagram of this new core called APU2L (APU for two–layer DT CNN) is given in Figure 6.1. There are also delay blocks to ensure data synchronization. The control I/O of the

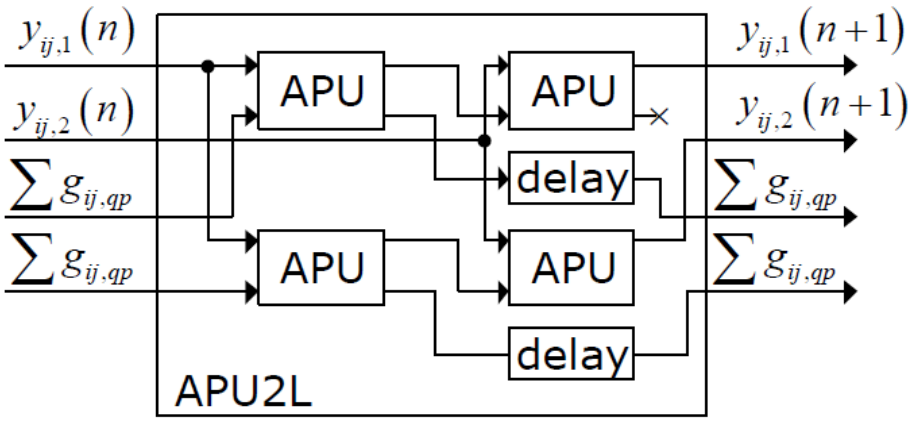


Figure 6.1 First approach for the processor core of the two–layer RTCNNP Architecture

APUs are not shown since the local control structure is the same as that of the previous works [15, 16, 6].

This approach is not chosen for implementation because there is an important setback in this design: the constant outputs of the APUs in the first stage are not saturated hence their bit widths are larger than the saturated outputs. As a result, APUs in the second stage require larger memory, wasting too much resources.

6.2 Second Approach to Design a Two–Layer Processor

In the second approach, (2.15a) and (2.15b) are both divided into four parts, similar to the first approach. Therefore (2.15) is rewritten to indicate the functions of each APU and BPU. However, the division is revised for the resource optimization, where an extra *addition and saturation* operation is also used:

$$y_{ij,1}(n+1) = \underbrace{f\Big(\underbrace{\bar{A}_{21}}_{\textbf{APU}_{21}} \circledast \boldsymbol{Y}_{ij,2}(n) + \underbrace{\bar{\boldsymbol{A}}_{11} \circledast \boldsymbol{Y}_{ij,1}(n) + \overbrace{\bar{\boldsymbol{B}}_{21} \circledast \boldsymbol{U}_{ij,2}}^{\textbf{BPU}_{21}} + \overbrace{\bar{\boldsymbol{B}}_{11} \circledast \boldsymbol{U}_{ij,1} + \bar{z}_{ij,1}}^{\textbf{BPU}_{11}}}_{\textbf{APU}_{11}}\Big)}_{\textbf{addition and saturation}} \tag{6.4a}$$

$$y_{ij,2}(n+1) = \underbrace{f\Big(\underbrace{\bar{A}_{22}}_{\textbf{APU}_{22}} \circledast \boldsymbol{Y}_{ij,2}(n) + \underbrace{\bar{\boldsymbol{A}}_{12} \circledast \boldsymbol{Y}_{ij,1}(n) + \overbrace{\bar{\boldsymbol{B}}_{12} \circledast \boldsymbol{U}_{ij,1}}^{\textbf{BPU}_{12}} + \overbrace{\bar{\boldsymbol{B}}_{22} \circledast \boldsymbol{U}_{ij,2} + \bar{z}_{ij,2}}^{\textbf{BPU}_{22}}}_{\textbf{APU}_{12}}\Big)}_{\textbf{addition and saturation}} \tag{6.4b}$$

The differences of the new approach are the new configuration of APUs and the new adder and saturator blocks in the topology. In this approach, the outputs of the layers are obtained from the saturator blocks instead of the APUs and the intermediate state values are calculated with the help of the adder blocks. Computation of each process in the first layer of an iteration is given in (6.5a) and that of the second layer is given in (6.5b).

$$
\begin{aligned}
&\mathrm{BPU}_{11} && : \quad g_{ij,11} && = \bar{\boldsymbol{B}}_{11} \circledast \boldsymbol{U}_{ij,1} + \bar{z}_{ij,1} \\
&\mathrm{BPU}_{21} && : \quad g_{ij,21} && = \bar{\boldsymbol{B}}_{21} \circledast \boldsymbol{U}_{ij,1} \\
&\mathrm{APU}_{11}(n+1) && : \quad h_{ij,1}(n+1) && = \boldsymbol{A}_{11} \circledast \boldsymbol{Y}_{ij,1}(n) + g_{ij,11} + g_{ij,21} \\
&\mathrm{APU}_{21}(n+1) && : \quad l_{ij,1}(n+1) && = \bar{\boldsymbol{A}}_{21} \circledast \boldsymbol{Y}_{ij,2}(n) \\
&\quad y_{ij,1}(n+1) && = f\left(x_{ij,1}(n+1)\right) && = f\left(h_{ij,1}(n+1) + l_{ij,1}(n+1)\right)
\end{aligned} \tag{6.5a}
$$

$$
\begin{aligned}
&\mathrm{BPU}_{22} && : \quad g_{ij,22} && = \bar{\boldsymbol{B}}_{22} \circledast \boldsymbol{U}_{ij,2} + \bar{z}_{ij,2} \\
&\mathrm{BPU}_{12} && : \quad g_{ij,12} && = \bar{\boldsymbol{B}}_{12} \circledast \boldsymbol{U}_{ij,1} \\
&\mathrm{APU}_{12}(n+1) && : \quad h_{ij,2}(n+1) && = \boldsymbol{A}_{12} \circledast \boldsymbol{Y}_{ij,1}(n) + g_{ij,22} + g_{ij,12} \\
&\mathrm{APU}_{22}(n+1) && : \quad l_{ij,2}(n+1) && = \bar{\boldsymbol{A}}_{22} \circledast \boldsymbol{Y}_{ij,2}(n) \\
&\quad y_{ij,2}(n+1) && = f\left(x_{ij,2}(n+1)\right) && = f\left(h_{ij,2}(n+1) + l_{ij,2}(n+1)\right)
\end{aligned} \tag{6.5b}
$$

In order to show the computation flow of each iteration by means of the processors, (6.5)

can be written in a sequence as given in (6.6)

$$
\begin{aligned}
&\mathrm{BPU}_{11} && : && g_{ij,11} && = \bar{\boldsymbol{B}}_{11} \circledast \boldsymbol{U}_{ij,1} + \bar{z}_{ij,1} \\
&\mathrm{BPU}_{21} && : && g_{ij,21} && = \bar{\boldsymbol{B}}_{21} \circledast \boldsymbol{U}_{ij,2} \\
&\mathrm{BPU}_{22} && : && g_{ij,22} && = \bar{\boldsymbol{B}}_{22} \circledast \boldsymbol{U}_{ij,2} + \bar{z}_{ij,2} \\
&\mathrm{BPU}_{12} && : && g_{ij,12} && = \bar{\boldsymbol{B}}_{12} \circledast \boldsymbol{U}_{ij,1} \\
&\mathrm{APU}_{11}(1) && : && h_{ij,1}(1) && = \boldsymbol{A}_{11} \circledast \boldsymbol{Y}_{ij,1}(0) + g_{ij,11} + g_{ij,21} \\
&\mathrm{APU}_{21}(1) && : && l_{ij,1}(1) && = \bar{\boldsymbol{A}}_{21} \circledast \boldsymbol{Y}_{ij,2}(0) \\
& y_{ij,1}(1) && = f(x_{ij,1}(1)) && = f(h_{ij,1}(1) + l_{ij,1}(1)) \\
&\mathrm{APU}_{12}(1) && : && h_{ij,2}(1) && = \boldsymbol{A}_{12} \circledast \boldsymbol{Y}_{ij,1}(0) + g_{ij,22} + g_{ij,12} \\
&\mathrm{APU}_{22}(1) && : && l_{ij,2}(1) && = \bar{\boldsymbol{A}}_{22} \circledast \boldsymbol{Y}_{ij,2}(0) \\
& y_{ij,2}(1) && = f(x_{ij,2}(1)) && = f(h_{ij,2}(1) + l_{ij,2}(1)) \\
& \vdots && \vdots && \vdots \\
&\mathrm{APU}_{11}(N) && : && h_{ij,1}(N) && = \boldsymbol{A}_{11} \circledast \boldsymbol{Y}_{ij,1}(N-1) + g_{ij,11} + g_{ij,21} \\
&\mathrm{APU}_{21}(N) && : && l_{ij,1}(N) && = \bar{\boldsymbol{A}}_{21} \circledast \boldsymbol{Y}_{ij,2}(N-1) \\
& y_{ij,1}(N) && = f(x_{ij,1}(N)) && = f(h_{ij,1}(N) + l_{ij,1}(N)) \\
&\mathrm{APU}_{12}(N) && : && h_{ij,2}(N) && = \boldsymbol{A}_{12} \circledast \boldsymbol{Y}_{ij,1}(N-1) + g_{ij,22} + g_{ij,12} \\
&\mathrm{APU}_{22}(N) && : && l_{ij,2}(N) && = f(\bar{\boldsymbol{A}}_{22} \circledast \boldsymbol{Y}_{ij,2}(N-1) + h_{ij,2}) \\
& y_{ij,2}(N) && = f(x_{ij,2}(N)) && = f(h_{ij,2}(N) + l_{ij,2}(N))
\end{aligned}
\tag{6.6}
$$

Since there are more intermediate values in the new processor core, saturators inside all of the APUs of the RTCNNP–v2 system are disabled. The simplified block diagram of this new APU2L is given in Figure 6.2. Similar to the first approach, the control I/O of the APUs are not shown since the local control structure is the same as that of the previous works [15, 16, 6]. Note that, all APUs are designed to be parallel in this approach, which reduces the memory usage and total delay of an APU2L block. Consequently, the additional delay of an APU2L is not caused by the line buffers of the streaming input data, but the registered adders and saturators in the second stage of the pipeline. In other words,

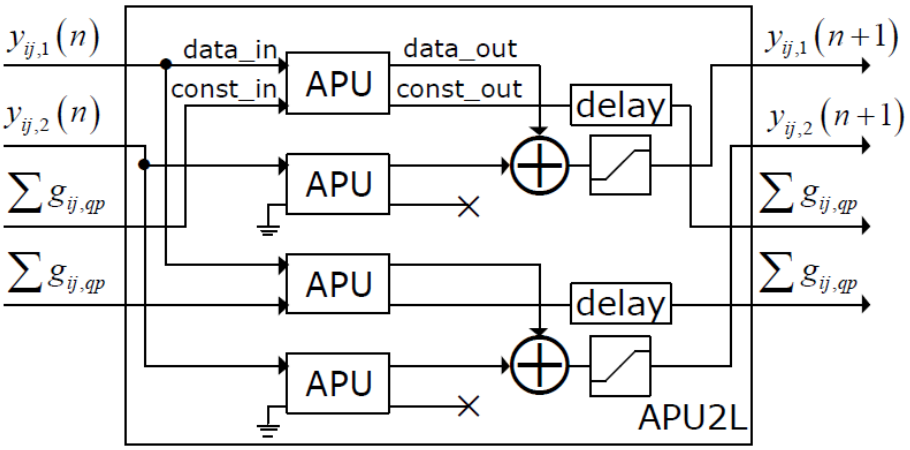


Figure 6.2 Second approach for the processor core of the two–layer RTCNNP Architecture

the delay of an APU2L is only two clock cycles longer than that of an APU, which is insignificant compared to the delay of two line buffers. Therefore, this design is chosen for the implementation.

