REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE YAHIA FARES DE MEDEA



Faculté de Technologie

Département du Génie Electrique

Mémoire de fin d'études de Master

Filière : télécommunications

Spécialité : systèmes de télécommunications

Thème

Contribution à l'implémentation d'un réseau de neurones convolutif (CNN) sur circuit FPGA

Présenté par :

Bensebsi ahmed

Lamri soufiane

Dirigé par :

Mr. Dali Mohammed

Promotion 2021

Remerciement

Avant tout nous tenons à remercier ALLAH qui nous a aidé et nous a donné la patience et le courage de finir le mémoire.

Évidemment, nous voulons commencer nos remerciements en montrant notre profond respect et notre gratitude à notre promoteur Mr **DALI Mohammed,** pour son encadrement et ses conseils patiemment pour ce travail.

Nous remercions les membres du jury qui nous ont fait l'honneur de juger ce travail.

Nos remerciements s'adressent aussi à tous les enseignants du département de Génie électrique de l'Université de Médéa

Nous remercions nos familles, spécialement, nos parents pour leur soutien inconditionnel, sans oublier nos frères et proches amis qui ont été si présent, et n'ont jamais cessé de nous encourager.

Dédicaces

Nous dédions ce modeste travail à :

Nos chers parents que Dieu nos les garde,

Nos frères et sœurs,

Toute les membres de nos familles

Tous ceux qui nous sont chers.

Résumé.

Ce travail est une contribution à l'implémentation sur circuit configurable de type FPGA d'un réseau de neurones convolutifs CNN. Les réseaux CNN sont largement utilisés dans plusieurs technologies modernes de vision par ordinateur comme la reconnaissance automatique de chiffres, de lettres, de textes manuscrits, de formes et de personnes. Les circuits FPGA sont considérés comme support privilégié pour l'implémentation des réseaux CNN vu leurs possibilités de reconfiguration, de calcul parallèle et de faible consommation de puissance. Dans notre travail nous avons fait la conception du réseau sous l'environnement Matlab, la conception des circuits des différentes couches du réseau sous l'environnement System Generator et l'implémentation sous l'environnement Vivado. Le réseau conçu est destiné à l'application de reconnaissance automatique de digits manuscrits.

Mots clés : Réseaux de neurones convolutifs CNN, circuit FPGA, reconnaissance digits manuscrits, System Generator, Vivado

ملخص

هذا العمل هو مساهمة في تنفيذ نوع خاص من الشبكات العصبونية المعروفة باسم الشبكات التلافيفية (CNN) على دارة قابلة لإعادة التشكل من نوع FPGA,. تستخدم شبكات CNN على نطاق واسع في العديد من تقنيات الرؤية بالكمبيوتر الحديثة مثل التعرف التلقائي على الأرقام , الحروف, النصوص المكتوبة بخط اليد , الأشكال والأشخاص. تعتبر دوائر FPGA وسيلة مفضلة لتنفيذ شبكات CNN نظرًا لإمكانياتها في إعادة التكوين والحوسبة المتوازية والاستهلاك المنخفض للطاقة. في عملنا هذا، قمنا بتصميم الشبكة تحت بيئة Matlab ، وتصميم الدوائر للطبقات المختلفة للشبكة تحت بيئة System Generator والتنفيذ تحت بيئة Vivado. الشبكة المصممة مخصصة لتطبيق التعرف التلقائي على الأرقام المكتوبة بخط اليد.

الكلمات المفتاحية : التعرف على الأرقام المكتوبة بخط اليد ,الشبكات التلافيفية CNN , دارة قابلة لإعادة الإعداد System Generator FPGA Vivado

Abstract

This work is a contribution to the implementation of a convolutional neural network CNN on Field Programmable Gate Array FPGA. CNN networks are widely used in several modern computer vision technologies such as automatic recognition of numbers, letters, handwritten texts, objects and person. FPGA circuits are considered as preferred platform for the implementation of CNN networks given their possibilities of reconfiguration, parallel computing and low power consumption. In this work, we have done the network design under the Matlab environment, the circuit design of the different layers of the network under the System Generator environment and the implementation under the Vivado environment. The network is targeting the automatic handwritten digits recognition application.

Keywords: convolutional neural network CNN, FPGA circuits, handwritten digits recognition, System Generator, Vivado.

Sommaire

Liste des figures

Liste des tableaux

Liste des acronymes

Introduction générale 11

Chapitre I Classifications des images et réseaux de neurones convolutifs

I.1 Introduction……14

I.2. Notion de base sur les images..... 14

I.2.1. Définition de l'image 14

I.2.2. Formes de représentation de l'image :..... 15

I.2.3. Les Types des images numériques :..... 15

I.2.4. Caractéristiques de l'image 16

I.3. Classification des images..... 17

I.3.1. Méthodes de classification 19

I.4. Rèseaux de neurones convolutifs 24

I.4.1. Architecture des réseaux CNN : 24

I.5. Exemples de modèles de CNN 30

I.5.1. Réseau AlexNet :..... 30

I.5.2. Réseau NiN..... 31

I.5.3. Réseau VGG..... 31

I.5.4. Réseau GoogLeNet..... 31

I.5.5. Réseau ResNet 31

I.6. Conclusion ..... ……32

Chapitre II Circuits FPGA : Architecture et outils de développement

II.1. Introduction 34

II.2. Architecture des circuits FPGA de Xilinx..... 34

II.3. Les blocs classiques d'un FPGA de xilinx [12] 36

II.3.1. Les CLB (configurable logic bloc) 36

II.3.2. Les IOB (Input Output Bloc)..... 39

II.3.3. Les interconnexions 39

II.4. Les blocs à usage spécifique 40

II.4.1. Les blocs RAM (BRAM)..... 41

II.4.2. Les blocs DSP 41
II.4.3. Les processeurs embarqués 42
II.5. Les outils de développement des FPGAs 44
II.5.1.Flot de conception 44
II.5.2. Vivado Design Suite 45
II.5.3. System Generator 46
II.6. Conclusion 47
Chapitre III Résultats de simulation et d'implémentations
III.1. Introduction 49
III.2. Les étapes de développement : 49
III.3. Conception, apprentissage et validation du réseau CNN : 49
III.3.1. Charger et explorer les données d'image : 50
III.3.2. Spécifier les ensembles d'apprentissage et de validation 50
III.3.3. Définition de la structure du réseau 50
III.3.4. Spécification des options de l'apprentissage : 52
III.3.5.L'apprentissage du réseau 53
III.4. Développement d'un programme Matlab pour la classification 54
III.5. Conception des circuits de calcul 54
III.5.1. Circuit de calcul d'activation ReLu 1 : 55
III.5.2. Circuit de calcul de Max pooling 1 : 55
III.5.3. Circuit de calcul de la convolution 2 : 57
III.6. Résultats de l'implémentation : 58
III.7.Conclusion 59
Conclusion générale 62

Liste de figures

Figure I.1 Apprentissage supervisées 19
Figure I.2 Apprentissage non supervisées 19
Figure I.3 Exemple de classification à base de l'arbre de décision 21
Figure I.4 Exemple de classification à base de SVM. 22
Figure I.5 Structure d'un neurone 23
Figure I.6 Exemple de réseau de neurone à deux couches cachées 23
Figure I.7 Structure générale d'un réseau de neurones convolutifs (CNN) 25
Figure I.8 Illustration des boucles de convolution à 4 niveaux et leurs dimensions 26
Figure I.9 Exemple de convolution 2D avec S=1, Nkx=3 et Nky=3 27
Figure I.10 Forme de la fonction ReLu 28
Figure I.11 Exemple d'une opération de maxpooling de taille 2 × 2. 28
Figure I.12 Exemple de trois couches entièrement connectées. ……29
Figure II.1 Architecture interne des anciennes version des circuits FPGA de Xilinx……...35
Figure II.2 Architecture interne de la famille Virtex 7 de Xilinx 36
Figure II.3 Arrangement des slice dans un CLB pour la série 7 de Xilinx. 36
Figure II.4 Structure interne de slice M de la série 7 de Xilinx 38
Figure II. 5 Structure interne d'un bloc IOB 39
Figure II.6 Les différentes lignes d'interconnexions. 40
Figure II.7 Architecture interne d'un bloc DSPE48. 41
Figure II.8 Schéma bloc d'un microcontrôleur embarqué de type PicoBlaze 43
Figure II.9 Schéma bloc d'un microcontrôleur embarqué de type MicroBlaze 43
Figure II.10 L'environnement de visualisation de l'IDE de Vivado..... ……46

Figure III.1 Exemples d'échantillons de la base MNIST. 50
Figure III.2 Architecture du réseau choisi pour la classification des digits manuscrits 51
Figure III.3 Fenêtre de progression de l'apprentissage 53
Figure III.4 Illustration des différentes phases de l'étape de conception des circuits de calcul 55
Figure III.5 Schéma bloc du circuit de calcul de la première couche de convolution 56

Figure III.6 Architecture détaillée de l'une des huit parties du circuit de calcul de la première couche de convolution. 56
Figure III.7 Structure de l'une des huit branches du circuit de calcul du max pooling 1. 57
Figure III.8 Schéma bloc d'un groupe du circuit de calcul de la convolution 2. 58

Liste des tableaux

Tableau III.1 Ressources du circuit FPGA Virtex7 XC7VX485T-2FFG1761C consommées par le ciruit de calcul de la convolution 1 59
Tableau III.2 Ressources du circuit FPGA Virtex7 XC7VX485T-2FFG1761C consommées par le circuit de calcul de première couche d'activation ReLu 1 59
Tableau III 3 Ressources du circuit FPGA Virtex7 XC7VX485T-2FFG1761C consommées par le circuit de calcul de première couche d'échantillonnage. 60
Tableau III.4 Ressources FPGA Virtex7 XC7VX485T-2FFG1761C consommées par le module de calcul de la deuxième couche de convolution…… 60

Liste des abréviations

CLB	Configurable Logic Block
CNN	Convolutional Neural Networks
CPU	Central Processing Unit
DCT	Discrete Cosine Transform
DPI	Dots Per Inch
DWT	Discrete Wavelet Transform
DWT	Discrete Wavelet Transform
DCT	Discrete Cosine Transform
FC	Fully Connected
FPGA	Field Programmable Gates Arrays
GPU	Graphic Processing Unit
GUI	Graphical User Interface
HDL	Hardware Description Language
IDE	Integrated Development Environment
IOB	Input Output Bloc
IP	Intellectual Property
KPPV	K-Plus Proches Voisins
LUT	Look_Up Table
MAC	Multiply ACcumulate
NiN	Network-in-Network
PPP	Points Par Pouce
RTL	Register Transfer Level
SGDM	Stochastic Gradient Descent with momentum
SVM	Support Vector Machine
VLSI	Very Large Scale Integrated Circuits

Introduction générale

À la fin du XXe siècle et au début du XXIe siècle, l'intelligence artificielle a émergé dans le monde de la technologie. Ce terme désigne l'ensemble des théories et des techniques développant des programmes informatiques complexes capables de simuler certains traits de l'intelligence humaine (raisonnement, apprentissage…). L'une des plus importantes des caractéristiques de ce domaine étant la capacité d'apprendre, de déduire et de réagir. Ou, comme l'informaticien John McCarthy l'a défini en 1956, c'est la science et l'ingénierie de la fabrication de machines intelligentes. L'apprentissage profond « deep learning » est une branche très importante de l'intelligence artificielle qui a connu un développement énorme ces dernières années. Le deep learning s'appuie sur un réseau de neurones artificiels s'inspirant du cerveau humain. Ce réseau est capable de décider de manière autonome, en fonction de certaines entrées fournies par le système.

La potentialité du deep learning est donnée principalement par les domaines d'application possibles, qui peuvent aller de la reconnaissance vocale à la conduite autonome de véhicules, de la reconnaissance d'images de vidéos à bien d'autres, comme la vision artificielle. Cependant il faut préciser que, si aujourd'hui l'application du deep learning se fait dans de nombreux secteurs, son potentiel est encore très large.

L'un des nombreux types de réseaux de neurones sont les réseaux convolutifs (ConvNet ou CNN), ainsi appelés en raison de l'application de l'opération de convolution qui est effectuée sur des matrices. Etant donné l'image d'entrée, un ConvNet a le but de fournir la classification de sortie à laquelle appartient l'image parmi les catégories de classes disponibles.

L'un des problèmes relatifs à l'utilisation des réseaux convolutifs (ConvNet) c'est qu'ils impliquent une complexité de traitement très élevée et nécessitent un espace mémoire important. Ce problème réduit énormément l'efficacité des processeurs basée sur la programmation comme les CPU et même les GPU.

L'une des solutions les plus efficaces pour surmonter le problème de la complexité de calcul est l'utilisation des circuits reconfigurables de type FPGA. En effet, l'utilisation de circuit FPGA permet d'accélérer le calcul par l'utilisation du parallélisme et permet aussi de réduire la puissance consommée par le système.

Notre projet de fin d'étude s'inscrit dans le cadre d'une contribution à l'implémentation sur FPGA d'un réseau de neurones convolutifs. Le réseau étudié est constitué de trois couches de convolution et une seule couche complètement connectée « fully connected ». L'application visée par le réseau est la reconnaissance automatique des chiffres manuscrits.

Afin de mieux présenter le contenu de notre travail, nous avons organisé le reste de ce mémoire comme suivant :

Dans le premier chapitre, nous présentons la classification des images et les réseaux de neurones convolutifs. Nous commençons par les notions de base sur les images numériques. Ensuite nous étudions le problème de classification des images et les différents types de classification. Nous finissons le premier chapitre par l'étude des réseaux de neurones convolutifs et leur utilisation dans le domaine de la classification des images.

Le deuxième chapitre est consacré à la présentation des circuits de type FPGA. Nous commençons par l'explication de l'architecture interne des circuits FPGAs en donnant les principaux composants. Puis, nous donnons le flot de conception et les outils utilisés pour le développement des applications à base de circuits FPGA.

Dans le troisième chapitre, nous présentons les différents schémas de conception, les résultats de simulation ainsi que les résultats d'implémentation sur FPGA.

Finalement, une conclusion générale résumant le fruit de ce projet est fournie à la fin de ce mémoire.

Chapitre.I

Classifications des images et réseaux de neurones convolutifs

I.1. Introduction

En théorie d'information, l'image est considérée comme l'une des sources les plus riches en information. De ce fait, un intérêt particulier a été donné au domaine de traitement d'image dont la classification d'images en fait partie importante. La classification d'images est une tâche fondamentale qui tente de comprendre une image entière dans son ensemble. Le but est de classer l'image en lui attribuant une étiquette spécifique. Typiquement, la classification d'images fait référence à des images dans lesquelles un seul objet apparaît et est analysé. Depuis l'apparition de l'apprentissage profond, la classification d'image a connu un progrès énorme sur le plan des domaines d'application et sur le plan de performance. La classification d'images avec apprentissage profond implique le plus souvent des réseaux de neurones convolutifs, ou CNN « Convolutional Neural Networks ». Dans les CNNs, les nœuds des couches cachées ne partagent pas toujours leur sortie avec chaque nœud de la couche suivante.

L'apprentissage en profondeur permet aux machines d'identifier et d'extraire des caractéristiques des images. Cela signifie qu'ils peuvent apprendre les caractéristiques à rechercher dans les images en analysant de nombreuses images. Ainsi, les programmeurs n'ont pas besoin d'entrer ces filtres à la main.

I.2. Notion de base sur les images

I.2.1. Définition de l'image

Une image est une représentation planaire d'une scène ou d'un objet situé en général dans un espace tridimensionnel, elle est issue du contact des rayons lumineux provenant des objets formants la scène avec un capteur (caméra, scanner, rayons X, ...). Il ne s'agit en réalité que d'une représentation spatiale de la lumière.

La casé totalité des systèmes actuels exploitent et traitent des images sous forme numérique. Même les systèmes d'acquisition d'images moderne (camera, scanner) fournissent directement des images numériques.

Le terme image numérique désigne, toute image sauvegardée sous une forme codée représentable par des valeurs numériques. La numérisation est le processus qui permet de passer de l'état d'image analogique, qui est caractérisée par l'aspect continu du signal qu'elle représente, à l'état d'image numérique qui est caractérisée par l'aspect discret. Le mot discret signifie que l'intensité lumineuse ne peut prendre que des valeurs quantifiées en un nombre

fini de points distincts. La forme numérique de l'image permet une exploitation efficace par des outils logiciels sur ordinateur. [1]

I.2.2. Formes de représentation de l'image

Une image numérique peut être représentée sous forme vectoriel ou sous forme matricielle.

I.2.2.1. Forme vectorielle

Une image sous forme vectoriel est une image numérique composée de plusieurs objets géométriques individuels (droites, polygones, arcs de cercle). L'image vectorielle est créée à partir d'équations mathématiques. Chaque forme dépend de plusieurs paramètres (hauteur, largeur, rayon) donnés sous forme de vecteurs.
Le point fort majeur d'une image vectorielle, par rapport à une image matricielle, c'est de pouvoir être agrandie à l'infini sans pour autant se déformer ou perdre de sa qualité. On utilise ce format pour toutes les créations destinées à être reproduites notamment pour l'impression grand format de logos, d'images, de textes, de dessins…

I.2.2.2. Forme matricielle

L'image sous format matriciel ou tramée est formée d'une grille composée de points carrés individuels nommés pixels. Pour une image en format matriciel, plus on zoom, plus les pixels deviennent apparents. Dans notre travail on s'intéresse seulement aux images matricielles.

I.2.3. Les Types des images numériques

I.2.3.1. Image binaire

Une image à 1 bit se compose uniquement de bits ***on*** et ***off***. Il s'agit donc du type d'image le plus simple. Chaque pixel est stocké sous forme de bit unique (0 ou 1). On l'appelle aussi parfois une image monochrome à 1 bit, car elle ne contient aucune couleur. Les images binaires ne nécessitent pas un espace de stockage important. A titre d'exemple, une image binaire 640×480 nécessite 38.4 kilo-octets (Ko) de stockage. Les images monochromes à 1 bit peuvent être satisfaisantes pour des images contenant uniquement des graphiques et du texte simples. De plus, les télécopieurs utilisent des données 1 bit. Par conséquent, les images 1 bit sont toujours importantes même si les capacités de stockage ont suffisamment augmenté pour permettre l'utilisation d'une imagerie contenant davantage d'informations.

I.2.3.2. Images à niveaux des gris

Une image ***à niveau des gris n bits,*** est une image pour laquelle chaque pixel a une valeur de gris comprise entre 0 et $N=2^n-1$. Couramment pour les images à niveaux de gris, chaque pixel est représenté par un seul octet (n=8 N=255).

L'ensemble de l'image peut être considéré comme un tableau bidimensionnel ou une matrice de valeurs de pixels. Nous appelons ce tableau (matrice) un bitmap.

I.2.3.3. Image couleur

La ***couleur*** est l'une des caractéristiques les plus importantes de l'image. Il est généralement invariant en translation, rotation et mise à l'échelle. L'image couleur peut être modélisée à l'aide de divers systèmes de couleur. Le ***RVB*** « RGB en englais » (rouge, vert bleu) est l'un des systèmes de couleurs couramment utilisés. Le modèle RVB est basé sur le fait que la couleur peut être vue comme une fonction vectorielle de trois coordonnées pour chaque position dans l'image. Parfois, ce modèle est appelé ***modèle additif***, car l'image est obtenue en ajoutant les composants en couleurs primaires. Chaque point de l'image peut être représenté par la somme des valeurs des trois couleurs primaires (R, G, B).

Pour le mode RVB, chaque pixel de l'image est représenté par trois valeurs indiquant l'intensité de chacune des couleurs rouge vert et bleu. Couramment chaque couleur est codée sur un octet. La valeur 0 signifie l'absence de couleur, tandis que la valeur 255 indique la couleur avec une intensité maximale. Dans une image couleur de 24 bits, chaque pixel est représenté par trois octets, représentant généralement le format RVB. Comme chaque valeur est comprise entre 0 et 255, ce format prend en charge 256 × 256 × 256, soit un total de 16 777 216 couleurs combinées possibles. Cependant, une telle flexibilité entraîne une pénalité de stockage : une image couleur 24 bits 640 × 480 nécessiterait 921,6 Ko de stockage sans compression.

I.2.4. Caractéristiques de l'image

L'image est un ensemble structuré d'information caractérisé par les paramètres suivants :

I.2.4.1. Définition

On appelle définition le nombre de pixels constituant l'image, c'est-à-dire sa «dimension informatique». La définition est égale au nombre de colonnes de l'image

multiplié par le nombre de de lignes. Une image possédant 640 pixels en largeur et 480 en hauteur aura une définition de 640 pixels par 480, notée *640×480*.

I.2.4.2. La résolution

Terme souvent confondu avec la définition, détermine par contre le nombre de points ou pixels par unité de surface, exprimé en *points par pouce* (PPP), en anglais DPI pour « *Dots Per Inch* ». La résolution permet ainsi d'établir le rapport entre le nombre de pixels d'une image et la taille réelle de sa représentation sur un support physique. Une résolution de 300 dpi signifie donc 300 colonnes et 300 lignes de pixels sur un pouce carré ce qui donne donc 90000 pixels sur un pouce carré. La résolution est directement liée à la clarté ou la finesse de détails atteinte par un moniteur ou une imprimante dans la production d'images.

I.2.4.3. La taille (poids)

Pour connaître la taille de l'image (généralement en octets), il est nécessaire de connaitre le nombre de pixels de l'image et le nombre de bits utilisés pour le codage de chaque pixel. La taille de l'image est alors égale à son nombre de pixels multiplié par le nombre de bits par pixel.

I.3. Classification des images

En raison de la numérisation et du développement rapide du Web, le monde est plein d'images numériques. Cependant, sans une classification appropriée, cette quantité gigantesque d'images ne sera pas très utile. De nombreuses recherches ont été effectuées au cours des dernières décennies pour organiser les images numériques en catégories afin qu'elles puissent être recherchées et récupérées facilement.

Fondamentalement, la classification des images consiste à organiser les images en différentes classes en fonction des caractéristiques des images. La classification des images consiste à utiliser des algorithmes de vision par ordinateur et d'apprentissage automatique pour extraire le sens d'une image. Cette action peut être aussi simple que d'attribuer une étiquette à ce que contient l'image, ou aussi avancée que d'interpréter le contenu d'une image et de renvoyer une phrase lisible par l'homme. Les applications de la classification d'images sont nombreuse, on retrouve ainsi la classification d'objets, de scènes, de textures, la reconnaissance de visages, d'empreintes digitales, de caractères parmi les applications courantes.

Un système de classification automatique d'images est composé des étapes suivantes [2] :

- Le **prétraitement** pour préparer l'image au traitement.
- **L'extraction de caractéristiques** : elle permet d'extraire l'information pertinente pour faire la distinction entre les images de classes différentes. L'extraction se fait à l'aide de descripteurs discriminants. La description obtenue peut être locale si elle tient compte des informations locales comme des motifs dans l'image ou globale si elle tient compte de la totalité des informations présentes à l'échelle macroscopique. Nous retrouvons des ***descripteurs de bas niveau*** s'intéressant à l'information contenue dans l'image au niveau du pixel et des ***descripteurs de plus haut niveau*** nécessitant une représentation intermédiaire de l'image plus adaptée. Parmi les descripteurs de bas niveau on peut citer :
  - les statistiques d'histogramme ;
  - les statistiques des matrices de co-occurrence ;
  - l'extracteur issu des caractéristiques spectrales ;

  Pour les extracteurs de plus haut niveau on trouve :
  - l'extracteur issu du d´etecteur de Harris ;
  - Scale Invariant Feature Transform (SIFT) ;
  - le Local Binary Pattern ;
- **L'apprentissage** permettant de construire des frontières de décision pour identifier la classe d'une image présentée à l'entrée du système de classification. La phase d'apprentissage consiste en la construction d'une règle de décision soit à partir d'un modèle, soit en élaborant une frontière dans l'espace de caractéristiques formé à l'étape d'extraction de caractéristiques.
- L**'évaluation** pour connaître les performances d'un système de classification d'images. Pour ce faire, il est nécessaire de définir une procédure d'évaluation consistant notamment dans le choix d'une mesure donnant les performances du système et des données d'évaluation. Généralement, les données d'évaluation sont constituées de bases de données connues dans le domaine.

I.3.1. Méthodes de classification

De nombreuses méthodes classiques ont été consacrés, elles peuvent être séparées en deux grandes catégories : les méthodes de classification supervisée et les méthodes de classification non supervisée. Nous étudions dans cette section les méthodes de classification des images du point de vue apprentissage.

I.3.1.1. Méthodes supervisées et méthodes non supervisées

a) Méthode supervisée

L'objectif de la classification supervisée est principalement de définir des règles permettant de classer des objets dans des classes à partir de variables qualitatives ou quantitatives caractérisant ces objets. On dispose au départ d'un échantillon dit d'apprentissage dont le classement est connu. Cet échantillon est utilisé pour l'apprentissage des règles de classement.

Il est nécessaire d'étudier la fiabilité de ces règles pour les comparer et les appliquer, évaluer les cas de sous apprentissage ou de sur apprentissage (complexité du modèle). On utilise souvent un deuxième échantillon indépendant, dit de validation ou de test.

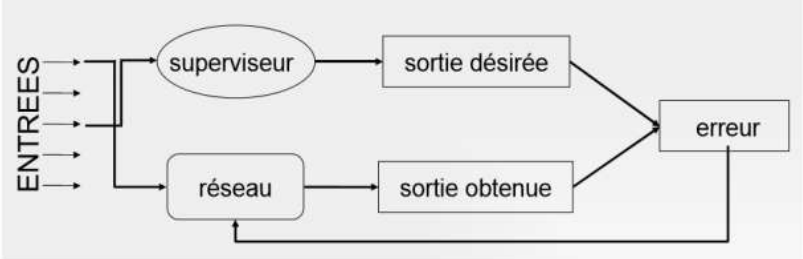


Figure I.1 Apprentissage supervisées

b) Méthodes non supervisées

Procède de la façon contraire. C'est à dire ne nécessitent aucun apprentissage et aucune tâche préalable d'étiquetage manuel. Elle consiste à représenter un nuage des points d'un espace quelconque en un ensemble de groupes appelé Cluster. Il est lié généralement au domaine de l'analyse des données comme l'ACP. Un «Cluster» est une collection d'objets qui sont «similaires» entre eux et qui sont «dissemblables » par rapport aux objets appartenant à d'autres groupes.

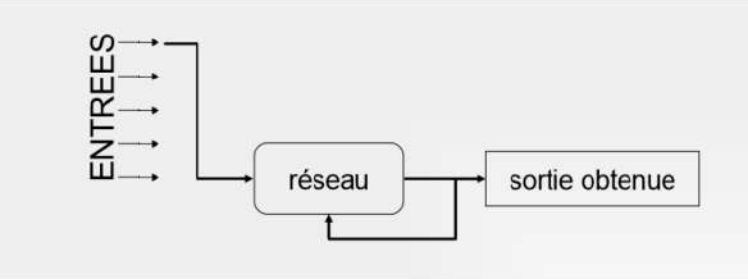


Figure I.2 Apprentissage non supervisées

I.3.1.2. Classification à base de K-plus proches voisins :

L'algorithme des K-Plus Proches Voisins (KPPV) est basé sur l'estimation de distance. Il n'a pas besoin d'apprentissage car le modèle de classification est directement le set d'apprentissage. Lors de la phase de décision, l'algorithme évalue la classe majoritaire des K individus du set d'apprentissage les plus proches de l'exemple à tester. K représente un paramètre important de l'approche car il conditionne la façon dont la classification va s'adapter aux données à classifier. Si la valeur de K est faible, on obtient un classifier de bonne résolution avec un biais d'autant plus faible que K est petit, mais la diminution de K rend le classifier très sensible aux bruits sur les données. Si, en revanche, K est élevé, on obtient un classifier peu sensible au bruit, avec une variance faible mais un biais élevé. Il est donc nécessaire de chercher un bon compromis entre biais et sensibilité aux bruits.

Malgré sa simplicité de mise en œuvre, les performances de l'algorithme KPPV sont encore mal connues et les coûts sont importants en stockage et temps de classification. Plusieurs améliorations ont été proposées par les chercheurs du domaine pour améliorer les performances du système de classification à base de l'algorithme KPPV.

I.3.1.3. Classification à base de l'arbre de décision

L'arbre de décision est une méthode de classification très populaire dans divers domaines comme la détection de tumeurs musculaires, la détection de tuberculose pulmonaire, la classification de l'émotion et la reconnaissance d'activités physiques réalisées par un individu.

La méthode est basée sur la décomposition hiérarchique séquentielle du domaine d'apprentissage en des sous-domaines les plus purs possibles contenant une population la plus homogène possible en termes de classes.

La construction des arbres de décision repose sur les étapes suivantes :

- le choix d'un attribut de partitionnement en chacun des nœuds de l'arbre,
- le choix d'un point de coupure pour la variable s´sélectionnée précédemment,
- le choix d'une mesure de score évaluant la qualité des coupures choisies afin de déterminer le meilleur partitionnement,
- le choix d'un critère d'arrêt de partitionnement,
- le choix d'une règle d'assignation d'une classe à un nœud terminal ou feuille.

La figure I.3 illustre un exemple de construction d'arbre de décision pour le problème classique de classification de variétés d'Iris proposé par *Fisher* [3]

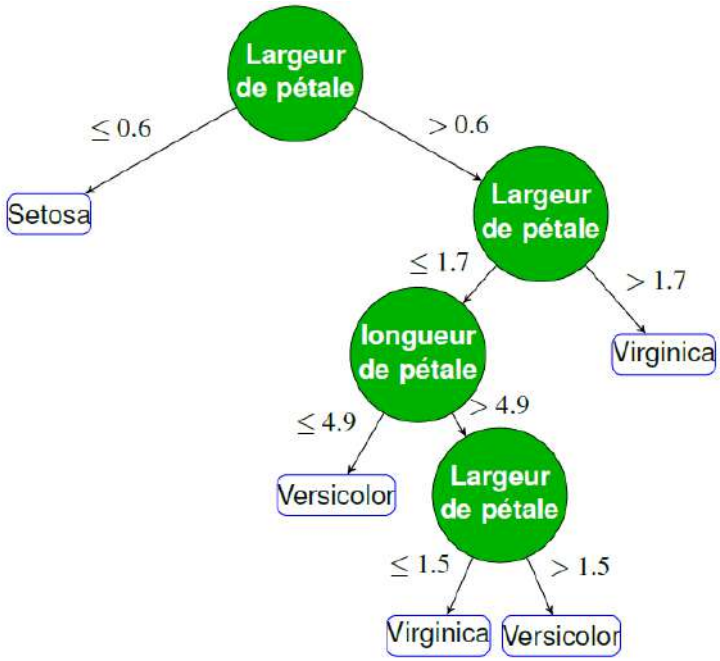


Figure I.3 Exemple de classification à base de l'arbre de décision

I.3.1.4. Classification à base de SVM « Support Vector Machine »

Le SVM [4] est un classifier discriminant paramétrique basé sur l'établissement d'un hyperplan qui sépare les exemples représentatifs de chacune des classes des données d'apprentissage.

La Figure I.4, présente le problème de classification à deux dimensions. Il existe une infinité de droites séparatrices des deux classes en présence mais une seule droite maximise la distance des exemples aux frontières des classes (i.e. maximise la distance $z2$ à l'hyperplan). Ces exemples frontières sont appelés vecteurs de support.

Sur la figure sont représentés deux jeux d'hyperplan notés direction 1 et direction 2 dont la direction 2 possède la marge la plus importante.

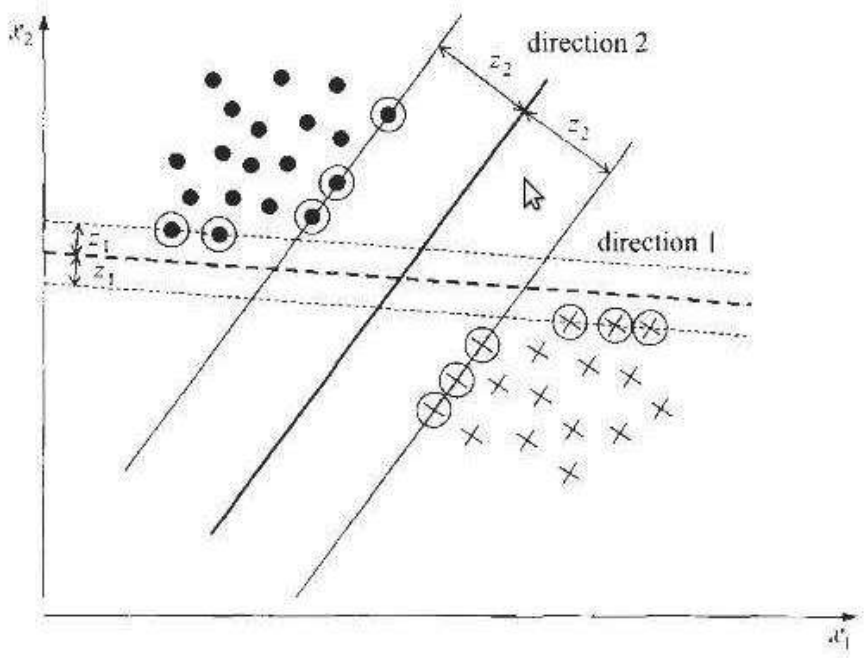


Figure I.4 Exemple de classification à base de SVM.

I.3.1.5. Classification à base de réseaux de neurones :

Les réseaux de neurones ont été proposés initialement comme modèle mathématique du fonctionnement des neurones physiologiques. Le modèle le plus simple de l'unité de base d'un réseau est le perceptron linéaire.

La sortie d'un neurone est évaluée par rapport à un seuil appliqué sur la somme pondérée des coefficients appliqués aux données en entrée comme suivant (figure I.5) :

$$sortie = f(d_w) = \begin{cases} +1 \ \ si \ \ d_w > 0 \\ -1 \ \ \ sinon \end{cases} \quad (1.1)$$

Où :

$$d_w(x) = w^t x \quad (1.2)$$

W : est le vecteur des coefficients du réseau

x : est l'entrée du réseau

$D_w(x)$: est la somme pondérée de l'entrée

f : est la fonction d'activation du réseau.

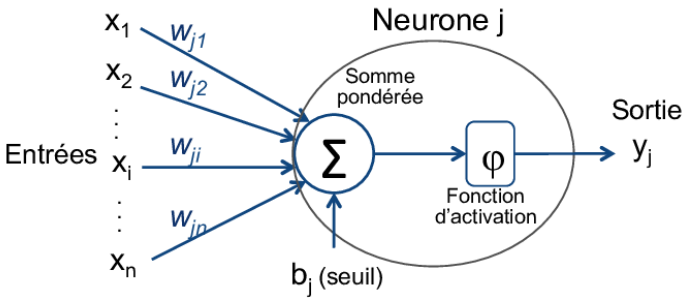


Figure I.5 Structure d'un neurone

Un réseau de neurone est constitué d'une couche d'entrée, d'une couche de sortie et d'une ou plusieurs couches intermédiaires appelées couches cachées. La figure I.6 est un exemple de réseau de neurones à deux couches cachées.