The processor computations and the output function in (6.5) and (6.6) can be generalized as (6.7).

$$\text{BPU}_{qp} \quad : g_{ij,qp} \quad = \begin{cases} \bar{\boldsymbol{B}}_{qp} \circledast \boldsymbol{U}_{ij,q} + \bar{z}_{ij,p} & p = q, \\ \bar{\boldsymbol{B}}_{qp} \circledast \boldsymbol{U}_{ij,q} & p \neq q. \end{cases} \tag{6.7a}$$

$$\text{APU}_{qp}(n) \quad : h_{ij,qp}(n) \quad = \begin{cases} \bar{\boldsymbol{A}}_{qp} \circledast \boldsymbol{Y}_{ij,q}(n-1) + \sum_{q=1}^{2} g_{ij,qp} & p = q, \\ \bar{\boldsymbol{A}}_{qp} \circledast \boldsymbol{Y}_{ij,q}(n-1) & p \neq q. \end{cases} \tag{6.7b}$$

$$y_{ij,p}(n) \quad = f(x_{ij,p}(n)) = f\left(\sum_{q=1}^{2} h_{ij,qp}(n)\right) \tag{6.7c}$$

Figure 6.2 computes one iteration of the two–layer DT CNN. For N iterations, N APU2L blocks are connected to each other to form a cascade. This structure is shown in Figure 6.3 which is the processor array of the two–layer RTCNNP. In this processor array, there are two constant values which should be carried from the first block to the N^{th} block without a change in their values. These constant values are the $g_{ij,qp}$ terms which are produced by the BPU_{qp} block using the $U_{ij,q}$ input of the q^{th} layer and $\bar{z}_{ij,p}$ bias input of the p^{th} layer of the CNN. These constant values are calculated once for every frame as in a single–

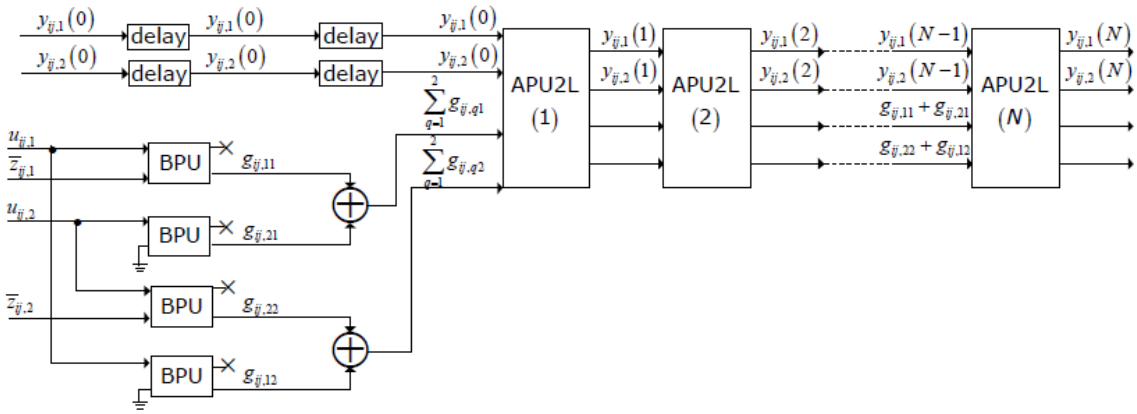


Figure 6.3 Processor array of the two–layer RTCNNP Architecture

layer CNN and carried through the iterations. As a result, APU2L(1) block takes the initial values of both layers and two constant values for each layer and carries out the first iteration. The results of the first iteration (($y_{ij,1}(1)$ and $y_{ij,2}(1)$) and the $g_{ij,qp}$ terms in each layer are given to the APU2L(2) block as inputs and the iterations continue until the output of the last APU2L(N) block gives its results. The number of xPUs in this processor array is $4N+4$, i.e., $4N$ APUs and 4 BPUs.

6.3 Proposed Specialized Multi–Layer DT CNN structure

In this thesis, the general MLCNN structure is specialized in order to reduce the resource usage on an FPGA device. The mathematical foundation and the architecture of the proposed MLCNN structure are given in this section.

6.3.1 The Mathematics of the Specialized Multi–Layer DT CNN structure

In a general multi–layer DT CNN structure, there are P^2 $\boldsymbol{A}$–templates and P^2 $\boldsymbol{B}$–templates that require $2P^2$ template dot product operations, as each layer should interact with all other layers. Inspired from the pixel–neigbourhood concept, a measure of layer–neigbourhood is defined, which limits the number of the inter–layer connections. For a layer–neighbourhood measure of R, both the number of $\boldsymbol{A}$– and $\boldsymbol{B}$–templates are reduced to $(1+2R)P-2$ from P^2. A measure of $R=1$ is used in the FPGA implementations discussed in this thesis, in other words nearest neighbours are related.

An optimization is done for $\boldsymbol{B}$–templates for defining only one input layer hence the num-

ber of $\boldsymbol{B}$–templates are reduced from P^2 to 1. In other words, an input is given to only one of the layers and all the neigbouring connections and inter–layer $\boldsymbol{B}$–templates are eliminated. As a result, the closed form of a specialized multi–layer DT CNN is obtained as (6.8)

$$y_{ij,p}(n+1) = \begin{cases} f\left(\sum_{q=1}^{2} \bar{\boldsymbol{A}}_{qp} \circledast \boldsymbol{Y}_{ij,q}(n) + \bar{\boldsymbol{B}}_{11} \circledast \boldsymbol{U}_{ij,1} + \bar{z}_1\right) & p = 1, \\ f\left(\sum_{q=P-1}^{P} \bar{\boldsymbol{A}}_{qp} \circledast \boldsymbol{Y}_{ij,q}(n) + \bar{z}_P\right) & p = P, \\ f\left(\sum_{q=p-1}^{p+1} \bar{\boldsymbol{A}}_{qp} \circledast \boldsymbol{Y}_{ij,q}(n) + \bar{z}_p\right) & \text{otherwise.} \end{cases} \tag{6.8}$$

This optimization is less crucial then the measure of layer–neigbourhood defined in this section. Note that, as discussed in the next section multiple $\boldsymbol{B}$–templates may be used when required. In this case, the closed form of a specialized multi–layer DT CNN can be given as (6.9)

$$y_{ij,p}(n+1) = \begin{cases} f\left(\sum_{q=1}^{2} \bar{\boldsymbol{A}}_{qp} \circledast \boldsymbol{Y}_{ij,q}(n) + \bar{\boldsymbol{B}}_{qp} \circledast \boldsymbol{U}_{ij,q} + \bar{z}_1\right) & p = 1, \\ f\left(\sum_{q=P-1}^{P} \bar{\boldsymbol{A}}_{qp} \circledast \boldsymbol{Y}_{ij,q}(n) + \bar{\boldsymbol{B}}_{qp} \circledast \boldsymbol{U}_{ij,q} + \bar{z}_P\right) & p = P, \\ f\left(\sum_{q=p-1}^{p+1} \bar{\boldsymbol{A}}_{qp} \circledast \boldsymbol{Y}_{ij,q}(n) + \bar{\boldsymbol{B}}_{qp} \circledast \boldsymbol{U}_{ij,q} + \bar{z}_p\right) & \text{otherwise.} \end{cases} \tag{6.9}$$

For the sake of simplicity, an example of a specialized six–layer DT CNN can be shown as (6.10) by taking $P = 6$ in (6.8), where layer–neighbourhood is one and the first layer is defined as the input layer. In this example, the number of $\boldsymbol{A}$– and $\boldsymbol{B}$–templates are reduced from 36 to 16 and 36 to 1, respectively.

$$y_{ij,1}(n+1) = f\left(\bar{\boldsymbol{A}}_{11} \circledast \boldsymbol{Y}_{ij,1}(n) + \bar{\boldsymbol{A}}_{21} \circledast \boldsymbol{Y}_{ij,2}(n) + \bar{\boldsymbol{B}}_{11} \circledast \boldsymbol{U}_{ij,1} + \bar{z}_{ij,1}\right), \tag{6.10a}$$

$$y_{ij,2}(n+1) = f\left(\bar{\boldsymbol{A}}_{12} \circledast \boldsymbol{Y}_{ij,1}(n) + \bar{\boldsymbol{A}}_{22} \circledast \boldsymbol{Y}_{ij,2}(n) + \bar{\boldsymbol{A}}_{32} \circledast \boldsymbol{Y}_{ij,3}(n) + \bar{z}_{ij,2}\right), \tag{6.10b}$$

$$y_{ij,3}(n+1) = f\left(\bar{\boldsymbol{A}}_{23} \circledast \boldsymbol{Y}_{ij,2}(n) + \bar{\boldsymbol{A}}_{33} \circledast \boldsymbol{Y}_{ij,3}(n) + \bar{\boldsymbol{A}}_{43} \circledast \boldsymbol{Y}_{ij,4}(n) + \bar{z}_{ij,3}\right), \tag{6.10c}$$

$$y_{ij,4}(n+1) = f\left(\bar{\boldsymbol{A}}_{34} \circledast \boldsymbol{Y}_{ij,3}(n) + \bar{\boldsymbol{A}}_{44} \circledast \boldsymbol{Y}_{ij,4}(n) + \bar{\boldsymbol{A}}_{54} \circledast \boldsymbol{Y}_{ij,5}(n) + \bar{z}_{ij,4}\right), \tag{6.10d}$$

$$y_{ij,5}(n+1) = f\left(\bar{\boldsymbol{A}}_{45} \circledast \boldsymbol{Y}_{ij,3}(n) + \bar{\boldsymbol{A}}_{55} \circledast \boldsymbol{Y}_{ij,4}(n) + \bar{\boldsymbol{A}}_{65} \circledast \boldsymbol{Y}_{ij,5}(n) + \bar{z}_{ij,5}\right), \tag{6.10e}$$

$$y_{ij,6}(n+1) = f\left(\bar{\boldsymbol{A}}_{56} \circledast \boldsymbol{Y}_{ij,3}(n) + \bar{\boldsymbol{A}}_{66} \circledast \boldsymbol{Y}_{ij,4}(n) + \bar{z}_{ij,6}\right) \tag{6.10f}$$