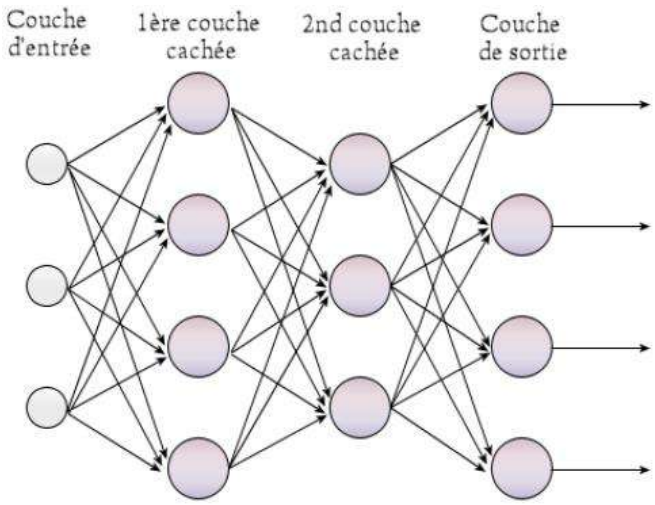


Figure I.6 Exemple de réseau de neurones à deux couches cachées

Des techniques d'apprentissage comme la rétro-propagation permettent de minimiser l'erreur quadratique empirique du réseau. L'algorithme de rétro-propagation propage dans le réseau, de la couche de sortie vers la couche d'entrée, l'erreur commise sur les exemples d'apprentissage et par le biais d'itérations successives permet d'optimiser le choix des pondérations. Le choix du nombre de couches, et du nombre de nœuds de chaque couche demeure un problème actuel. En effet le dimensionnement du réseau est à figer avant la phase d'apprentissage des pondérations ; surtout, aucune règle ne permet de d´déterminer le nombre de neurones adéquat dans la couche cachée.

I.4. Réseaux de neurone convolutifs

Les réseaux de neurones convolutifs ou CNN (Convolutional Neural Network), sont des réseaux de neurones multicouches spécialisés dans la classification des images et la reconnaissance de formes.

Les réseaux CNN comportent deux parties distinctes, la première partie est la partie convolutionnelle, la deuxième partie est la partie entièrement connecté (fully connected).

L'architecture de la partie convolutionelle repose sur des couches de convolution alternant avec des couches d'agrégation (pooling) et des couches d'activation. L'image en entrée est fournie sous forme de matrice de pixels, elle passe par une succession de filtres au niveau de chaque couche de convolution, créant des nouvelles images appelées cartes (maps) de convolutions. La couche d'agrégation (pooling) réduit la taille de l'image par une fonction qui prend le maximum ou la moyenne de plusieurs pixels voisins. Les sorties de la partie convolutionelle qui sont des matrices sont transformées en un vecteur unique appelé caractéristique du CNN.

Ce vecteur en sortie de la partie convolutionnelle est ensuite branché en entrée de la deuxième partie qui est constituée d'une ou plusieurs couches complètement connectées. Le rôle de cette partie est de combiner les caractéristiques du CNN pour classer l'image. La dernière couche de la partie entièrement connectée contient un nombre de neurone égale au nombre de catégories de classement (un neurone pour chaque catégorie). Les valeurs numériques obtenues sont généralement normalisées entre 0 et 1, de somme 1, pour produire une distribution de probabilité sur les catégories. [5], [6]

I.4.1. Architecture des réseaux CNN :

Les réseaux de neurones convolutifs tels qu'illustrés à la figure I.7 intègrent généralement plusieurs couches de *convolution*, de *regroupement* (pooling) et de *normalisation* qui extraient des caractéristiques de bas niveau à haut niveau de l'entrée pendant le processus d'inférence en aval. Ces caractéristiques peuvent être classées en un nombre fini de classes de sortie par les couches de classification finales telles que le perceptron multicouche ou les couches entièrement connectées. [7], [8]

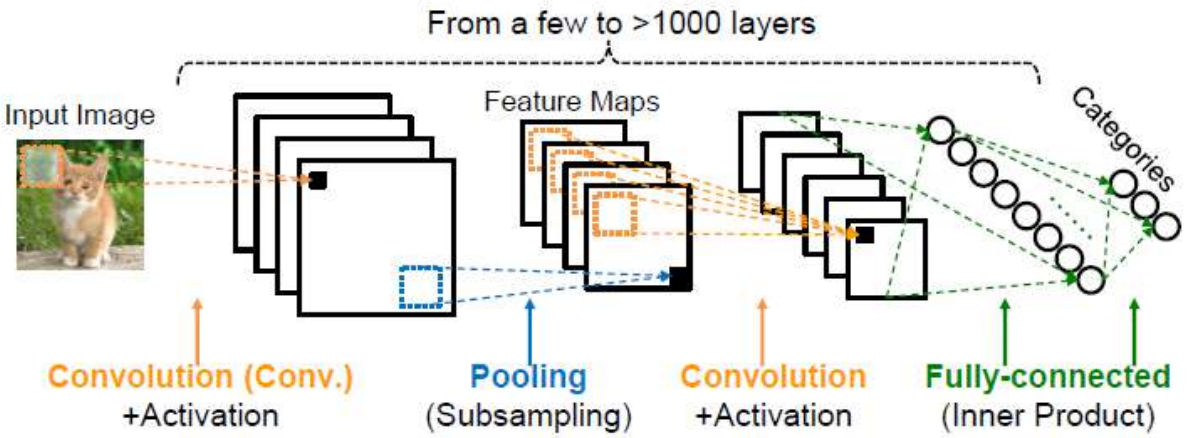


*Figure **I.7** Structure générale d'un réseau de neurones convolutifs (CNN)*

I.4.1.1. Couche d'entrée

Les données d'entrée sont les images à classer, chaque image x est un volume de dimension $M \times M \times C$, où M est la hauteur et la largeur de l'image et C est le nombre de canaux de l'image. Pour la commodité de la formulation, la hauteur et la largeur des images sont supposées être les mêmes. Typiquement, pour les images couleur RVB $C = 3$ et pour les images en niveaux de gris, $C = 1$.

I.4.1.2. Couches de convolution

Un réseau de neurones convolutifs est un réseau de neurones qui utilise une opération mathématique qui s'appelle convolution ou produit de convolution. La convolution est l'opération principale des algorithmes CNN, qui implique des opérations de multiplication et d'accumulation tridimensionnelles (MAC) des pixels d'entrée et des poids du noyau.

Comme le montre la figure I.8, plusieurs dimensions sont utilisées pour décrire les tailles des caractéristiques et des cartes de noyau de chaque couche de convolution pour un CNN donné. Chaque couche de convolution possède une entrée sous forme de plusieurs matrices de même dimension connues sous le nom « input feature maps », un ensemble de groupe de filtres nommé « Kernel maps » et une sortie sous forme de plusieurs matrices aussi de même dimension appelé « Output feature maps ». Dans la figure I.8 les différentes dimensions sont données comme suivant :

Pour input feature maps : *Nix* et *Niy* représentent les dimensions (nombre de colonnes et nombre de lignes) de chacune des matrices d'entrée et *Nif* est le nombre de matrice d'entrée (nombre de feature maps).

Pour kernel maps : *Nkx* et *Nky* sont les dimensions d'un kernel, *Nif* est le nombre de kernels par groupe et *Nof* et le nombre total de groupes.

Pour out feature maps : *Nox* et *Noy* représentent les dimensions (nombre de colonnes et nombre de lignes) de chacune des matrices de sortie et *Nof* est le nombre de matrice de sorties.

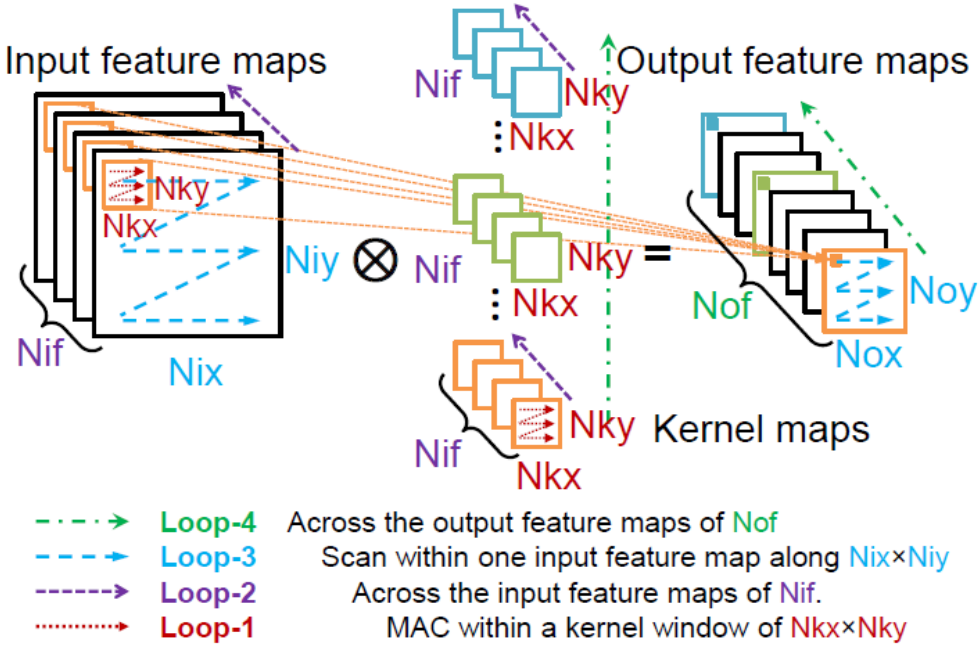


Figure I.8 Illustration des boucles de convolution à 4 niveaux et leurs dimensions

Le pixel de position *x*, *y* dans la matrice de sortie du numéro *no* est calculé comme suivant :

$$pixel_l(no; x, y) =$$

$$\sum_{ni=1}^{Nif} \sum_{ky=1}^{Nky} \sum_{kx=1}^{Nkx} pixel_l\,(ni, S * x + Kx.S * y + Ky) * weight(ni, no; Kx, Ky + bias(no); \quad (1.3)$$

$où\ x \in \{1, 2 \dots Nox\},\ y \in \{1,2 \dots Noy\}$ et $no \in \{1,2 \dots Nof\}$ et $L \in \{1,2 \dots \#conv\}$, #conv est le nombre de couche de convolution dans le réseau.

Le paramètre *S* dans l'équation est le pas de convolution (stride). C'est le pas de décalage du kenel de convolution à chaque calcul d'un nouveau pixel de sortie. Par exemple dans le cas ou *S=1* (figure I.9) le kernel de convolution sera déplacé d'un seul pixel vers la droite pour chaque mouvement horizontal du noyau et d'un seul pixel vers le bas pour chaque mouvement vertical du noyau lors du calcul d'une matrice de sortie.

L'équation 1.3, est une convolution 3D qui est obtenu en sommant sur toutes les valeurs de *ni* des convolutions 2D. Pour bien comprendre la convolution 2D, nous avons présenté dans la figure I.9 un exemple pour *S=1, Nkx=3 et Nky=3*

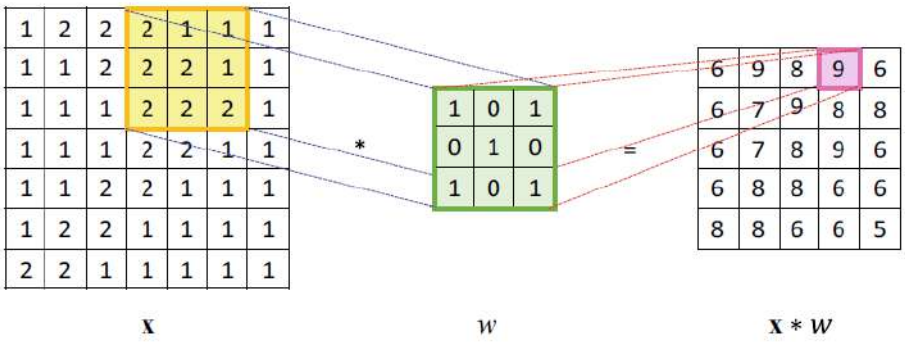


Figure I.9 Exemple de convolution 2D avec S=1, Nkx=3 et Nky=3

La relation qui existe entre les dimensions d'entrée et les dimensions de sortie d'une couche de convolution est donnée par les deux équations suivantes :

$$N_{ix} = (N_{ox} - 1) \times S + N_{kx} \quad (1.4)$$

$$N_{iy} = (N_{oy} - 1) \times S + N_{ky} \quad (1.5)$$

Pour contrôler la taille spatiale des cartes de caractéristiques en sortie (*Nox* et *Noy*), on ajoute parfois des pixels autour de la bordure des maps de caractéristiques en entrée, et la taille de remplissage des zéros est incluse dans *Nix* et *Niy*. Cette opération est connue sous le nom de padding et elle est réalisée en ajoutant des zéros ou en répétant les pixels de la bordure.

I.4.1.3. Couche de ReLu (Rectified Linear unit)

La sortie d'une couche de convolution est essentiellement la somme pondérée de la couche d'entrée. Comme dans un réseau de neurones ordinaire, cette sortie doit passer une fonction d'activation. Il a été trouvé que dans une couche de convolution, la fonction *max (0, x)* est plus efficace qu'une fonction *sigmoïde* pour l'activation [9]. La fonction *max (0, x)* est fondamentalement une fonction linéaire rectifiée car elle rectifie ou réfracte simplement la moitié négative de y = x à 0 (figure I.10). Par conséquent, on l'appelle souvent un redresseur et un nœud activé par le redresseur est également appelée unité linéaire rectifiée ou ReLU. L'activation ReLU est généralement implémentée en tant que couche séparée après la couche de convolution.

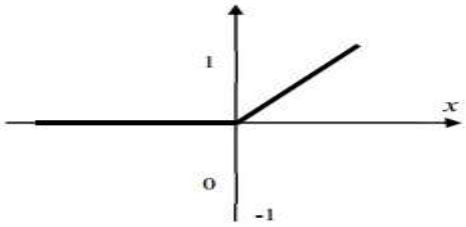


Figure I.10 Forme de la fonction ReLu

I.4.1.4. Couche d'échantillonnage (Pooling)

La couche d'échantillonnage est chargée de réduire la taille spatiale des maps de caractéristiques, mais elle conserve les informations les plus importantes. Il existe différents types d'échantillonnage comme *max( ), average( ), L2 norm* ou transformée spectralle comme DWT «discrete wavelet transform » et DCT «discrete cosine transform ».

L'échantillonnage consiste à appliquer un kernel sur les matrices des maps d'activation en le faisant glisser avec un pas préalablement défini (le pas est généralement égal à la taille du noyau pour éviter le phénomène de chevauchement). Le *Max Pooling* renvoie la valeur maximale de la partie de l'image couverte par le noyau. Au lieu de prendre le maximum, nous pourrions prendre la moyenne de tous les éléments couverts par le noyau, cela est assuré par le *Average Pooling*. En pratique, le Max Pooling est beaucoup plus utilisé que le Average Pooling puisqu'il fonctionne mieux. La forme la plus courante est une couche de Pooling avec des noyaux de taille 2×2 appliqués avec un pas de 2 réduisant ainsi la dimension de l'entrée par 2, en éliminant 75% des activations et en laissant la dimension de profondeur inchangée. Dans la Figure I.11, le nombre des activations passe de 16 à 4 en appliquant le Max pooling. [10]

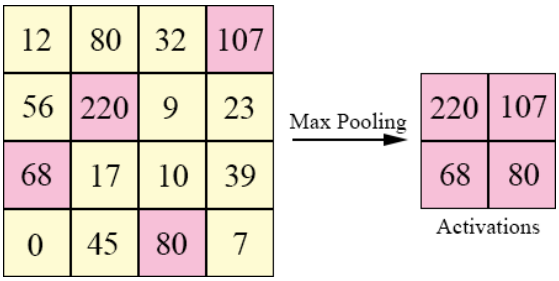


Figure I.11 Exemple d'une opération de maxpooling de taille 2 × 2.

L'échantillonnage a pour fonction de réduire progressivement la taille spatiale de la donnée d'entrée. En effet, il permet de :

- rendre les représentations d'entrée (carte des caractéristiques) plus petites et plus faciles à gérer.
- réduire la puissance de calcul requise pour traiter les données par réduction des dimensions et donc il contrôle le sur-apprentissage.
- extraire les caractéristiques dominantes qui sont invariantes en rotation et en position, maintenant ainsi l'efficacité du processus d'apprentissage du modèle.
- aider à obtenir une représentation équivariante de l'image. Ceci est très puissant puisqu'il permet de détecter des objets dans une image, peu importe où ils se trouvent.

I.4.1.5. Couches complètement connectées

Pour les réseaux de neurones réguliers et à la fin des couches de convolution, le type de couche le plus courant que nous trouvons est la couche entièrement connectée, dans laquelle les neurones entre deux couches adjacentes sont simplement entièrement connectés entre eux, comme le montre la figure I.12.

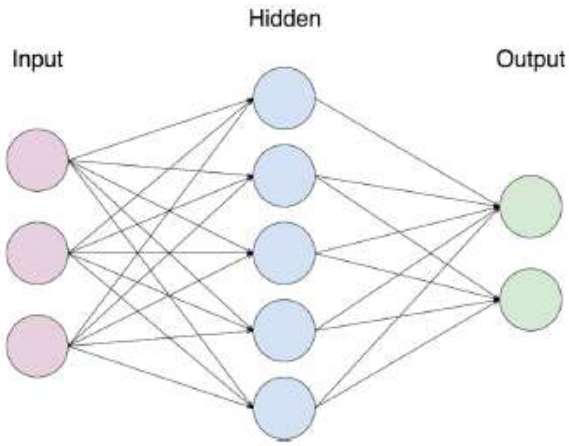


Figure I.12 Exemple de trois couches entièrement connectées

Pour obtenir l'entrée de la première couche complétement connectée, on applique une transformation matrice vers vecteur de toutes les matrices représentant la sortie de la couche qui la précède puis on fait la concaténation de tous les vecteurs pour obtenir un vecteur unique. Les poids de chaque couche complètement connectée sont représentés sous forme d'une matrice unique à deux dimensions dont une dimension représente le nombre d'entrées de la couche

et l'autre dimension représente le nombre de sortie de la couche. La sortie d'une couche ni autre que le produit matrice-vecteur de la matrice des poids et du vecteur des caractéristiques d'entrée suivit de l'ajout d'un vecteur des biais.

La sortie des couches de convolution et d'échantillonnage représente des caractéristiques de haut niveau de l'image d'entrée. L'objectif de la couche complètement connectée est d'utiliser ces caractéristiques pour classer l'image d'entrée du réseau en différentes classes en fonction de la base de données d'apprentissage. La plupart des cartes de caractéristiques des couches de convolution et d'échantillonnage peuvent être utiles pour la tâche de classification, mais une combinaison de ces fonctionnalités peut être encore meilleure.

Pour la plupart des réseaux CNN, la dernière couche complètement connectée est suivie d'une couche d'activation de type *softmax*. Cette dernière agit sur les sorties pour les transformer en distribution de probabilité (somme égale à un) pour rendre facile la tache de classification. La fonction *softmax* prend un vecteur de scores arbitraires à valeurs réelles et le réduit à un vecteur de valeurs comprises entre zéro et un. Elle s'écrit sous la forme : [11]

$$y_i = \text{softmax}(x)_i = \frac{e_i^{x_j}}{-e^{x_j}} \tag{1.6}$$

I.5. Exemples de modèles de CNN

La forme la plus commune d'une architecture CNN empile quelques couches de Convolution et de ReLU, et les suit avec des couches de Poolling, et répète ce schéma jusqu'à ce que l'entrée soit réduite dans un espace d'une taille suffisamment petite. À un moment, il est fréquent de placer des couches entièrement connectées (FC). La dernière couche entièrement connectée est reliée vers la sortie. Voici quelques exemples d'architectures de réseaux *CNN* proposé dans la littérature :

I.5.1. Réseau AlexNet

Le réseau est proposé par *Krizhevsky et al* [9] il augmente considérablement la précision de la classification par rapport aux méthodes traditionnelles d'apprentissage automatique basées sur des fonctionnalités fabriquées à la main et est le premier modèle CNN à remporter le concours ILSVRC en 2012. AlexNet est composé de 5 couches de convolution et de 3 couches FC, il a 61 millions de poids de noyau (paramètres) et nécessite environ 1,4 milliard d'opérations de multiplication et d'addition. Le top 5 de la précision du modèle AlexNet Caffe sur la base de données ImageNet Russakovsky [12] est de 80,2 %. Il

existe trois tailles de noyau différentes (Nkx Nky) dans les couches de convolution AlexNet : 11× 11, 5×5 et 3×3.

I.5.2. Réseau NiN

Le réseau NiN « network-in-network » a été proposé par Lin et al. [13]. Il remplace les couches FC par une couche de type average pooling globale à la fin pour économiser un grand nombre de poids, tout en pouvant toujours atteindre le même niveau de précision qu'AlexNet sur la base de données ImageNet. NiN se compose de petites couches de convolution empilées avec une taille de noyau 1×1 pour une meilleure abstraction des caractéristiques locales. Comme AlexNet, NiN a également des tailles de noyau de : 11× 11, 5×5 et 3×3.

I.5.3. Réseau VGG

Le réseau VGG a été proposé par Simonyan et Zisserman [14] a une structure très régulière, avec seulement 3×3 couches de convolution et 2×2 couches de max-pooling. Cependant, les trois couches FC nécessitent environ 123 millions de poids et les couches de convolution dans VGG-16 nécessitent également environ 15 millions de poids. Le VGG-16 global a besoin d'environ 31 milliards d'opérations pour traiter une image dans le processus d'anticipation. Le top 5 de la précision du modèle VGG-16 sur la base de validation ImageNet est de 88,7 %.

I.5.4. Réseau GoogLeNet

Le réseau VGG a été proposé par *Szegedy et al.* [15], il atteint le même niveau de précision que VGG-16 avec une précision de 89,0 % dans le top 5 sur ImageNet, tout en n'exigeant que 6,1 millions de poids et 3,2 milliards d'opérations, ce qui est obtenu par une architecture plus complexe ave. Un module de démarrage se compose de six couches de convolution avec des tailles de noyau 1×1, 3×3 et 5×5 et une couche de mise en commun maximale dans quatre branches parallèles.

I.5.5. Réseau ResNet

Le réseau ResNet a été proposé par He et al. [16] résout le problème de la formation de CNN très profonds avec des centaines de couches et peut toujours obtenir une précision convaincante avec une précision du top 5 supérieure à 92,2% sur ImageNet. Les réseaux très profonds sont difficiles à entraîner car les gradients disparaissent après s'être propagés en arrière à travers un trop grand nombre de couches. ResNet incorpore des connexions de saut (\raccourci") entre deux couches non adjacentes afin que les gradients puissent revenir plus

facilement aux couches précédentes pendant l'entraînement. ResNet-50 a 53 couches de convolution, une mise en commun maximale et une couche FC, où chaque convolution est suivie d'une couche Bach normalisation. Les tailles de noyau utilisées dans ResNet sont 7×7, 3×3 et 1×1. Le nombre de poids dans ResNet-50 est d'environ 26 millions et le nombre d'opérations est d'environ 7,7 milliards.

I.6. Conclusion

Dans la première partie de ce chapitre nous avons commencé par donner les concepts de base sur les images numériques, leurs modes de représentation, leurs différents types ainsi que leurs caractéristiques principales. Ensuite, nous avons passé à la présentation du problème de classification des images qui trouve un large domaine d'application. Nous avons discuté les différentes étapes de classification à savoir l'étape d'extraction de caractéristiques et l'étape d'apprentissage permettant d'´établir la fonction de décision du classifier. Nous avons, aussi, présenté les différentes méthodes de classification couramment utilisées dans la littérature.

Dans la deuxième partie du présent chapitre, nous avons étudié une classe importante de réseaux de neurones qui ont connu un grand succès dans le domaine de classification des images. Il s'agit de réseaux de neurones convolutifs CNN qu'ils sont basé sur l'opération de convolution. L'architecture de ce type de réseau est basée sur deux parties, à savoir la partie de convolution et la partie entièrement connectée. La partie convolution est basée essentiellement sur la succession de trois opérations de base qui sont la convolution, la rectification ReLu et le Pooling. La partie entièrement connecté se termine par une couche *softmax* qui donne les probabilités d'appartenance de l'image d'entrée à chacune des classes possibles

Chapitre.II : Circuits FPGA : Architecture et outils de développement

II.1. Introduction

Les circuits logique programmable de type FPGA, « Field Programmable Gates Arrays » sont des circuits intégrés formés par des blocs élémentaires dont les liaisons physique entre eux est reprogrammables par logiciel. De ce fait, ils offrent la possibilité de réaliser des fonctions numériques plus ou moins complexes selon le choix de l'utilisateur toute en offrant la possibilité de changer les fonctions réalisées à volonté.

Actuellement, les FPGAs sont de plus en plus utilisés dans des applications de traitement numérique des signaux et en communication numérique. Les FPGAs sont des composants VLSI (Very Large Scale Integrated Circuits), entièrement reconfigurables, ce qui permet de les reprogrammer afin d'accélérer notablement certaines phases de calculs. L'avantage de ce genre de circuit est sa grande souplesse, qui permet de les réutiliser dans des algorithmes différents en un temps très court. Le deuxième avantage des FPGAs, c'est qu'ils contiennent de multiples ressources identiques ce qui facilite de paralléliser des phases de calculs. Ces atouts justifient le succès de ces composants dans tous les domaines d'applications de l'électronique, l'aéronautique, l'industrie, les télécommunications, les appareils médicaux l'électroménager…etc,

Un circuit FPGA dispose d'une importante quantité de ressources matérielles opérationnelles dont on peut configurer la fonction. Ces ressources sont, principalement, des blocs élémentaires logiques, des mémoires RAM, des opérateurs arithmétiques, des ressources de routage interne et des entrées/sorties. Ces ressources configurables sont reliées par un réseau dense de lignes de routage et de lignes de transport des horloges. Ces lignes de routage sont aussi configurables.

II.2. Architecture des circuits FPGA de Xilinx

Pour réussir à implanter efficacement un système dans un circuit de type FPGA, il est indispensable de bien connaitre sa structure interne et ses limites du point de vue performances. Les composants logiques programmables sont des circuits composés de nombreuses cellules logiques élémentaires librement assemblables.

Les circuits FPGA sont constitués d'une matrice de blocs logiques programmables entourés de blocs d'entrée sortie programmable [17]. L'ensemble est relié par un réseau d'interconnexions. Il existe actuellement plusieurs fabricants de circuits FPGA et plusieurs technologies et principes organisationnels. Dans ce chapitre on se limite à la présentation de

l'architecture des FPGAs du constructeur Xilinx.

Les anciennes versions des circuits FPGA de Xilinx, comme illustré dans la figure II.1 reposent sur trois types de ressources qui sont :

- Les blocs logiques configurable CLB « Configurable Logic Block »
- Les blocs entrées/sorties IOB « Input Output Bloc ».
- Les blocs d'interconnexion

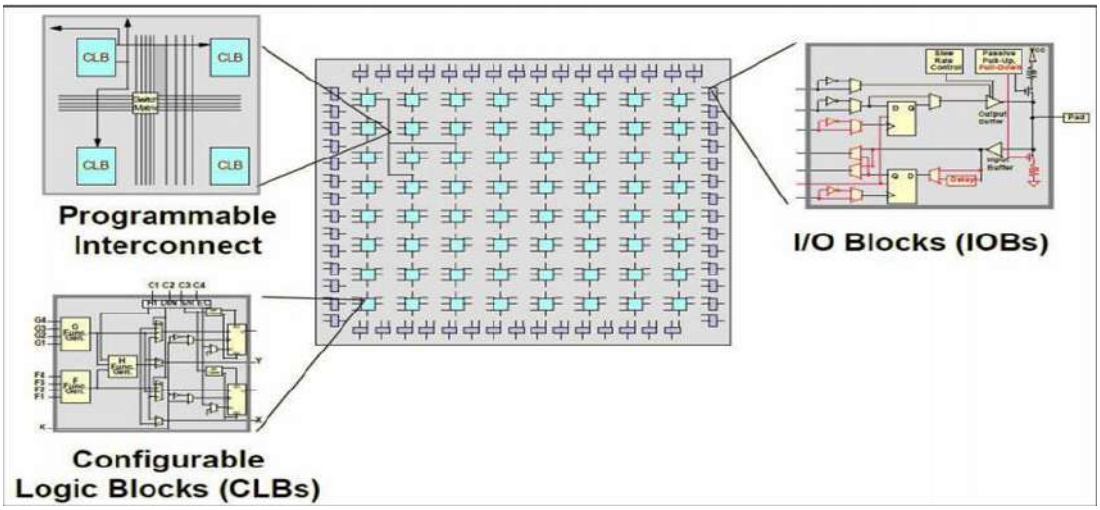


Figure II.1 Architecture interne des anciennes version des circuits FPGA de Xilinx.

Les architectures des versions modernes des circuits FPGA de Xilinx intègre, en plus des blocs classiques, d'autres blocs permettant d'améliorer les performances et d'optimiser l'implémentation des opérations de traitement avancé de signal qui nécessite des ressources spécifique. La figure II.2 donne les blocs incorporés dans les circuits FPGA de la version Virtex 7. [18]

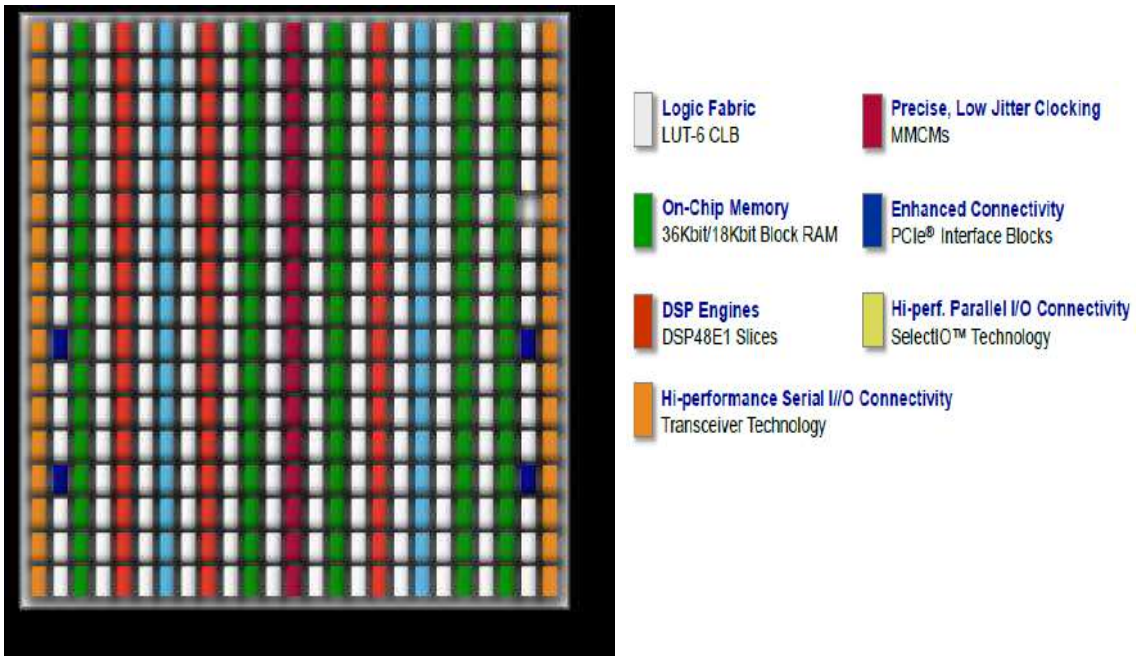


Figure II.2 Architecture interne de la famille Virtex 7 de Xilinx

II.3. Les blocs classiques d'un FPGA de Xilinx [12]

II.3.1. Les CLB (configurable logic bloc)

Les blocs logiques configurables (CLB) sont les principales ressources logiques pour l'implémentation des fonctions séquentielles et combinatoires. Chaque élément CLB est connecté à une matrice de commutation (switch matrix) pour l'accès à la matrice générale d'interconnexion. Pour la série 7 des FPGAs de Xilinx, chaque CLB contient une paire de slices qui ne sont pas en connexion directe et dont chacun s'organise en colonne ; étiqueté slice (1) pour la partie supérieur et slice(0) pour la partie inférieure comme le montre la figure II.3 [19].

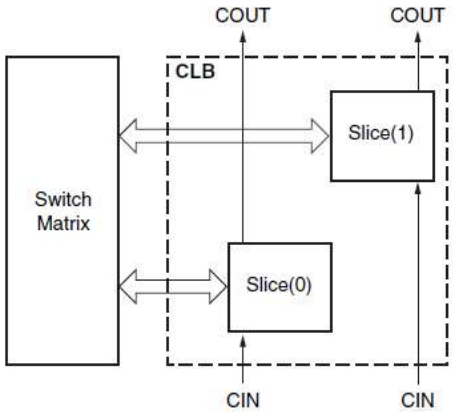


Figure II.3 Arrangement des slices dans un CLB pour la série 7 de Xilinx.

Les slices contiennent essentiellement des générateurs de fonction logique LUT (look_up table), des bascules D, des multiplexeurs, des RAM distribuées et des registres à décalage. Pour la série 7 de Xilinx, chaque slice contient :

- Quatre générateurs de fonctions logiques LUT.
- Huit éléments de mémorisation.
- Des multiplexeurs
- Une logique de retenu

Ces éléments sont utilisés par toutes les slices pour fournir des fonctions logiques, arithmétiques et de type ROM. De plus, certaines slices prennent en charge deux fonctions supplémentaires : stocker des données à l'aide de la RAM distribuée et décaler des données avec des registres 32 bits. Les slices qui prennent en charge ces fonctions supplémentaires sont appelées SLICEM (figure II.4); d'autres sont appelés SLICEL. Chaque CLB peut contenir deux SLICEL ou un SLICEL et un SLICEM.

Les générateurs de fonctions des FPGA de la série 7 sont implémentés sous forme de tables LUT à six entrées. Il existe six entrées indépendantes (entrées A1 à A6) et deux sorties indépendantes (O5 et O6) pour chacun des quatre générateurs de fonctions d'une slice (A, B, C et D).

Les multiplexeurs sont utilisés pour combiner jusqu'à quatre générateurs de fonctions pour fournir n'importe quelle fonction de sept ou huit entrées dans une slice. Ainsi on peut combiner :

- Des fonctions à six entrées générées par LUTs A et B pour former une fonction à sept entrées.
- Des fonctions à six entrées générées par LUTs C et D pour former une fonction à sept entrées.
- Des fonctions à six entrées générées par tous les LUTs d’une slice pour former une fonction à huit entrées.

Pour générer une fonction à plus de huit entrées il faut utiliser plusieurs slices.

Il y a huit éléments de stockage par slice. Quatre peuvent être configurés comme des bascules de type D à déclenchement par front ou des verrous sensibles au niveau. Les quatre autre éléments de stockage ne peuvent être configurés que comme des bascules de type D à déclenchement par front.