However, many MLCNN applications have non–zero far-layer interconnections, consequently the proposed layer–neighbourhood measure is incomplete. For example, it may

be necessary for an application to have two additional templates over (6.10) as given in (6.11)

$$y_{ij,1}(n+1) = f\left(\bar{\boldsymbol{A}}_{11} \circledast \boldsymbol{Y}_{ij,1}(n) + \bar{\boldsymbol{A}}_{21} \circledast \boldsymbol{Y}_{ij,2}(n) + \bar{\boldsymbol{B}}_{11} \circledast \boldsymbol{U}_{ij,1} + \bar{z}_{ij,1} + \bar{\boldsymbol{A}}_{61} \circledast \boldsymbol{Y}_{ij,6}(n)\right), \tag{6.11a}$$

$$y_{ij,2}(n+1) = f\left(\bar{\boldsymbol{A}}_{12} \circledast \boldsymbol{Y}_{ij,1}(n) + \bar{\boldsymbol{A}}_{22} \circledast \boldsymbol{Y}_{ij,2}(n) + \bar{\boldsymbol{A}}_{32} \circledast \boldsymbol{Y}_{ij,3}(n) + \bar{z}_{ij,2}\right), \tag{6.11b}$$

$$y_{ij,3}(n+1) = f\left(\bar{\boldsymbol{A}}_{23} \circledast \boldsymbol{Y}_{ij,2}(n) + \bar{\boldsymbol{A}}_{33} \circledast \boldsymbol{Y}_{ij,3}(n) + \bar{\boldsymbol{A}}_{43} \circledast \boldsymbol{Y}_{ij,4}(n) + \bar{z}_{ij,3}\right), \tag{6.11c}$$

$$y_{ij,4}(n+1) = f\left(\bar{\boldsymbol{A}}_{34} \circledast \boldsymbol{Y}_{ij,3}(n) + \bar{\boldsymbol{A}}_{44} \circledast \boldsymbol{Y}_{ij,4}(n) + \bar{\boldsymbol{A}}_{54} \circledast \boldsymbol{Y}_{ij,5}(n) + \bar{z}_{ij,4} + \bar{\boldsymbol{A}}_{14} \circledast \boldsymbol{Y}_{ij,1}(n)\right), \tag{6.11d}$$

$$y_{ij,5}(n+1) = f\left(\bar{\boldsymbol{A}}_{45} \circledast \boldsymbol{Y}_{ij,3}(n) + \bar{\boldsymbol{A}}_{55} \circledast \boldsymbol{Y}_{ij,4}(n) + \bar{\boldsymbol{A}}_{65} \circledast \boldsymbol{Y}_{ij,5}(n) + \bar{z}_{ij,5}\right), \tag{6.11e}$$

$$y_{ij,6}(n+1) = f\left(\bar{\boldsymbol{A}}_{56} \circledast \boldsymbol{Y}_{ij,3}(n) + \bar{\boldsymbol{A}}_{66} \circledast \boldsymbol{Y}_{ij,4}(n) + \bar{z}_{ij,6}\right) \tag{6.11f}$$

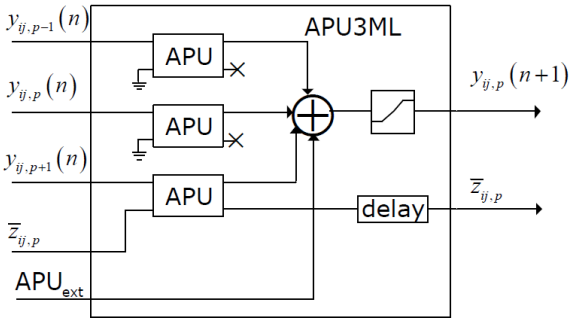
Consequently, an extra addition operation is added to both (6.11a) and (6.11d), in order to support applications with far–neighbourhood interconnections. To avoid clutter, the closed form of a specialized multi–layer DT CNN that has also several far–neighbourhood templates is not shown in this section as it is discussed in the next section in terms of processing units.

6.3.2 Design of a Multi–Layer Processor

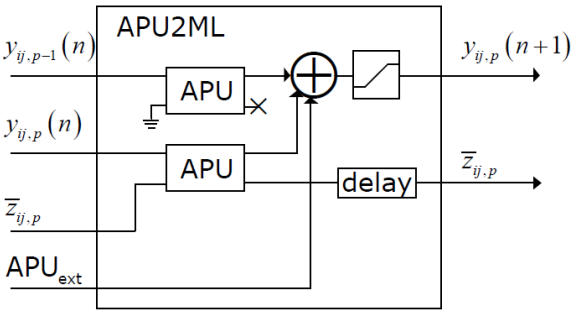
(6.8) can be rewritten in the open form similar to (6.10) indicating the functions of each APU and BPU:

$$
\begin{aligned}
y_{ij,1}(n+1) &= \underbrace{f\Bigg(\overbrace{\bar{A}_{11}\circledast \boldsymbol{Y}_{ij,1}(n)}^{\mathbf{APU_{11}}}+\overbrace{\bar{A}_{21}\circledast \boldsymbol{Y}_{ij,2}(n)+\underbrace{\bar{B}_{11}\circledast \boldsymbol{U}_{ij,1}+\bar{z}_{ij,1}}_{\mathbf{BPU}}}^{\mathbf{APU_{21}}}\Bigg)}_{\mathbf{addition\ and\ saturation}}\\
y_{ij,2}(n+1) &= \underbrace{f\Bigg(\overbrace{\bar{A}_{12}\circledast \boldsymbol{Y}_{ij,1}(n)}^{\mathbf{APU_{12}}}+\overbrace{\bar{A}_{22}\circledast \boldsymbol{Y}_{ij,2}(n)}^{\mathbf{APU_{22}}}+\overbrace{\bar{A}_{32}\circledast \boldsymbol{Y}_{ij,3}(n)+\bar{z}_{ij,2}}^{\mathbf{APU_{32}}}\Bigg)}_{\mathbf{addition\ and\ saturation}}\\
y_{ij,3}(n+1) &= \underbrace{f\Bigg(\overbrace{\bar{A}_{23}\circledast \boldsymbol{Y}_{ij,2}(n)}^{\mathbf{APU_{23}}}+\overbrace{\bar{A}_{33}\circledast \boldsymbol{Y}_{ij,3}(n)}^{\mathbf{APU_{33}}}+\overbrace{\bar{A}_{43}\circledast \boldsymbol{Y}_{ij,4}(n)+\bar{z}_{ij,3}}^{\mathbf{APU_{43}}}\Bigg)}_{\mathbf{addition\ and\ saturation}}\\
y_{ij,4}(n+1) &= \underbrace{f\Bigg(\overbrace{\bar{A}_{34}\circledast \boldsymbol{Y}_{ij,3}(n)}^{\mathbf{APU_{34}}}+\overbrace{\bar{A}_{44}\circledast \boldsymbol{Y}_{ij,4}(n)}^{\mathbf{APU_{44}}}+\overbrace{\bar{A}_{54}\circledast \boldsymbol{Y}_{ij,5}(n)+\bar{z}_{ij,4}}^{\mathbf{APU_{54}}}\Bigg)}_{\mathbf{addition\ and\ saturation}}\\
&\vdots \qquad\qquad\qquad \vdots\\
y_{ij,p}(n+1) &= \underbrace{f\Bigg(\overbrace{\bar{A}_{p-1p}\circledast \boldsymbol{Y}_{ij,p-1}(n)}^{\mathbf{APU_{p\text{-}1p}}}+\overbrace{\bar{A}_{pp}\circledast \boldsymbol{Y}_{ij,p}(n)}^{\mathbf{APU_{pp}}}+\overbrace{\bar{A}_{p+1p}\circledast \boldsymbol{Y}_{ij,p+1}(n)+\bar{z}_{ij,p}}^{\mathbf{APU_{p+1p}}}\Bigg)}_{\mathbf{addition\ and\ saturation}}\\
&\vdots \qquad\qquad\qquad \vdots\\
y_{ij,P-1}(n+1) &= \underbrace{f\Bigg(\overbrace{\bar{A}_{P-2P-1}\circledast \boldsymbol{Y}_{ij,P-2}(n)}^{\mathbf{APU_{P\text{-}2P\text{-}1}}}+\overbrace{\bar{A}_{P-1P-1}\circledast \boldsymbol{Y}_{ij,P-1}(n)}^{\mathbf{APU_{P\text{-}1P\text{-}1}}}+\\
&\qquad \overbrace{\bar{A}_{PP-1}\circledast \boldsymbol{Y}_{ij,P}(n)+\bar{z}_{ij,P-1}}^{\mathbf{APU_{PP\text{-}1}}}\Bigg)}_{\mathbf{addition\ and\ saturation}}\\
y_{ij,P}(n+1) &= \underbrace{f\Bigg(\overbrace{\bar{A}_{P-1P}\circledast \boldsymbol{Y}_{ij,P-1}(n)}^{\mathbf{APU_{P\text{-}1P}}}+\overbrace{\bar{A}_{PP}\circledast \boldsymbol{Y}_{ij,P}(n)+\bar{z}_{ij,P}}^{\mathbf{APU_{PP}}}\Bigg)}_{\mathbf{addition\ and\ saturation}}
\end{aligned}
\tag{6.12}
$$

The processor computations and the output functions in (6.12) can be generalized as



(a) 3–input APU sub–block for multi–layer RTCNNP core



(b) 2–input APU sub–block for multi–layer RTCNNP core

Figure 6.4 Sub–blocks of multi–layer RTCNNP core

(6.13) which indicates the computation flow of each process for an iteration.

$$
\begin{aligned}
&\text{BPU} : g_{ij} = \bar{\boldsymbol{B}}_{11} \circledast \boldsymbol{U}_{ij,1} + \bar{z}_{ij,1} \\
&\text{APU}_{p-1p}(n) : h_{ij,p}(n) = \begin{cases} 0 & p = 1, \\ \bar{\boldsymbol{A}}_{p-1p} \circledast \boldsymbol{Y}_{ij,p-1}(n) & \text{otherwise}, \end{cases} \\
&\text{APU}_{pp}(n) : l_{ij,p}(n) = \bar{\boldsymbol{A}}_{pp} \circledast \boldsymbol{Y}_{ij,p}(n) + I_{ij,p} \\
&I_{ij,p} = \begin{cases} g_{ij} & p = 1, \\ \bar{z}_{ij,p} & \text{otherwise}, \end{cases} \\
&\text{APU}_{p+1p}(n) : r_{ij,p}(n) = \begin{cases} 0 & p = P, \\ \bar{\boldsymbol{A}}_{p+1p} \circledast \boldsymbol{Y}_{ij,p+1}(n) & \text{otherwise}, \end{cases} \\
&y_{ij,p}(n) = f\left(x_{ij,p}(n)\right) = f\left(h_{ij,p}(n) + l_{ij,p}(n) + r_{ij,p}(n)\right)
\end{aligned}
\tag{6.13}
$$

$3P-2$ APUs are required for an iteration. Since the difference equations are independent from each other in the model given in (6.12), the design of the architecture can be real-

ized with simple sub–blocks instead of one complex block. It is obvious that the layers in the middle of (6.12) contain three APU blocks while the layers at the first and last row of (6.12) contain two. The processor named 3–input APU for multi–layer DT CNN (APU3ML), which is labeled $\mathrm{APU}_{pp}(n)$ in (6.13), is designed for the layers between the first and last layers of the multi–layer DT CNN. Similarly, the processors named 2–input APU for multi–layer DT CNN (APU2ML), which are labeled $\mathrm{APU}_{p-1p}(n)$ and $\mathrm{APU}_{p+1p}(n)$ in (6.13), are designed for the first and last layers of the architecture, respectively. The simplified block diagrams of APU3ML and APU2ML sub–blocks are given in Figure 6.4a and Figure 6.4b respectively. As a matter of fact, the APU2L block given in Figure 6.2 contains two APU2ML sub–blocks. The introduction of APU3ML and APU2ML sub–blocks is the result of dividing the computation among the different layers. These two sub–blocks both have APU_{ext} input which can be used to take far–neighbourhood inter–layer feedback template result from a single APU, if needed.