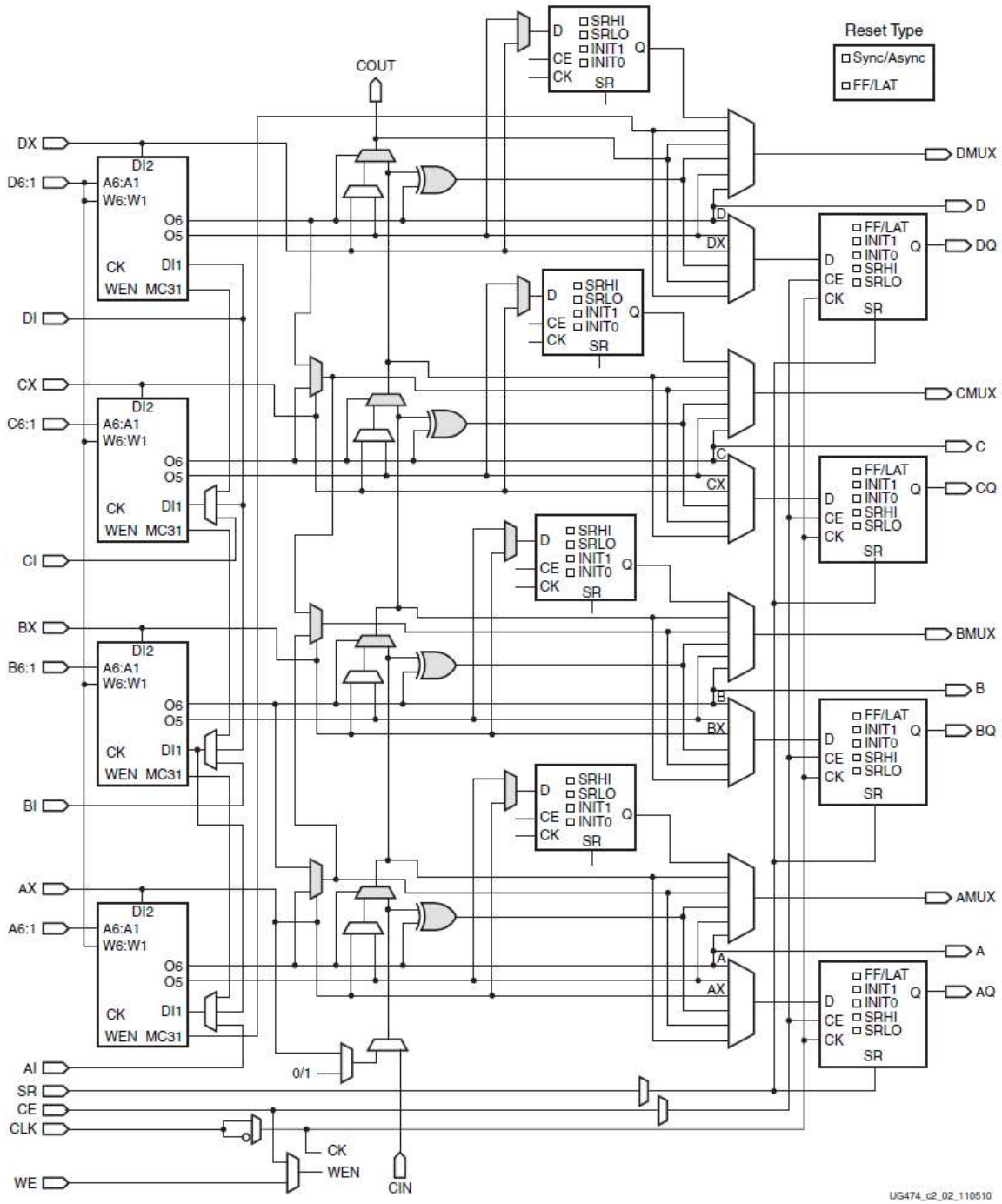


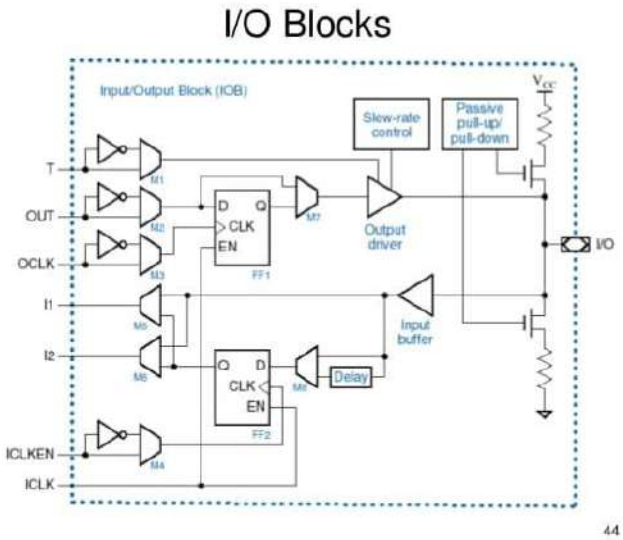
Figure II.4 Structure interne de slice M de la série 7 de Xilinx.

Les générateurs de fonctions (LUT) dans les SLICEM peuvent être implémentés en tant que ressource RAM synchrone appelée élément RAM distribué. Plusieurs LUT dans un SLICEM peuvent être combinés de différentes manières pour stocker une plus grande quantité de données. Un générateur de fonctions LUT de SLICEM peut également être configuré comme un registre à décalage 32 bits sans utiliser les bascules disponibles dans une tranche. Utilisé de cette manière, chaque LUT peut retarder les données série de 1 à 32

cycles d'horloge.

II.3.2. Les IOB (Input Output Bloc)

Les blocs entrée/sortie permettent l'interface entre les composants interne du circuit FPGA et le circuit externe à l'FPGA. Ils sont présents sur toute la périphérie du circuit FPGA. Chaque bloc IOB contrôle une broche du composant et il peut être défini en entrée, en sortie, en signaux bidirectionnels ou être inutilisé (haute impédance). La figue II.5 donne la structure interne d'un block IOB.



FigureII.5 Structure interne d'un bloc IOB

II.3.3. Les interconnexions

Les connexions internes dans les circuits FPGA sont composées de segments métallisés. Parallèlement à ces lignes, nous trouvons des matrices programmables réparties sur la totalité du circuit, horizontalement et verticalement entre les divers CLB. Elles permettent les connexions entre les diverses lignes, celles-ci sont assurées par des transistors MOS dont l'état est contrôlé par des cellules de mémoire RAM. Le rôle de ces interconnexions est de relier avec un maximum d'efficacité les blocs logiques et les entrées/sorties afin que le taux d'utilisation dans un circuit donné soit le plus élevé. Pour parvenir à cet objectif, Xilinx propose trois sortes d'interconnexions selon la longueur et la destination des liaisons (figure II.6) :

- **Les interconnexions directes :** Ces interconnexions permettent l'établissement des liaisons entre les CLB et les IOB. Ce type permet de connecter directement certaines entrées d'un CLB aux sorties d'un autre.

- **Les longues lignes :** Ce sont de longs segments métallisés parcourant toute la longueur et la largeur du FPGA, elles permettent éventuellement de transmettre avec un minimum de retard les signaux entre les différents éléments dans le but d'assurer un synchronisme aussi parfait que possible. De plus, ces longues lignes permettent d'éviter la multiplicité des points d'interconnexion.
- **Les matrices d'interconnexion :** Ce sont des aiguilleurs situés à chaque intersection. Leur rôle est de raccorder les longues lignes entre elles selon diverses configurations. Ces interconnexions sont utilisées pour relier un CLB à n'importe quel autre CLB sur le FPGA pour assurer la communication des signaux. Pour éviter l'affaiblissement des signaux traversant les longues lignes, des buffers sont implantés dans chaque matrice d'interconnexion.

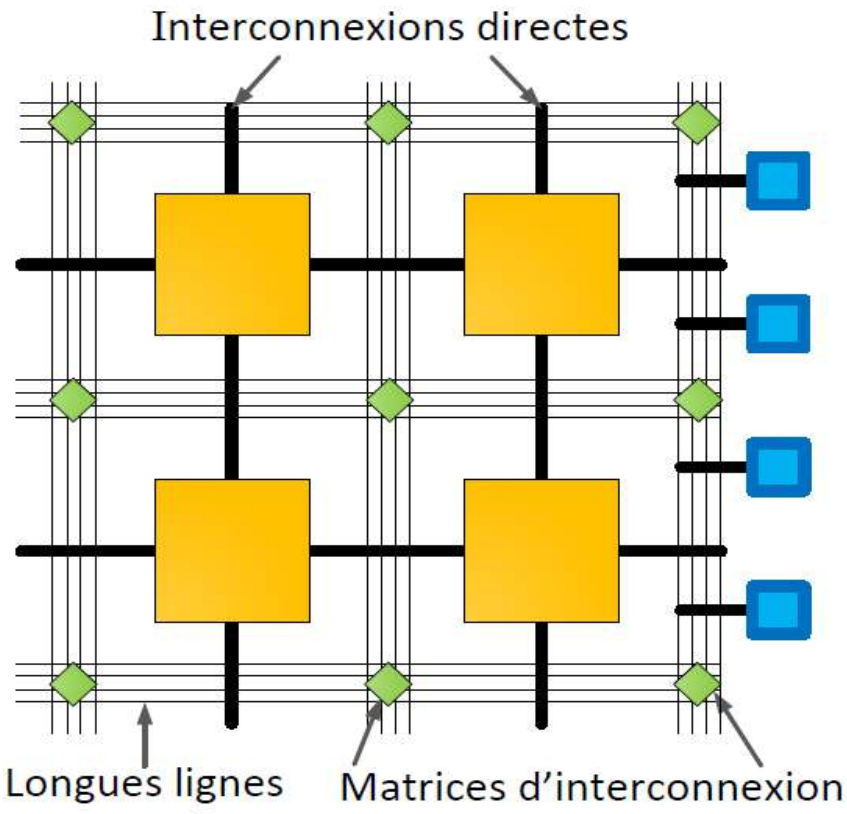


Figure II.6 Les différentes lignes d'interconnexions.

II.4. Les blocs à usage spécifique

Pour améliorer les capacités de certains circuits FPGA en termes de puissance de calcul, les constructeurs ont intégré d'autres blocs spécifiques en plus des trois blocs décrits dans le paragraphe précédent.

II.4.1. Les blocs RAM (BRAM)

Les blocs RAM sont des mémoires embarquées sur le circuit FPGA, l'utilisateur du circuit peut utiliser ces blocs de mémoire pour stocker des ensembles de données. La taille de chaque bloc RAM dépend de la catégorie du circuit FPGA. Par exemple une BRAM des FPGA de la série 7 de Xilinx stocke jusqu'à 36 Kbits de données et peut être configuré comme deux RAM indépendantes de 18 Kb, ou une RAM de 36 Kb. Chaque bloc RAM de 36 Ko peut être configuré en tant que 64K x 1 (lorsqu'il est monté en cascade avec un bloc RAM adjacent de 36 Ko), 32K x 1, 16K x 2, 8K x 4, 4K x 9, 2K x 18, 1K x 36 512 x 72 en mode double port simple. Chaque bloc RAM de 18 Ko peut être configuré en tant que 16K x 1, 8K x2, 4K x 4, 2K x 9, 1K x 18 ou 512 x 36 en mode double port simple.

II.4.2. Les blocs DSP

Les blocs DSP sont des blocs qui permettent l'implémentation efficace des opérations complexes qui sont nécessaire pour le traitement numérique du signal et pour d'autres opérations. Par exemple, on peut les utiliser pour faire l'implémentation efficace des additionneurs et des multiplieurs complexes, et pour l'implémentation des générateurs d'adresses mémoire et des multiplexeurs à nombre d'entrées très élevé. Les blocs DSP sont conçus pour améliorer la vitesse, la consommation de puissance et l'efficacité de nombreuses applications en traitement du signal et au-delà du traitement du signal. Le nombre de blocs DSP embarqués sur le circuit FPGA dépend de sa catégorie. La figure II.7 montre l'architecture interne d'un bloc DSPE48 de Xilinx.

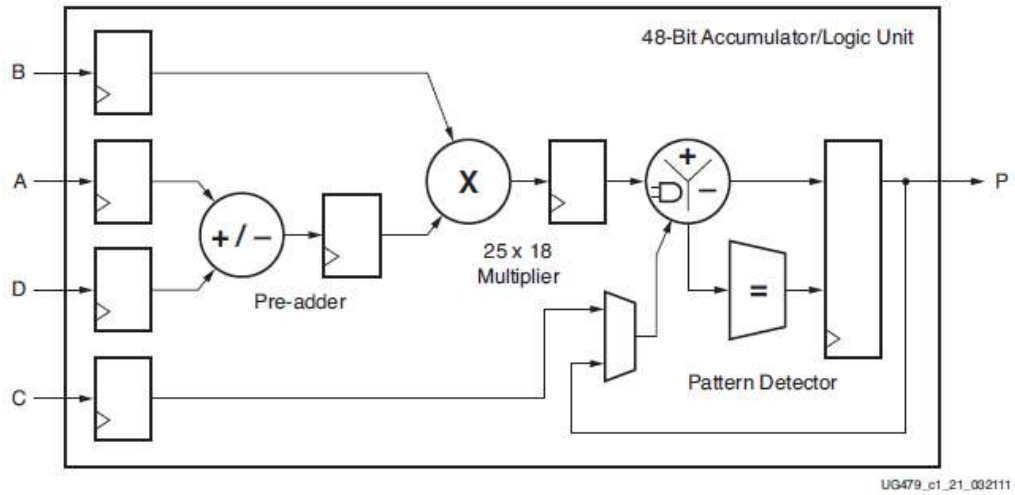


Figure II.7 Architecture interne d'un bloc DSPE48. [20]

Les fonctionnalités importantes des DSPs sont :

- Multiplication 25 × 18.

- Accumulateur 48 bits.
- Pré-additionneur économique en d'énergie.
- Unité arithmétique à instruction unique et données multiples (SIMD).
- Unité logique en option qui peut générer l'une des dix fonctions logiques différentes à deux opérandes
- Logique d'arrondi
- Fonctionnalités avancées, comme pipelining en option et bus dédiés pour la mise en cascade

II.4.3. Les processeurs embarqués

Les processeurs embarqués sont l'un des blocs spécifiques les plus importants pour les circuits FPGA car beaucoup d'applications nécessitent l'utilisation d'un processeur. L'utilisation d'un circuit FPGA qui contient un processeur embarqué (comme le Virtex de Xilinx 5) permet au développeur de simplifier grandement le processus de conception et de réduire l'utilisation des ressources FPGA et la consommation d'énergie. Le constructeurs Xilinx intègre dans ces circuits FPGA deux familles de processeurs embarqués qui sont les picoBlaze [21] et microBlaze [22].

Le microcontrôleur PicoBlaze™ est un cœur de microcontrôleur RISC 8 bits entièrement intégré compact, performant et économique, optimisé pour les circuits FPGA de Xilinx. La version KCPSM6 du microcontrôleur PicoBlaze est optimisée pour les FPGA Spartan-6, Virtex-6 et 7 et exploite les progrès technologiques pour fournir un degré d'efficacité encore plus élevé que la version KCPSM3. Le microcontrôleur KCPSM6 est incroyablement petit, occupant seulement 26 slices. Le cœur du microcontrôleur PicoBlaze est totalement intégré au FPGA cible et ne nécessite aucune ressource externe. Le microcontrôleur PicoBlaze est extrêmement flexible. La fonctionnalité de base est facilement étendue et améliorée en connectant une logique FPGA supplémentaire aux ports d'entrée et de sortie du microcontrôleur. La figure II.8 est un schéma bloc de la structure interne du microcontrôleur de type PicoBlaze.

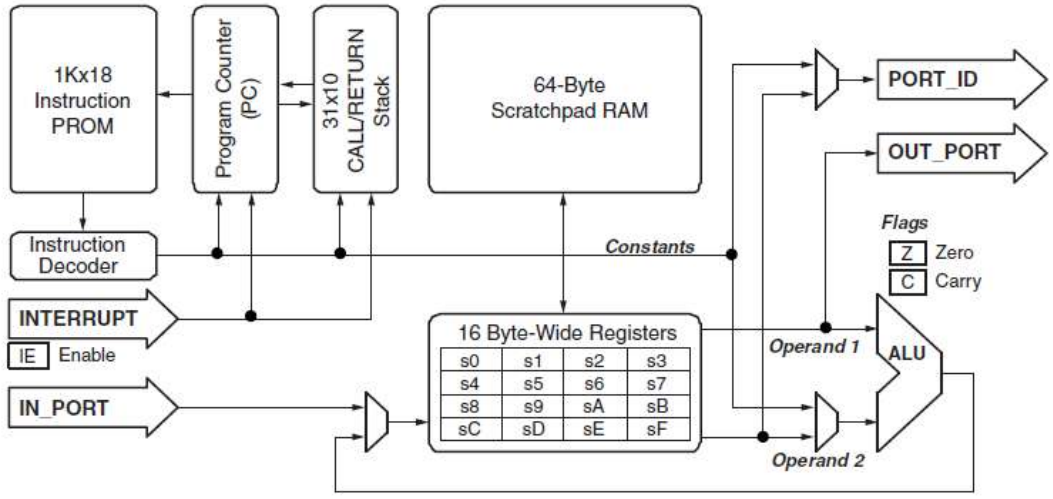


Figure II.8 Schéma bloc d'un microcontrôleur embarqué de type PicoBlaze. [21]

Le processeur embarqué ou soft core MicroBlaze est un microcontrôleur à jeu d'instructions réduit (RISC) optimisé pour la mise en œuvre dans les FPGA de Xilinx. La figure II.9 montre un schéma fonctionnel du cœur MicroBlaze [22].

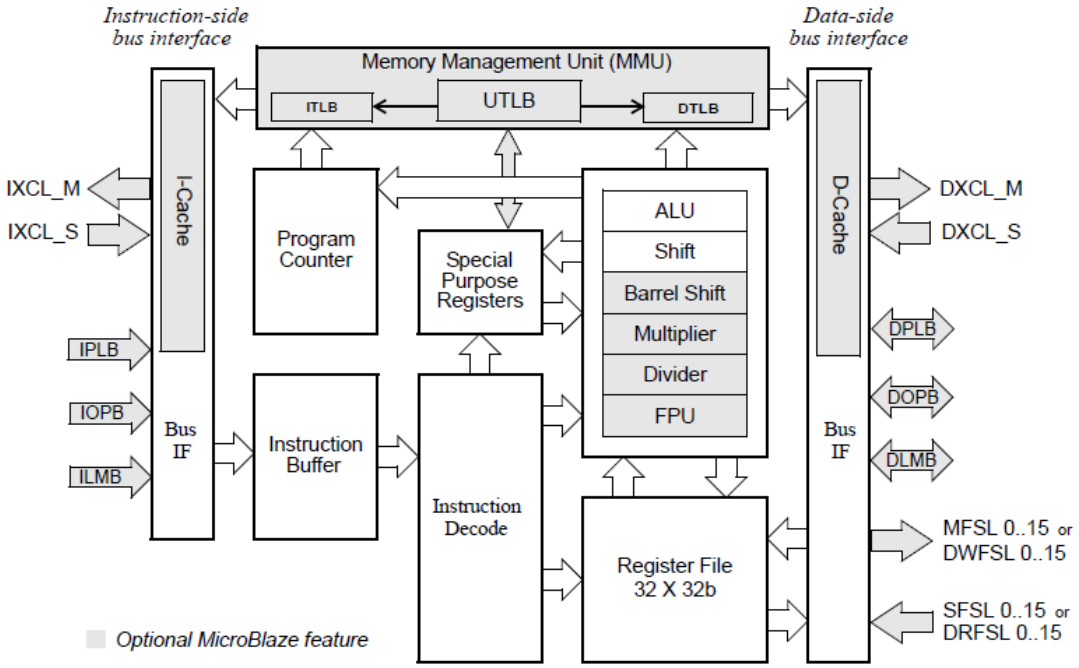


Figure II.9 Schéma bloc d'un microcontrôleur embarqué de type MicroBlaze. [22]

Le processeur Soft Core MicroBlaze est hautement configurable, il permet de sélectionner un ensemble spécifique de fonctionnalités requises par la conception.