In order to construct a new block to be used in the processor array of a multi–layer RTCNNP, the APU3ML and APU2ML sub–blocks are connected to each other like shown in Figure 6.5, which is called an APU for multi–layer DT CNN (APUML). An APUML block can be modified to realize far–neighbourhood inter–layer connections using extra APU blocks. An example of this can be given by adding an extra feedback template to (6.12). To avoid clutter, only one extra template is added in (6.14), hence the modified APUML block is given in Figure 6.6. Note that, the APU_{ext} input is inside the APUML block and it is not an external input for the APUML block. In this example, it is assumed that there is a feedback template relating the second layer to the p^{th} layer. This assumption also can be generalized as a feedback template relating the q^{th} layer to the

p^{th} layer.

$$
\begin{aligned}
y_{ij,1}(n+1) &= \underbrace{f\Big(\overbrace{\bar{A}_{11}\circledast \boldsymbol{Y}_{ij,1}(n)}^{\text{APU}_{11}}+\overbrace{\bar{A}_{21}\circledast \boldsymbol{Y}_{ij,2}(n)+\underbrace{\bar{B}_{11}\circledast \boldsymbol{U}_{ij,1}+\bar{z}_{ij,1}}_{\text{BPU}}}^{\text{APU}_{21}}\Big)}_{\text{addition and saturation}} \\
y_{ij,2}(n+1) &= \underbrace{f\Big(\overbrace{\bar{A}_{12}\circledast \boldsymbol{Y}_{ij,1}(n)}^{\text{APU}_{12}}+\overbrace{\bar{A}_{22}\circledast \boldsymbol{Y}_{ij,2}(n)}^{\text{APU}_{22}}+\overbrace{\bar{A}_{32}\circledast \boldsymbol{Y}_{ij,3}(n)+\bar{z}_{ij,2}}^{\text{APU}_{32}}\Big)}_{\text{addition and saturation}} \\
y_{ij,3}(n+1) &= \underbrace{f\Big(\overbrace{\bar{A}_{23}\circledast \boldsymbol{Y}_{ij,2}(n)}^{\text{APU}_{23}}+\overbrace{\bar{A}_{33}\circledast \boldsymbol{Y}_{ij,3}(n)}^{\text{APU}_{33}}+\overbrace{\bar{A}_{43}\circledast \boldsymbol{Y}_{ij,4}(n)+\bar{z}_{ij,3}}^{\text{APU}_{43}}\Big)}_{\text{addition and saturation}} \\
y_{ij,4}(n+1) &= \underbrace{f\Big(\overbrace{\bar{A}_{34}\circledast \boldsymbol{Y}_{ij,3}(n)}^{\text{APU}_{34}}+\overbrace{\bar{A}_{44}\circledast \boldsymbol{Y}_{ij,4}(n)}^{\text{APU}_{44}}+\overbrace{\bar{A}_{54}\circledast \boldsymbol{Y}_{ij,5}(n)+\bar{z}_{ij,4}}^{\text{APU}_{54}}\Big)}_{\text{addition and saturation}} \\
&\vdots \qquad\qquad \vdots \\
y_{ij,p}(n+1) &= f\Big(\overbrace{\bar{A}_{p-1p}\circledast \boldsymbol{Y}_{ij,p-1}(n)}^{\text{APU}_{\text{p-1p}}}+\overbrace{\bar{A}_{pp}\circledast \boldsymbol{Y}_{ij,p}(n)}^{\text{APU}_{\text{pp}}}+\overbrace{\bar{A}_{p+1p}\circledast \boldsymbol{Y}_{ij,p+1}(n)+\bar{z}_{ij,p}}^{\text{APU}_{\text{p+1p}}}\Big)+ \\
&\quad \underbrace{\overbrace{\bar{A}_{2p}\circledast \boldsymbol{Y}_{ij,2}(n)}^{\text{APU}_{\text{2p}}}}_{\text{addition and saturation}} \\
&\vdots \qquad\qquad \vdots \\
y_{ij,P-1}(n+1) &= f\Big(\overbrace{\bar{A}_{P-2P-1}\circledast \boldsymbol{Y}_{ij,P-2}(n)}^{\text{APU}_{\text{P-2P-1}}}+\overbrace{\bar{A}_{P-1P-1}\circledast \boldsymbol{Y}_{ij,P-1}(n)}^{\text{APU}_{\text{P-1P-1}}}+ \\
&\quad \underbrace{\overbrace{\bar{A}_{PP-1}\circledast \boldsymbol{Y}_{ij,P}(n)+\bar{z}_{ij,P-1}}^{\text{APU}_{\text{PP-1}}}\Big)}_{\text{addition and saturation}} \\
y_{ij,P}(n+1) &= \underbrace{f\Big(\overbrace{\bar{A}_{P-1P}\circledast \boldsymbol{Y}_{ij,P-1}(n)}^{\text{APU}_{\text{P-1P}}}+\overbrace{\bar{A}_{PP}\circledast \boldsymbol{Y}_{ij,P}(n)+\bar{z}_{ij,P}}^{\text{APU}_{\text{PP}}}\Big)}_{\text{addition and saturation}}
\end{aligned}
\tag{6.14}
$$

For N iterations, N APUML blocks forming a cascade are required as given in Figure 6.7. For a P layer APUML design P sub–blocks, which is composed of 2 APU2ML sub–blocks and $P-2$ APU3ML sub–blocks, are needed. The introduction of these sub–blocks

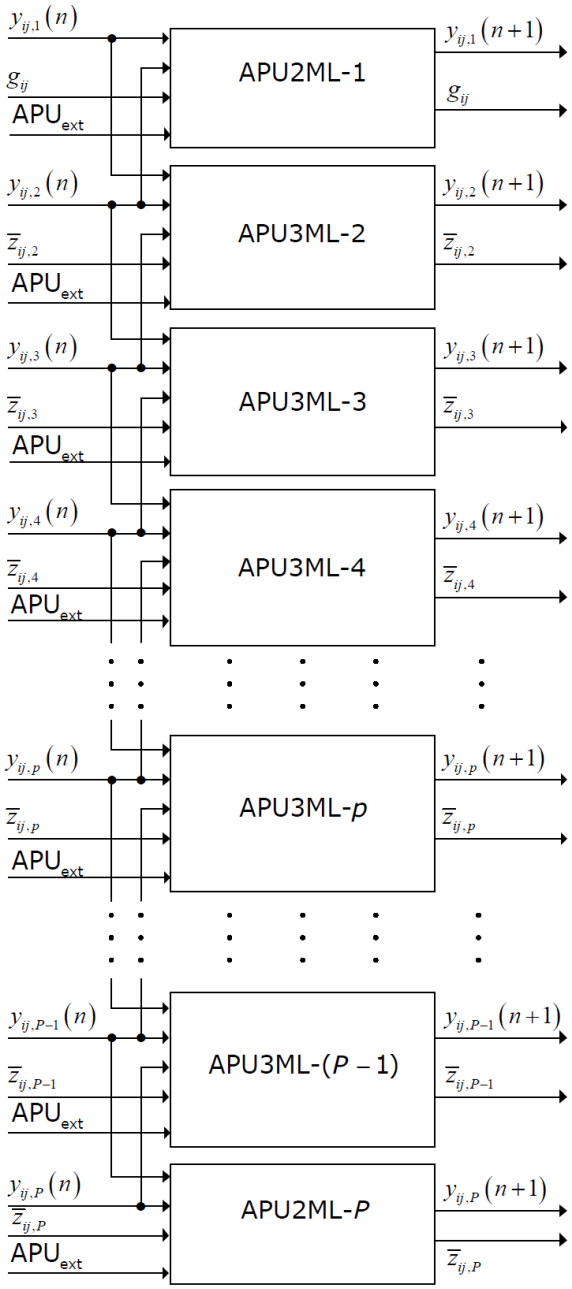


Figure 6.5 Processor core of the multi–layer RTCNNP architecture

make the design process much more easier. The number of xPUs in this processor array is $(3P-2)N+1$, excluding any far–neigbourhood interconnections and having one input and one feed–forward template at the first layer, i.e., $(3P-2)N$ APUs and 1 BPU. Since the BPU is used once before the APUML array, the number of BPUs can be increased up to $3P-2$ if needed. In this case, the number of xPUs in the processor array is $(3P-2)(N+1)$, excluding any far–neigbourhood interconnections. This modification can be done by using the same design steps, which are used for the APU blocks for creating a APUML

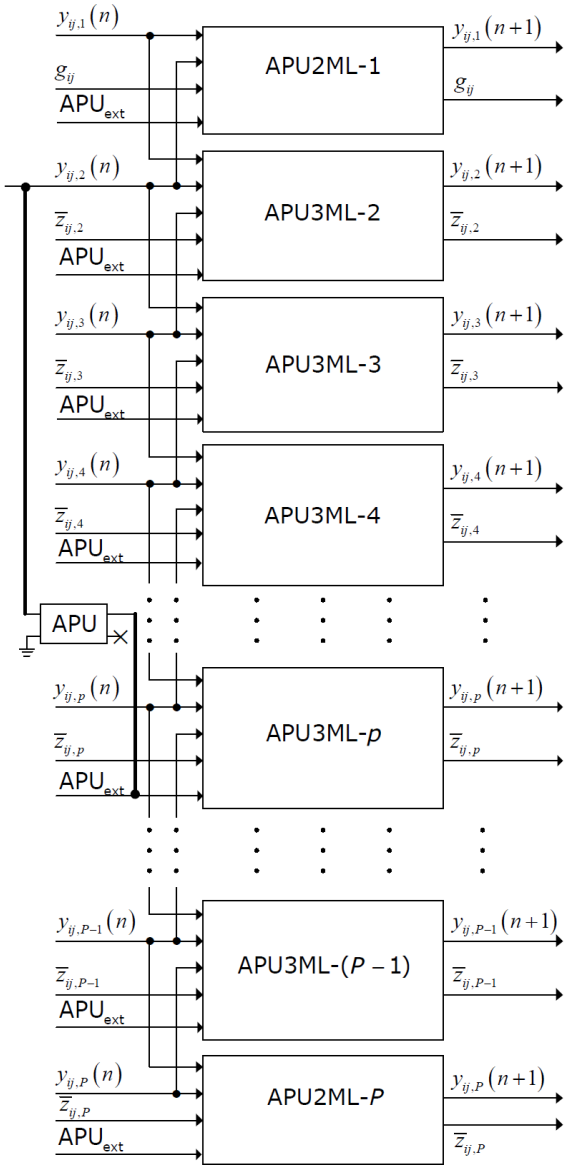


Figure 6.6 Extended processor core of the multi–layer RTCNNP architecture

block, for the BPU blocks and creating a BPU for multi–layer DT CNN (BPUML). The extended version of the processor array is given in Figure 6.8 including the BPUML block. If it is assumed that a and b extra feedback and feedforward templates are required for far–neigbourhood interconnections respectively, then the number of xPUs in the processor array is $(3P-2)(N+1)+aN+b$. In this case, there are $(3P-2+a)N$ APUs and $3P-2+b$ BPUs. The working principle of this processor array is also similar to a two–layer RTCNNP processor array.

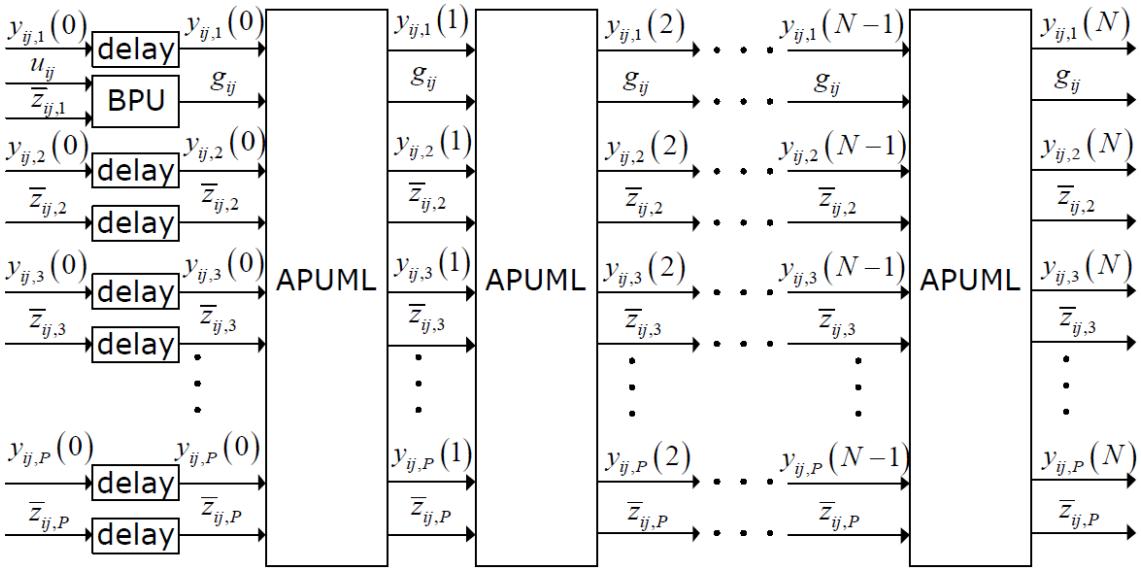


Figure 6.7 Processor array of the multi–layer RTCNNP architecture

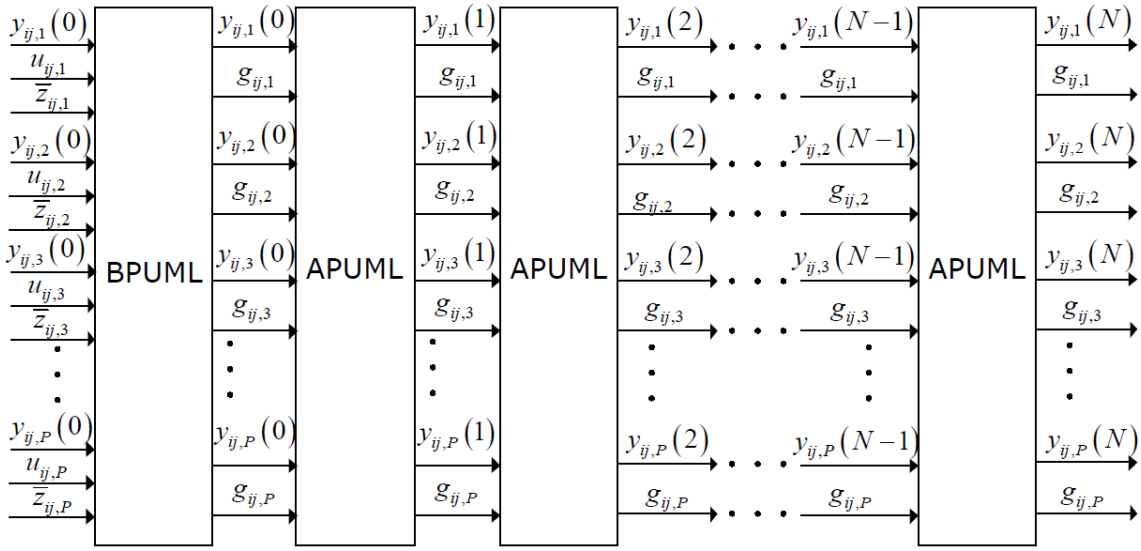


Figure 6.8 Extended processor array of the multi–layer RTCNNP architecture

6.4 Overview of the FPGA implementation

In this section, support blocks of the FPGA implementation are given in order to show the complete architecture. These support blocks are the same as that of the RTCNNP–v2 system, hence one can refer to [6] for details.

The inner blocks are either two–layer or multi–layer cores explained in Section 6.2 and Section 6.3.2. For the sake of simplicity, a two–layer CNN architecture with its support blocks is chosen as an example. A two–layer CNN video processor unit (VPU2L) is given

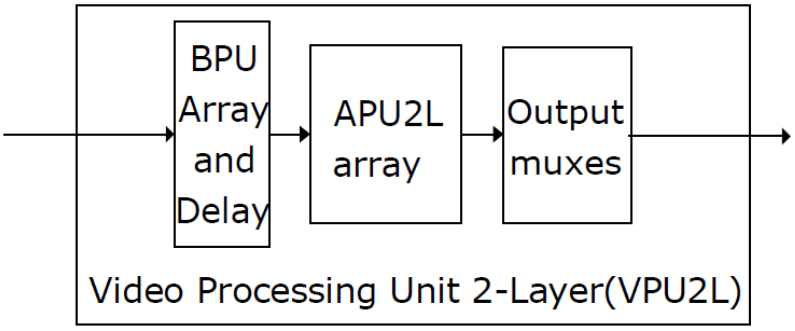


Figure 6.9 Two–layer video processing unit

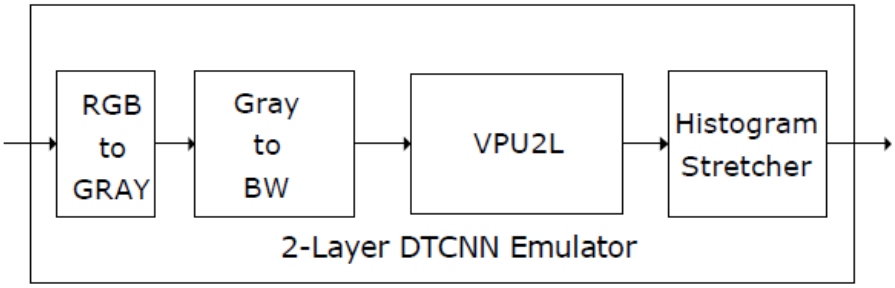


Figure 6.10 Two–layer DT CNN emulator

in Figure 6.9. To summarize, a VPU2L contains a BPU array, delay blocks, an APU2L array and output muxes. BPU array, delay blocks and APU2L array are explained in Section 6.2. Output muxes route the output data to the next block.

One upper block of VPU2L is a two–layer DT CNN emulator (Figure 6.10). This block contains RGB to Gray, Gray to BW and histogram stretcher blocks along with the VPU2L block. Of these, RGB to Gray is mandatory while the others can be configured for specific tasks or bypassed for testing purposes.

The topmost block of the FPGA implementation is the two–layer RTCNNP (Figure 6.11). It contains the DVI input and DVI output blocks which gets the data from video receiver hardware and gives the processed data to the video transmitter hardware, respectively.

As discussed in [15, 16, 6], both systems, which have two– and multi–layer RTCNNP architectures, are designed to capture a progressive video stream, process it with two– or multi–layer DT CNN, and convert the result to a progressive video stream again. The simplified block diagram of this system is given in Figure 6.12.

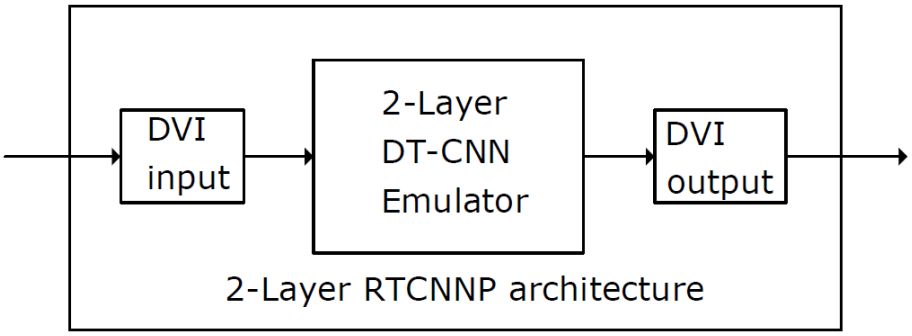


Figure 6.11 Two–layer RTCNNP

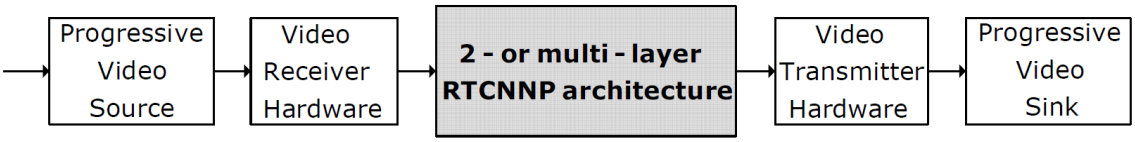


Figure 6.12 Simplified block diagram of the whole system

6.5 Comparisons of the Proposed Multi–Layer CNN Architecture with the Multi–Layer Falcon Architecture

In this section, the multi–layer Falcon architecture [9, 32], which is discussed in Section 4.2, is compared to the multi–layer RTCNNP architecture, proposed in Section 6.3.2.

Both structures support real–time operation and use the FSR model of DT CNN. These two issues can be considered as the similarities of these structures.

In [32], it is reported that the multi–layer Falcon architecture emulates a general multi–layer CNN array where every layer is connected together in all possible ways while the multi–layer RTCNNP architecture emulates a special multi–layer CNN array where each layer is connected to their nearest neigbour layers by the feedback cloning templates. Therefore, for a P–layer CNN design, the multi–layer Falcon architecture is capable of having a maximum number of $2P^2$ templates however the multi–layer RTCNNP architecture is capable of having a maximum number of $6P-4+a+b$ templates. Although the proposed multi–layer RTCNNP architecture is based on a specialized model, it can be modified to realize far–neighbourhood interconnections by adding xPU blocks for the required templates depending on the application as given in Section 6.3.2, if needed.