L'ensemble de fonctionnalités fixes du processeur comprend :

• Trente-deux registres à usage général 32 bits

• Mot d'instruction de 32 bits avec trois opérandes et deux modes d'adressage

• Bus d'adresses 32 bits

• Pipeline des instructions.

En plus de ces fonctionnalités fixes, le processeur MicroBlaze est paramétré pour permettre en œuvre sélective de fonctionnalités supplémentaires comme les mémoires caches, les diviseurs, les multiplieurs à résultats sur 64 bits, les opérateurs en virgule flottante…

II.5. Les outils de développement des FPGAs

Les outils de développement logiciels permettent de réaliser la totalité du flot de conception d'une application à base de circuit FPGA.

II.5.1. Flot de conception

Pour mener à bien une application sur FPGA, le concepteur suit les étapes de développement suivantes :

II.5.1.1. Spécification de l'entrée

Cette étape consiste à décrire le circuit électronique à implémenter sur FPGA. Pour la spécification on peut utiliser un langage de description de matériel (hardware description language : HDL) comme VHDL ou verilog ou un schéma électrique du circuit en utilisant des éditeurs de schéma électriques intégrés avec les logiciels de conception. Les versions récentes permettent de faire la spécification en utilisant un langage usuel de haut niveau comme le langage C, Matlab et on peut utiliser même le Simulink pour cela.

II.5.1.2. Simulation et vérification

La simulation est une étape importante dans le flot de conception sur FPGA. A ce point, nous avons une conception qui doit être vérifiée. Le but de la simulation est de vérifier que la conception effectue les fonctions requises. Typiquement, la simulation peut être effectuée à trois endroits dans le processus de développement : après la spécification (simulation fonctionnelle), après la synthèse (simulation fonctionnelle) et après le routage (simulation temporelle).

II.5.1.3. Synthèse

La synthèse d'un circuit consiste à traduire la description du circuit en blocs disponibles dans la technologie utilisée. Par exemple, pour un circuit décrit avec un schéma et qui doit être réalisé sur un FPGA, le processus de synthèse convertit et regroupe les portes logiques du schéma en composantes réalisables sur le FPGA choisi. L'étape de synthèse génère un fichier appelé NETLIST qui contient les ressources FPGA utilisées ainsi que la

description de leurs interconnexions.

II.5.1.4. Mapping, placement et routage :

Le mapping est une opération de dimensionnement de la conception en fonction des ressources internes du composant cible. Le placement consiste à sélectionner les modules élémentaires du circuit FPGA cible (slices, CLB, LUTs, BRAM…) où les éléments de la conception seront placés. Le routage permet de spécifier les interconnexions physiques entre les différentes ressources issues de l'étape de placement.

II.5.1.5. Configuration :

Après l'étape de routage qui prend en compte l'architecture du FPGA, un fichier de configuration appelé bitstream est généré. Celui-ci permet de spécifier au FPGA les différents points d'interconnexion. Lors de la configuration le fichier bitstream est chargé dans une RAM interne du circuit FPGA.

II.5.2 Vivado Design Suite

Vivado Design Suite est une suite logicielle produite par Xilinx pour la synthèse et l'analyse de conceptions HDL, remplaçant Xilinx ISE avec des fonctionnalités supplémentaires pour le développement de systèmes sur puce et la synthèse de haut niveau [23]. Vivado a été introduit en 2012 et est un environnement de conception intégré (IDE) avec des outils de niveau système à IC construits sur un modèle de données évolutif partagé et un environnement de débogage commun.

L'environnement de conception intégré (IDE) de Vivado fournit une interface utilisateur graphique (GUI) intuitive avec des fonctionnalités puissantes. L'analyse et l'affectation des contraintes sont activées tout au long du processus de conception. Par exemple, on peut exécuter des estimations de puissance après la synthèse, le placement ou le routage. Les modifications des contraintes, de la configuration de conception ou des paramètres d'outils se produisent en temps réel, souvent sans forcer la ré implémentation.

On peut améliorer les performances de conception en utilisant les nouveaux algorithmes fournis par l'IDE Vivado, comprenant :

• Conception au niveau de transfert de registre (RTL) utilisant VHDL, Verilog et

SystemVerilog ;

• Intégration de blocs IP « Intellectual property »

• Simulation comportementale, fonctionnelle et temporelle avec le simulateur Vivado

• Synthèse Vivado

• Implémentation Vivado pour le placement et le routage

• E/S série et analyseur logique pour le débogage

• Analyse de puissance

• Analyse de timing statique

• Aménagement d'étage de haut niveau

• Génération de flux de bits.

La figure II.10 donne une vue globale de l'environnement de visualisation Vivado IDE

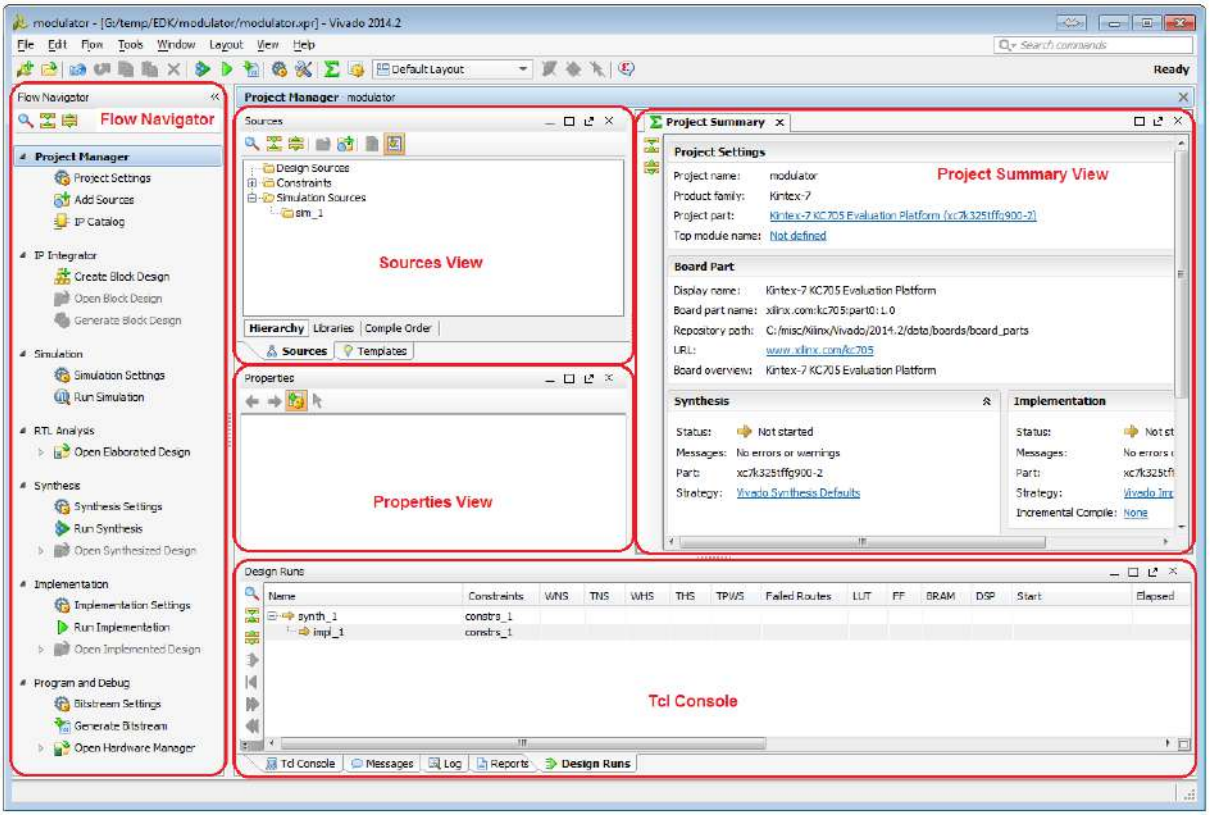


Figure II.10 L'environnement de visualisation de l'IDE de Vivado.

II.5.3 System Generator [24]

System Generator est un outil qui permet de développer, d'une façon simple et rapide des applications sur FPGA. Constituant une interface entre MATLAB/Simulink et VIVADO, il permet de concevoir et de simuler des systèmes numériques dans un environnement graphique à partir de blocs IP et des codes MATLAB. Ces blocs permettent

de réaliser toutes les opérations communément utilisé par un système numérique (registre, mémoire, compteur, multiplexeur…). En outre, System Generator contient des blocs DSP pour réaliser d'une façon efficace les opérations liées au traitement numérique du signal (additionneurs, multiplicateurs, FFT, filtres ...). Dans System Generator sont aussi inclus des blocs Mcode qui permettent l'utilisation de programmes MATLAB pour la modélisation et l'implémentation des opérations de contrôle simple. System Generator fournit aussi un bloc estimateur de ressources qui permet de faire une estimation rapide des ressources nécessaires pour l'implémentation d'un design sur FPGA.

II.6. Conclusion

Ce chapitre a été consacré à la présentation des circuits FPGA qui d'inscrivent dans la gamme des circuits numériques à haute performance, haut niveau de flexibilité et cout de réalisation raisonnable. Nous avons commencé par la présentation de l'architecture de base qui est basée sur des éléments à usage générale et des éléments à usage spécifique. Les trois éléments à usage générale sont les CLB, les IOB et les interconnexions. Parmi les éléments à usage spécifique les plus importants on peut citer : les blocs de calcul DSP, les mémoires en bloc BRAM, les processeurs embarqué et les gestionnaire d'horloge numérique. Nous avons vue aussi que le développement d'une application à base de FPGA est facilite par l'introduction de plusieurs logiciels de conception. Enfin nous avons présenté les différentes étapes de développement d'une application à base de circuit FPGA et les outils logiciels qui facilitent la conception. La programmation d'un circuit FPGA peut être achevée par les langages de description matérielle ou l'environnement Matlab-Simulink en utilisant l'outil« Xilinx-System-Generator ». Pour notre cas, nous avons utilisé l'outil « Xilinx-System-Générator » pour la conception des différentes architectures de notre projet.

Chapitre III

Résultantes de simulation et d'implémentations

III.1 Introduction

Ce chapitre représente le fruit de notre travail de fin d'étude. Il est consacré à l'explication détaillée des schémas de conception des programmes Matlab utilisés et à la présentation des résultats de simulation et des résultats d'implémentation d'un réseau de neurones CNN sur circuit de type FPGA. Pour mener à bien la conception nous avons utilisé un environnement mixte Matlab, System Generator et simulink. Nous avons ciblé l'application de reconnaissance automatique de digits manuscrits car elle ne nécessite pas des réseaux de neurones très compliqués. En outre, la base de donnée nécessaire pour le test et l'apprentissage du réseau pour cette application est disponible gratuitement sur internet. Les simulations sont réalisées en virgule flottante simple précision sous l'environnement system Generator et en virgule flottante double précisions sous l'environnement Matlab. Pour valider les conceptions on fait la comparaison entre les résultats de simulation de System Generator et les résultats de Matlab. Après les deux étapes de conception et de simulation nous avons passé à l'étape d'implémentation en utilisant l'outil Vivado de Xilinx. Le circuit FPGA cible d'implémentation que nous avons choisi pour notre application est le Virtex7 XC7VX485T-2FFG1761C.

III.2 Les étapes de développement :

Pour la réalisation de notre projet nous avons suivi les étapes suivantes :

1. Conception, apprentissage et validation du réseau CNN sous l'environnement Matlab et en utilisant les fonctions de « ***Deep Learning Toolbox*** ».
2. Développement d'un programme Matlab pour la classification d'une image quelconque de la base de données en utilisant les paramètres (poids et biais) du réseau conçu dans l'étape 1.
3. Conception des circuits de calcul de convolution, de relu et de MaxPooling pour les différentes couches du réseau CNN en utilisant le logiciel System Generator.
4. Implémentations des différents designs en utilisant le logiciel VIVADO.

III.3 Conception, apprentissage et validation du réseau CNN :

L'application envisagée par notre réseau convolutif est de faire une reconnaissance automatique des digits manuscrits en utilisant l'apprentissage profond. L'apprentissage consiste à classer l'image d'un digit écrit à la main dans l'une des dix classes possible de zéros à neuf. Le programme développé contient les taches suivantes :

III.3.1 Charger et explorer les données d'image :

La première partie du programme consiste à charger la base de données des échantillons de chiffres en tant que banque de données d'images «imageDatastore ». Pour ce faire, on doit d'abord spécifier le répertoire où se trouve la base de donnée puis en utilise la fonction *imageDatastore* qui étiquette automatiquement les images en fonction des noms de dossier et stocke les données en tant qu'objet *ImageDatastore.*

III.3.2 Spécifier les ensembles d'apprentissage et de validation

Pour faire l'apprentissage et la validation du réseau CNN, on doit diviser les données en ensembles de données d'apprentissage et de validation, de sorte que chaque catégorie de l'ensemble d'apprentissage contienne pratiquement 75% des images et que l'ensemble de validation contienne les images restantes de chaque étiquette (25%). Pour cela on utilise la fonction s*plitEachLabel* pour divise le datastore total en deux nouveaux datastores, *trainDigitData* et *valDigitData.*

Pour notre application nous avons choisi la base de données MNIST. Cette base contient 70000 échantillons d'images de digits écrits à la main sous format png. La base est divisée en dix sous ensemble dont chaque ensemble contient environ 7000 images. Des échantillons de cette base sont montrés dans la figure III.1.



Figure III.1 Exemples d'échantillons de la base MNIST.

III.3.3 Définition de la structure du réseau

La définition de la structure du réseau consiste à déterminer les paramètres des différentes couches du réseau CNN qui sont :

- La couche de l'image d'entrée (image input layer)
- Les couches de convolution (convolution layers)
- Les couches d'activations ReLu (ReLu layers)
- Les couches d'échantillonnage (Pooling Layers)
- Les couches entièrement connectées (Fully connected layers)
- La couche Sofmax (Softmax Layer)

- La couche de classification (Classification layer).

Pour notre application nous avons cherché un compromis entre la précision de classification et la complexité d'implémentation du réseau. Pour cela nous avons choisi la structure du réseau de la figure III.2.

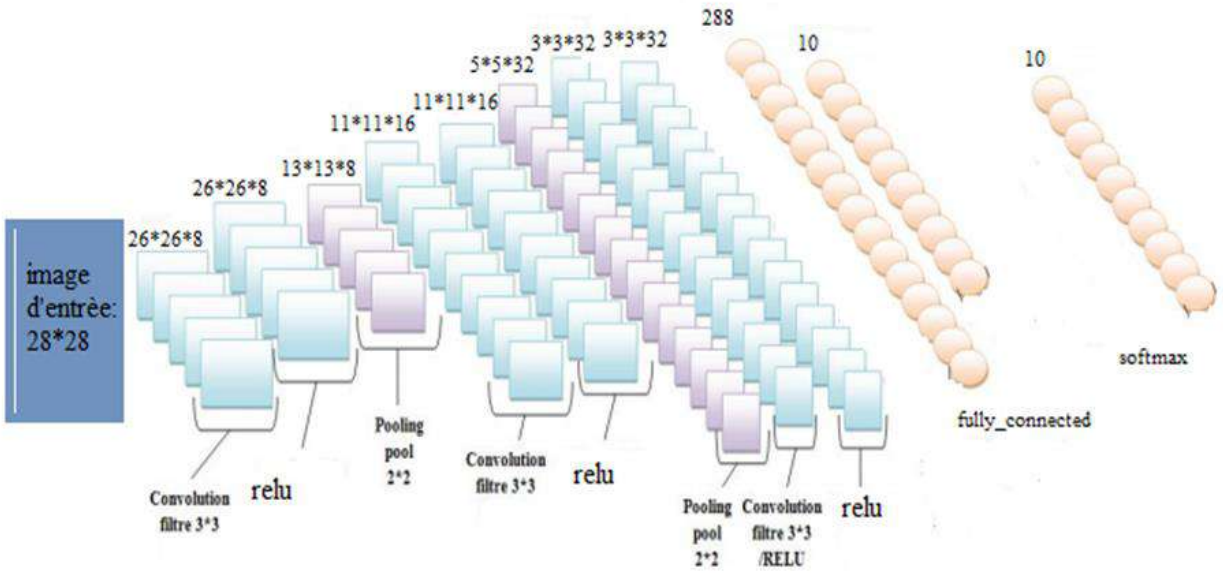


Figure III. 2 Architecture du réseau choisi pour la classification des digits manuscrits

Le modèle proposé est composé de trois couches de convolution, trois couches ReLu, deux couches de max pooling, une seule couche entièrement connectée, une couche Soft max et une couche de classification.

L'image en entrée est une image en niveau de gris codé sur 8 bits sous format png de taille 28×28. La première couche de convolution est composée de 08 filtres de taille 3×3 et utilise un pas égale à 1, ce qui permet de calculer 8 maps de caractéristiques de dimension 26×26 chacune. La couche de convolution est suivie d'une fonction d'activation ReLu cette fonction force les neurones à retourner des valeurs positives ou nulles. Ensuite on applique Max pooling de taille 2×2 et de pas 2 pour réduire la taille des maps de caractéristiques et le nombre de paramètres des couches ultérieures.

À la sortie de la première couche de Max pooling nous aurons 08 maps de caractéristiques de taille 13×13. Ces 08 maps forment l'entrée de la deuxième couche de convolution. Cette couche est composé de 16 groupes de filtre dont chaque groupe contient 8 filtres de taille 3×3 ce qui donne 128 filtres de taille 3×3. La sortie de la deuxième convolution est formée de 16 maps de caractéristiques de taille 11×11. Comme pour la première convolution, la deuxième

convolution est aussi suivie d'une couche d'activation ReLU et d'une couche Max pooling.
A la sortie de la deuxième couche Maxpooling, on aura 16 maps de taille 5×5 qui sont injecté à l'entrée de la troisième couche de convolution. La troisième couche de convolution est formée de 32 groupes de filtres dont chaque groupe contient 16 filtres de taille 3×3 (512 filtres de taille 3×3). Cette couche de convolution est suivie d'une fonction d'activation ReLU.