The block diagram design of the multi–layer Falcon architecture is very different from the

multi–layer RTCNNP architecture. Single–layer processor core of Falcon architecture, which is given in Figure 4.6, is redesigned by means of a different arithmetic-unit structure in order to create the multi–layer processor core, which is given in Figure 4.9. However, single–layer RTCNNP-v2 [16] core is not redesigned and used only by bypassing a block in its structure to create multi–layer processor core.

In the multi–layer Falcon architecture, rows and columns of the 2–D processor array of core processors respectively process stripes of the 2–D input and carry out the iterations. On the other hand, in the multi–layer RTCNNP architecture all of the 2–D input image is row–wise packed, layers are created in the columns and iterations are processed in the rows of the 2–D processor array of core processors. In other words, the multi–layer Falcon architecture creates the layers via template selection whereas the the multi–layer RTCNNP architecture creates the layers via assigning separate processor blocks having the templates required for each layer.

Another difference is their control style. The multi–layer Falcon architecture uses global control like both single–layer Falcon and RTCNNP-v1 architectures [14], while the multi–layer RTCNNP uses local control as the RTCNNP-v2.

Finally, in [32], it is reported that the multi–layer Falcon architecture does not have a configurable VHDL description to generate layers generically whereas layer generation of multi–layer RTCNNP is reconfigurable.

CHAPTER 7

ANALYSIS, SIMULATION RESULTS AND IMPLEMENTATION RESULTS OF THE PROPOSED ARCHITECTURES

In this chapter, the chosen numeral system for the proposed architecture is discussed and simulation and implementation results are given.

7.1 Analysis of the Chosen Numeral System for the Proposed Architectures

Fixed–point arithmetic is preferred for the implementation, as bit widths of the signals are significantly larger when floating–point arithmetic is used, leading to a waste of resources. Moreover, floating–point arithmetic is mainly used in DSP devices since their software and hardware supports efficient floating–point computations. On the other hand, in this work the proposed architectures are designed to be implemented on an FPGA device, hence a fixed–point implementation would be much more efficient.

The total bit widths of data inputs of the two– and multi–layer RTCNNP emulators are designed to be configurable, like that of the RTCNNP–v2 emulator. In order to discuss the numeral system, let tbw denote the value of total bit width of the inputs. The tbw value is then separated into two parts by a dot. The left and right sides are called the integer and fractional parts which can be denoted as *int* and $frac$, respectively. The tbw and $frac$ values can be chosen according to the application hence $int = tbw - frac$. For example, if $tbw = 16$ and $frac = 9$, the fixed–point number is in 7.9 data format and since it is a signed number, the most significant bit indicates the sign. While these are the abbreviated forms to be used from this point to the end of the thesis, there is also an open form, which can be shown as $Xxxxxx.xxxxxxxxx$ where X represents the sign bit and x represents any other bit.

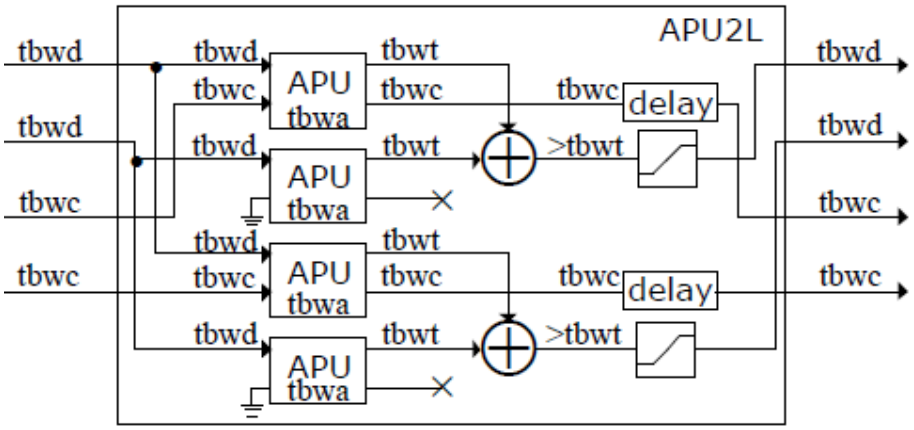


Figure 7.1 Bit width representation of an APU2L block (second approach)

Bit widths of an APU2L block are given in Figure 7.1. Bit widths of the data input, constant input and template coefficients are denoted as *tbwd*, *tbwc* and *tbwa*, respectively, which are independent from each other. There are not any specific method for choosing the values of these bit widths, hence the values are chosen according to the application desired. Note that, in the most general case, the bit width values of different APU blocks can also be different, hence for simplicity it is assumed that for all APUs, *tbwd*, *tbwc* and *tbwa* values are the same. There are multiplication and addition operations inside each APU, consequently, the bit width of the data output gets larger (*tbwt*). At the next stage, the data outputs of two APUs are summed, therefore the output of the addition block has a bit width larger than *tbwt*. Finally, the saturator block takes this signal as input and not only reduces the bit width value to *tbwd* but also applies the activation function to obtain the output. This is the data output of an APU2L block and it is given as an input to the next APU2L block for the next iteration. Note that, the *tbwt* value is a function of *tbwd*, *tbwc* and *tbwa* values.

Bit widths of the first APU2L architecture proposed in Section 6.2 are given in Figure 7.2. It is obviously seen that the intermediate *tbwt* signals taken from the data outputs of the APUs at the first stage are given to the constant inputs of the APUs at the second stage. As a result, constant RAM of the APUs at the second stage should be significantly larger.

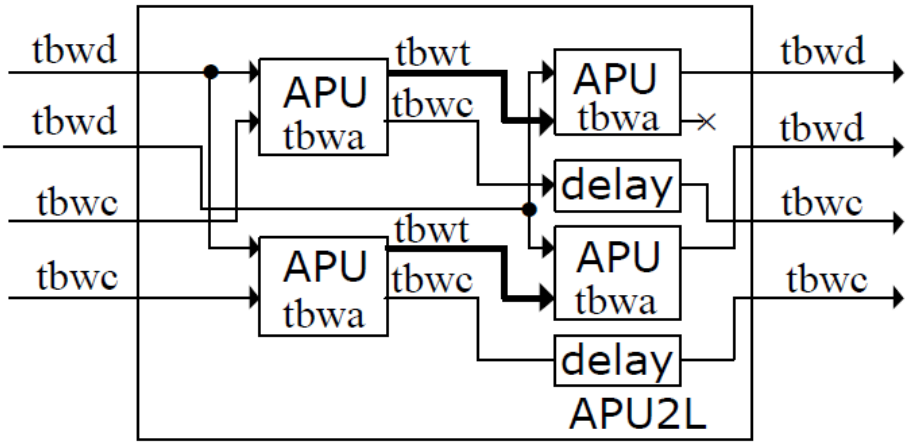


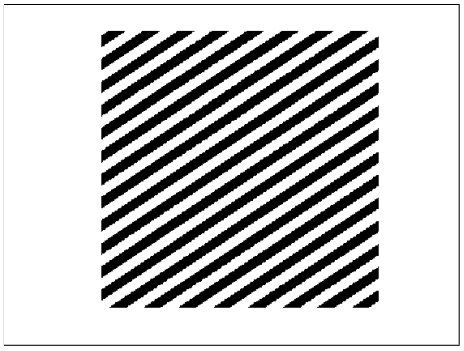
Figure 7.2 Bit width representation of APU2L block (first approach)

7.2 Simulation Results for the Proposed Architectures

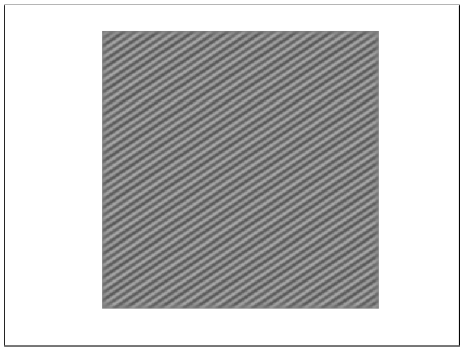
MATLAB simulations are carried out on a PC, with a 2.63 GHz AMD Phenom II X3 710 processor, 4 GB RAM and a 64–bit operating system. An m–file is written which simulates (2.13) using a configurable fixed–point numeral system.

Both the two– and multi–layer RTCNNP architectures are simulated on both MATLAB and Modelsim and the simulation results are compared to each other. The output images obtained from both simulators are the same. The video input image is given to the first layer and the output image is taken from one of the layers as there is one video input unit and one video output unit.

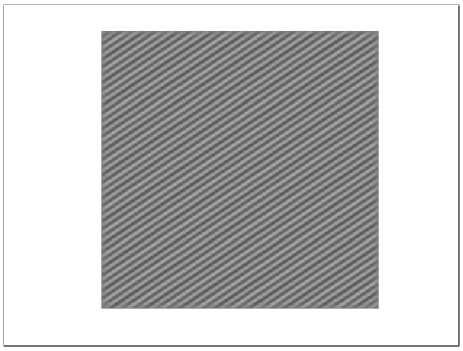
A CNN–based Gabor–type filter (GTF) proposed in [22] is chosen as an application example for the simulations. Note that, a CNN–based GTF implementation is also reported in [35], where the xPU core of the RTCNNP–v2 design is modified to realize an optimized version of the butterfly structure. It is observed that the same results are obtained by both implementations. In order to increase the speed of the simulation, a video input image that has a resolution of 256×256 is preferred. The input image is chosen as a striped image oriented in a specific angle. As a simulation example, the frequency parameters of the GTF are tuned to pass the third spatial harmonic of a specific oriented striped image, where the output of the first layer is obtained as the third harmonic of the input, and the output of the second layer is a spatially shifted version of the output of the first layer (Figure 7.3). The simulation is run for 24 iterations.



(a) A striped input image



(b) Output image of the 1st layer



(c) Output image of the 2nd layer

Figure 7.3 Two–layer CNN–based Gabor–type filter simulation results for a striped input image

Note that, a CNN–based GTF is a linear system whose $\boldsymbol{A}$– and $\boldsymbol{B}$–templates have many zero entries, which means that not all of the implemented primitives are tested with GTF simulations. Consequently, in order to test all parts of the design, the simulation is repeated many times with random templates with all non–zero entries. Furthermore, some simulations are carried out for images with 1080×1920 resolution and successful operation of the design at Full–HD 1080p is verified.

7.3 Implementation Results for the Proposed Architectures

The APU2L block and APU2ML and APU3ML sub–blocks are synthesized on a Quartus II 10.0 software, where the *Register Transfer Level* (RTL) schematics given in Figure 7.4, Figure 7.5 and Figure 7.6 are obtained, respectively.