Après la troisième couche de Relu, on obtient 32 maps de taille 3×3. Ces maps sont aplati et concaténé pour former un vecteur unique de 288 éléments formant l'entrée de la couche entièrement connectée. La couche entièrement connecté contient une matrice de pondération de dimension 288×10 éléments et un vecteur de biais de 10 éléments. Cette couche donne un vecteur de 10 caractéristiques. Le vecteur de caractéristiques et utilisé par la couche SoftMax pour produire une distribution de probabilité à 10 valeurs. Les valeurs de probabilité sont utilisées par la couche de classification pour attribuer une classe à l'image d'entrée selon la probabilité maximale.

La définition de la structure du réseau sous Matlab est réalisée en définissant d'une manière adéquate les paramètres des fonctions suivantes du Deep learning toolbox :
Pour la couche d'entrée : *imageInputLayer*
Pour la convolution 2D : *convolution2dLayer*
Pour l'activation ReLu : *reluLayer*
Pour le Maxpooling : *maxPooling2dLayer*

III.3.4 Spécification des options de l'apprentissage :

Après avoir défini la structure du réseau, on spécifie les options de formation. La spécification consiste à déterminer :

- L'algorithme utilisé pour l'apprentissage, pour notre cas nous avons utilisé l'algorithme de la descente de gradient stochastique avec (SGDM).
- Le taux d'apprentissage initial, la valeur souvent utilisée est de 0,01.
- Le nombre maximum d'époques. Une époque est un cycle d'entraînement complet sur l'ensemble des données d'entraînement. Pour notre cas nous avons choisi 4 époques.
- Surveillance de la précision du réseau pendant la formation en spécifiant les données de validation et la fréquence de validation.
- Mélange des données à chaque époque. Le logiciel entraîne le réseau sur les données d'entraînement et calcule la précision des données de validation à intervalles réguliers pendant l'entraînement. Les données de validation ne sont pas utilisées pour mettre à jour les pondérations du réseau.

- Activation du tracé de la progression de l'entraînement.

III.3.5 L'apprentissage du réseau

Pour entrainer un réseau profond nous devons réunir un volume de données labellisées très important. Pour notre application nous avons utilisé 75% de la base pour faire l'apprentissage. Cette étape est réalisée sous Matlab en utilisant la fonction :

net= trainNetwork(imdsTrain,layers,options);

Avec *imdsTrain* est le data store des images d'apprentissage, *Layers* est la structure du réseau et *options* sont les options d'apprentissage.

Le graphique de progression de l'apprentissage montre la perte et la précision du mini-lot et la perte et la précision de la validation. La perte est la perte d'entropie croisée. La précision est le pourcentage d'images que le réseau classe correctement.

La figure III.3 illustre la fenêtre de progression de l'apprentissage du réseau CNN que nous avons choisie.

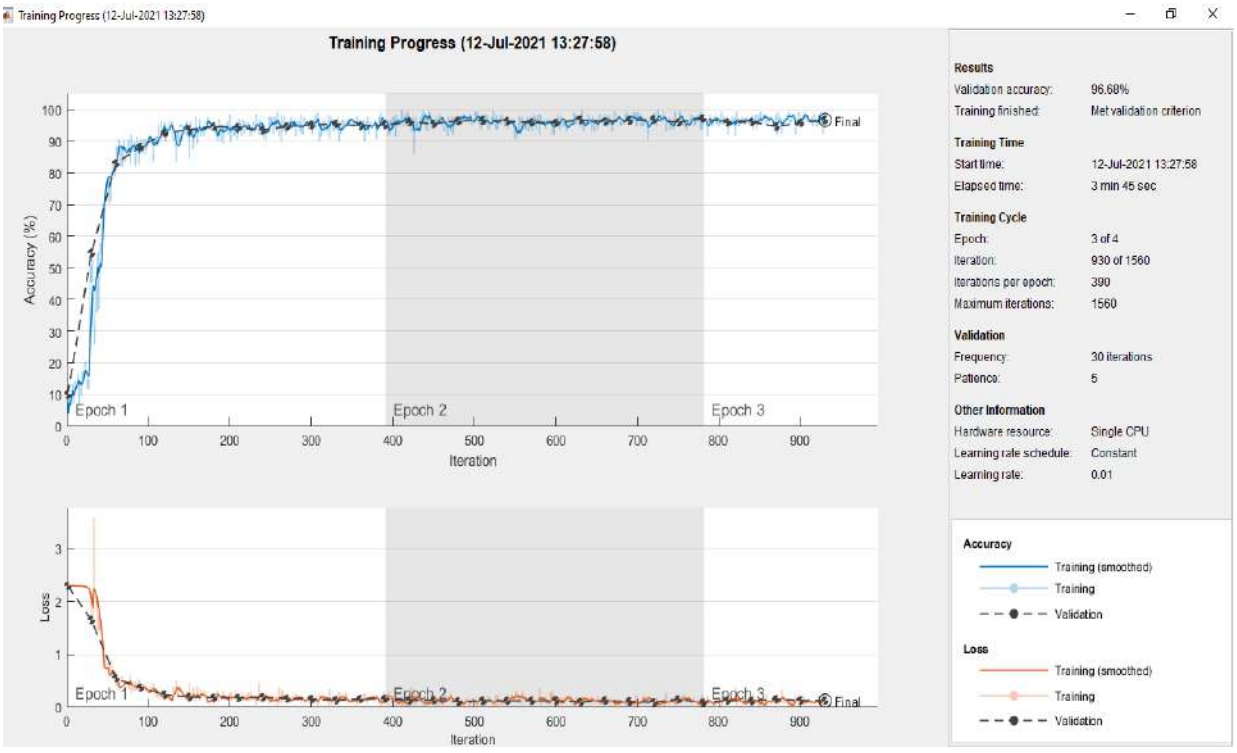


Figure III. 3.Fenêtre de progression de l'apprentissage

Le graphe de précision montre que la précision reste presque constante à partir de la fin du premier epoch. A la fin de l'apprentissage on obtient une précision de 96.68% ce qui est largement acceptable vue la simplicité du réseau.

Après l'étape d'entrainement du réseau il est possible de classifier n'importe quel image de digit

même s'il n'est pas dans la base de donnée en utilisant les paramètres du réseau et en utilisant la fonction *calssifiy(net, 'image_name').*

Pour ne pas refaire la tâche d'apprentissage à chaque fois qu'on a besoin de faire une classification, il est possible de sauvegarder le réseau entrainé en utilisant la fonction *save name_net,* ou *name_net* désigne le nom du réseau de destination. Le réseau sauvegardé sous le nom *name_net* peut être chargé à tout moment en utilisant *load name_net.*

III.4 Développement d'un programme Matlab pour la classification

Dans cette partie du travail, nous avons développé notre propre programme sous Matlab pour calculer les sorties des différentes couches du réseau pour une image donnée en entrée du réseau. Cette partie nous servie beaucoup à la compréhension de la méthode de calcul des différentes sortie afin de faire la conception d'une façon correcte pour l'étape suivante. La vérification du bon fonctionnement du programme a été réalisée en comparant les matrices représentant les maps de caractéristiques de chaque couche obtenues par notre propre programme avec les maps de caractéristiques calculées en utilisant les fonctions du Deep Learning Toolbox.
Le calcul des sorties des couches convolution1, Maxpooling 1, ReLu 1, Maxpooling 2, ReLu 2, ReLu 3 nécessite l'utilisation de trois boucles imbriquées. Le calcul des sorties des deux couches Convolution 2 et convolution 3 nécessite quatre boucles imbriquées.

III.5 Conception des circuits de calcul

Cette phase est pratiquement la partie la plus importante de notre travail car c'est dans cette étape qu'on a fait la conception des circuits qui font le calcul de convolution, de ReLu et de Maxpooling pour les différentes couches du réseau. Ces circuits vont être ensuite convertis en RTL (registre transfert Level) et implémentés sur le circuit FPGA cible.
Dans cette partie nous avons utilisé le logiciel System Generator pour faire la conception des circuits de calcul et le logiciel Matlab pour faire la mise en forme des résultats intermédiaire de calcul.
Donc, cette étape est réalisée à travers des altérations consécutives entre calcul (convolution, ReLu et maxppoling) et mise en forme comme montré dans la figure III.4.

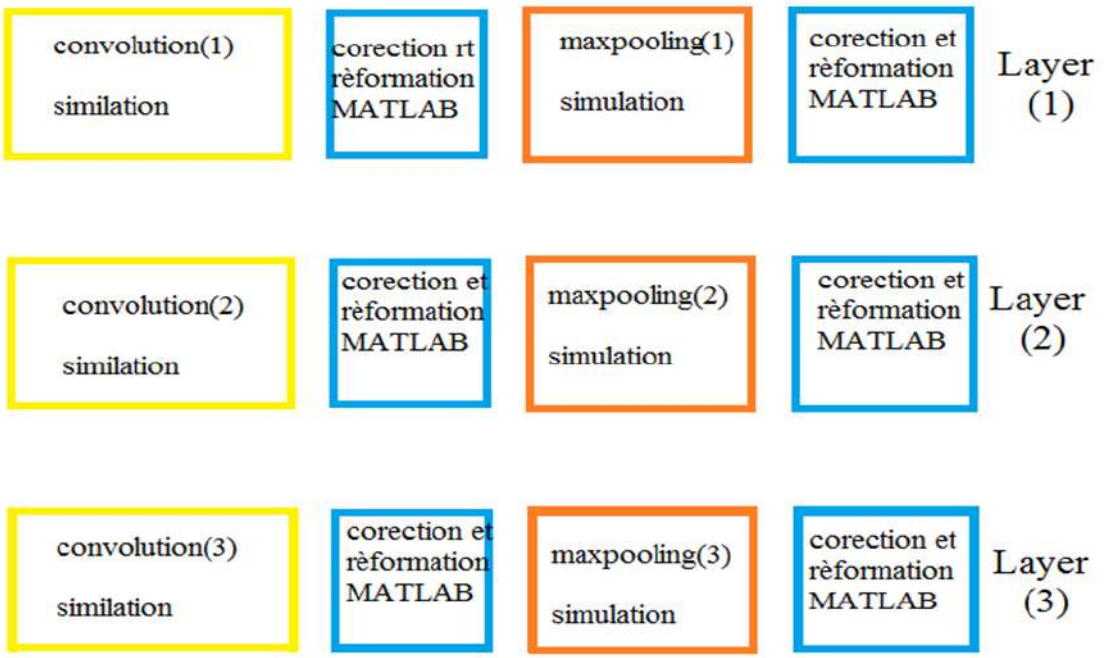


Figure III.4. Illustration des différentes phases de l'étape de conception des circuits de calcul

III.5.1 Circuit de calcul de la convolution 1

Le circuit réalisé permet de calculer d'une façon parallèle les huit maps de caractéristiques de la première couche de convolution. Comme montré sur la figure III.5, le circuit est composé de huit parties identiques notées conv_1 à conv_8 dont chaque partie est chargée de calculer une seule convolution.

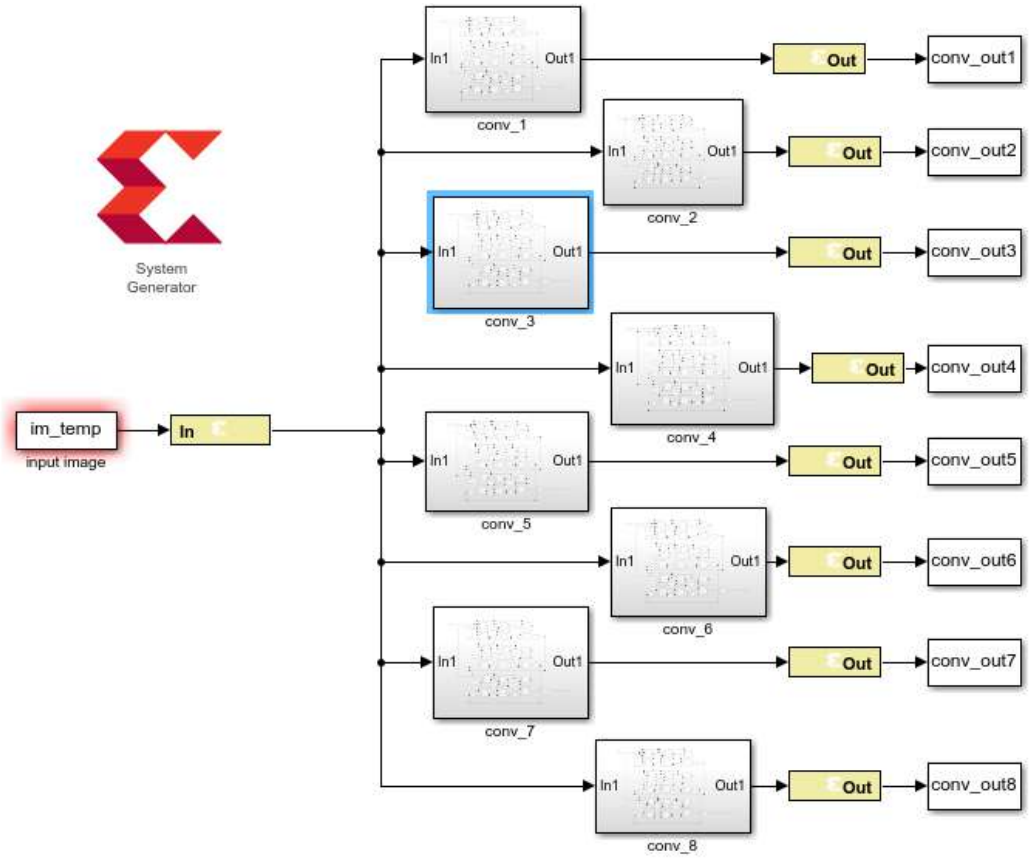


Figure III. 5 Schéma bloc du circuit de calcul de la première couche de convolution

La structure détaillée de l'une des parties conv_1 à conv_8 est donnée dans la figure III.6.

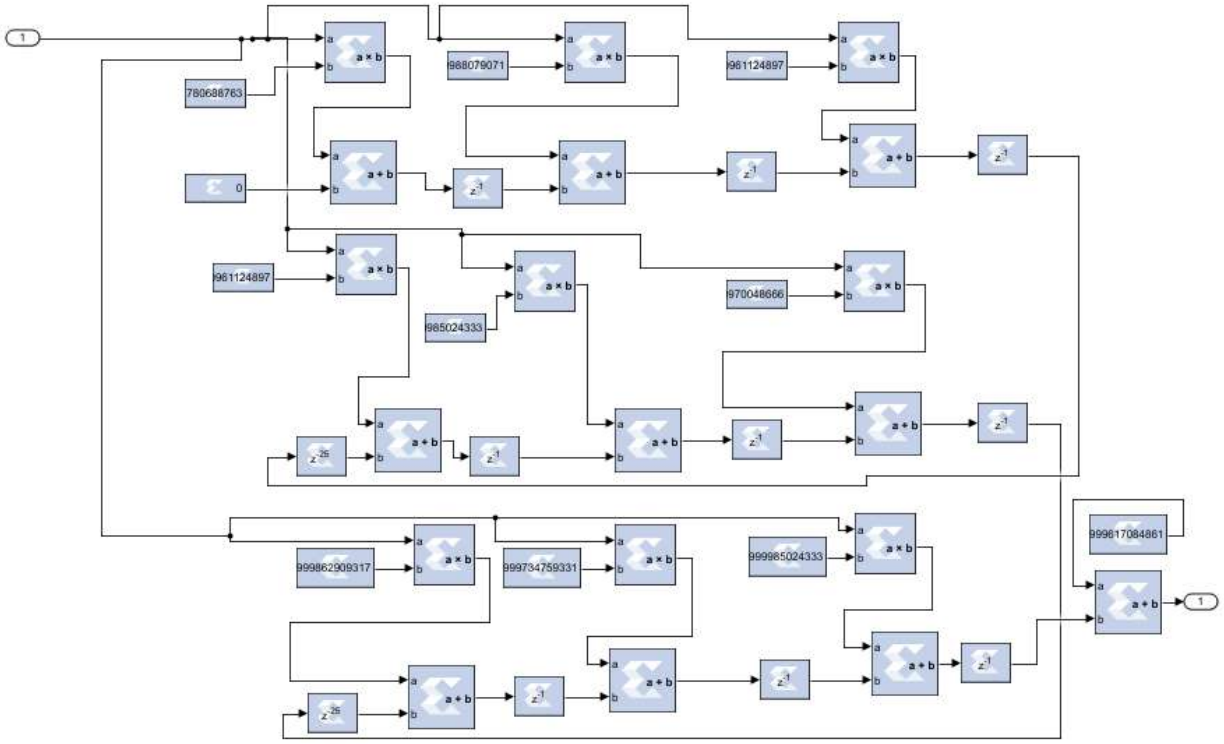


Figure III.6 Architecture détaillée de l'une des huit parties du circuit de calcul de la première couche de convolution.

Chaque partie contient : 9 multiplieurs de nombres réels, 10 additionneurs de nombres réels, 9 éléments de retard à cycle unique, 2 éléments de retard à 26 cycles chacun et 10 éléments de mémorisation pour sauvegarder les neuf paramètres d'un filtre de dimension 3×3 et un seul biais. Donc, en tous, le circuit de calcul de la première couche de convolution nécessite : 72 multiplieurs, 80 additionneur, 72 éléments de retard à cycle unique, 16 éléments de retard à 26 cycles et 80 éléments de mémorisation.