A two–layer RTCNNP architecture is implemented on an Altera Stratix IV GX 230 FPGA device, whose resource usage statistics are given in Table 7.1. The maximum number of iterations that can be implemented on an Altera Stratix IV GX 230 FPGA device depends

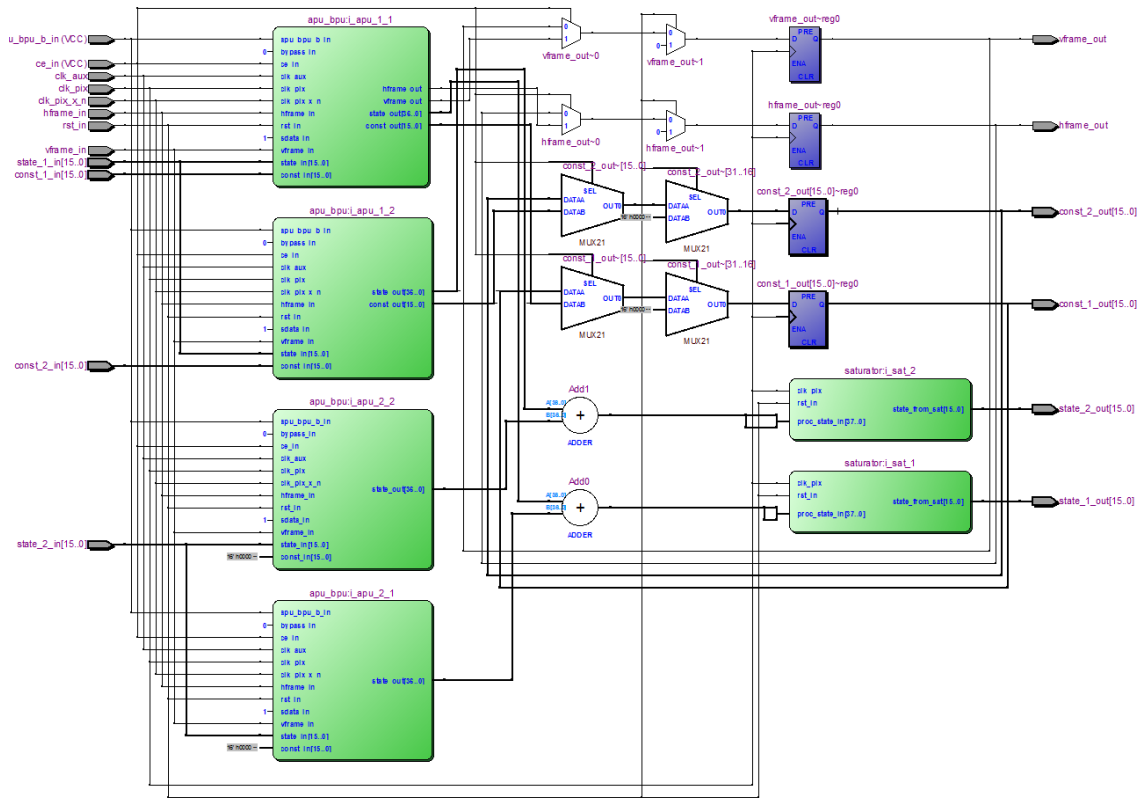


Figure 7.4 RTL schematic of APU2L block

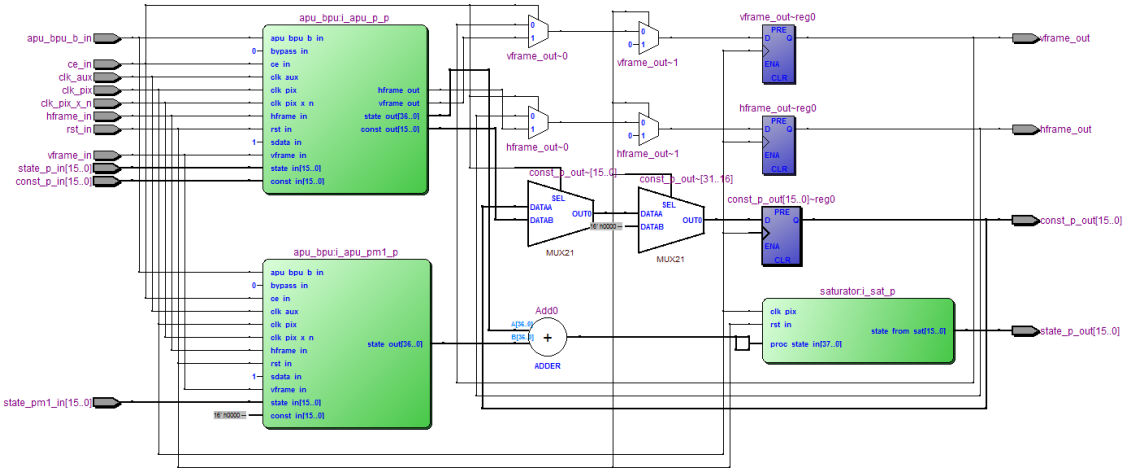


Figure 7.5 RTL schematic of APU2ML sub–block

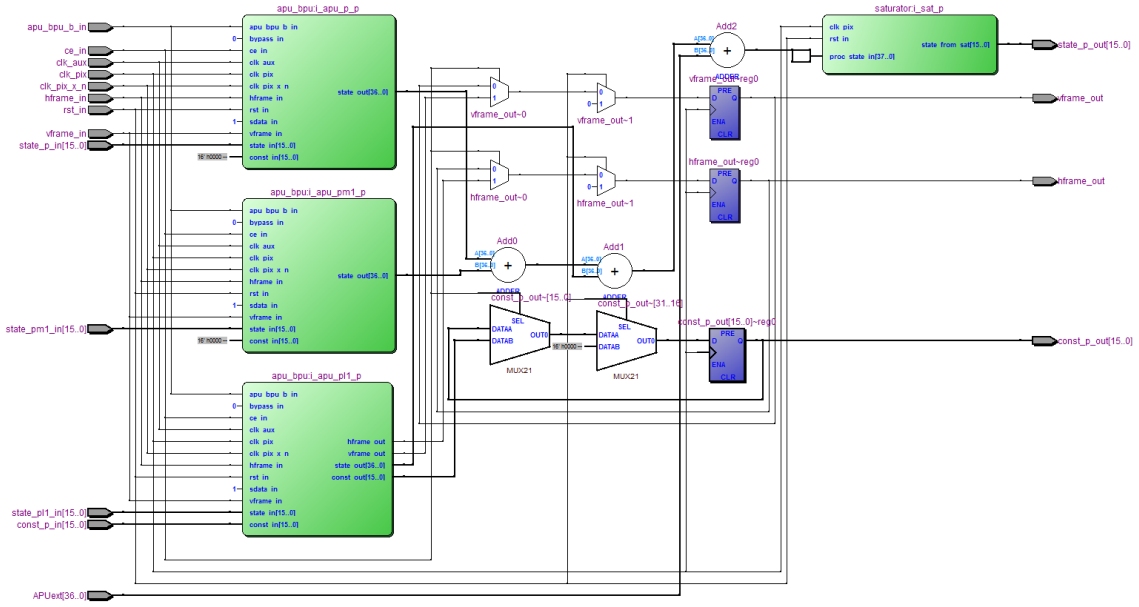


Figure 7.6 RTL schematic of APU3ML sub–block

on many reconfigurable parameters of the design: fixed–point bit widths of constants, states and template coefficients, maximum number of visible pixels configured for a single line, sizes of the templates and clock multipliers of the APUs. For respective fixed–point bit widths of 16, 16 and 18, pixel count support of a line of 2048, 3×3 templates and clock multipliers of 1, the maximum number of iterations that can fit in a single Altera Stratix IV GX 230 FPGA device is 24. For a P–layer RTCNNP implementation on the same device it can be estimated as $\lfloor 100/(3P-2+a) \rfloor$. For example, for P values of 3, 4 and 5, the maximum number of iterations that can fit on the same FPGA device are 14, 10 and 7, respectively. Estimations for different FPGA vendors and/or families can be done by comparing their maximum number of resources given in their data manuals with the implementation results given in this section.

The maximum pixel clock frequency estimated by the Quartus II software is 225.68 MHz with no special optimization. Note that, the maximum clock rate can be increased with further optimization, if desired. Furthermore, for full–HD 1080p@60, the visible pixel rate is 124.4 MP/s and the pixel clock frequency is 148.5 MHz, hence the maximum throughput of the prototype is restricted to 124.4 MP/s by the DVI I/O.

In the RTCNNP–v2 architecture [15, 16, 6], a single APU is used for an iteration, hence

Table 7.1 Resource usage of the 2–layer RTCNNP architecture

Device	**EP4SGX230KF40C2**	
Number of iterations	**1**	**24**
Number of APUs	**4**	**4 × 24**
Combinational ALUTs	11,036 / 182,400 (6%)	110,190 / 182,400 (60%)
Memory ALUTs	60 / 91,200 (< 1%)	0 / 91,200 (0%)
Dedicated logic registers	14,888 / 182,400 (8%)	145,793 / 182,400 (80%)
Logic utilization	9%	90%
Total registers	14888	163259
Total block memory bits	766,456 / 14,625,792 (5%)	9,832,196 / 14,625,792 (67%)
DSP block 18–bit elements	82 / 1,288 (6%)	1,002 / 1,288 (78%)

the maximum number of iterations that can be implemented on the same FPGA device with the same reconfigurable parameters is 100 for full–HD 1080p@60. In [35] the implemented core is highly optimized for a GTF, hence it is considerably smaller than the two–layer CNN architecture proposed in this thesis, therefore the maximum number of iterations implemented on the same FPGA device is 100 for the same parameters.

Finally, the class of the proposed architecture is completely different from that of the MLCNN structures found in the literature, as layer neighborhood is a new phenomena proposed in this thesis. Consequently, comparing different architectures will be difficult and the comparison result will not be feasible.

CHAPTER 8

RESULTS AND DISCUSSION

In this thesis, first, a novel architecture to emulate a two–layer DT CNN is proposed, which is based on the RTCNNP–v2 architecture proposed in [6]. The reconfigurability, flexibility and reusability properties of the RTCNNP–v2 architecture is preserved in this design. Second, the proposed two–layer DT CNN emulator is generalized to emulate a specialized discrete–time multi–layer CNN. The specialization method of the traditional multi–layer CNN is also proposed in this thesis by defining a layer–neighbourhood measure. The layer–neighbourhood concept cuts off the resources required to implement a multi–layer DT CNN significantly, from P^2 template dot product operations to $(1+2R)P-2$ for each Euler iteration. However, the proposed measure is not readily suitable for many multi–layer CNN structures in the literature. Consequently the specialized architecture is modified to be expendable to support far–neighbourhood interconnections.

The most original contribution of this thesis is the method of extending the RTCNNP–v2 to two– and multi–layer DT CNN architectures. To this end, new topologies of the xPUs of the RTCNNP–v2 architecture is proposed, where the computation is divided not only in the time domain as in [6], but also in the layer domain. In other words, different processing units are assigned to each inter– and intra–layer template dot product operations. Furthermore, the fully pipelined architecture of the RTCNNP–v2 is preserved.

The proposed real–time two–layer DT CNN architecture is implemented on an Altera Stratix IV GX 230 FPGA device and tested up to 24 iterations. The number of iterations is only limited by the hardware resources of the FPGA device, hence it is trivial to increase the number by using more FPGA devices and connecting them end to end. The timing

analysis results show that the theoretical maximum pixel clock frequency of the prototype is 225.68 MHz. However, like many state of the art visual systems, the video I/O interface used in the design support only up to Full–HD@60 (1920×1080 resolution at 60 Hz frame rate) with a pixel clock frequency of 148.5 MHz, consequently, the frequency limit is not an issue in most cases. In the case of Full–HD@60, the pixel rate and the throughput of the system is 124.4 MP/s, and the total computation power of the system is 124.65 Giga multiplication operations per second for 24 iterations.