Il est important de noter que chaque branche du circuit de calcul de la première convolution introduit un retard global au début de calcul égal à 58 cycles horloge en plus d'un retard de deux cycles horloge au début de chaque nouvelle ligne de la matrice de convolution. On outre, chaque branche donne le résultat de convolution sous forme d'un vecteur unique. Pour reformuler les résultats de chaque branche sous forme d'une matrice 26×26 on doit :

- Supprimer le retard global de 58 cycles,
- Supprimer les retards de 2 cycles au début de chaque ligne,
- Reformuler le vecteur résultant de 676 en matrice 26×26,

Ces opérations sont réalisées à l'aide du programme de reformulation que nous avons développé sous l'environnement Matlab.

III.5.2 Circuit de calcul d'activation ReLu 1 :

Le circuit de calcul de ReLu est simple, il est composé de huit branches identiques dont chaque branche contient un multiplexeur, un extracteur du bit de signe et d'un élément de mémorisation.

III.5.3 Circuit de calcul de Max pooling 1 :

Le circuit de calcul du max pooling permet d'appliquer l'opération de max pooling sur chacune des 8 maps de la sortie de la couche ReLu1. Il est formé de huit branches dont chaque branche possède la structure montrée dans la figure III.7.

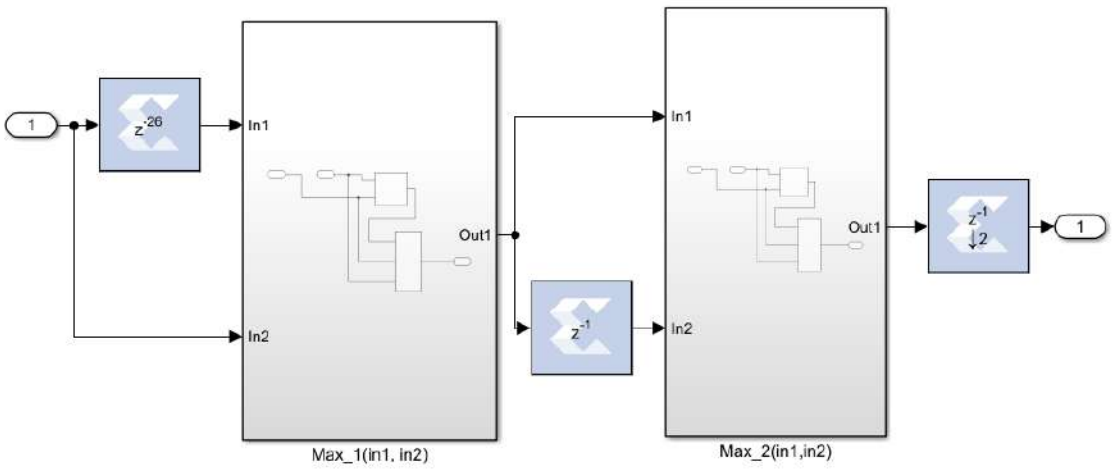


Figure III.7 Structure de l'une des huit branches du circuit de calcul du max pooling 1.

Chaque branche est composée d'un élément de retard de 26 cycles horloge, d'un élément de retard d'un seul cycle, d'un élément de sous échantillonnage et de deux éléments de calcul de maximum. Un élément de calcul de maximum permet de calculer le maximum de deux entrées, il est composé d'un comparateur et d'un multiplexeur.

III.5.4 Circuit de calcul de la convolution 2 :

Le circuit de calcul pour la deuxième convolution est trop compliqué en comparaison avec le circuit de calcul de la première couche de convolution. Il est composé de 16 groupes de calcul identiques dont chaque groupe permet de calculer l'une des 16 maps de caractéristiques de la couche. La structure d'un groupe de ce circuit est détaillée dans la figure III.8.

Chaque groupe du circuit est formé de huit branches de calcul de convolution notées conv_1 à conv_8. Les sorties des huit branches sont additionnées à l'aide de 7 additionneurs pour former une convolution unique. Cette dernière est additionnée à un biais pour former une map de caractéristiques. Chacune des huit branches de convolution possède une structure identique à celle de la figure III.6.

Le circuit de calcul de la convolution 2 nécessite : 1280 additionneurs, 1152 multiplieurs, 1152 éléments de retard à un seul cycle et 304 éléments de retard à 10 cycles.

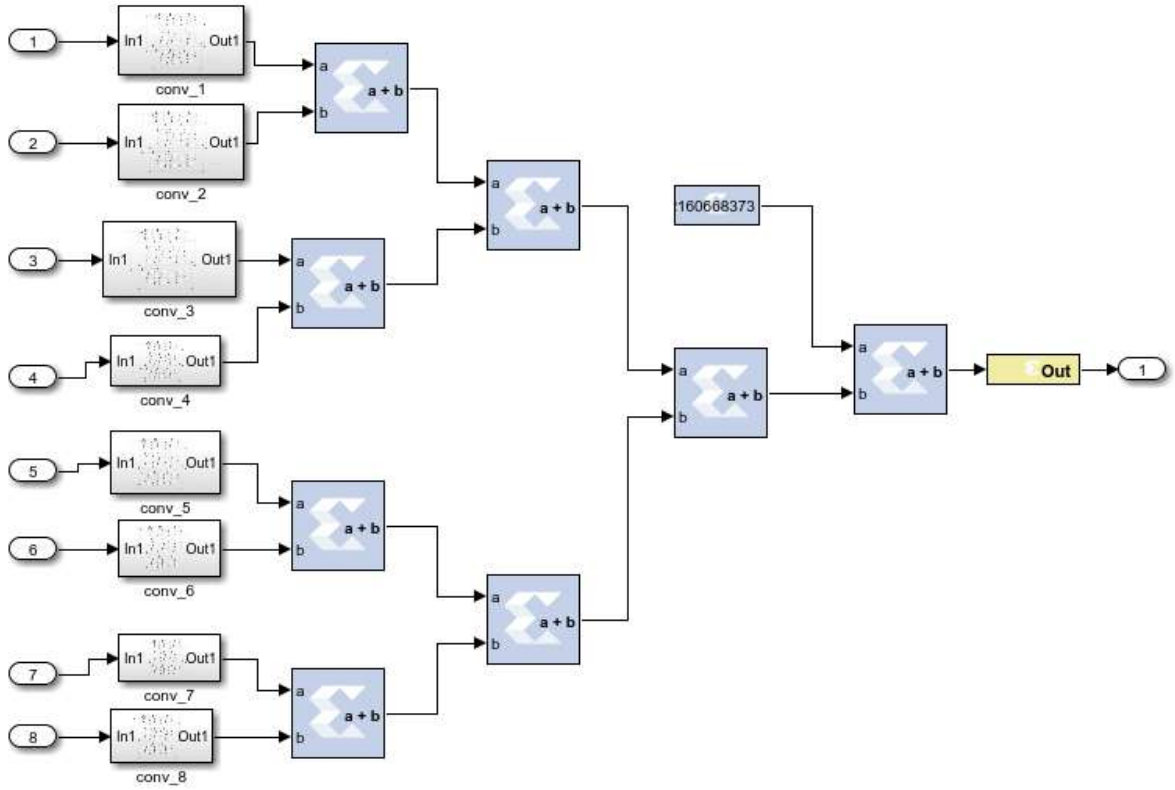


Figure III.8 Schéma bloc d'un groupe du circuit de calcul de la convolution 2.

Les circuits de calcul de ReLu 2 et de max pooling 2 sont identiques aux circuits de calcul de

ReLu1 et de max pooling 1 respectivement.

III.6 Résultats de l'implémentation :

Après vérification du bon fonctionnement de chacun des circuits de conception expliqués dans le paragraphe précédent, nous avons généré leurs model HDL « Hardware Discription Language ». Ensuite nous avons passés à l'environnent Vivado pour faire l'implémentation.
Le tableau III.1 donne les ressources du circuit FPGA Virtex7 XC7VX485T-2FFG1761C consommées par le circuit de calcul de la première couche de convolution. Ce tableau montre que le pourcentage le plus élevé des ressources est celui des IOB (inputs outputs blocks). Cela est dû à la représentation des entrées et des sorties en virgule flottante. Les ressources rares du circuit FPGA qui sont les DSP sont utilisées à moins de 10%.

Tableau.III 1.Ressources du circuit FPGA Virtex7 XC7VX485T-2FFG1761C consommées par le circuit de calcul de la convolution 1

	Ressources					
	Slice	**LUT**	**Flip Flop pairs**	**Muxs**	**DSP**	**IOB**
Utilisée	8633	30197	1891	146	216	289
Disponible	75900	303600	303600	151800	2800	700
Pourcentage	11.37%	9.95%	0.62%	0.1%	7.71%	41.29%

Dans le tableau III.2 nous avons donné les ressources du circuit FPGA cible utilisées par le circuit de calcul de la première couche d'activation ReLu.

Tableau.III 2.Ressources du circuit FPGA Virtex7 XC7VX485T-2FFG1761C consommées par le circuit de calcul de première couche d'activation ReLu 1

	Ressources					
	Slice	**LUT**	**Flip Flop pairs**	**Muxs**	**DSP**	**IOB**
Utilisée	52	128	0	0	0	512
Disponible	75900	303600	303600	151800	2800	700
Pourcentage	0.06%	0.042%	0%	0%	0%	73.14%

A partir du tableau III.2 on peut constater qu'à l'exception des IOBs, la couche de ReLu consomme une portion négligeable des ressources du FPGA Virtex 7.

Le tableau III.3 donne les ressources FPGA consommées par le circuit de calcul de la première couche de max pooling. On remarque que cette couche consomme une portion négligeable de ressources FPGA disponible.

Tableau.III 3.Ressources du circuit FPGA Virtex7 XC7VX485T-2FFG1761C consommées par le circuit de calcul de première couche d'échantillonnage.

	Ressources					
	Slice	**LUT**	**Flip Flop pairs**	**Muxs**	**DSP**	**IOB**
utilisée	**1345**	**509**	**523**	**523**	**0**	**512**
disponible	**75900**	**303600**	**303600**	**151800**	**2800**	**700**
Pourcentage	**1.77%**	**0.17%**	**0.17%**	**0.34%**	**0%**	**73.14%**

Dans le tableau III.4 sont données les ressources consommées par le module de calcul de la deuxième couche de convolution. Les résultats reportés dans le tableau montre que le la deuxième couche de convolution consomme une portion importante de ressources surtout en terme de DSP, de slices et des de LUTs ce qui nous permit de conclure que l'architecture complètent parallèle et en virgule flottante n'est plus convenable pour l'implémentation de la deuxième couche de convolution sur un seul circuit FPGA. Cela est attendu puisque comme mentionné dans le paragraphe III.5.4 cette couche est trop compliquée en comparaison avec la première couche de convolution.

Tableau III.4 Ressources FPGA Virtex7 XC7VX485T-2FFG1761C consommées par le module de calcul de la deuxième couche de convolution.

	Ressources				
	Slice	**LUT**	**Flip Flop pairs**	**Muxs**	**DSP**
utilisée	75900	303600	30256	2336	3456
disponible	75900	303600	303600	151800	3456
Pourcentage	100%	100%	9.96%	1.53%	123%

Les ressources consommées par la deuxième couche d'activation (ReLu 2) et la deuxième couche d'échantillonnage (maxpooling 2) sont pratiquement le double de ressources consommées

par la première couche d'activation (ReLu 1) et la première couche d'échantillonnage (Maxpooling 1) données dans les deux tableaux 3.2 et 3.3 respectivement.

III.7 Conclusion

Dans ce chapitre nous avons présenté les différentes étapes de réalisation de notre projet. Nous avons commencé par l'étape de conception, d'apprentissage et de validation du réseau convolutif utilisé pour la reconnaissance automatique de digits manuscrits. Pour le choix de l'architecture du réseau nous avons cherché un bon compromis entre simplicité et précision. Pour cela nous avons choisi un réseau contenant trois couches de convolution, trois couches de ReLu deux couches de Maxpooling et une seule couche entièrement connectée. Le réseau choisi donne une précision presque égale à 97%.

Après la première étape nous avons passé à l'étape de développement d'un programme Matlab pour faire la classification des images de digits manuscrits.

Dans la troisième étape nous avons fait la conception des circuits qui réalisent les opérations de convolution de la première et de la deuxième couche, de ReLu de la première couche et de Maxpooling de la première couche en utilisant System Generator. Durant cette étape nous avons constaté que les circuits de calcul de convolution sont les plus compliqués.

Dans la dernière étape de notre projet nous avons fait l'implémentation des circuits développés dans l'étape précédente sur FPGA en utilisant le logiciel Vivado. Les rapports d'utilisation de ressources montrent que les circuits de convolution sont les plus exigent en termes de ressources consommées.

Conclusion générale

Ce travail de projet de fin d'étude est une contribution à l'implémentation sur circuit FPGA d'un réseau de neurones convolutifs CNN.

Nous avons commencé notre projet par une étude théorique sur les deux parties principale en relation directe. La première partie concerne la classification des images et les réseaux de neurones convolutifs. La deuxième partie est en relation avec les circuits FPGA, leurs architectures internes et les outils qui facilitent le développement des applications sur FPGA. Cette partie de l'étude théorique nous a permis d'acquérir des connaissances très utiles pour notre domaine de formation.

Dans la deuxième partie de notre travail nous avons fait la conception, la simulation et l'implémentation des différentes couches d'un réseau CNN destiné à l'application de reconnaissance automatique de digits manuscrits. La partie pratique du travail a été achevée avec réussite à travers plusieurs étapes. Nous avons commencé par le choix de l'architecture appropriée du réseau. Ensuite, nous avons fait la conception, l'apprentissage et la validation du réseau en utilisant les fonctions propre de Deep learning ToolBox de Matlab. Dans la troisième étape nous avons développé un programme Matlab pour calculer les sorties des différentes couches du réseau pour bien maitriser l'architecture du réseau. Dans la quatrième phase pratique nous avons fait la conception sous l'environnement system Generator de Xilinx des circuits permettant de calculer les sorties des différentes couches. Dans la dernière partie pratique nous avons passé à l'implémentation sur FPGA des différents circuits que nous avons déjà conçu dans l'étape précédente.

Durant la réalisation du projet nous avons rencontré quelques difficultés surtout durant la conception et l'implémentation. D'abord l'aspect du logiciel System Generator est tout à fait différent par rapport à Simulink ce qui nous a posé beaucoup de problèmes au début. En outre, la simulation et l'implémentation ont été réalisé en utilisant les versions 2018 de Matlab, de System Generator et de Vivado. Ces versions nécessitent un ordinateur performant ce qui n'été pas à notre disposition. Cela a rendait trop longues les simulations et surtout les implémentations.

Comme perspectives de notre travail nous pouvons citer :

- Faire la conception en utilisant la représentation en virgule fixe des données au lieu de la virgule flottante qui nécessite beaucoup de ressources FPGA.
- Intégrer les parties de mise en forme des données dans les circuits de calcul conçus sous System Generator.
- Faire la vérification du résultat de classification sur une carte FPGA.

Bibliographie

[1] Hadjila F. & Bouabdellah R, « Reconnaissance des visages par les réseaux de neurones », mémoire de PFE, Université de Tlemcen, 2003.

[2] Chesner Desir, « Classification Automatique d'Images, Application à l'Imagerie du Poumon Profond », thèse de doctorat, Université de Rouen, 2013.

[3] Fisher, R. «The use of multiple measurements in taxonomic problems», *Annals of Human Genetics*, 1936.

[4] Cortes, C. and Vapnik, V., «Support-vector networks. *Machine learning*», 1995

[5] Dengsheng Zhang, «Fundamentals of Image Data Mining Analysis, Features, Classification and Retrieval», Springer 2021

[6] Giulia Altamura, « Development of hardware accelerators on FPGA for convolutional neural networks », Politecnico di Torino 2018.

[7] Yufei Ma, « Hardware Acceleration of Deep Convolutional Neural Networks on FPGA », Doctor of Philosophy thesis, Arizona State University, USA, 2018.

[8] Ilja Ivanovs, « Implementation Analysis of Convolutional Neural Networks on FPGAs », Master Thesis, University of technology Eindhoven, 2019.

[9] Krizhevsky, A., I. Sutskever and G. E. Hinton, « Imagenet classi_cation with deep convolutional neural networks », in Conf. on Neural Information Processing Systems (NIPS)", 2012.

[10] Khouloud Dahmane, « Analyse d'images par méthode de Deep Learning appliquée au contexte routier en conditions météorologiques dégradées », thèse de doctorat, l'Université Clermont Auvergne, 2020.

[11] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. « Dropout A Simple Way to Prevent Neural Networks from Overfitting », *Journal of Machine Learning Research*, , 2014.

[12] Russakovsky, O., J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg and L. Fei-Fei, « ImageNet Large Scale Visual Recognition Challenge », International Journal of Computer Vision (IJCV), 2015.

[13] Lin, M., Q. Chen and S. Yan, « Network In Network », CoRR URL http://arxiv. org/abs/1312.4400, 2013.

[14] Simonyan, K. and A. Zisserman, « Very deep convolutional networks for large-scale image recognition », URL http://arxiv.org/abs/1409.1556, 2014.

[15] Szegedy, C., W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, « Going deeper with convolutions », in IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)", 2015.

[16] He, K., X. Zhang, S. Ren and J. Sun, « Deep residual learning for image recognition », in IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), 2016.

[17] Gina Smith, « FPGAs 101: Everything You Need to Know to Get Started », Newnes 2010

[18] Xilinx, « 7 Series FPGA Overview », 2011

[19] Xilinx, « 7 Series FPGAs Configurable Logic Block *User Guide*, UG474 (v1.8) », 2016

[20] Xilinx, « 7 Series DSP48E1 Slice *User Guide* UG479 (v1.10) », 2018

[21] Xilinx, « PicoBlaze 8-bit Embedded Microcontroller User Guide *for Extended Spartan®-3 and Virtex®-5 FPGAs Introducing PicoBlaze for Spartan-6, Virtex-6, and 7 Series FPGAs*, UG129 », 2011

[22] Xilinx, « MicroBlaze Processor Reference Guide *Embedded Development Kit EDK 10.1i* UG081», 2008

[23] Xilinx, « Vivado Design Suite User Guide *Getting Started* UG910 (v2018.2) », 2018

[24] https://www.xilinx.com/support/documentation-navigation/design-hubs/dh0014-vivado-system-generator-hub.html consulté le 11/07/2021