For future work, first, some minor reconfigurability issues of the implemented prototype can be solved. Second, multi–layer DT CNN algorithms can be implemented. Third, the upper frequency limit can be eliminated by dividing the computation in a spatial domain as well.

REFERENCES

[1] Chua, L. and Yang, L., (1988). "Cellular Neural Networks: Theory", Circuits and Systems, IEEE Transactions on, 35: 1257-1272.

[2] Chua, L. and Roska, T., (2002). Cellular Neural Networks and Visual Computing: Foundations and Applications, Cambridge University Press.

[3] Galan, R., Jimenez-Garrido, F., Dominguez-Castro, R., Espejo, S., Roska, T., Rekeczky, C., Petras, I. and Rodriguez-Vazquez, A., (2003). "A bio-inspired two-layer mixed-signal flexible programmable chip for early vision", Neural Networks, IEEE Transactions on, 14: 1313-1336.

[4] Carmona, R., Jimenez-Garrido, F., Dominguez-Castro, R., Espejo, S. and Rodriguez-Vazquez, A., (2002). "CMOS realization of a 2-layer CNN universal machine chip", Cellular Neural Networks and Their Applications, 2002. (CNNA 2002). Proceedings of the 2002 7th IEEE International Workshop on, 2002.

[5] Rekeczky, C., Serrano-Gotarredona, T., Roska, T. and Rodriguez-Vazquez, A., (2000). "A stored program 2nd order/3-layer complex cell CNN-UM", Cellular Neural Networks and Their Applications, 2000. (CNNA 2000). Proceedings of the 2000 6th IEEE International Workshop on, 2000.

[6] Yıldız, N., (2013). Design of A Cellular Neural Network Emulator and Its Implementation on An FPGA Device, PhD Thesis, Yildiz Technical University, Graduate School of Natural and Applied Sciences, İstanbul.

[7] Shi, B., (2003). "Cortically-inspired visual processing with a four layer cellular neural network", Neural Networks, 2003. Proceedings of the International Joint Conference on, 2003.

[8] Rodríguez-Vázquez, n., Domínguez-Castro, R., Jiménez-Garrido, F., Morillas, S., Listán, J., Alba, L., Utrera, C., Espejo, S. and Romay, R., (2008). The Eye-RIS CMOS Vision System, Springer Netherlands.

[9] Nagy, Z. and Szolgay, P., (2003). "Configurable Multilayer CNN-UM Emulator on FPGA", Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on, 50: 774-778.

[10] Hidvegi, T., Keresztes, P. and Szolgay, P., (2002). "An Accelerated Digital CNN-UM (CASTLE) Architecture by Using the Pipe-Line Technique", Cellular Neural Networks and Their Applications, 2002. (CNNA 2002). Proceedings of the 2002 7th IEEE International Workshop on, July 2002.

[11] Martinez, J. J., Toledo, F. J., Fernandez, E. and Ferrandez, J. M., (2008). "A Retinomorphic Architecture Based on Discrete-Time Cellular Neural Networks Using Reconfigurable Computing", Neurocomputing, 71: 766-775.

[12] Yeniçeri, R. and Yalçın, M., (2008). "An implementation of 2D locally coupled relaxation oscillators on an FPGA for real-time autowave generation", Cellular Neural Networks and Their Applications, 2008. CNNA 2008. 11th International Workshop on, 2008.

[13] Pardo, F., Lopez, P. and Cabello, D., (2008). "DT-CNN emulator: 3D heat equation solver with applications on the non-destructive soil inspection", Cellular Neural Networks and Their Applications, 2008. CNNA 2008. 11th International Workshop on, 2008.

[14] Kayaer, K. and Tavşanoğlu, V., (2008). "A New Approach to Emulate CNN on FPGAs for Real Time Video Processing", Cellular Neural Networks and Their Applications, 2008. CNNA 2008. 11th International Workshop on, July 2008.

[15] Yıldız, N., Cesur, E. and Tavşanoğlu, V., (2010). "A New Control Structure for the Pipelined CNN Processor Arrays", Cellular Nanoscale Networks and Their Applications (CNNA), 2010 12th International Workshop on, February 2010.

[16] Cesur, E., Yıldız, N. and Tavşanoğlu, V., (2010). "Architecture of The Next Generation Real Time CNN Processor: RTCNNP-v2", Nonlinear Theory and its Applications (NOLTA), 2010 International Symposium on, September 2010.

[17] Karahaliloğlu, K. and Balkır, S., (2005). "Bio-inspired compact cell circuit for reaction-diffusion systems", Circuits and Systems II: Express Briefs, IEEE Transactions on, 52: 558-562.

[18] Kayaer, K., (2008). Gerçek Zamanlı Video İşleyen Yeni Bir Hücresel Sinir Ağı Emülatörü Tasarımı ve FPGA ile Gerçeklenmesi, PhD Thesis, Yildiz Technical University, Graduate School of Natural and Applied Sciences, İstanbul.

[19] Cesur, E., (2013). Gerçek Zamanlı Bir Hücresel Sinir Ağı Yapısının Tasarımı ve Bu Yapıyla Gabor Filtrelerinin FPGA Üzerinde Gerçeklenmesi, PhD Thesis, Yildiz Technical University, Graduate School of Natural and Applied Sciences, İstanbul.

[20] Yang, T., (2003). "Multi–layer Cellular Neural Networks: Theory and Applications to Modelling Nitric Oxide Diffusion in Nervous Systems", International Journal of Computational Cognition, 1: 1-23.

[21] Shi, B. E., (2006). "An eight layer cellular neural network for spatio-temporal image filtering: Research Articles", Int. J. Circuit Theory Appl., 34: 141–164.

[22] Shi, B., (1998). "Gabor–type filtering in space and time with cellular neural networks", Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on, 45: 121 -132.

[23] Arena, P., Branciforte, M., Di Bernardo, G., Lavorgna, M. and Occhipin, L., (2000). "Reaction-diffusion CNN chip. I. IC implementation", Circuits and Systems, 2000. Proceedings. ISCAS 2000 Geneva. The 2000 IEEE International Symposium on, 2000.

[24] Yalçın, M., (2008). "A Simple Programmable Autowave Generator Network for Wave Computing Applications", Circuits and Systems II: Express Briefs, IEEE Transactions on, 55: 1173-1177.

[25] Espejo, S., Rodriguez-Vazquez, A., Dominguez-Castro, R. and Carmona, R., (1994). "Convergence and Stability of the FSR CNN Model", Cellular Neural Networks and their Applications, 1994. CNNA-94., Proceedings of the Third IEEE International Workshop on, December 1994.

[26] Harrer, H., (1993). "Multiple layer discrete-time cellular neural networks using time-variant templates", Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on, 40: 191-199.

[27] Harrer, H., Nossek, J., Roska, T. and Chua, L., (1994). "A current-mode DTCNN universal chip", Circuits and Systems, 1994. ISCAS '94., 1994 IEEE International Symposium on, 1994.

[28] Martinez-Alvarez, J., Toledo-Moreo, J., Garrigos-Guerrero, J. and Ferrandez-Vicente, J., (2010). "A multi-FPGA distributed embedded system for the emulation of Multi-Layer CNNs in real time video applications", Cellular Nanoscale Networks and Their Applications (CNNA), 2010 12th International Workshop on, February 2010.

[29] Martinez-Alvarez, J., Toledo-Moreo, F. and Ferrandez-Vicente, J., (2007). "Discrete-Time Cellular Neural Networks in FPGA", Field-Programmable Custom Computing Machines, 2007. FCCM 2007. 15th Annual IEEE Symposium on, 2007.

[30] Martínez, J., Toledo, F. and Ferrández, J., (2003). "New emulated discrete model of CNN architecture for FPGA and DSP applications", Proceedings of the 7th International Work-Conference on Artificial and Natural Neural Networks: Part II: Artificial Neural Nets Problem Solving Methods, 2003.Berlin, Heidelberg.

[31] Malki, S., (2008). On Hardware Implementation of Discrete-Time Cellular Neural Networks, Ph.D. thesis, Lunds University, Sweden.

[32] Nagy, Z., (2007). Implementation of Emulated Digital CNN-UM Architecture on Programmable Logic Devices and its Applications, University of Pannonia, Department of Image Processing and Neurocomputing, .

[33] Alpay, M., Yıldız, N. and Tavşanoğlu, V., (2010). "Alanda Programlanabilen Kapı Dizileri ile İki Katmanlı Hücresel Sinir Ağı Gerçeklemesi", Akıllı Sistemlerde Yenilikler ve Uygulamaları Sempozyumu (ASYU), June 2010.

[34] Alpay, M., Yıldız, N. and Tavşanoğlu, V., (2012). "Görüntü İşleme Uygulamalarına Yönelik Gerçek Zamanlı İki Katmanlı Bir Hücresel Sinir Ağı Emülatörünün Gerçekleştirilmesi", Elektrik - Elektronik ve Bilgisayar Mühendisliği Sempozyumu (ELECO), November 2012.

[35] Cesur, E., Yıldız, N. and Tavşanoğlu, V., (2012). “On an Improved FPGA Implementation of CNN-Based Gabor-Type Filters”, Circuits and Systems II: Express Briefs, IEEE Transactions on, 59: 815-819.

CURRICULUM VITAE

PERSONAL INFORMATION

Name Surname : Murathan ALPAY

Date of birth and place : March 24, 1981 – İstanbul

Foreign Languages : English

E-mail : murathan1981@gmail.com

EDUCATION

Degree	Department	University	Date of Graduation
Masters	Electronics	Yıldız Technical University	2007
Undergraduate	Electronics and Communications Engineering	Yıldız Technical University	2004
High School	Science–Mathematics	Kadıköy Anatolian High School	2000

WORK EXPERIENCE

Year	Corporation/Institution	Job Description
2005–2013	Yıldız Technical University	Research Assistant

PUBLISHMENTS

Conference Papers

1. Alpay. M., Yıldız. N. Tavşanoğlu. V., (2012). "Görüntü İşleme Uygulamalarına Yönelik Gerçek Zamanlı İki Katmanlı Bir Hücresel Sinir Ağı Emülatörünün Gerçekleştirilmesi", Elektrik - Elektronik ve Bilgisayar Mühendisliği Sempozyumu (ELECO), November 2012.

2. Alpay. M., Yıldız. N. Tavşanoğlu. V., (2010). "Alanda Programlanabilen Kapı Dizileri ile İki Katmanlı Hücresel Sinir Ağı Gerçeklemesi", Akıllı Sistemlerde Yenilikler ve Uygulamaları Sempozyumu (ASYU), June 2010.

Projects

1. Scholar, “Durağan Ve Video Görüntü İşleyen Hücresel Sinir Ağı Yapısının Yeni Bir Fpga Mimarisi İle Tasarım Ve Gerçeklemesi,” TÜBİTAK, 108E023, 2009–2011

2. Scholar, “Parmak izi tanıyan hızlı bir sistemin hücresel analog işlemci dizileri kullanarak tasarımı ve uyarlanması,” Yıldız Technical University BAPK, 25-04-03-01, 2005–2008